W. Caban, *Architecting and Operating OpenShift Clusters*
https://doi.org/10.1007/978-1-4842-4985-7_7

7. Administration

William Caban [1]

(1) Columbia, MD, USA

After deploying OpenShift platform as presented in Chapter 6, the administrative tasks of the platform start. The interaction with an OpenShift cluster is governed by the role-based access control (RBAC) objects. The RBAC determines whether a User is authorized to perform a given action within a Project. A User is an account that is used to interact with the OpenShift API. A User will be associated to one or more Groups that are used to assign privileges to multiple users at the same time.

This chapter focuses on the main tasks of user management (basic user management, groups, virtual users, and service accounts), security, quotas, and templates, which are powerful features for enabling self-service capabilities.

User and Groups

There are several types of users in OpenShift. The default user types are documented in Table 7-1.

Table 7-1 OpenShift Virtual Groups

User Type	Description
Regular users	Regular users are represented by the *User* object. This is the most common way users interact with OpenShift.
System users	This type of user is usually created automatically during the deployment and is used by the platform to interact with the OpenShift API.

User Type	Description
Service accounts	The service accounts users are represented by the *ServiceAccount* object. These are special system users associated with projects. The service accounts can be created automatically during Project creation or by a *Project* administrator.

Examples of some of the *system users* created during the deployment of OpenShift are

- Cluster administrators (i.e., system:admin)
- Per-node users (i.e., system:node:node1.ocp.example.com (http://node1.ocp.example.com))
- An anonymous user (system:anonymous)

During the creation of a new *Project*, OpenShift creates three service accounts that are used when executing certain actions in the *Project*:

- system:serviceaccount:<project-name>:deployer
- system:serviceaccount:<project-name>:builder
- system:serviceaccount:<project-name>:default

To access *OpenShift*, every user must be authenticated (i.e., using access tokens, certificates, etc.). The policy associated to the *User* object determines what the user is authorized to do in the cluster. When the user is authenticated, the policy associated to the *User* dictates the authorizations. When the API receives a request with no authentication or invalid authentication, these requests are processed as a request by the anonymous user *system:anonymous*.

Virtual Groups and Virtual Users

OpenShift provisions a series of system groups as the base classification for any user interacting with the platform. These special groups are referred to as *Virtual Groups* . Similarly, there is a special *Virtual User* used to identify for anonymous interactions. Table 7-2 lists the *Virtual Groups* and *Virtual Users.*

Table 7-2 OpenShift Virtual Groups

Virtual Group or Virtual User	Description
system:authenticated	This *Virtual Group* represents all the authenticated users.
system:authenticated:oauth	This *Virtual Group* represents authenticated users with an OAuth access token.

Virtual Group or Virtual User	Description
system:unauthenticated	This *Virtual Group* represents all the unauthenticated users.
system:anonymous	This *Virtual User* is used in conjunction with the system:unauthenticated *Virtual Group* to represent an unauthenticated user interacting with the OpenShift API.

Authentication, Authorization, and OpenShift RBAC

The OpenShift Master has a built-in OAuth server [1] used by the users to obtain an access token to interact with the API. The request for an OAuth token must specify the OAuth client that will receive and use the token (see Table 7-3).

Table 7-3 OpenShift OAuth Clients

OAuth Clients	Description
openshift-web-console	Request tokens to use for the web console
openshift-browser-client	Token requests at https://<master>/oauth/token/request with a user-agent that can handle interactive logins
openshift-challenging-client	Token requests with a user-agent that supports OAuth *WWW-Authenticate* challenges.

When a new OAuth Token request arrives to the OAuth server (#2 on Figure 7-1), the OAuth server uses the identity provider to determine the identity of the user making the request (#3 on Figure 7-1). Once the user identity is established, it maps the identity to the corresponding *User* (#4 on Figure 7-1). After successfully mapping the identity to the *User*, the OAuth server creates a token for that *User* and returns it to the original requester.

