OpenShift Container Platform 3.5

Using the OpenShift REST API

Comprehensive guide to using the OpenShift REST API

Last Updated: 2018-03-20

OpenShift Container Platform 3.5 Using the OpenShift REST API

Comprehensive guide to using the OpenShift REST API

Legal Notice

Abstract

Guide to using the OpenShift REST API Guide with end to end examples

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. INTRODUCTION

1.1.1. Using this Guide

This guide provides step-by-step examples for interacting with the OpenShift and Kubernetes REST APIs. Each section contains a description of the resource or function and a set of applicable actions. Each action features a request breakdown that lists the requests types and expected results, followed by applicable example requests and responses to help guide the user through the process. Example configurations for all resources in this guide can be found in the Resource Examples chapter.

This guide does not include complete listings of all available options for each resource object and should be considered a companion to the comprehensive information found in the REST API Reference Guide.

1.1.1.1. Request Types

GET requests are used to retrieve a resource configuration if that resource is specified by name in the target URL.
For example, a GET request to **`{$OCP_CLUSTER}/oapi/namespaces/{$PROJECT}/routes/{$ROUTE}`** returns that **`{$ROUTE}`** configuration.

Alternatively, GET requests to resource collections return the list objects in that resource collection.
For example, a GET request to **`{$OCP_CLUSTER}/oapi/namespaces/{$PROJECT}/routes`** returns a list of the routes in the **`{$PROJECT}`** namespace.

GET requests do not require a request body.

POST requests are used to create new resources based on the configuration in the request body.
For example, a POST request with a new route configuration in the request body to **`{$OCP_CLUSTER}/api/namespaces/{$PROJECT}/routes`** creates a new route in the **`{$PROJECT}`** namespace.

PUT requests are used to update or modify configurations. A PUT request targets specific resources and requires a complete and modified configuration in the request body.
For example, A PUT request to with a complete route configuration that has had one or more field values updated to **`{$OCP_CLUSTER}/oapi/namespaces/{$PROJECT}/routes/{$ROUTE}`** will replace that **`{$ROUTE}`** configuration.

PATCH requests are used to selectively update or modify field values in a resource configuration. How the change is merged is defined per field.

PATCH requests require JSON formatting and **Content-Type:** header **text/strategic-merge-patch+json**.
For example, a PATCH request with the following request body:

```
{
  "spec": {
     "to": {
      "weight": 1
     }
  }
}
```

to **{$OCP_CLUSTER}/oapi/namespaces/{$PROJECT}/routes/{$ROUTE}** will update the **weight** of that route to **1**.

DELETE requests are used to remove resource configurations.
For example, a request to **{$OCP_CLUSTER}/oapi/namespaces/{$PROJECT}/routes/{$ROUTE}** removes that route from the **{$PROJECT}** namespace.

DELETE requests do not require a request body.

1.1.1.2. Request Examples

Request bodies are typically shown in their simplest form. Detailed object configurations can be found in the Resource Examples chapter. Comprehensive information for each API object can be found in the REST API Reference Guide.

Request and response body snippets are used in some examples. These snippets use ellipses (**…**) to delimit the relevant content from the redacted parts of the example configuration.

Request headers use generic **{$BEARER_TOKEN}** and **{$OCP_CLUSTER}** variables in place of the request authentication bearer token hash, and the OpenShift cluster hostname in the request target url, respectively.

An example OpenShift cluster hostname is:

```
https://openshift.example.com:8443
```

Additional variables are included in place of object names. For example, **{$PROJECT}** is commonly used in place of a particular namespace.

Request examples in this guide use YAML for readability, and **application/yaml** is specified in the **Accept:** and **Content-Type:** headers to accommodate this. However, **PATCH** requests require JSON formatting and are documented as such.

1.1.2. API Authentication

Requests to the OpenShift Container Platform API are authenticated using the following methods:

OAuth Access Tokens

- Obtained from the OpenShift Container Platform OAuth server using the **<*master*>/oauth/authorize** and **<*master*>/oauth/token** endpoints.
- Sent as an **Authorization: Bearer…** header
- Sent as an **access_token=…** query parameter for websocket requests prior to OpenShift Container Platform server version 3.6.
- Sent as a websocket subprotocol header in the form **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** for websocket requests in OpenShift Container Platform server version 3.6 and later.

X.509 Client Certificates

- Requires an HTTPS connection to the API server.

- Verified by the API server against a trusted certificate authority bundle.
- The API server creates and distributes certificates to controllers to authenticate themselves.

Any request with an invalid access token or an invalid certificate is rejected by the authentication layer with a 401 error.

If no access token or certificate is presented, the authentication layer assigns the **`system:anonymous`** virtual user and the **`system:unauthenticated`** virtual group to the request. This allows the authorization layer to determine which requests, if any, an anonymous user is allowed to make.

See the REST API Overview for more information and examples.

CHAPTER 2. PREPARING RESOURCES

2.1. PROJECTS

2.1.1. Description of Resource

Projects are the unit of isolation and collaboration in OpenShift. A project has one or more members, a quota on the resources that the project may consume, and the security controls on the resources in the project. Within a project, members may have different roles - project administrators can set membership, editors can create and manage the resources, and viewers can see but not access running containers. In typical clusters, cluster administrators, rather than project administrators, can alter project quotas.

Listing or watching projects will return only projects the user has the reader role on.

An OpenShift project is an alternative representation of a Kubernetes namespace. Projects are exposed as editable to end users while namespaces are not. Direct creation of a project is typically restricted to administrators, while end users should use the **`ProjectRequest`** resource.

2.1.2. Creating a New Project as Administrator

2.1.2.1. Request Breakdown

Creating a new project as an administrator requires one request:

1. A **POST** request to the **`projects`** resource.

The **POST** request to the **`projects`** resource uses a **`Project`** configuration in the request body.

2.1.2.2. POST Request to Create a New Project

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/projects
```

Request Body

```
apiVersion: v1
kind: Project
metadata:
  name: {$PROJECT_NAME}
```

2.1.3. Creating a New Project as User

2.1.3.1. Request Breakdown

Creating a new project as a user requires one request:

1. A **POST** request to the **`projectrequests`** resource.

The **POST** request uses a **`ProjectRequest`** configuration in the request body.

2.1.3.2. POST Request to Create a New Project

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/projectrequests
```

Request Body

```
apiVersion: v1
kind: ProjectRequest
metadata:
  name: {$PROJECT_NAME}
```

2.1.4. Listing all Projects in an Environment

2.1.4.1. Request Breakdown

Listing all the projects in an environment requires one request:

1. A **GET** request to the **`projects`** resource.

The **GET** request returns a **`ProjectList`** configuration.

2.1.4.2. GET Request to List all Projects

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/projects
```

2.1.5. Listing a Specific Project

2.1.5.1. Request Breakdown

Listing the details for a project requires three requests:

1. A **GET** request to the project name in the **`projects`** resource.

2. A **GET** request to the **`resourcequotas`** subresource of the project namespace.

3. A **GET** request to the **limitranges** subresource of the project namespace.

These requests return the **Project**, **ResourceQuotaList**, and **LimitRangeList** configurations respectively for the project namespace.

2.1.5.2. GET Request to List the Project Configuration

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/projects
```

2.1.5.3. GET Request to List the ResourceQuotaList Configuration

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/resourcequotas
```

2.1.5.4. GET Request to List the LimitRangeList Configuration

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/limitranges
```

2.1.6. Deleting a Project

2.1.6.1. Request Breakdown

Deleting a project requires one request:

1. A **DELETE** request to the project namespace in the **projects** resource of the namespace.

The **DELETE** request returns a **code: 200** and the project is deleted.

2.1.6.2. DELETE Request to Delete a Project

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/projects/{$PROJECT}
```

2.2. SERVICE ACCOUNTS

2.2.1. Description of Resource

ServiceAccount binds together:

- a name, understood by users, and perhaps by peripheral systems, for an identity
- a principal that can be authenticated and authorized
- a set of secrets

2.2.2. Creating a New Service Account

2.2.2.1. Request Breakdown

Creating a new service account requires one request:

1. A **POST** request to the **`serviceaccounts`** subresource of the namespace.

The **POST** request uses a **`ServiceAccount`** configuration in the request body.

2.2.2.2. POST Request to Create a New Service Account

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts
```

Request Body

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {$SERVICEACCOUNT}
```

2.2.3. Listing All Service Accounts in an Environment

2.2.3.1. Request Breakdown

Listing all the service accounts in an environment requires one request:

1. A **GET** request to the **`serviceaccounts`** resource.

The **GET** request returns the **`ServiceAccountList`**, listing the details of all service accounts in the environment.

2.2.3.2. GET Request to Return All Service Accounts in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/serviceaccounts
```

2.2.4. Listing all Service Accounts in a Namespace

2.2.4.1. Request Breakdown

Listing all the service accounts in a namespace requires one request:

1. A **GET** request to the **`serviceaccounts`** subresource of the namespace.

The **GET** request returns the **`ServiceAccountList`**, listing the details of all service accounts in the namespace.

2.2.4.2. GET Request to Return Service Accounts in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts
```

2.2.5. Listing a Specific Service Account Configuration

2.2.5.1. Request Breakdown

Listing a specific service account configuration requires one request:

1. A **GET** request to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **GET** request returns the **`ServiceAccount`** configuration for the specified service account.

2.2.5.2. GET Request to Return Specific Service Account Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

2.2.6. Adding a Role to a Service Account

2.2.6.1. Request Breakdown

Adding a role to a service account requires two requests, where the second request type is dependent on the response of the first:

1. A **GET** request to the **`policybindings`** subresource of the namespace.
2. A **POST** request to the **`rolebindings`** subresource of the namespace (if the role has not been previously bound to an object in the namespace) +. **OR**
3. A **PUT** request to the role name in the **`rolebindings`** subresource of the namespace.

The **GET** request returns the **`PolicyBindingList`**, listing all the **`RoleBinding`** configurations in the namespace.

Create a **`RoleBinding`** with a **POST** request to bind an object to a role. Once the role binding exists in the namespace, binding another object to that role requires modification of the **`RoleBinding`** with a **PUT** request.

2.2.6.2. GET Request to Return PolicyBindingList in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/policybindings
```

The **GET** request returns a list of the role bindings that are present in the namespace.

Response Body Snippet

```
...
RoleBindings:
  name: {$ROLE}
  roleBinding:
    metadata:
      name: {$ROLE}
      namespace: {$PROJECT}
      uid: {$UID}
```

```
      resourceVersion: "{$VERSION_NUMBER}"
    roleRef:
      name: {$ROLE}
    subjects:
    - kind: ServiceAccount
      name: {$SERVICEACCOUNT}
      namespace: {$PROJECT}
    userNames:
    - system:serviceaccount:{$PROJECT}:{$SERVICEACCOUNT}
...
```

If the intended role is not listed, send a **POST** request to create the **`RoleBinding`** configuration. Include the service account as a **`subject`** for the role, and the **`system:serviceaccount:{$PROJECT}:{$SERVICEACCOUNT}`** user name that distinguishes the service account.

If the role is listed, as in the response in the example above, update the returned **`RoleBinding`** configuration with the **`subject`** and **`userName`** of the service account being added to the role and send it as a **PUT** request to the role name in the **`rolebindings`** subresource of the namespace.

2.2.6.3. POST Request to Create a RoleBinding in a Namespace

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/rolebindings
```

Request Body

```
kind: RoleBinding
metadata:
  name: {$ROLE}
  namespace: {$PROJECT}
roleRef:
  name: {$ROLE}
subjects:
- kind: ServiceAccount
  name: {$SERVICEACCOUNT}
  namespace: {$PROJECT}
userNames:
- system:serviceaccount:{$PROJECT}:{$SERVICEACCOUNT}
```

2.2.6.4. PUT Request to Add a Service Account to an Existing Role

Request Header

```
curl -k -v \
-X PUTv \
-H "Authorization: {$BEARER_TOKEN}" \
```

```
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/rolebindings/{$ROLE}
```

Request Body

```
kind: RoleBinding
metadata:
  name: {$ROLE}
  namespace: {$PROJECT}
  uid: {$UID}
  resourceVersion: "{$VERSION_NUMBER}"
roleRef:
  name: {$ROLE}
subjects:
- kind: ServiceAccount
  name: {$SERVICEACCOUNT}
  namespace: {$PROJECT}
subjects:
- kind: ServiceAccount
  name: {$SERVICEACCOUNT-2}
  namespace: {$PROJECT}
userNames:
- system:serviceaccount:{$PROJECT}:{$SERVICEACCOUNT}
- system:serviceaccount:{$PROJECT}:{$SERVICEACCOUNT-2}
```

2.2.7. Linking a Secret to a Service Account

2.2.7.1. Request Breakdown

Linking a secret to a service account requires two requests:

1. A **GET** request to the service account name in the **`serviceaccounts`** subresource of the namespace.
2. A **PUT** request with updated **`secrets`** parameter to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **GET** request returns the **`ServiceAccount`** configuration.

The **PUT** request uses the returned **`ServiceAccount`** configuration with an updated **`secrets`** parameter to include the secret to link to the service account.

2.2.7.2. GET Request to Return Service Account Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
...
```

2.2.7.3. PUT Request to Add Secret to Service Account Configuration

Request Header

```
curl -k -v \
-X PUT  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
- name: {$SECRET_NAME}
...
```

2.2.8. Unlinking a Secret from a Service Account

2.2.8.1. Request Breakdown

Unlinking a secret from a service account requires two requests:

1. A **GET** request to the service account name in the **`serviceaccounts`** subresource of the namespace.
2. A **PUT** request with modified **`secrets`** parameter to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **GET** request returns the **`ServiceAccount`** configuration.