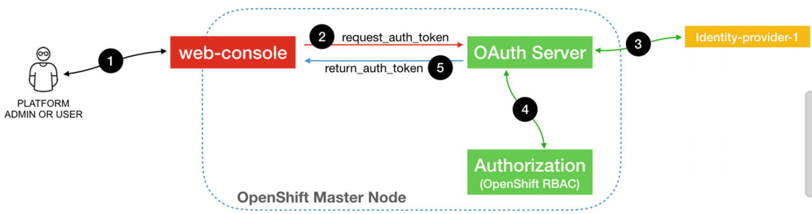

Figure 7-1 Sample flow for an OAuth Token request

> **NOTE** OpenShift supports the use of Service Account as OAuth clients [2] and the addition of OAuth client [3] definitions.

RBAC

The RBAC objects determine if a user is allowed to perform a specific action within a *Project*. The RBAC authorization is comprised of Rules, Roles, and Bindings (see Table 7-4 for more details).

Table 7-4 Authorization Constructs

Construct	Description
Rules	Represent the *Verbs* permitted on a set of Kubernetes and OpenShift objects.
Roles	Represent a collection of policy *Rules*. *Users* and *Groups* can be associated to multiple *Roles* at the same time.
Bindings	Represent the association of *Users* or *Groups* with a *Role*.
Verb	The *Verbs* are get, list, create, update, delete, delete collection, or watch.

Construct	Description
Identity	Represents the *User Name* and the list of *Groups* the *User* belongs to.

There are two levels of RBAC authorization in an OpenShift Cluster (see Table 7-5 for details).

Table 7-5 Levels of RBAC Authorizations

Construct	Description
Cluster RBAC	Refers to *Roles* and *Bindings* that are applicable cluster-wide and not scoped to a particular *Project*. *Cluster Role Bindings* can only reference *Cluster Roles* (*Roles* that exist cluster-wide).
Local RBAC	Refers to *Roles* and *Bindings* scoped to a particular *Project*. *Local Role Bindings* can reference *Cluster Roles* or *Local Roles* (*Roles* that only exist in a *Project*).

DEFAULT CLUSTER ROLES

OpenShift predefines a series of default *Cluster Roles* (see Table 7-6) that can be bound to *Users* or *Groups*. In addition, a cluster-admin user can define additional *Roles*.

Table 7-6 Default Cluster Roles

Default Cluster Role	**Description**
cluster-admin	A super-user that can perform any action on any Project. **Note:** When the *cluster-admin Rol*e is bound to a *User* with a *Local Binding*, that user will have full control over quota and actions on every resource in the *Project*.
admin	A Project manager. **Note:** When used in a *Local Binding*, a *User* with *admin Role* will have rights to view and modify any resource in the *Project* (except for Quota).
basic-user	A user that can get basic information about *Projects* and *Users*.

Default Cluster Role	Description
cluster-status	A user that can get basic cluster status information.
edit	A user that can modify most objects in a *Project* but does not have rights to view or modify *Roles* or *Bindings*.
self-provisioner	A user that can create their own *Projects.*
view	A user who can see, but not modify, most objects in a *Project*. They cannot view or modify *Roles* or *Bindings*.
cluster-reader	A user who can *read*, but not *view*, objects in the cluster.

SECURITY CONTEXT CONSTRAINTS

OpenShift provides granular control of the actions and access of a *Pod* with the capabilities provided by the *Security Context Constraints (SCC)*.

The SCC objects define the conditions that a Pod must met in order to be accepted into the system. The SCC controls the following:

1. Ability to run privileged *Containers*
2. Additional capabilities that can be requested by a *Container*
3. Ability to use *Host* directories as Volumes
4. *SELinux* context of the *Container*
5. The User ID
6. The use of *Host* namespaces and networking
7. Allocating an *FSGroup* [4] that owns the *Pod's* Volumes
8. Configuring allowable supplemental *Groups*
9. Requiring the use of a *read-only* root filesystem
10. Controlling the usage of *Volume types*
11. Configuring allowable *SECCOMP* profiles

OpenShift defines seven default SCC in a cluster. These default SCC are listed on Figure 7-2.

```
$ oc get scc
NAME              PRIV    CAPS  SELINUX    RUNASUSER         FSGROUP    SUPGROUP   PRIORITY  READONLYROOTFS  VOLUMES
anyuid            false   []    MustRunAs  RunAsAny          RunAsAny   RunAsAny   10        false           [configMap downwardAPI emptyDir persistentVolumeClaim projected secret]
hostaccess        false   []    MustRunAs  MustRunAsRange    MustRunAs  RunAsAny   <none>    false           [configMap downwardAPI emptyDir hostPath persistentVolumeClaim projected secret]
hostmount-anyuid  false   []    MustRunAs  RunAsAny          RunAsAny   RunAsAny   <none>    false           [configMap downwardAPI emptyDir hostPath nfs persistentVolumeClaim projected secret]
hostnetwork       false   []    MustRunAs  MustRunAsRange    MustRunAs  MustRunAs  <none>    false           [configMap downwardAPI emptyDir persistentVolumeClaim projected secret]
node-exporter     false   []    RunAsAny   RunAsAny          RunAsAny   RunAsAny   <none>    false           [*]
nonroot           false   []    MustRunAs  MustRunAsNonRoot  RunAsAny   RunAsAny   <none>    false           [configMap downwardAPI emptyDir persistentVolumeClaim projected secret]
privileged        true    [*]   RunAsAny   RunAsAny          RunAsAny   RunAsAny   <none>    false           [*]
restricted        false   []    MustRunAs  MustRunAsRange    MustRunAs  RunAsAny   <none>    false           [configMap downwardAPI emptyDir persistentVolumeClaim projected secret]
```

Figure 7-2 List of default SCC

By default, authenticated users are granted access to the *restricted SCC* (line #10 on Figures 7-2 and 7-3), while cluster administrators, Nodes, and the build controller are granted the *privileged SCC* (line #9 on Figure 7-2).

```
1  $ oc get --export scc/restricted -o yaml
2  allowHostDirVolumePlugin: false
3  allowHostIPC: false
4  allowHostNetwork: false
5  allowHostPID: false
6  allowHostPorts: false
7  allowPrivilegeEscalation: true
8  allowPrivilegedContainer: false
9  allowedCapabilities: null
10 apiVersion: security.openshift.io/v1
11 defaultAddCapabilities: null
12 fsGroup:
13   type: MustRunAs
14 groups:
15 - system:authenticated
16 kind: SecurityContextConstraints
17 metadata:
18   annotations:
19     kubernetes.io/description: restricted denies access to all host features and requires
20       pods to be run with a UID, and SELinux context that are allocated to the namespace.  This
21       is the most restrictive SCC and it is used by default for authenticated users.
22   creationTimestamp: null
23   name: restricted
24   selfLink: /apis/security.openshift.io/v1/securitycontextconstraints/restricted
25 priority: null
26 readOnlyRootFilesystem: false
27 requiredDropCapabilities:
28 - KILL
29 - MKNOD
30 - SETUID
31 - SETGID
32 runAsUser:
33   type: MustRunAsRange
34 seLinuxContext:
35   type: MustRunAs
36 supplementalGroups:
37   type: RunAsAny
38 users: []
39 volumes:
40 - configMap
41 - downwardAPI
42 - emptyDir
43 - persistentVolumeClaim
44 - projected
45 - secret
```

Figure 7-3 The “restricted SCC” definition

As it can be seen from the *restricted SCC* definition (Figure 7-3), this SCC enforces the following restrictions:

- Pods cannot run as privileged (line #8 on Figure 7-3).
- Pods cannot use Host directory Volumes (lines #39 to #45 on Figure 7-3).
- Pods run as a user in a preallocated range of UID (lines #32 and #33 on Figure 7-3).
- Pods run with a preallocated SELinux MCS label (lines #34 and #35 on Figure 7-3).

- Pods can use any supplemental Group (lines #36 and #37 on Figure 7-3).