The **PUT** request uses the returned **`ServiceAccount`** configuration with a modified **`secrets`** parameter to unlink the secret from the service account.

2.2.8.2. GET Request to Return Service Account Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
```

```
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
- name: {$SECRET_NAME}
...
```

2.2.8.3. PUT Request to Remove Secret from Service Account Configuration

Request Header

```
curl -k -v \
-X PUT  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
...
```

2.2.9. Deleting a Service Account

2.2.9.1. Request Breakdown

Deleting a service account requires one request:

1. A **DELETE** request to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the service account is deleted.

2.2.9.2. DELETE Request to Delete Service Account

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
```

```
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

2.3. SECRETS

2.3.1. Description of Resource

The **`Secret`** resource type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, **`dockercfg`** files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods.

2.3.2. Creating Secrets from File

2.3.2.1. Request Breakdown

Creating a new secret requires one request:

1. A **POST** request to the **`secrets`** subresource of the namespace.

The **POST** request to the **`secret`** subresource uses a **`Secret`** configuration in the request body.

2.3.2.2. POST Request to Create a New Secret

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/secrets
```

Request Body

```
apiVersion: v1
kind: Secret
metadata:
  name: {$SECRET}
  namespace: {$PROJECT}
data:
  username: ZGVtby11c2Vy
  password: cGFzc3dvcmQ=
  ca.crt: {$BASE64-ENCODED-CERTIFICATE}
type: kubernetes.io/basic-auth
```

Table 2.1. data Parameter Descriptions

Parameter Name	Description	Additional Notes

Parameter Name	Description	Additional Notes
{$FILENAME}	Base64-encoded contents of file, where the {$FILENAME} parameter is the plaintext file name	If the file is already encrypted, the **type** field for the secret needs a value of 'Opaque'. See the example secret configuration for an example.
username	Base64-encoded user name for basic authentication	
password	Base64-encoded password for the supplied basic authentication user name	
ca.crt	Base64-encoded contents of certificate file	
ssh-privatekey	Base64-encoded contents of SSH private key file	

2.3.3. Listing Secrets in an Environment

2.3.3.1. Request Breakdown

Listing all the secrets in an environment requires one request:

1. A **GET** request to the **secrets** resource.

The **GET** request to the **secrets** resource returns returns the **SecretList**, listing the details for all secrets in the environment.

2.3.3.2. GET Request to Return All Secrets in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/secrets
```

2.3.4. Listing Secrets in a Namespace

2.3.4.1. Request Breakdown

Listing all the secrets in a namespace requires one request:

1. A **GET** request to the **secrets** subresource of the namespace.

The **GET** request to the **secrets** resource returns the **SecretList**, listing the details of all secrets in the namespace.

2.3.4.2. GET Request to Return Secrets in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/secrets
```

2.3.5. Listing a Specific Secret Configuration

2.3.5.1. Request Breakdown

Listing a specific secret configuration requires one request:

1. A **GET** request to the secret name in the **secrets** subresource of the namespace.

The **GET** request returns the **Secret** configuration for the specified secret.

2.3.5.2. GET Request to Return Specific Secret Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/secrets/{$SECRET}
```

2.3.6. Linking a Secret to a Service Account

2.3.6.1. Request Breakdown

Linking a secret to a service account requires two requests:

1. A **GET** request to the service account name in the **serviceaccounts** subresource of the namespace.

2. A **PUT** request with updated **secrets** parameter to the service account name in the **serviceaccounts** subresource of the namespace.

The **GET** request returns the **ServiceAccount** configuration.

The **PUT** request uses the returned **`ServiceAccount`** configuration with an updated **`secrets`** parameter to include the secret to link to the service account.

2.3.6.2. GET Request to Return Service Account Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
...
```

2.3.6.3. PUT Request to Add Secret to Service Account Configuration

Request Header

```
curl -k -v \
-X PUT  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
- name: {$SECRET_NAME}
...
```

2.3.7. Unlinking a Secret from a Service Account

2.3.7.1. Request Breakdown

Unlinking a secret from a service account requires two requests:

1. A **GET** request to the service account name in the **`serviceaccounts`** subresource of the namespace.

2. A **PUT** request with modified **`secrets`** parameter to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **GET** request returns the **`ServiceAccount`** configuration.

The **PUT** request uses the returned **`ServiceAccount`** configuration with a modified **`secrets`** parameter to unlink the secret from the service account.

2.3.7.2. GET Request to Return Service Account Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
- name: {$SECRET_NAME}
...
```

2.3.7.3. PUT Request to Remove Secret from Service Account Configuration

Request Header

```
curl -k -v \
-X PUT  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

Request Body Snippet

```
...
secrets:
- name: {$SERVICEACCOUNT}-token-x4tx5
- name: {$SERVICEACCOUNT}-dockercfg-7qfh6
...
```

2.3.8. Deleting a Service Account

2.3.8.1. Request Breakdown

Deleting a service account requires one request:

1. A **DELETE** request to the service account name in the **`serviceaccounts`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the service account is deleted.

2.3.8.2. DELETE Request to Delete Service Account

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/serviceaccounts/{$SERVICEACCOU
NT}
```

2.3.9. Deleting Secrets

2.3.9.1. Request Breakdown

Deleting a secret requires one request:

1. A **DELETE** request to the secret name in the **`secret`** subresource of the namespace.

The *DELETE request returns a **`code: 200`** and the secret is deleted.

2.3.9.2. DELETE Request to Delete Secret

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/secrets/{$SECRET}
```

2.4. PERSISTENT VOLUMES

2.4.1. Description of Resource

PersistentVolume (PV) is a storage resource provisioned by an administrator. It is analogous to a node.

2.4.2. Creating a New Persistent Volume

2.4.2.1. Request Breakdown

Creating a new persistent volume requires one request:

1. A **POST** request to the **`persistentvolumes`** resource.

The **POST** request uses a **`PersistentVolume`** configuration in the request body.

2.4.2.2. POST Request to Create a Persistent Volume

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/persistentvolumes
```

Request Body

```
kind: PersistentVolume
metadata:
  name: {$VOLUME}
spec:
  capacity:
    storage: {$SIZE}Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: {$/STORAGE/FILE/PATH}
    server: {$STORAGE_SERVER_HOSTNAME}
```

2.4.3. Listing All Persistent Volumes in an Environment

2.4.3.1. Request Breakdown

Listing all the persistent volumes in an environment requires one request:

1. A **GET** request to the **`persistentvolumes`** resource.

The **GET** request returns returns the **`PersistentVolumeList`**, listing the details of all persistent volumes in the environment.

2.4.3.2. GET Request to Return All Persistent Volumes in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/persistentvolumes
```

2.4.4. Listing a Specific Persistent Volume Configuration

2.4.4.1. Request Breakdown

Listing a specific persistent volumes configuration requires one request:

1. A **GET** request to the persistent volume name in the **`persistentvolumes`** resource.

The **GET** request returns the **`PersistentVolume`** configuration for the specified persistent volume.

2.4.4.2. GET Request to Return Specific Persistent Volume Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/persistentvolumes/{$VOLUME}
```

2.4.5. Deleting a Persistent Volume

2.4.5.1. Request Breakdown

Deleting a persistent volume requires one request:

1. A **DELETE** request to the persistent volume name in the **`persistentvolumes`** resource.

The **DELETE** request returns a **`code: 200`** and the persistent volume is deleted.

2.4.5.2. DELETE Request to Delete Persistent Volume

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/persistentvolumes/{$VOLUME}
```

2.5. PERSISTENT VOLUME CLAIMS

2.5.1. Description of Resource

PersistentVolumeClaim is a user's request for and claim to a persistent volume.

2.5.2. Creating a New Persistent Volume Claim

2.5.2.1. Request Breakdown

Creating a new persistent volume claim requires one request:

1. A **POST** request to the **`persistentvolumeclaims`** subresource of the namespace.

The **POST** request uses a **`PersistentVolumeClaim`** configuration in the request body.

2.5.2.2. POST Request to Create a Persistent Volume Claim

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/persistentvolumeclaims
```

Request Body

```
kind: PersistentVolumeClaim
metadata:
  name: {$CLAIM}
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: {$SIZE}Gi
```

2.5.3. Listing All Persistent Volume Claims in an Environment

2.5.3.1. Request Breakdown

Listing all the persistent volume claims in an environment requires one request:

1. A **GET** request to the **`persistentvolumeclaims`** resource.

The **GET** request returns the **`PersistentVolumeClaimList`**, listing the details of all persistent volume claims in the environment.

2.5.3.2. GET Request to Return All Persistent Volume Claims in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/persistentvolumeclaims
```

2.5.4. Listing all Persistent Volume Claims in a Namespace

2.5.4.1. Request Breakdown

Listing all the persistent volume claims in a namespace requires one request:

1. A **GET** request to the **`persistentvolumeclaims`** subresource of the namespace.

The **GET** request returns the **`PersistentVolumeClaimList`**, listing the details of all persistent volume claims in the namespace.

2.5.4.2. GET Request to Return Persistent Volume Claims in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/persistentvolumeclaims
```

2.5.5. Listing a Specific Persistent Volume Claim Configuration

2.5.5.1. Request Breakdown

Listing a specific persistent volume claim configuration requires one request:

1. A **GET** request to the persistent volume claim name in the **`persistentvolumeclaims`** subresource of the namespace.

The **GET** request returns the **`PersistentVolumeClaim`** configuration for the specified persistent volume claim.

2.5.5.2. GET Request to Return Specific Persistent Volume Claim Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/persistentvolumeclaims/{$CLAIM
}
```

2.5.6. Adding a Persistent Volume Claim to an Object

Persistent volume claims can be added to objects that have a pod template: deploymentconfigs, replication controllers, and pods. However, you can not change a pod's volumes once it has been created.

If you alter a volume setting on a deployment config, a deployment will be triggered. Changing a replication controller will not affect running pods.

Adding a persistent volume claim to an object configuration requires two requests:

1. A **GET** request to the object name in the **{$OBJECT}** subresource of the namespace.
2. A **PATCH** request with updated field values to the object name in the **{$OBJECT}** subresource of the namespace.

The **GET** request is optional as the **PATCH** request body is not determined by the returned configuration, however it can be useful for preparing the **PATCH** request body.

The syntax for the **PATCH** request is the same for both **deploymentconfig** and **replicationcontroller** objects.

2.5.6.1. PATCH Request to Add Persistent Volume Claim

NOTE

PATCH requests require JSON formatting.

The following **PATCH** request example adds a persistent volume claim {$CLAIM}, which is bound to persistent volume {$VOLUME), to **deploymentconfig** {$DEPLOYMENTCONFIG}. The same **PATCH** can be applied to a **replicationcontroller** object with modified target:

Request Header

```
curl -k -v \
-X PATCH \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/json" \
-H "Content-Type: text/strategic-merge-patch+json" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfig/{$DEPLOYMENT
CONFIG}
```

Request Body

```
{
    "spec": {
        "template": {
            "spec": {
                "volumes": [
                    {
                        "name": "{$VOLUME}",
                        "persistentVolumeClaim": {
                            "claimName": "{$CLAIM}"
                        }
                    }
                ]
            }
        }
    }
}
```

2.5.7. Removing a Persistent Volume Claim from an Object

If you alter a volume setting on a deployment config, a deployment will be triggered. Changing a replication controller will not affect running pods.

Removing a persistent volume claim from an object configuration requires two requests:

1. A **GET** request to the object name in the **`{$OBJECT}`** subresource of the namespace.
2. A **PATCH** request with a **`delete $patch`** type to the object name in the **`{$OBJECT}`** subresource of the namespace.

The **GET** request is optional as the **PATCH** request body is not determined by the returned configuration, however it can be useful for preparing the **PATCH** request body.

The syntax for the **PATCH** request is the same for both **`deploymentconfig`** and **`replicationcontroller`** objects.

2.5.7.1. PATCH Request to Add Persistent Volume Claim

NOTE

PATCH requests require JSON formatting.

The following **PATCH** request example adds a persistent volume claim {$CLAIM}, which is bound to persistent volume {$VOLUME), to **`deploymentconfig`** {$DEPLOYMENTCONFIG}. The same **PATCH** can be applied to a **`replicationcontroller`** object with modified target:

Request Header

```
curl -k -v \
-X PATCH \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/json" \
-H "Content-Type: text/strategic-merge-patch+json" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfig/{$DEPLOYMENT
CONFIG}
```

Request Body

```
{
    "spec": {
        "template": {
            "spec": {
                "volumes": [
                    {
                        "$patch": "delete",
                        "name": "{$VOLUME}"
                    }
                ]
            }
        }
    }
}
```

2.5.8. Deleting a Persistent Volume Claim

2.5.8.1. Request Breakdown

Deleting a persistent volume claim requires one request:

1. A **DELETE** request to the persistent volume claim name in the **`persistentvolumeclaims`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the persistent volume claim is deleted.