The SCC strategies [5] are settings and strategies that fall into three categories:

- Controlled by a boolean (default to the most restrictive value)
- Controlled by an allowable set specifying the allowed values
- Controlled by a strategy in which a mechanism generates the value and ensures the value is allowed (see Table 7-7)

Table 7-7 SCC Strategies

SCC Strategy	Options
RUNASUSER	MustRunAs, MustRunAsRange, MustRunAsNonRoot, RunAsAny
SELINUXCONTEXT	MustRunAs, RunAsAny
SUPPLEMENTALGROUPS	MustRunAs, RunAsAny
FSGROUP	MustRunAs, RunAsAny

SCC Strategy	Options
volumes	azureFile, azureDisk, flocker, flexVolume, hostPath, emptyDir, gcePersistentDisk, awsElasticBlockStore, gitRepo, secret, nfs, iscsi, glusterfs, persistentVolumeClaim, rbd, cinder, cephFS, downwardAPI, fc, configMap, vsphereVolume, quo byte, photonPersistenDisk, projected, portworxVolume, scaleIO, storageos, “*”, none

SECCOMP PROFILES

SECCOMP (secure computing mode) is a security facility in the Linux Kernel that allows a system administrator to limit access by Containers to the system features. The combination of restricted and allowed calls are arranged in profiles. Different profiles can be passed to different *Containers*. This provides a fine-grained control over the *syscalls* available from a *Container*.

> **NOTE** [6] SECCOMP is a Kernel feature, and as such, it must be enabled on the system.

To enable SECCOMP for a Pod, the following annotations are required in the Pod configuration:

- `seccomp.security.alpha.kubernetes.io/pod: <unconfined>`
- `container.seccomp.security.alpha.kubernetes.io/<container_name>: <localhost/profile_name>`

In addition, edit the `/etc/origin/node/node-config.yaml` to define the `seccomp-profile-root` directory where the local SECCOMP profiles will be stored. (See Listing 7-1.)

```
# Edit /etc/origin/node/node-config.yaml
kubeletArguments:
  ...
  seccomp-profile-root:
    - "/path/to/seccomp/profiles"
# Restart the Node services
$ sudo systemctl restart atomic-openshift-node
```

Listing 7-1 Defining SECCOMP profiles directory

To control the *SECCOMP* profiles that may be used in the *OpenShift* platform and to set the default *SECCOMP* profile, configure the *SCC* with the `seccompProfiles` field. When using a custom SECCOMP profile, the format for the field is `localhost/<profile-name>.` (See Listing 7-2.)

```
seccompProfiles:
- localhost/<profile-name>
```

Listing 7-2 Configuring SECCOMP in SCC profiles

ENABLING UNSAFE SYSCTL

When SYSCTL are *namespaced*, their value can be set independently for each *Pod*. This is a requirement for *SYSCTLS* to be accessible in a *Pod* within *Kubernetes.*

A SYSCTL is considered safe for a Pod if

- Does not influence any other *Pod* on the *Node*

- Does not harm the *Node’s* health

- Does not gain *CPU* or *memory* resources outside the resource limits of a *Pod*

All safe [7] SYSCTLS are enabled by default. All other SYSCTLS are considered unsafe and are disabled by default. A user with cluster-admin privileges can manually enable unsafe SYSCTLS on a per-node basis.

Enabling unsafe sysctls requires modifying the `kubeletArguments` on the `/etc/origin/node/node-config.yaml` in the Nodes that will be supporting the unsafe SYSCTLS (see Listing 7-3).

```
# Edit /etc/origin/node/node-config.yaml
kubeletArguments:
  ...
  allowed-unsafe-sysctls:
    - "kernel.msg*,net.ipv4.route.min_pmtu"
# Restart the Node services
$ sudo systemctl restart atomic-openshift-node
```

Listing 7-3 Enabling unsafe SYSCTLS

The configuration of SYSCTLS for a Pod is done by setting the values under the securityContext in the Pod configuration (see Listing 7-4).

> **NOTE** There is no distinction between safe and unsafe sysctls in the Pod configuration.