2.5.8.2. DELETE Request to Delete Persistent Volume Claim

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/persistentvolumeclaims/{$CLAIM
}
```

CHAPTER 3. BUILDS AND IMAGES

3.1. BUILDCONFIGS

3.1.1. Description of Resource

Build configurations define a build process for new Docker images. There are three types of builds possible - a Docker build using a Dockerfile, a Source-to-Image build that uses a specially prepared base image that accepts source code that it can make runnable, and a custom build that can run arbitrary Docker images as a base and accept the build parameters. Builds run on the cluster and on completion are pushed to the Docker registry specified in the "output" section. A build can be triggered via a webhook, when the base image changes, or when a user manually requests a new build be created.

Each build created by a build configuration is numbered and refers back to its parent configuration. Multiple builds can be triggered at once. Builds that do not have "output" set can be used to test code or run a verification build.

3.1.2. Creating a New BuildConfig

3.1.2.1. Request Breakdown

Creating a new buildconfig requires one request:

1. A **POST** request to the **`buildconfigs`** subresource of the namespace.

The **POST** request uses a **`BuildConfig`** configuration in the request body. The following example uses a **`BuildConfig`** for the Jboss Developer **`kitchensink`** quickstart found at https://github.com/jboss-developer/jboss-eap-quickstarts.

3.1.2.2. POST Request to Create a New BuildConfig

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/buildconfigs
```

Request Body

```
apiVersion: v1
kind: BuildConfig
metadata:
  labels:
    app: {$PROJECT}
  name: {$BUILD}
spec:
  runPolicy: Serial
  source:
    contextDir: kitchensink
    git:
```

```
      ref: 7.1.0.Beta
      uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
    type: Git
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jboss-eap70-openshift:1.4
        namespace: openshift
    type: Source
  triggers:
  - github:
      secret: 8d1d7bbf3313324f
    type: GitHub
  - generic:
      secret: 2a20ab9ac1316fcf
    type: Generic
  - imageChange: {}
    type: ImageChange
  - type: ConfigChange
```

3.1.3. Listing All BuildConfigs in an Environment

3.1.3.1. Request Breakdown

Listing all the buildconfigs in an environment requires one request:

1. A **GET** request to the **`buildconfigs`** resource.

The **GET** request returns the **`BuildConfigList`**, listing the details of all buildconfigs in the environment.

3.1.3.2. GET Request to Return All BuildConfigs in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/buildconfigs
```

3.1.4. Listing all BuildConfigs in a Namespace

3.1.4.1. Request Breakdown

Listing all the buildconfigs in a namespace requires one request:

1. A **GET** request to the **`buildconfigs`** subresource of the namespace.

The **GET** request returns the **`BuildConfigList`**, listing the details of all buildconfigs in the namespace.

3.1.4.2. GET Request to Return BuildConfigs in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/buildconfigs
```

3.1.5. Listing a Specific BuildConfig Configuration

3.1.5.1. Request Breakdown

Listing a specific buildconfig configuration requires one request:

1. A **GET** request to the buildconfig name in the **`buildconfigs`** subresource of the namespace.

The **GET** request returns the **`BuildConfig`** configuration for the specified build.

3.1.5.2. GET Request to Return Specific BuildConfig Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/buildconfigs/{$BUILDCONFIG}
```

3.1.6. Deleting a BuildConfig

3.1.6.1. Request Breakdown

Deleting a buildconfig requires one request:

1. A **DELETE** request to the buildconfig name in the **`buildconfigs`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the buildconfig is deleted.

3.1.6.2. DELETE Request to Delete BuildConfig

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/buildconfigs/{BUILDCONFIG}
```

3.2. BUILDS

3.2.1. Description of Resource

A build is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A BuildConfig object is the definition of the entire build process.

3.2.2. Listing All Builds in an Environment

3.2.2.1. Request Breakdown

Listing all the builds in an environment requires one request:

1. A **GET** request to the **`builds`** resource.

The **GET** request returns the **`BuildList`**, listing the details of all builds in the environment.

3.2.2.2. GET Request to Return All Builds in an Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/builds
```

3.2.3. Listing all Builds in a Namespace

3.2.3.1. Request Breakdown

Listing all the builds in a namespace requires one request:

1. A **GET** request to the **`builds`** subresource of the namespace.

The **GET** request returns the **`BuildList`**, listing the details of all builds in the namespace.

3.2.3.2. GET Request to Return Builds in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/builds
```

3.2.4. Listing a Specific Build Configuration

3.2.4.1. Request Breakdown

Listing a specific build configuration requires one request:

1. A **GET** request to the build name in the **`builds`** subresource of the namespace.

The **GET** request returns the **`Build`** configuration for the specified build.

3.2.4.2. GET Request to Return Specific Build Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/builds/{$BUILD}
```

3.2.5. Starting a Build

3.2.5.1. Request Breakdown

Starting a build requires one request:

1. A **POST** request to the **`instantiate`** subresource of the **`BuildConfig`** name.

The **POST** request returns a **`BuildConfig`** configuration.

3.2.5.2. POST Request to Start a Build

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/buildconfigs/{$BUILDCONFIG}
```

Request Body

```
kind: BuildRequest
apiVersion: v1
metadata:
  name: {$BUILDCONFIG}
triggeredBy:
- message: Manually triggered
```

3.2.6. Cancelling a Build

3.2.6.1. Request Breakdown

Cancelling a build consists of two requests:

1. A **GET** request to the build name in the **`builds`** subresource of the namespace.
2. A **PUT** request with an added **`cancelled`** field to the build name in the **`builds`** subresource of the namespace.

The **GET** request returns a **`Build`** configuration.
The **GET** request is optional if an updated **`Build`** configuration can otherwise be provided for the **PUT** request.

Include a **`spec: status: cancelled`** parameter with value **`true`** in the **`Build`** configuration. Send the updated configuration as the request body in a **PUT** request to the build name.

3.2.6.2. GET Request to Return Build Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/builds/{$BUILD}
```

3.2.6.3. PUT Request to Cancel a Build

Request Header

```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/builds/{$BUILD}
```

Request Body Snippet

```
...
spec:
  status:
    cancelled: "true"
...
```

3.2.7. Deleting a Build

3.2.7.1. Request Breakdown

Deleting a build requires one request:

1. A **DELETE** request to the build name in the **`builds`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the build is deleted.

3.2.7.2. DELETE Request to Delete Build

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/builds/{$BUILD}
```

3.3. IMAGES

3.3.1. Description of Resource

An image is a binary that includes all of the requirements for running a single container, as well as metadata describing its needs and capabilities.

3.3.2. Creating an Image

3.3.2.1. Request Breakdown

Creating a new image requires one request:

1. A **POST** request to the **`images`** subresource of the namespace.

The **POST** request uses an **`Image`** configuration in the request body.

3.3.2.2. POST Request to Create a New Image

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/images
```

Request Body

```
apiVersion: v1
kind: Image
metadata:
  name: {$IMAGE}
dockerImageReference: registry.access.redhat.com/openshift3/jenkins-1-
rhel7@sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e18
7a0
```

3.3.3. Listing All Images in a Cluster

3.3.3.1. Request Breakdown

Listing all the images in a cluster requires one request:

1. A **GET** request to the **`images`** resource.

The **GET** request returns the **`ImageList`**, listing the details of all images in the cluster.

3.3.3.2. GET Request to Return All Images in a Cluster

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/images
```

3.3.4. Listing a Specific Image Configuration

3.3.4.1. Request Breakdown

Listing a specific image configuration in a cluster requires one request:

1. A **GET** request to the **`sha256:`**-prefixed image hash in the **`images`** resource.

The **GET** request returns the **`Image`** configuration for the specified image.

3.3.4.2. GET Request to Return Specific Image Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/images/sha256:{$IMAGEHASH}
```

3.3.5. Deleting an Image

3.3.5.1. Request Breakdown

Deleting an Image requires one request:

1. A **DELETE** request to the **`sha256:`**-prefixed image hash in the **`images`** resource.

The **DELETE** request returns a **`code: 200`** and the Image is deleted.

3.3.5.2. DELETE Request to Delete Image

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/images/sha256:{$IMAGEHASH}
```

3.4. IMAGE STREAMS

3.4.1. Description of Resource

An ImageStream stores a mapping of tags to images, metadata overrides that are applied when images are tagged in a stream, and an optional reference to a Docker image repository on a registry.

3.4.2. Creating a New Image Stream

3.4.2.1. Request Breakdown

Creating a new image stream requires one request:

1. A **POST** request to the **`imagestreams`** subresource of the namespace.

The **POST** request uses an **`ImageStream`** configuration in the request body.

3.4.2.2. POST Request to Create a New Image Stream

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/imagestreams
```

Request Body

```
kind: ImageStream
apiVersion: v1
metadata:
  name: test
  annotations:
    openshift.io/display-name: Test
spec:
  tags:
  - name: 1.0
    annotations:
      openshift.io/display-name: test (1.0)
      description: Vague description of this early test image.
```

```
      iconClass: icon-ruby
      tags: test
      sampleRepo: https://github.com/openshift/ruby-ex.git
    from:
      kind: ImageStreamTag
      name: '1.0'
```

3.4.3. Importing an Image Stream

3.4.3.1. Request Breakdown

Importing an image stream requires one request:

1. A **POST** request to the **`imagestreamimports`** subresource of the namespace.

The **POST** request uses an **`ImageStreamImport`** configuration in the request body.

The following example imports the **`ruby-23-rhel7:latest`** image into a namespace as a custom imagestream.

3.4.3.2. POST Request to Create a New Image Stream

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/imagestreamimports
```

Request Body

```
kind: ImageStreamImport
apiVersion: v1
metadata:
  name: {$IMAGESTREAM}
spec:
  import: true
  images:
  - from:
      kind: DockerImage
      name: registry.access.redhat.com/rhscl/ruby-23-rhel7:latest
    to:
      name: '{$IMAGESTREAMTAG}'
```

3.4.4. Listing All Image Streams in a Cluster

3.4.4.1. Request Breakdown

Listing all the image streams in a cluster requires one request:

1. A **GET** request to the **`imagestreams`** resource.

The **GET** request returns the **`ImageStreamList`**, listing the details of all image streams in the cluster.

3.4.4.2. GET Request to Return All Image Streams in a Cluster

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/imagestreams
```

3.4.5. Listing all Image Streams in a Namespace

3.4.5.1. Request Breakdown

Listing all the image streams in a namespace requires one request:

1. A **GET** request to the **`imagestreams`** subresource of the namespace.

The **GET** request returns the **`ImageStreamList`**, listing the details of all image streams in the namespace.

3.4.5.2. GET Request to Return Image Streams in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/imagestreams
```

3.4.6. Listing a Specific Image Stream Configuration

3.4.6.1. Request Breakdown

Listing a specific image stream configuration requires one request:

1. A **GET** request to the image stream name in the **`imagestreams`** subresource of the namespace.

The **GET** request returns the **`ImageStream`** configuration for the specified image stream.

3.4.6.2. GET Request to Return Specific Image Stream Configuration

Request Header

```
curl -k -v \
-X GET  \
```

```
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/imagestreams/{$IMAGESTREAM}
```

3.4.7. Deleting an Image Stream

3.4.7.1. Request Breakdown

Deleting a image stream requires one request:

1. A **DELETE** request to the image stream name in the **`imagestreams`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the image stream is deleted.

3.4.7.2. DELETE Request to Delete Image Stream

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/imagestreams/{$IMAGESTREAM}
```

3.5. IMAGE STREAM TAGS

3.5.1. Description of Resource

An ImageStreamTag represents an Image that is retrieved by tag name from an ImageStream.

3.5.2. Adding an Image Stream Tag

3.5.2.1. Request Breakdown

Adding a tag to an image stream requires one request:

1. A **PUT** request to the image stream tag name in the **`imagestreamtags`** subresource of the namespace.

The **PUT** request uses an **`ImageStreamTag`** configuration in the request body.

The following example adds an image stream tag to the default **`ruby`** image stream in the **`openshift`** namespace.

3.5.2.2. PUT Request to Add an Image Stream Tag

Request Header

```
curl -k -v \
```

```
-X PUT  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/openshift/imagestreamtags/ruby:
{$IMAGESTREAMTAG}
```

Request Body

```
kind: ImageStreamTag
metadata:
  name: ruby:{$IMAGESTREAMTAG}
tag:
  from:
    kind: ImageStreamImage
    namespace: openshift
    name:
ruby@sha256:9cfdf4b811ace13d4c555335b249ab831832a384113035512abc9d4d5cc597
16
```

3.5.3. Deleting an Image Stream Tag

3.5.3.1. Request Breakdown

Deleting an image stream tag requires one request:

1. A **DELETE** request to the image stream tag name in the **`imagestreamtags`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the image stream tag is deleted.

The following example deletes the image stream tag, added in the previous example, from the default **`ruby`** image stream in the **`openshift`** namespace.

3.5.3.2. DELETE Request to Delete Image Stream Tag

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/openshift/imagestreamtags/ruby:
{$IMAGESTREAMTAG}
```

CHAPTER 4. PROJECT MANAGEMENT

4.1. POD CONFIGURATIONS

4.1.1. Description of Function

Display the **Pod** configurations of pods in an OpenShift cluster. Depending on the endpoint of the request, you can list the **Pod** configurations for all the pods in an environment, all the pods in a namespace, or a specific **Pod** configuration in a namespace.