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
spec:
  securityContext:
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.ipv4.route.min_pmtu
      value: "552"
```

```
    - name: kernel.msgmax
      value: "65536"
  ...
```

Listing 7-4 Example setting SYSCTLS for Pod

> **NOTE** A Pod using unsafe SYSCTLS will fail to run on any *Node* where the unsafe SYSCTLS have not been explicitly enabled.

IDENTITY PROVIDERS

Configuring the identity provider [8] for the built-in OAuth server can be done during the installation or after the installation.

The OpenShift 3.11.x supported identity providers are

- **Deny All**: Default identity provider. Denies access for all usernames and passwords.
- **Allow All**: Allows access to any non-empty username with any non-empty password to log in. Used for testing purposes. (Used as default if running without a master configuration file.)
- **HTPasswd**: Validates usernames and passwords against a flat file generated using *htpasswd.*
- **Keystone**: Uses the OpenStack identity project for authentication.
- **LDAP**: Validates usernames and password against an LDAPv3 server using simple bind authentication.
- **Basic Authentication** (remote): Allows users to log in to OpenShift with credentials validated against a remote identity provider. (Must use an HTTPS connection to remote server.)
- **Request Header**: Identifies users from request header values like X-Remote-User.
- **GitHub**: Uses the OAuth authentication from GitHub.

- **GitLab**: Uses the OAuth authentication from GitLab (versions 7.7.0 to 11.0). If using GitLab version 11.1 or later, use the OpenID Connect.

- **Google**: Uses Google's OpenID Connect integration.

- **OpenID Connect**: Integrates with an OpenID Connect identity provider.

The configuration of the identity provider uses a `mappingMethod` to define how new identities are mapped to users when they log in to *OpenShift*. The value will be one of the following:

- **claim**: Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity. (This is the default configuration.)

- **lookup**: Looks up an existing identity, user identity mapping, and user. It does not provision users or identities if they don't exist. Using this method requires cluster administrators to set up identities and users manually or by an external process.

- **generate**: Provisions a user with the identity's preferred user name. If a user with the preferred user name already exists, a unique user name is generated (i.e., username2).

- **add**: Provisions a user with the identity's preferred user name. If a user with that user name already exists, the identity is mapped to the existing user. (Required when multiple identity providers are configured that identify the same set of users.)

Managing Users and Groups

The creation of a user depends on the configuration of the `mappingMethod` in the *identity provider*. The manual creation of a user is as shown in Listing 7-5.

```
$ oc create user <username> --full-name="User Name"
```

Listing 7-5 Manual creation of a user

Managing the roles, groups, and SCC for a user can be done with the `oc` client command with the options as shown in Figure 7-4.

```
$ oc adm policy
Manage policy on the cluster

These commands allow you to assign and manage the roles and policies that apply to users. The reconcile commands allow
you to reset and upgrade your system policies to the latest default policies.

To see more information on roles and policies, use the 'get' and 'describe' commands on the following resources:
'clusterroles', 'clusterpolicy', 'clusterrolebindings', 'roles', 'policy', 'rolebindings', and 'scc'.

Usage:
  oc adm policy [flags]

Discover:
  who-can                         List who can perform the specified action on a resource
  scc-subject-review              Check whether a user or a ServiceAccount can create a Pod.
  scc-review                      Checks which ServiceAccount can create a Pod

Manage project membership:
  remove-user                     Remove user from the current project
  remove-group                    Remove group from the current project

Assign roles to users and groups:
  add-role-to-user                Add a role to users or serviceaccounts for the current project
  add-role-to-group               Add a role to groups for the current project
  remove-role-from-user           Remove a role from users for the current project
  remove-role-from-group          Remove a role from groups for the current project

Assign cluster roles to users and groups:
  add-cluster-role-to-user        Add a role to users for all projects in the cluster
  add-cluster-role-to-group       Add a role to groups for all projects in the cluster
  remove-cluster-role-from-user   Remove a role from users for all projects in the cluster
  remove-cluster-role-from-group  Remove a role from groups for all projects in the cluster