4.1.2. Listing Configurations for All Pods in an Environment

4.1.2.1. Request Breakdown

Listing all the pods in an environment requires one request:

1. A **GET** request to the **pods** resource.

The **GET** request returns the **PodList**, listing the details of all pods in the environment.

4.1.2.2. GET Request to Return All Pod Configurations in OpenShift Environment

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/pods
```

4.1.3. Listing All Pods in a Namespace

4.1.3.1. Request Breakdown

Listing all the pods in a namespace requires one request:

1. A **GET** request to the **pods** subresource of the namespace.

The **GET** request returns the **PodList**, listing the details of all pods in the namespace.

4.1.3.2. GET Request to Return All Pod Configurations in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/pods
```

4.1.4. Listing a Specific Pod Configuration in a Namespace

4.1.4.1. Request Breakdown

Listing a specific pod configuration requires one request:

1. A **GET** request to the pod name in the **pods** subresource of the namespace.

The **GET** request returns the **Pod** configuration for the specified pod.

4.1.4.2. GET Request to Return Specific Pod Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/pods/{$POD}
```

4.2. IDLE

4.2.1. Description of Function

Idling discovers the scalable resources (such as deployment configurations and replication controllers) associated with a series of services by examining the endpoints of the service. Each service is then marked as idled, the associated resources are recorded, and the resources are scaled down to zero replicas.

Upon receiving network traffic, the services (and any associated routes) will "wake up" the associated resources by scaling them back up to their previous scale.

4.2.2. Idling a DeploymentConfig

4.2.2.1. Request Breakdown

Idling consists of a series of requests:

1. A **GET** request to the endpoint name of the **endpoints** subresource of the namespace to discover corresponding pods.
2. A **GET** request to each pod name in the **pods** subresource to discover the owner reference (**deploymentconfig** or **replicationcontroller**) for that pod.
3. A **GET** request to the referenced owner name (**deploymentconfig** or **replicationcontroller**) to discover if that referenced owner has an owner reference.
4. A **GET** request to the **scale** subresource for all referenced **deploymentconfig** or **replicationcontroller** objects to discover the replica value for the resource.
5. A **PATCH** request with updated annotations, including idle start time and corresponding owner reference, to each endpoint name in the **endpoint** subresource of the namespace.

6. A **PUT** request with updated annotations for the idle time and previous scale to the endpoint owner name (`deploymentconfig` or `replicationcontroller`) to set the replicas to `0`.

Idling an endpoint requires a **PATCH** request to the endpoint, and a **PUT** request to the referenced owner of the resources. **GET** requests 1-3 discover the pods corresponding to the endpoint, the owner resource of the pods, and then discover if that owner resource is owned by another resource.

GET request 4 is optional as the replica value for the resource is in the configuration returned in a previous **GET** request; however, returning the scale subresource can simplify the process of retrieving the replica value.

The **PATCH** request adds annotations to the endpoint that include the time in which the resource is considered idle, and an owner reference so that the owner resource is scaled accordingly when a service receives traffic. The **PUT** request to the owner resource includes the same idle time annotation as the endpoint, the previous replica value of the resource to scale to when unidled, and replaces the replica value to 0 to idle the resource.

Table 4.1. Annotations for Determining Resource Owner

Annotation	Description
kubernetes.io/created-by	Owner reference for Kubernetes objects.
openshift.io/deployment-config.name	Owner reference for resource belonging to a `deploymentconfig`.

Table 4.2. Annotations for Endpoint

Annotation	Description
idling.alpha.openshift.io/idled-at	Indicates the time at which the resource is considered idle.
idling.alpha.openshift.io/unidle-targets	References the owner resource that the endpoint will unidle.

Table 4.3. Annotations for Idling Owner Resource

Annotation	Description
idling.alpha.openshift.io/idled-at	Indicates the time at which the resource is considered idle.
idling.alpha.openshift.io/previous-scale	Indicates the previous scale of the resource prior to idle. When the endpoint receives traffic, the resource will scale to this value.

4.2.2.2. GET Request to Return `endpoint`

Request Header

-

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/api/v1/namespaces/{$PROJECT}/end
points/{$ENDPOINT}
```

Response Body Snippet

```
...
subsets:
- addresses:
  - ip: 192.0.2.1
    nodeName: {$OPENSHIFT_NODE_URL}
    targetRef:
      kind: Pod
      name: {$POD1}
      namespace: {$PROJECT}
      resourceVersion: "213472"
      uid: 6acc03a4-41cd-11e7-a37b-1418776f4b43
  - ip: 192.0.2.2
    nodeName: {$OPENSHIFT_NODE_URL}
    targetRef:
      kind: Pod
      name: {$POD2}
      namespace: {$PROJECT}
      resourceVersion: "213472"
      uid: 6acc02e0-41cd-11e7-a37b-1418776f4b43
...
```

The request returns two pods corresponding to the endpoint.

4.2.2.3. GET Request to Return pod Owner

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/api/v1/namespaces/{$PROJECT}/pod
s/{$POD1}
```

Response Body Snippet

```
...
kind: Pod
metadata:
  annotations:
    kubernetes.io/created-by: |
      {"kind":"SerializedReference","apiVersion":"v1","reference":
{"kind":"ReplicationController","namespace":"{$PROJECT}","name":"
```

```
{$REPLICATIONCONTROLLER}","uid":"e45f3d69-3b9a-11e7-a37b-
1418776f4b43","apiVersion":"v1","resourceVersion":"213449"}}
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: {$DEPLOYMENTCONFIG}
    openshift.io/deployment.name: {$DEPLOYMENT}
    openshift.io/scc: restricted
...
```

The pod owner is discovered from the **kubernetes.io/created-by:** and the **openshift.io/deployment-config.name:** annotations. The pod in this example has two owners: {$REPLICATIONCONTROLLER} and {$DEPLOYMENTCONFIG}.

For the purposes of the example, {$POD2} has the same owners.

4.2.2.4. GET Request to Return `replicationcontroller` Owner

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/api/v1/namespaces/{$PROJECT}/rep
licationcontrollers/{$REPLICATIONCONTROLLER}
```

Response Body Snippet

```
...
kind: ReplicationController
metadata:
  annotations:
    openshift.io/deployer-pod.name: {$DEPLOYER_POD}
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: {$DEPLOYMENTCONFIG}
...
```

The replication owner is discovered with the **openshift.io/deployment-config.name:** annotation. The replication controller in this example is also owned by {$DEPLOYMENTCONFIG}.

4.2.2.5. GET Request to Return the `deploymentconfig` Owner

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/oapi/v1/namespaces/{$PROJECT}/de
ploymentconfigs/{$DEPLOYMENTCONFIG}
```

Response Body Snippet

```
...
kind: DeploymentConfig
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
...
```

The returned **deploymentconfig** shows no owner, confirming that {$DEPLOYMENTCONFIG} is the sole owner of {$POD1} and {$POD2}.

4.2.2.6. GET Request to Return Scale

Request Header

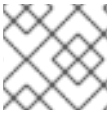
```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/oapi/v1/namespaces/{$PROJECT}/de
ploymentconfigs/{$DEPLOYMENTCONFIG}/scale
```

Response Body Snippet

```
...
kind: Scale
spec:
  replicas: 2
...
```

The **deploymentconfig** has a replica value of **2**. This will be used when updating the **deploymentconfig** to idle the resources.

4.2.2.7. PATCH Request to Update endpoint



NOTE

PATCH requests require JSON formatting.

Request Header

```
curl -k -v \
-X PATCH \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/json" \
-H "Content-Type: text/strategic-merge-patch+json" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/api/v1/namespaces/{$PROJECT}/end
points/{$ENDPOINT}
```

Request Body

```
{
```

```
  "metadata":
    "annotations": {
      "idling.alpha.openshift.io/idled-at": "2017-06-21T04:18:54Z",
      "idling.alpha.openshift.io/unidle-targets": "
[{\"kind\":\"DeploymentConfig\",\"name\":\"
{$DEPLOYMENTCONFIG}\",\"replicas\":2}]"
    }
}
```

The **idling.alpha.openshift.io/idled-at:** annotation indicates the time at which the endpoint is considered idled. The same annotation is used for the owner in the next step.
The **idling.alpha.openshift.io/unidle-targets:** annotation indicates that the endpoint will unidle the owner.

4.2.2.8. PUT Request to Update the `deploymentconfig`

Request Header

```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
https://{$OPENSHIFT_ENVIRONMENT_URL}:8443/oapi/v1/namespaces/{$PROJECT}/de
ploymentconfigs/{$DEPLOYMENTCONFIG}
```

Request Body Snippet

```
...
kind: DeploymentConfig
metadata:
  annotations:
    idling.alpha.openshift.io/idled-at: 2017-06-21T04:18:54Z
    idling.alpha.openshift.io/previous-scale: "2"
    openshift.io/generated-by: OpenShiftWebConsole
...
spec:
  replicas: 0
...
```

The **PUT** request uses the **`deploymentconfig`** returned in a previous request, modified with the following:

- The **idling.alpha.openshift.io/idled-at:** annotation is added to indicate the time at which the endpoint is considered idled.
- The **idling.alpha.openshift.io/previous-scale:** annotation is added, using the value from the previous **`scale`** request. When the endpoint is unidled, this value will determine the number of replicas to scale.
- The **replicas** value is modified to **`0`**.

4.3. ROLLBACK

4.3.1. Description of Function

Revert an application back to a previous deployment.

When you run this command your deployment configuration will be updated to match a previous deployment. By default only the pod and container configuration will be changed and scaling or trigger settings will be left as-is. Note that environment variables and volumes are included in rollbacks; if you have recently updated security credentials in your environment your previous deployment may not have the correct values.

Any image triggers present in the rolled back configuration will be disabled with a warning. This is to help prevent your rolled back deployment from being replaced by a triggered deployment soon after your rollback.

Any version of the **`deploymentconfig`** can be used for the rollback request. Rolling back a deployment creates a new deployment version. All deployment reversions use rollback requests, whether rolling the version back or forward.

4.3.2. Rolling Back a Deployment

4.3.2.1. Request Breakdown

Deployment rollback consists of two requests:

1. A **POST** request to the **`rollback`** subresource of the deployment.
2. A **PUT** request to the deployment name in the **`deploymentconfig`** subresource of the namespace.

The **POST** request body specifies which elements of the deployment configuration version spec to be included in the returned **`deploymentconfig`**. The request returns a new **`deploymentconfig`** that represents the rollback state and can be used verbatim in the request body of the **PUT** request.

The **PUT** request body sends the returned **`deploymentconfig`** configuration to the deployment name in the subresource. This triggers a new deployment that effectively rolls back the deployment to the version specified in the returned **`deploymentconfig`**.

4.3.2.2. POST Request to Return deploymentconfig

Request Header

```
curl -k -v \
-X POST  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfigs/{$DEPLOYMEN
T}/rollback
```

Request Body

```
apiVersion: v1
kind: DeploymentConfigRollback
name: {$DEPLOYMENT}
spec:
```

```
  from:
    name: {$DEPLOYMENT}-{$VERSION}
  includeTriggers: false
  includeTemplate: true
  includeReplicationMeta: false
  includeStrategy: false
```

Table 4.4. Field Descriptions

Field Name	Description	Required	Type
includeTriggers	Returns the **triggers** parameters of the specified deployment version for rollback	True	Boolean
includeTemplate	Returns the **template** parameters of the specified deployment version for rollback	True	Boolean
includeReplicationMeta	Returns the **replicas** and **selector** parameters of the specified deployment version for rollback	True	Boolean
includeStrategy	Returns the **strategy** parameters of the specified deployment version for rollback	True	Boolean

4.3.2.3. PUT Request to Revert Application to Previous Deployment

Request Header

```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfigs/{$DEPLOYMEN
T}
```

Request Body

Use the response body returned by the **POST** request as the request body. No changes are necessary.

4.4. SCALE

4.4.1. Description of Function

Set a new size for a **deployment**, **replicationcontroller**, or **job** resources:

- **deploymentconfig**: Modify the number of replica pods for the deployment.
- **replicationcontroller**: Modify the number of replicas maintained by a replication controller in a deployment.
- **job**: Modify the number of pod replicas running in parallel, executing a job.

4.4.2. Scaling a Deployment

4.4.2.1. Request Breakdown

Scaling a deployment consists of two requests:

1. A **GET** request to the **scale** subresource of the deployment.
2. A **PUT** request with an updated **replicas** value to the **scale** subresource of the deployment.

The **GET** request returns the **Scale** configuration.
The **GET** request is optional if an updated **Scale** configuration can otherwise be provided for the **PUT** request.

Update the **spec: replicas** parameter in the **Scale** configuration. Send the configuration as the request body in a **PUT** request to the **scale** subresource of the deployment.

4.4.2.2. GET Request to Scale Deployment

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfigs/{$DEPLOYMEN
T}/scale
```

4.4.2.3. PUT Request to Scale Deployment

Request Header

```
curl -k -v
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/deploymentconfigs/{$DEPLOYMEN
T}/scale
```

Request Body Snippet

```
...
spec:
  replicas: 3
...
```

4.4.3. Scaling a Replication Controller

4.4.3.1. Request Breakdown

Scaling a replication controller consists of two requests:

1. A **GET** request to the deployment name and version in the **`replicationcontrollers`** subresource of the namespace.
2. A **PUT** request with an updated **`replicas`** value to the same deployment in the **`replicationcontrollers`** of the namespace.