Manage policy on pods and containers:
  add-scc-to-user                 Add security context constraint to users or a service account
  add-scc-to-group                Add security context constraint to groups
  remove-scc-from-user            Remove user from scc
  remove-scc-from-group           Remove group from scc

Upgrade and repair system policy:
  reconcile-cluster-roles         Update cluster roles to match the recommended bootstrap policy
  reconcile-cluster-role-bindings Update cluster role bindings to match the recommended bootstrap policy
  reconcile-sccs                  Replace cluster SCCs to match the recommended bootstrap policy

Use "oc adm policy <command> --help" for more information about a given command.
Use "oc adm options" for a list of global command-line options (applies to all commands).
```

Figure 7-4 Manage user roles, groups, and SCC

USING SERVICE ACCOUNTS

Service Accounts (SA) provide a flexible way to control API access without sharing a regular *User* credential.

The user name of a *Service Account (SA)* is derived from its Project and name (see Listing 7-6). The *Service Account* can be granted *Roles* (see Listing 7-6) as any other user in the system.

```
# Format of a Service Account name
```

```
system:serviceaccount:<project-name>:<name>
# Assigning Role to a Service Account
$ oc policy add-role-to-user <role-name>
system:serviceaccount:<project-name>:<name>
# Assigning Role to a Service Account from the Project
it belongs to
$ oc policy add-role-to-user <role-name> -z <SA-name>
```

Listing 7-6 Assigning Roles to Service Account

Each Service Account belongs to two groups:

- `system:serviceaccount`
- `system:serviceaccount:<project-name>`

During the creation of a new *Service Account*, [9] the system ensures to add two secrets to it (see Listing 7-7):

- An API token
- Credentials for the OpenShift Container Registry

> **NOTE** The generated API token and registry credentials do not expire. If the secret is deleted, a new one is automatically generated to replace it.

```
# Creating a Service Account name
$ oc create sa sa-demo (or) oc create serviceaccount
sa-demo
serviceaccount/sa-demo created
$ oc describe sa sa-demo
Name:                  sa-demo
Namespace:             demo
Labels:                <none>
Annotations:           <none>
Image pull secrets:    sa-demo-dockercfg-rj875
Mountable secrets:     sa-demo-token-xph4v
                       sa-demo-dockercfg-rj875
Tokens:                sa-demo-token-txlcq
                       sa-demo-token-xph4v
Events:                <none>
```

Listing 7-7 Creating a Service Account

To associate a *ServiceAccount* to a *Pod*, use the `serviceAccountName` under the *Pod's* `spec` definition (see Listing 7-8).

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
spec:
  serviceAccountName: sa-demo
  ...
```

Listing 7-8 Creating a Service Account

The API tokens from the ServiceAccount associated to the Pod are mounted as a file at `/var/run/secrets/kubernetes.io/serviceaccount/token` inside the *Container*.

> **NOTE** The default *ServiceAccount* is used when no explicit *ServiceAccount* is specified in the Pod definition.

Quotas and Limit Ranges

Quotas and *Limit Ranges* are objects that can be set by a cluster administrator to limit the number of objects or amount of compute resources that are used by a particular *Project*. While *LimitRanges* specify the limits of compute resources in a *Project* on per-object basis, *Quotas* act as the upper limit for the total compute resources or number of objects in the Project.

LimitRange object can set up compute resource constraints in a *Project* at the following level:

- Pod
- Container

- Image
- ImageStream
- PersistentVolumeClaim

To apply a *LimitRange*[10] to a *Project,* create the object definition with the specification (see definition in Figure 7-5).

```
$ oc create -f demo-limit-range.yaml -n demo
limitrange/demo-limit-range created

$ oc describe limitrange demo-limit-range
Name:                      demo-limit-range
Namespace:                 demo
Type                       Resource                Min   Max   Default Request  Default Limit  Max Limit/Request Ratio
----                       --------                ---   ---   ---------------  -------------  -----------------------
Pod                        cpu                     200m  2     -                -              -
Pod                        memory                  6Mi   1Gi   -                -              -
Container                  memory                  4Mi   1Gi   100Mi            200Mi          -
Container                  cpu                     100m  2     200m             300m           10
openshift.io/Image         storage                 -     1Gi   -                -              -
openshift.io/ImageStream   openshift.io/image-tags -     20    -                -              -
openshift.io/ImageStream   openshift.io/images     -     30    -                -              -
```

Figure 7-5 Creating and verifying LimitRange

All resource creation or modification requests are checked against the *LimitRange* in the *Project*. The resource creation or modification is rejected if it violates the constraints (see Figure 7-6).