The **GET** request returns the **`ReplicationController`** configuration.
The **GET** request is optional if an updated **`ReplicationController`** configuration can otherwise be provided for the **PUT** request. The **`ReplicationController`** configuration includes an encoded **`deploymentconfig`** for the deployment being scaled.

Update the **`spec: replicas`** parameter in the **`ReplicationController`** configuration. Send the configuration as the request body in a **PUT** request to the deployment **`replicationcontrollers`** resource.

4.4.3.2. GET Request for Deployment Replication Controller

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
GET
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/replicationcontrollers/{$DEPLO
YMENT}-{$VERSION}
```

4.4.3.3. PUT Request to Scale Replication Controller

Request Header

```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/api/v1/namespaces/{$PROJECT}/replicationcontrollers/{$DEPLO
YMENT}-{$VERSION}
```

Request Body Snippet

```
...
spec:
  replicas: 3
...
```

4.4.4. Scaling a Job

4.4.4.1. Request Breakdown

Scaling a job consists of two requests:

1. A **GET** request to the job name in the **`jobs`** subresource of the namespace .
2. A **PUT** request with an updated **`parallelism`** value to the job name in the **`jobs`** subresource of the namespace .

The **GET** request returns a **`Job`** configuration.
The **GET** request is optional if an updated **`Job`** configuration can otherwise be provided for the **PUT** request.

Update the **`spec: parallelism`** parameter in the **`Job`** configuration. Send the configuration as the request body in a **PUT** request to the deployment **`jobs`** subresource.

4.4.4.2. GET Request for Deployment Job

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/apis/batch/v1/namespaces/{$PROJECT}/jobs/{$JOB}
```

4.4.4.3. PUT Request to Scale Job

Request Header

```
curl -k -v \
-X PUT \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/apis/batch/v1/namespaces/{$PROJECT}/jobs/{$JOB}
```

Request Body Snippet

```
...
spec:
  parallelism: 3
...
```

4.5. ROUTES

4.5.1. Description of Resource

A route allows developers to expose services through an HTTP(S) aware load balancing and proxy layer via a public DNS entry. The route may further specify TLS options and a certificate, or specify a public CNAME that the router should also accept for HTTP and HTTPS traffic. An administrator typically configures their router to be visible outside the cluster firewall, and may also add additional security, caching, or traffic controls on the service content. Routers usually talk directly to the service endpoints.

Once a route is created, the **`host`** field may not be changed. Generally, routers use the oldest route with a given host when resolving conflicts.

Routers are subject to additional customization and may support additional controls via the annotations field.

Because administrators may configure multiple routers, the route status field is used to return information to clients about the names and states of the route under each router. If a client chooses a duplicate name, for instance, the route status conditions are used to indicate the route cannot be chosen.

See the Architecture Guide and the Developer Guide for more information on routes, and the REST API Guide for complete field descriptions.

4.5.2. Creating a New Route

4.5.2.1. Request Breakdown

Creating a new route requires one request:

1. A **POST** request to the **`routes`** subresource of the namespace.

The **POST** request uses a **`Route`** configuration in the request body.
The following request example uses the minimum required fields to expose a service with an insecure route.

4.5.2.2. POST Request to Create a New Route

Request Header

```
curl -k -v \
-X POST \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/routes
```

Request Body

```
apiVersion: v1
kind: Route
metadata:
  name: {$ROUTE_NAME}
spec:
```

```
  to:
    kind: Service
    name: {$SERVICE_NAME}
```

Table 4.5. Route `metadata` Fields

Field.subfield	Description	Type
`labels.app`	Map of string keys and values that can be used to organize and categorize (scope and select) objects. Used primarily for router sharding. May match selectors of replication controllers and services.	*projectlabel*
`name`	Name of the route.	*routename*

Table 4.6. Route spec Fields

Field.subfield	Description	Type
`host`	Host is an optional alias/DNS that points to the service. If not specified, a route name will typically be automatically chosen. Must follow DNS952 subdomain conventions.	*route-project.router.default.svc.cluster.local*
`port.targetPort`	If specified, the port to be used by the router. Most routers will use all endpoints exposed by the service by default - set this value to instruct routers which port to use.	*8080-tcp*
`to.name`	Name of the service/target used for the route.	*servicename*
`to.weight`	Weight as an integer between 1 and 256 that specifies the target's relative weight against other target. reference objects	*1*
`alternateBackends.name`	Name of an alternate service/target that is being referred to.	*altservice*

Field.subfield	Description	Type
`alternateBackends.weight`	Weight as an integer between 1 and 256 that specifies the target's relative weight against other target.	*1*
`tls.termination`	The tls field provides the ability to configure certificates and termination for the route. Options are 'edge', 'passthrough', 'reencrypt'.	*edge*
`wildcardPolicy`	Wildcard policy, if any, for the route. Currently only 'Subdomain' or 'None' is allowed.	*Subdomain*

4.5.3. Listing All Service Configurations in an Environment

4.5.3.1. Request Breakdown

Listing the configurations for all routes in an environment requires one request:

1. A **GET** request to the `routes` resource.

The **GET** request returns the `RouteList`, listing the details of all routes in the environment.

4.5.3.2. GET Request to Return All Service Configurations in an Environment

Request Header

```
curl -k -v \
-X GET \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/routes
```

4.5.4. Listing All Service Configurations in a Namespace

4.5.4.1. Request Breakdown

Listing the configurations for all services in a namespace requires one request:

1. A **GET** request to the `routes` subresource of the namespace.

The **GET** request returns the `RouteList`, listing the details of all routes in the namespace.

4.5.4.2. GET Request to Return All Service Configurations in Namespace

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/routes
```

4.5.5. Listing a Specific Route Configuration in a Namespace

4.5.5.1. Request Breakdown

Listing a specific **Route** configuration requires one request:

1. A **GET** request to the route name in the **routes** subresource of the namespace.

The **GET** request returns the **Route** configuration for the specified route.

4.5.5.2. GET Request to Return Specific Route Configuration

Request Header

```
curl -k -v \
-X GET  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/routes/{$ROUTE}
```

4.5.6. Updating a Route Configuration

Updating a **Route** configuration requires two requests:

1. A **GET** request to the route name in the **routes** subresource of the namespace.
2. A **PATCH** request with updated field values to the route name in the **routes** subresource of the namespace.

The **GET** request is optional as the **PATCH** request body is not determined by the returned configuration, however it can be useful for preparing the **PATCH** request body.

4.5.6.1. PATCH Request to Update Route Configuration

NOTE

PATCH requests require JSON formatting.

The following **PATCH** request example adds an alternate service to the route and updates the original service with an equal **weight**:

Request Header

```
curl -k -v \
```

```
-X PATCH \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/json" \
-H "Content-Type: text/strategic-merge-patch+json" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/
routes/{$ROUTE}
```

Request Body

```
{
  "spec": {
     "to": {
      "weight": 1
     },
     "alternateBackends": [
       {
        "kind": "Service",
        "name": "{$ALT_SERVICE}",
        "weight": 1
       }
    ]
  }
}
```

4.5.7. Deleting a Route

4.5.7.1. Request Breakdown

Deleting a route requires one request:

1. A **DELETE** request to the route name in the **`routes`** subresource of the namespace.

The **DELETE** request returns a **`code: 200`** and the route is deleted.

4.5.7.2. DELETE Request to Delete a Route

Request Header

```
curl -k -v \
-X DELETE  \
-H "Authorization: {$BEARER_TOKEN}" \
-H "Accept: application/yaml" \
-H "Content-Type: application/yaml" \
{$OCP_CLUSTER}/oapi/v1/namespaces/{$PROJECT}/routes/{$ROUTE}
```

CHAPTER 5. RESOURCE EXAMPLES

5.1. PREPARING RESOURCES EXAMPLES

5.1.1. Project Configuration

```
apiVersion: v1
kind: Project
metadata:
  annotations:
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: admin
    openshift.io/sa.scc.mcs: s0:c8,c2
    openshift.io/sa.scc.supplemental-groups: 1000060000/10000
    openshift.io/sa.scc.uid-range: 1000060000/10000
  creationTimestamp: 2017-05-18T06:35:06Z
  name: demo-app
  resourceVersion: "2548"
  selfLink: /oapi/v1/projectsdemo-app
  uid: 21f17420-3b94-11e7-a37b-1418776f4b43
spec:
  finalizers:
  - openshift.io/origin
  - kubernetes
status:
  phase: Active
```

5.1.2. ServiceAccount Configuration

```
apiVersion: v1
imagePullSecrets:
- name: demo-service-account-dockercfg-7qfh6
kind: ServiceAccount
metadata:
  creationTimestamp: 2017-07-28T05:00:25Z
  name: demo-service-account
  namespace: demo-app
  resourceVersion: "1799954"
  selfLink: /api/v1/namespaces/{$PROJECT}/serviceaccounts/demo-service-
account
  uid: aacbb53d-7351-11e7-9e2b-1418776f4b43
secrets:
- name: demo-service-account-token-x4tx5
- name: demo-service-account-dockercfg-7qfh6
- name: demo-secret
```

5.1.3. RoleBinding Configuration

```
apiVersion: v1
groupNames: null
kind: RoleBinding
```

```
metadata:
  creationTimestamp: 2017-07-28T04:24:37Z
  name: edit
  namespace: demo-app
  resourceVersion: "1795828"
  selfLink: /oapi/v1/namespaces/demo-app/rolebindings/edit
  uid: aad4c7ea-734c-11e7-9e2b-1418776f4b43
roleRef:
  name: edit
subjects:
- kind: ServiceAccount
  name: demo-service-account
  namespace: demo-app
userNames:
- system:serviceaccount:demo-app:demo-service-account
```

5.1.4. Secret Configuration

```
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: 2017-07-18T06:31:12Z
  name: demo-secret
  namespace: demo-app
  resourceVersion: "1536805"
  selfLink: /api/v1/namespaces/demo-app/secrets/demo-secret
  uid: b13d63e0-6b82-11e7-9e2b-1418776f4b43
data:
  keystore.jks:
/u3+7QAAAAIAAAABAAAAAQAKc2VsZnNpZ25lZAAAAVyvkZlZAAAFAzCCBP8wDgYKKwYBBAEqAh
EBAQUABIIE67eZLBiEKM8qQGBmFLjGb+HWeI/um3I7x8udbr2qukVjNseb5s+I8P3rbmxvHuN4
zyOL3P1eEH4j9wEVZXoomtn7DOz9Ro+dEOsxT9Hj51te+zpJcMHpVLySsvV1Wf4HDKCafxBHwZ
tY2l7vUFjEKQhoEu+r6FeI1FU3D8ZUF+hWFJ5/esIDw4/d3wIjwrEmUYL9OITGpoy0V4Vi9VhX
GO+xPCNUqxJdzMhKcM+mURf88+yB5F2fSQ8HHDzhxKzxTzX3CmSdg3Gr8o/TkN9i51eg3rZPl+
yBTq7vXo43V7jbi4OM8WhVkfsexj1SKoV5PLRGgWK9BieIEP4DCkQ7iM6qFiQxXeIUXyamjhhl
IQAq7BvGy7abxu/m2Q08SGx0TfSwDmi/NLCAxnnPZuEJbzoRBNDfKaTJCCaSxkRfwLyfsPqBTI
3EZgYRTFswAlm1DkvuHLftO270SRDGwWnm8/xTQ+hvK/E+M9S0be31ehDLlyaLJEHz8mst/vxy
m89NnGCSETuv3X4HPxGUMCbF4TdtSClA+zXexhqmbOTupLkuVIRu+a33yQaFb3Owb4iC2oUJvq
9H4mPmas0sxMUrZ2bvIqoK9kCMIeXVJ7uh495fjZp9xjT1mTlNxs9KSavSpdHatwypoQsiAvE1
w4efljyYY2ppqQNi3hBmgkvQnI2OLDrvJTpnZQv2CewxPGU8UCTzCTQ6WnI8NO6JrxV/h8Ak0E
xjbHI3zUDdYLvRkp5W0f05Fho2PbeTOYYF/Vn5IU8T41wnDQc4rqvF/cfmeMQOeAAVgYvLRdML
dJPBimxIciIYbOjk4yuCRh0cBI+6qK1Tk9VnSC9A354ihpTEg5Utm67q9wV5Voq2OP3MTQTbbk
0bmNtCn8ybuM9CeXJlSRstBuevakAvNuOo2d5TcUW0KM4MqDfMBNq1U5JzD+GyXI2iwItgp9l7
64vhxjFR+RzS/zq1orxP5staUIpC8eI2mkb3jdAsLIiR+TyDHpAEk7G7j6OghyXrpTPA4wWJDr
0RAAbOBdWPPYB8wCph6424eGMP3JowIODASJwxJ6UqLHVFh0Rwzhz1RnNzHL2kruLASLoyvp7Q
jvSqaMsfLyaeZpsH32R90KX2EHKp9G+8Y8ujdeD4SuHo57PRXyFWPtOli9fkbRyPugKNrD5Wfr
74jx9tNuelWb4SquVS2SaIbjmtzuylS5aOVW92FEM4OlRBdrScrIic+u8lpoGzrEvgO2lGj4Ma
HfpibC3pv0K92ksYZ0X1kQbrNfbCmqDPKq1Eid9vI+ZMW+SHvJJqg+axT3a3+ojVpEW2ymiwJs
ZfcSFNa9JzfDRtH2K2Ellf14WyFAAzB6j+ZlWiMCqwFLDh370FAvKzOUACYLItX7s52NTqAlPl
EyZIPNxMYTRVvMNpiTJ8ZrZRmnd9gsnVNgxwdUu31uDxvIbRQdwR+PbxRNfbAj8jM6dPeCAXvS
PQEt+qWB9LPuNW0zyvQzw85Dmda3x60o28LDWjAFT1QhAN1KepX3vrP6GgmNrK9nmwHa0xOx5O
XINDhXNZxpmV1YKvZE4lpdZFUikYYjasMp9AI38wK+fJUTY19bX3g2lfmIeM2nmuRabpgQ3wN/
FxdDwqn6XsNOAgKT86bmwc8vIy0P5d+7OoGvpSIC2FDFlfBYi5tehxVqz4JBBOAAAAAQAFWC41
MDkAAAN7MIIDdzCCAl+gAwIBAgIEMo+1QjANBgkqhkiG9w0BAQsFADBsMRAwDgYDVQQGEwdVbm
tub3duMRAwDgYDVQQIEwdVbmtub3duMRAwDgYDVQQHEwdVbmtub3duMRAwDgYDVQQKEwdVbmtu
```

```
b3duMRAwDgYDVQQLEwdVbmtub3duMRAwDgYDVQQDEwdVbmtub3duMB4XDTE3MDYxNjA2MjI1MV
oXDTE4MDYxMTA2MjI1MVowbDEQMA4GA1UEBhMHVW5rbm93bjEQMA4GA1UECBMHVW5rbm93bjEQ
MA4GA1UEBxMHVW5rbm93bjEQMA4GA1UEChMHVW5rbm93bjEQMA4GA1UECxMHVW5rbm93bjEQMA
4GA1UEAxMHVW5rbm93bjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALgOLtnUc/O0
3eYxZNr5N8wMeONdjaV/N7QxbtqL1feozbux3aJ1vQHn0heqj4irnJJJTnoCt6yVhpInixC7F9
tm0+zWVxAXWBAL5w7zXSMeVdjTOjKC6qwbsA/jGGQXa5+RCyuMwLb8RNp0YjvNEVqUumyIdipE
QiyZNLGV9C1vZcSMRIf8Ba5b+DaD2PssXKscmEf8J2noU096NyowQJJprb/rQ/TbXmM9xca+HZ
GSEd5qUQ8o7fMEnnUnIdv0ftyBxvrpfwRQza//KtMa/8uaMltmbbTBXQsSTJoJb4vKkZu4Uxij
rBOH4wNWJpWL8LpBOC3HZqDBc1UVN61nQ+cCAwEAAaMhMB8wHQYDVR0OBBYEFCFeYqiTHAeyIc
zCorbqzaQ/hBzoMA0GCSqGSIb3DQEBCwUAA4IBAQANvxKg+KYg2EExKa5iHq8TxSTlUnz/gSaX
YL70nFL65i1eIkvksUcduQRMfTbop8wvx5hlX3N7suSjeBtDMSFJ/ZG4PPmQCdwQkshgMgsvd9
yTsbeV7Wb8tc8JRSAJFhtxAHSTLJ8MxLBlwPtu+oBkmtvLQ7xhQarhKD7GPTYCI3EQwbrhCvKO
Gx4uH17b8ZshilUOVtE3XiRLE3zkcCc8Sy+rDm0FSkkJ6AN4xoHXKmLkqZoKiM1S5c8Nf6tRQA
UmsNIbNxG5lCTnE76ZCF4wj+ukO3t8d5x42D+vKVNEnufJ4hcY4IMXet4kAPRE/CrEDNPqA3/4
MVcvHT7pmgNITpqKCkToVp57Jl+CRyJVxmdscaA=
type: Opaque
```

5.1.5. PersistentVolume Configuration

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/bound-by-controller: "yes"
  creationTimestamp: 2017-08-07T05:30:05Z
  name: volume01
  resourceVersion: "2105343"
  selfLink: /api/v1/persistentvolumesvolume01
  uid: 7831e1bc-7b31-11e7-9e2b-1418776f4b43
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 3Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: volume01-volume-claim
    namespace: demo-app
    resourceVersion: "2105341"
    uid: 1ca84c8a-7b37-11e7-9e2b-1418776f4b43
  nfs:
    path: /root/storage
    server: openshift-1.example.com
  persistentVolumeReclaimPolicy: Retain
status:
  phase: Bound
```

5.1.6. PersistentVolumeClaim Configuration

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
```

```
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
  creationTimestamp: 2017-08-08T05:20:53Z
  name: volume05-claim
  namespace: demo-app
  resourceVersion: "2136937"
  selfLink: /api/v1/namespaces/demo-app/persistentvolumeclaims/volume05-
claim
  uid: 599e61db-7bf9-11e7-9e2b-1418776f4b43
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 3Gi
  volumeName: volume05
status:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 3Gi
  phase: Bound
```

5.1.7. DeploymentConfig Configuration

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
  creationTimestamp: 2017-04-20T02:24:46Z
  generation: 43
  labels:
    app: demo-app
  name: demo-app
  namespace: demo-project
  resourceVersion: "913151"
  selfLink: /oapi/v1/namespaces/demo-project/deploymentconfigs/demo-app
  uid: 858f0351-2570-11e7-8669-1418776f4b43
spec:
  replicas: 3
  selector:
    deploymentconfig: demo-app
  strategy:
    resources: {}
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      updatePeriodSeconds: 1
    type: Rolling
  template:
    metadata:
      creationTimestamp: null
```

```
      labels:
        app: demo-app
        deploymentconfig: demo-app
    spec:
      containers:
      - image: 172.30.192.109:5000/demo-project/demo-
app@sha256:9e7db6c87501bf88b652cf99c9573f4909cad6ce7e91bc54297f88d39d9a9a7
7
        name: demo-app
        ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 8443
          protocol: TCP
        - containerPort: 8778
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      securityContext: {}
      terminationGracePeriodSeconds: 30
  test: false
  triggers:
  - type: ConfigChange
  - imageChangeParams:
      containerNames:
      - demo-app
      from:
        kind: ImageStreamTag
        name: demo-app:latest
        namespace: demo-project
      lastTriggeredImage: 172.30.192.109:5000/demo-project/demo-
app@sha256:ec2c64a5ca48a0c0ccb2c58556dbbf3e3f44ef7d2b2514dace06860cf3e53be
c
    type: ImageChange
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: 2017-04-28T02:21:54Z
    message: Replication controller "demo-app-36" has completed
progressing
    reason: NewReplicationControllerAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: 2017-05-02T06:24:50Z
    message: Deployment config has minimum availability.
    status: "True"
    type: Available
  details:
    causes:
    - type: ConfigChange
    message: config change
  latestVersion: 36
```

```
  observedGeneration: 43
  replicas: 3
  updatedReplicas: 3
```

5.2. BUILD AND IMAGE EXAMPLES

5.2.1. Build Configuration

```
apiVersion: v1
kind: Build
metadata:
  annotations:
    openshift.io/build-config.name: control-app
    openshift.io/build.number: "1"
    openshift.io/build.pod-name: control-app-1-build
  creationTimestamp: 2017-05-18T08:22:36Z
  labels:
    app: control-app
    buildconfig: control-app
    openshift.io/build-config.name: control-app
    openshift.io/build.start-policy: Serial
  name: control-app-1
  namespace: demo-app
  resourceVersion: "5117"
  selfLink: /oapi/v1/namespaces/demo-app/builds/control-app-1
  uid: 25f29fe0-3ba3-11e7-a37b-1418776f4b43
spec:
  nodeSelector: null
  output:
    pushSecret:
      name: builder-dockercfg-f6301
    to:
      kind: ImageStreamTag
      name: control-app:latest
  postCommit: {}
  resources: {}
  revision:
    git:
      author:
        email: author@example.com
        name: Author Person
      commit: d9281fa6c7ca7a498bdf95049c64bbbe41b989cf
      committer:
        email: author@example.com
        name: Author Person
      message: Update POM versions for EAP 7.0.0.GA release
    type: Git
  serviceAccount: builder
  source:
    contextDir: kitchensink
    git:
      ref: 7.0.0.GA
      uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
    type: Git
  strategy:
```

```
    sourceStrategy:
      from:
        kind: DockerImage
        name: registry.access.redhat.com/jboss-eap-7/eap70-
openshift@sha256:2aa466daf9d45c93ba6ea6ec4d7cd8f26a14b9850e03f3353c85bac03
fdf6df9
    type: Source
  triggeredBy:
  - message: Build configuration change
status:
  completionTimestamp: 2017-05-18T08:26:17Z
  config:
    kind: BuildConfig
    name: control-app
    namespace: demo-app
  duration: 214000000000
  output:
    to:
      imageDigest:
sha256:e2a66141c9c77c5507ed8237efd97d6c440b8ee0a50f5899c44d8f475be3d94c
  phase: Complete
  startTimestamp: 2017-05-18T08:22:43Z
```

5.2.2. BuildConfig Configuration

```
apiVersion: v1
kind: BuildConfig
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
  creationTimestamp: 2017-05-18T08:22:36Z
  labels:
    app: control-app
  name: control-app
  namespace: demo-app
  resourceVersion: "4972"
  selfLink: /oapi/v1/namespaces/demo-app/buildconfigs/control-app
  uid: 25ee117e-3ba3-11e7-a37b-1418776f4b43
spec:
  nodeSelector: null
  output:
    to:
      kind: ImageStreamTag
      name: control-app:latest
  postCommit: {}
  resources: {}
  runPolicy: Serial
  source:
    contextDir: kitchensink
    git:
      ref: 7.0.0.GA
      uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
    type: Git
  strategy:
    sourceStrategy:
```

```
      from:
        kind: ImageStreamTag
        name: jboss-eap70-openshift:1.4
        namespace: openshift
    type: Source
  triggers:
  - generic:
      secret: dbc5f4f2ab54b2cc
    type: Generic
  - github:
      secret: eea95016a468c783
    type: GitHub
  - imageChange:
      lastTriggeredImageID: registry.access.redhat.com/jboss-eap-7/eap70-
openshift@sha256:2aa466daf9d45c93ba6ea6ec4d7cd8f26a14b9850e03f3353c85bac03
fdf6df9
    type: ImageChange
  - type: ConfigChange
status:
  lastVersion: 1
```