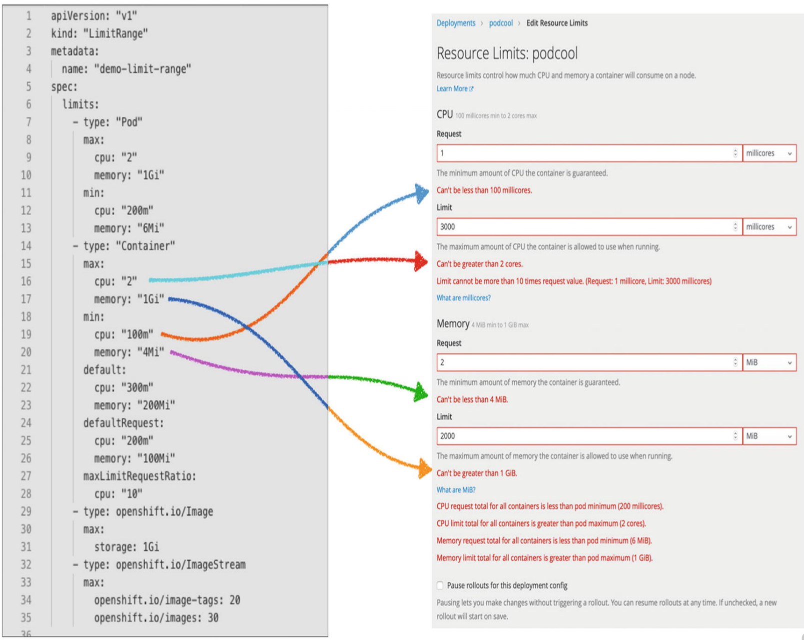

Figure 7-6 LimitRange and its effect on Pod requests

The *ResourceQuota* object is used to set up *Project*-level *Quota* to limit the number of objects in a *Project* or the total *Limits* for a *Project.* Figure 7-7 shows an example defining and verifying the creation of a *ResourceQuota.*

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: demo-quota
spec:
  hard:
    pods: "3"
    requests.cpu: "300m"
    requests.memory: 512Mi
    limits.cpu: "2"
    limits.memory: 2Gi
```