5.2.3. Image Configuration

```
apiVersion: v1
dockerImageLayers:
- mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
  name:
sha256:8642dd241e54ecb57f49345f135e9bcedb0546e7e61c1ca4d0008a9925f50444
  size: 0
- mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
  name:
sha256:fdd633d880f736958e14a036256b2def325acf6b438b7c849139fe92d5cbe4ce
  size: 0
- mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
  name:
sha256:9ba7fddb59304bf6233e7b8e699208cc908f236e261bb9da9a0c9f63c06cb80d
  size: 227928358
dockerImageManifestMediaType:
application/vnd.docker.distribution.manifest.v1+json
dockerImageMetadata:
  Architecture: amd64
  Author: Author Person <author@example.com>
  Config:
    Cmd:
    - /usr/libexec/s2i/run
    Env:
    - PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    - container=oci
    - JENKINS_VERSION=1.651.2
    - HOME=/var/lib/jenkins
    - JENKINS_HOME=/var/lib/jenkins
    - STI_SCRIPTS_URL=image:///usr/libexec/s2i
    ExposedPorts:
      8080/tcp: {}
      50000/tcp: {}
```

```
    Hostname: 06c09501cb72
    Image:
b9ff160d3f23457a8ab1db52f7df0d7dd3d5ffa3ac51e3fd2f0021a7e83417b5
    Labels:
      architecture: x86_64
      authoritative-source-url: registry.access.redhat.com
      build-date: 2017-04-22T13:35:47.460129
      com.redhat.build-host: ip-10-29-120-106.ec2.internal
      com.redhat.component: openshift-jenkins-docker
      description: The Red Hat Enterprise Linux Base image is designed to
be a fully
        supported foundation for your containerized applications.  This
base image
        provides your operations and application teams with the packages,
language
        runtimes and tools necessary to run, maintain, and troubleshoot
all of your
        applications. This image is maintained by Red Hat and updated
regularly. It
        is designed and engineered to be the base layer for all of your
containerized
        applications, middleware and utilites. When used as the source for
all of
        your containers, only one copy will ever be downloaded and cached
in your
        production environment. Use this image just like you would a
regular Red Hat
        Enterprise Linux distribution. Tools like yum, gzip, and bash are
provided
        by default. For further information on how this image was built
look at the
        /root/anacanda-ks.cfg file.
      distribution-scope: public
      io.k8s.description: Jenkins is a continuous integration server
      io.k8s.display-name: Jenkins 1.651.2
      io.openshift.expose-services: 8080:http
      io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
      io.openshift.tags: jenkins,jenkins1,ci
      name: openshift3/jenkins-1-rhel7
      release: "57"
      summary: Provides the latest release of Red Hat Enterprise Linux 7
in a fully
        featured and supported base image.
      vcs-ref: 7f2c3c18da5c321d16e403f7905d86fb3fa89fcd
      vcs-type: git
      vendor: Red Hat, Inc.
      version: 1.651.2
    User: "1001"
    Volumes:
      /var/lib/jenkins: {}
  ContainerConfig:
    Cmd:
    - /bin/sh
    - -c
    - '#(nop) '
    - USER [1001]
```

```
    Env:
    - PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    - container=oci
    - JENKINS_VERSION=1.651.2
    - HOME=/var/lib/jenkins
    - JENKINS_HOME=/var/lib/jenkins
    - STI_SCRIPTS_URL=image:///usr/libexec/s2i
    ExposedPorts:
      8080/tcp: {}
      50000/tcp: {}
    Hostname: 06c09501cb72
    Image:
sha256:1a6180458fa11cf049d4bb9bef78f5379e9983fc023548b1fee76e790f27acfa
    Labels:
      architecture: x86_64
      authoritative-source-url: registry.access.redhat.com
      build-date: 2017-04-22T13:35:47.460129
      com.redhat.build-host: ip-10-29-120-106.ec2.internal
      com.redhat.component: openshift-jenkins-docker
      description: The Red Hat Enterprise Linux Base image is designed to
be a fully
        supported foundation for your containerized applications.  This
base image
        provides your operations and application teams with the packages,
language
        runtimes and tools necessary to run, maintain, and troubleshoot
all of your
        applications. This image is maintained by Red Hat and updated
regularly. It
        is designed and engineered to be the base layer for all of your
containerized
        applications, middleware and utilites. When used as the source for
all of
        your containers, only one copy will ever be downloaded and cached
in your
        production environment. Use this image just like you would a
regular Red Hat
        Enterprise Linux distribution. Tools like yum, gzip, and bash are
provided
        by default. For further information on how this image was built
look at the
        /root/anacanda-ks.cfg file.
      distribution-scope: public
      io.k8s.description: Jenkins is a continuous integration server
      io.k8s.display-name: Jenkins 1.651.2
      io.openshift.expose-services: 8080:http
      io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
      io.openshift.tags: jenkins,jenkins1,ci
      name: openshift3/jenkins-1-rhel7
      release: "57"
      summary: Provides the latest release of Red Hat Enterprise Linux 7
in a fully
        featured and supported base image.
      vcs-ref: 7f2c3c18da5c321d16e403f7905d86fb3fa89fcd
      vcs-type: git
      vendor: Red Hat, Inc.
```

```
      version: 1.651.2
    User: "1001"
    Volumes:
      /var/lib/jenkins: {}
  Created: 2017-04-22T13:38:24Z
  DockerVersion: 1.12.6
  Id: 3029164ce7ca6ba5ca2fce59d0fcdd0590b582e022419a59d504ed0a354d70dc
  Parent: 6baf74cede21e20f9229b4558e06d38d5cd61816abb69f6c8126f8bceaa2518f
  Size: 300045578
  apiVersion: "1.0"
  kind: DockerImage
dockerImageMetadataVersion: "1.0"
dockerImageReference: registry.access.redhat.com/openshift3/jenkins-1-
rhel7@sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e18
7a0
kind: Image
metadata:
  creationTimestamp: null
  name:
sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e187a0
```

5.2.4. Image Stream Import Configuration

```
kind: ImageStreamImport
apiVersion: v1
metadata:
  name: ruby
  namespace: openshift
  resourceVersion: '654'
  creationTimestamp:
spec:
  import: true
  images:
  - from:
      kind: DockerImage
      name: registry.access.redhat.com/rhscl/ruby-23-rhel7:latest
    to:
      name: '2.3'
```

5.2.5. Image Stream Configuration

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/display-name: Jenkins
    openshift.io/image.dockerRepositoryCheck: 2017-05-18T05:20:22Z
  creationTimestamp: 2017-05-18T05:18:41Z
  generation: 2
  name: jenkins
  namespace: openshift
  resourceVersion: "697"
  selfLink: /oapi/v1/namespaces/openshift/imagestreams/jenkins
  uid: 75043e39-3b89-11e7-8867-1418776f4b43
```

```
spec:
  tags:
  - annotations:
      description: Provides a Jenkins 1.X server on RHEL 7. For more
information about
        using this container image, including OpenShift considerations,
see https://github.com/openshift/jenkins/blob/master/README.md.
      iconClass: icon-jenkins
      openshift.io/display-name: Jenkins 1.X
      tags: hidden,jenkins
      version: 1.x
    from:
      kind: DockerImage
      name: registry.access.redhat.com/openshift3/jenkins-1-rhel7:latest
    generation: 2
    importPolicy: {}
    name: "1"
    referencePolicy:
      type: Source
  - annotations:
      description: Provides a Jenkins 2.X server on RHEL 7. For more
information about
        using this container image, including OpenShift considerations,
see https://github.com/openshift/jenkins/blob/master/README.md.
      iconClass: icon-jenkins
      openshift.io/display-name: Jenkins 2.X
      tags: jenkins
      version: 2.x
    from:
      kind: DockerImage
      name: registry.access.redhat.com/openshift3/jenkins-2-rhel7:latest
    generation: 2
    importPolicy: {}
    name: "2"
    referencePolicy:
      type: Source
  - annotations:
      description: |-
        Provides a Jenkins server on RHEL 7. For more information about
using this container image, including OpenShift considerations, see
https://github.com/openshift/jenkins/blob/master/README.md.

        WARNING: By selecting this tag, your application will
automatically update to use the latest version of Jenkins available on
OpenShift, including major versions updates.
      iconClass: icon-jenkins
      openshift.io/display-name: Jenkins (Latest)
      tags: jenkins
    from:
      kind: ImageStreamTag
      name: "2"
    generation: 1
    importPolicy: {}
    name: latest
    referencePolicy:
      type: Source
```

```
status:
  dockerImageRepository: 172.30.252.150:5000/openshift/jenkins
  tags:
  - items:
    - created: 2017-05-18T05:20:22Z
      dockerImageReference: registry.access.redhat.com/openshift3/jenkins-
2-
rhel7@sha256:105dd6e8e518b5f632e550ac8edeef52c5079c7fb102fc55db45d3b52d805
3c1
      generation: 2
      image:
sha256:105dd6e8e518b5f632e550ac8edeef52c5079c7fb102fc55db45d3b52d8053c1
    tag: latest
  - items:
    - created: 2017-05-18T05:20:22Z
      dockerImageReference: registry.access.redhat.com/openshift3/jenkins-
1-
rhel7@sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e18
7a0
      generation: 2
      image:
sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e187a0
    tag: "1"
  - items:
    - created: 2017-05-18T05:20:22Z
      dockerImageReference: registry.access.redhat.com/openshift3/jenkins-
2-
rhel7@sha256:105dd6e8e518b5f632e550ac8edeef52c5079c7fb102fc55db45d3b52d805
3c1
      generation: 2
      image:
sha256:105dd6e8e518b5f632e550ac8edeef52c5079c7fb102fc55db45d3b52d8053c1
    tag: "2"
```

5.2.6. Image Stream Tag Configuration

```
apiVersion: v1
generation: 2
image:
  dockerImageLayers:
  - mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
    name:
sha256:8642dd241e54ecb57f49345f135e9bcedb0546e7e61c1ca4d0008a9925f50444
    size: 0
  - mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
    name:
sha256:fdd633d880f736958e14a036256b2def325acf6b438b7c849139fe92d5cbe4ce
    size: 0
  - mediaType: application/vnd.docker.container.image.rootfs.diff+x-gtar
    name:
sha256:9ba7fddb59304bf6233e7b8e699208cc908f236e261bb9da9a0c9f63c06cb80d
    size: 227928358
  dockerImageManifestMediaType:
application/vnd.docker.distribution.manifest.v1+json
  dockerImageMetadata:
```

```
    Architecture: amd64
    Author: Author Person <author@example.com>
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Env:
      - PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - container=oci
      - JENKINS_VERSION=1.651.2
      - HOME=/var/lib/jenkins
      - JENKINS_HOME=/var/lib/jenkins
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      ExposedPorts:
        8080/tcp: {}
        50000/tcp: {}
      Hostname: 06c09501cb72
      Image:
b9ff160d3f23457a8ab1db52f7df0d7dd3d5ffa3ac51e3fd2f0021a7e83417b5
      Labels:
        architecture: x86_64
        authoritative-source-url: registry.access.redhat.com
        build-date: 2017-04-22T13:35:47.460129
        com.redhat.build-host: ip-10-29-120-106.ec2.internal
        com.redhat.component: openshift-jenkins-docker
        description: The Red Hat Enterprise Linux Base image is designed
to be a fully
          supported foundation for your containerized applications.  This
base image
          provides your operations and application teams with the
packages, language
          runtimes and tools necessary to run, maintain, and troubleshoot
all of your
          applications. This image is maintained by Red Hat and updated
regularly.
          It is designed and engineered to be the base layer for all of
your containerized
          applications, middleware and utilites. When used as the source
for all of
          your containers, only one copy will ever be downloaded and
cached in your
          production environment. Use this image just like you would a
regular Red
          Hat Enterprise Linux distribution. Tools like yum, gzip, and
bash are provided
          by default. For further information on how this image was built
look at
          the /root/anacanda-ks.cfg file.
        distribution-scope: public
        io.k8s.description: Jenkins is a continuous integration server
        io.k8s.display-name: Jenkins 1.651.2
        io.openshift.expose-services: 8080:http
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: jenkins,jenkins1,ci
        name: openshift3/jenkins-1-rhel7
        release: "57"
        summary: Provides the latest release of Red Hat Enterprise Linux 7
```

```
in a fully
          featured and supported base image.
        vcs-ref: 7f2c3c18da5c321d16e403f7905d86fb3fa89fcd
        vcs-type: git
        vendor: Red Hat, Inc.
        version: 1.651.2
      User: "1001"
      Volumes:
        /var/lib/jenkins: {}
    ContainerConfig:
      Cmd:
      - /bin/sh
      - -c
      - '#(nop) '
      - USER [1001]
      Env:
      - PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - container=oci
      - JENKINS_VERSION=1.651.2
      - HOME=/var/lib/jenkins
      - JENKINS_HOME=/var/lib/jenkins
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      ExposedPorts:
        8080/tcp: {}
        50000/tcp: {}
      Hostname: 06c09501cb72
      Image:
sha256:1a6180458fa11cf049d4bb9bef78f5379e9983fc023548b1fee76e790f27acfa
      Labels:
        architecture: x86_64
        authoritative-source-url: registry.access.redhat.com
        build-date: 2017-04-22T13:35:47.460129
        com.redhat.build-host: ip-10-29-120-106.ec2.internal
        com.redhat.component: openshift-jenkins-docker
        description: The Red Hat Enterprise Linux Base image is designed
to be a fully
          supported foundation for your containerized applications.  This
base image
          provides your operations and application teams with the
packages, language
          runtimes and tools necessary to run, maintain, and troubleshoot
all of your
          applications. This image is maintained by Red Hat and updated
regularly.
          It is designed and engineered to be the base layer for all of
your containerized
          applications, middleware and utilites. When used as the source
for all of
          your containers, only one copy will ever be downloaded and
cached in your
          production environment. Use this image just like you would a
regular Red
          Hat Enterprise Linux distribution. Tools like yum, gzip, and
bash are provided
          by default. For further information on how this image was built
look at
```

```
          the /root/anacanda-ks.cfg file.
        distribution-scope: public
        io.k8s.description: Jenkins is a continuous integration server
        io.k8s.display-name: Jenkins 1.651.2
        io.openshift.expose-services: 8080:http
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: jenkins,jenkins1,ci
        name: openshift3/jenkins-1-rhel7
        release: "57"
        summary: Provides the latest release of Red Hat Enterprise Linux 7
in a fully
          featured and supported base image.
        vcs-ref: 7f2c3c18da5c321d16e403f7905d86fb3fa89fcd
        vcs-type: git
        vendor: Red Hat, Inc.
        version: 1.651.2
      User: "1001"
      Volumes:
        /var/lib/jenkins: {}
    Created: 2017-04-22T13:38:24Z
    DockerVersion: 1.12.6
    Id: 3029164ce7ca6ba5ca2fce59d0fcdd0590b582e022419a59d504ed0a354d70dc
    Parent:
6baf74cede21e20f9229b4558e06d38d5cd61816abb69f6c8126f8bceaa2518f
    Size: 300045578
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: registry.access.redhat.com/openshift3/jenkins-1-
rhel7@sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e18
7a0
  metadata:
    annotations:
      description: Provides a Jenkins 1.X server on RHEL 7. For more
information about
        using this container image, including OpenShift considerations,
see https://github.com/openshift/jenkins/blob/master/README.md.
      iconClass: icon-jenkins
      openshift.io/display-name: Jenkins 1.X
      tags: hidden,jenkins
      version: 1.x
    creationTimestamp: 2017-05-18T05:20:22Z
    name:
sha256:9a370e38aca93da91bda03107f74fc245b169a8c642daf431a93289f44e187a0
    resourceVersion: "695"
    uid: b0e3ef77-3b89-11e7-8867-1418776f4b43
kind: ImageStreamTag
metadata:
  annotations:
    description: Provides a Jenkins 1.X server on RHEL 7. For more
information about
      using this container image, including OpenShift considerations, see
https://github.com/openshift/jenkins/blob/master/README.md.
    iconClass: icon-jenkins
    openshift.io/display-name: Jenkins 1.X
    tags: hidden,jenkins
```

```
    version: 1.x
  creationTimestamp: 2017-05-18T05:20:22Z
  name: jenkins:1
  namespace: openshift
  resourceVersion: "697"
  selfLink: /oapi/v1/namespaces/openshift/imagestreamtags/jenkins%3A1
  uid: 75043e39-3b89-11e7-8867-1418776f4b43
tag:
  annotations:
    description: Provides a Jenkins 1.X server on RHEL 7. For more
information about
      using this container image, including OpenShift considerations, see
https://github.com/openshift/jenkins/blob/master/README.md.
    iconClass: icon-jenkins
    openshift.io/display-name: Jenkins 1.X
    tags: hidden,jenkins
    version: 1.x
  from:
    kind: DockerImage
    name: registry.access.redhat.com/openshift3/jenkins-1-rhel7:latest
  generation: 2
  importPolicy: {}
  name: "1"
  referencePolicy:
    type: Source
```

5.3. PROJECT MANAGEMENT RESOURCE EXAMPLES

5.3.1. Scale Configuration

```
apiVersion: extensions/v1beta1
kind: Scale
metadata:
  creationTimestamp: 2017-04-20T02:24:46Z
  name: demo-app
  namespace: demo-project
  resourceVersion: "913151"
  selfLink: /oapi/v1/namespaces/demo-project/deploymentconfigs/demo-
app/scale
  uid: 858f0351-2570-11e7-8669-1418776f4b43
spec:
  replicas: 3
status:
  replicas: 3
  selector:
    deploymentconfig: demo-app
  targetSelector: deploymentconfig=demo-app
```

5.3.2. ReplicationController Configuration

```
apiVersion: v1
kind: ReplicationController
metadata:
```

```
  annotations:
    kubectl.kubernetes.io/original-replicas: "1"
    openshift.io/deployer-pod.name: demo-app-8-deploy
    openshift.io/deployment-config.latest-version: "8"
    openshift.io/deployment-config.name: demo-app
    openshift.io/deployment.phase: Complete
    openshift.io/deployment.replicas: ""
    openshift.io/deployment.status-reason: image change
    openshift.io/encoded-deployment-config: |
      {"kind":"DeploymentConfig","apiVersion":"v1","metadata":
{"name":"demo-app","namespace":"demo-
project","selfLink":"/oapi/v1/namespaces/demo-
project/deploymentconfigs/demo-app","uid":"858f0351-2570-11e7-8669-
1418776f4b43","resourceVersion":"574636","generation":10,"creationTimestam
p":"2017-04-20T02:24:46Z","labels":{"app":"demo-app"},"annotations":
{"openshift.io/generated-by":"OpenShiftWebConsole"}},"spec":{"strategy":
{"type":"Rolling","rollingParams":
{"updatePeriodSeconds":1,"intervalSeconds":1,"timeoutSeconds":600,"maxUnav
ailable":"25%","maxSurge":"25%"},"resources":{}},"triggers":
[{"type":"ConfigChange"},{"type":"ImageChange","imageChangeParams":
{"automatic":true,"containerNames":["demo-app"],"from":
{"kind":"ImageStreamTag","namespace":"demo-project","name":"demo-
app:latest"},"lastTriggeredImage":"172.30.192.109:5000/demo-project/demo-
app@sha256:9e7db6c87501bf88b652cf99c9573f4909cad6ce7e91bc54297f88d39d9a9a7
7"}}],"replicas":1,"test":false,"selector":{"deploymentconfig":"demo-
app"},"template":{"metadata":{"creationTimestamp":null,"labels":
{"app":"demo-app","deploymentconfig":"demo-app"}},"spec":{"containers":
[{"name":"demo-app","image":"172.30.192.109:5000/demo-project/demo-
app@sha256:9e7db6c87501bf88b652cf99c9573f4909cad6ce7e91bc54297f88d39d9a9a7
7","ports":[{"containerPort":8080,"protocol":"TCP"},
{"containerPort":8443,"protocol":"TCP"},
{"containerPort":8778,"protocol":"TCP"}],"resources":
{},"terminationMessagePath":"/dev/termination-
log","imagePullPolicy":"Always"}],"restartPolicy":"Always","terminationGra
cePeriodSeconds":30,"dnsPolicy":"ClusterFirst","securityContext":
{}}}},"status":
{"latestVersion":8,"observedGeneration":9,"replicas":1,"updatedReplicas":1
,"availableReplicas":1,"details":{"message":"image change","causes":
[{"type":"ImageChange","imageTrigger":{"from":
{"kind":"ImageStreamTag","namespace":"demo-project","name":"demo-
app:latest"}}}]},"conditions":
[{"type":"Available","status":"True","lastTransitionTime":"2017-04-
20T02:52:12Z","message":"Deployment config has minimum availability."},
{"type":"Progressing","status":"True","lastTransitionTime":"2017-04-
20T06:51:26Z","reason":"NewReplicationControllerAvailable","message":"Repl
ication controller \"demo-app-7\" has completed progressing"}]}}
  creationTimestamp: 2017-04-20T07:06:50Z
  generation: 7
  labels:
    app: demo-app
    openshift.io/deployment-config.name: demo-app
  name: demo-app-8
  namespace: demo-project
  resourceVersion: "913150"
  selfLink: /api/v1/namespaces/demo-project/replicationcontrollers/demo-
app-8
```

```
  uid: ed20f473-2597-11e7-8669-1418776f4b43
spec:
  replicas: 0
  selector:
    deployment: demo-app-8
    deploymentconfig: demo-app
  template:
    metadata:
      annotations:
        openshift.io/deployment-config.latest-version: "8"
        openshift.io/deployment-config.name: demo-app
        openshift.io/deployment.name: demo-app-8
      creationTimestamp: null
      labels:
        app: demo-app
        deployment: demo-app-8
        deploymentconfig: demo-app
    spec:
      containers:
      - image: 172.30.192.109:5000/demo-project/demo-
app@sha256:9e7db6c87501bf88b652cf99c9573f4909cad6ce7e91bc54297f88d39d9a9a7
7
        imagePullPolicy: Always
        name: demo-app
        ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 8443
          protocol: TCP
        - containerPort: 8778
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  observedGeneration: 7
  replicas: 0
```

5.3.3. Job Configuration

```
apiVersion: batch/v1
kind: Job
metadata:
  creationTimestamp: 2017-04-25T05:18:31Z
  labels:
    controller-uid: 9f3c1695-2976-11e7-8669-1418776f4b43
    job-name: demo-job
  name: demo-job
  namespace: demo-project
  resourceVersion: "703715"
  selfLink: /apis/batch/v1/namespaces/demo-project/jobs/demo-job
  uid: 9f3c1695-2976-11e7-8669-1418776f4b43
```

```
spec:
  completions: 1
  parallelism: 3
  selector:
    matchLabels:
      controller-uid: 9f3c1695-2976-11e7-8669-1418776f4b43
  template:
    metadata:
      creationTimestamp: null
      labels:
        controller-uid: 9f3c1695-2976-11e7-8669-1418776f4b43
        job-name: demo-job
      name: demo-job
    spec:
      containers:
      - command:
        - perl
        - -Mbignum=bpi
        - -wle
        - print bpi(2000)
        image: perl
        imagePullPolicy: Always
        name: demo-job
        resources: {}
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
      restartPolicy: Never
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  completionTime: 2017-04-25T05:19:26Z
  conditions:
  - lastProbeTime: 2017-04-25T05:19:26Z
    lastTransitionTime: 2017-04-25T05:19:26Z
    status: "True"
    type: Complete
  startTime: 2017-04-25T05:18:31Z
  succeeded: 1
```

5.3.4. Pod Configuration

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/created-by: |
      {"kind":"SerializedReference","apiVersion":"v1","reference":
{"kind":"ReplicationController","namespace":"demo-app","name":"demo-app-
1","uid":"e45f3d69-3b9a-11e7-a37b-
1418776f4b43","apiVersion":"v1","resourceVersion":"25862"}}
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: demo-app
    openshift.io/deployment.name: demo-app-1
    openshift.io/scc: restricted
  creationTimestamp: 2017-05-19T01:59:43Z
```

```
  generateName: demo-app-1-
  labels:
    app: demo-app
    deployment: demo-app-1
    deploymentconfig: demo-app
  name: demo-app-1-6vh7q
  namespace: demo-app
  resourceVersion: "25887"
  selfLink: /api/v1/namespaces/demo-app/pods/demo-app-1-6vh7q
  uid: d3630fd0-3c36-11e7-a37b-1418776f4b43
spec:
  containers:
  - image: 172.30.252.150:5000/demo-app/demo-
app@sha256:f8788dfceee980b67b041a4a7eca955f48790b294892670969d45de81470d36
2
    imagePullPolicy: Always
    name: demo-app
    ports:
    - containerPort: 8080
      protocol: TCP
    - containerPort: 8443
      protocol: TCP
    - containerPort: 8778
      protocol: TCP
    resources: {}
    securityContext:
      capabilities:
        drop:
        - KILL
        - MKNOD
        - SETGID
        - SETUID
        - SYS_CHROOT
      privileged: false
      runAsUser: 1000060000
      seLinuxOptions:
        level: s0:c8,c2
    terminationMessagePath: /dev/termination-log
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-30zcw
      readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-wlhh3
  nodeName: openshift-2.example.com
  restartPolicy: Always
  securityContext:
    fsGroup: 1000060000
    seLinuxOptions:
      level: s0:c8,c2
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes:
  - name: default-token-30zcw
```

```
    secret:
      defaultMode: 420
      secretName: default-token-30zcw
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2017-05-19T01:59:43Z
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: 2017-05-19T01:59:47Z
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: 2017-05-19T01:59:43Z
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID:
docker://0dcfd5d9d4fe864381bfb0a2c4d6441cdcded4a88708be473f2b8a1c7ebb4e17
    image: 172.30.252.150:5000/demo-app/demo-
app@sha256:f8788dfceee980b67b041a4a7eca955f48790b294892670969d45de81470d36
2
    imageID: docker-pullable://172.30.252.150:5000/demo-app/demo-
app@sha256:f8788dfceee980b67b041a4a7eca955f48790b294892670969d45de81470d36
2
    lastState: {}
    name: demo-app
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2017-05-19T01:59:46Z
  hostIP: 192.0.2.2
  phase: Running
  podIP: 192.128.0.21
  startTime: 2017-05-19T01:59:43Z
```

5.3.5. Route Configuration

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
    openshift.io/host.generated: "true"
  creationTimestamp: 2017-05-18T07:09:39Z
  labels:
    app: demo-app
  name: demo-app
  namespace: demo-app
  resourceVersion: "351291"
  selfLink: /oapi/v1/namespaces/demo-app/routes/demo-app
  uid: f589cd2f-3b98-11e7-a37b-1418776f4b43
spec:
```

```
  host: demo-app-demo-app.router.default.svc.cluster.local
  port:
    targetPort: 8080-tcp
  to:
    kind: Service
    name: demo-app
    weight: 90
  wildcardPolicy: None
status:
  ingress:
  - conditions:
    - lastTransitionTime: 2017-05-18T07:09:39Z
      status: "True"
      type: Admitted
    host: demo-app-demo-app.router.default.svc.cluster.local
    routerName: router
    wildcardPolicy: None
```