```
$ oc create -f demo-quota.yaml -n demo
resourcequota/demo-quota created

$ oc describe resourcequota demo-quota
Name:              demo-quota
Namespace:         demo
Resource           Used  Hard
--------           ----  ----
limits.cpu         0     2
limits.memory      0     2Gi
pods               1     3
requests.cpu       0     300m
requests.memory    0     512Mi
```

Figure 7-7 Definition and creation of ResourceQuota

When a particular request for creation or modification of a resource violates a Quota, the system will prevent the creation or modification of the resource (see Figure 7-8).

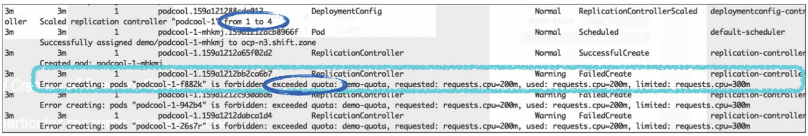

Figure 7-8 Example of quota enforcement

OpenShift Service Catalogs

OpenShift includes a *Service Catalog* which implements the Open Service API[11] (OSP API) for Kubernetes. This capability allows users to connect applications deployed in OpenShift to services instantiated through service brokers.

A user with cluster-admin privileges registers one or more *Service Brokers* with *OpenShift* cluster. Each Service Broker defines a set of *Cluster Service Classes* and *Service Plans* available to users.

Users request to provision or deprovision a resource provided by a *Service Class*. When provisioning a new resource, the *Users bind* the *service instance* with their local application *Pods*.

OpenShift provides two Service Brokers with the Service Catalog:

- **Template Service Broker (TSB)** gives the visibility into the *Instant App* and *Quickstart Templates*[12] that are shipped with *OpenShift*. In addition, the TSB makes available as a service any services defined as an *OpenShift Template.*

- **OpenShift Ansible Broker (OAB)**[13] is an implementation of the *OSB API* that manages application defined by *Ansible Playbook Bundles (APBs).*

OPENSHIFT TEMPLATES

OpenShift Templates provide a way to parameterize the creation of any OpenShift and Kubernetes objects. A template can be processed to create anything the user executing the Template has the permission to create within a Project (i.e., Services, BuildConfig, Deployments, Routes, etc.).

Templates are one of the mechanisms used to provide self-service capabilities with OpenShift. They provide a way for developers to deploy, on self-serve style, applications or backend stacks, when needed, while administrators retain full control on how a particular application or backend stack is implemented.

A Template can be executed from CLI or using the web console if the Template has been uploaded to the Project or Global Template library. Installing a Template can be done over GUI or CLI (see Figure 7-9).

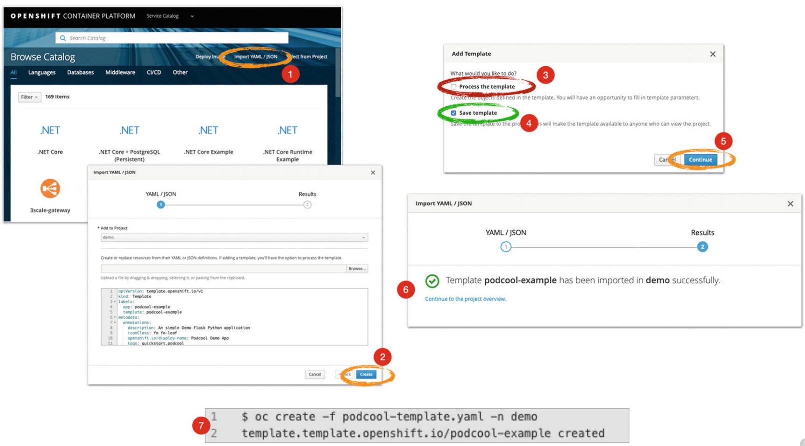

Figure 7-9 Installing OpenShift Template

When using the *GUI* to install an *OpenShift Template,* there are two options: an option to immediately process the *Template* (#3 on Figure 7-9) and another option to save the template to the service catalog (#4 on Figure 7-9).

> **NOTE** When installing a *Template,* it needs to be associated to a namespace. To make the *Template* available cluster-wide, it should be installed into the *openshift Project.*

An example of an OpenShift Template is shown in Listing 7-9.

```
apiVersion: template.openshift.io/v1
kind: Template
labels:
  app: podcool-example
  template: podcool-example
metadata:
  annotations:
    description: An simple Demo Flask Python
application
    iconClass: fa fa-leaf
    openshift.io/display-name: Podcool Demo App
    tags: quickstart,podcool
  name: podcool-example
objects:
- apiVersion: v1
  kind: Service
  metadata:
    annotations:
      description: Exposes and load balances the
application pods
    name: podcool-example
  spec:
    ports:
    - name: web
      port: 8080
      targetPort: 8080
    selector:
      name: podcool-example
- apiVersion: v1
```

```
  kind: ImageStream
  metadata:
    annotations:
      description: Keeps track of changes in the
application image
    name: podcool-example
- apiVersion: v1
  kind: BuildConfig
  metadata:
    annotations:
      description: Defines how to build the
application
    name: podcool-example
  spec:
    output:
      to:
        kind: ImageStreamTag
        name: podcool-example:latest
    source:
      contextDir: ${CONTEXT_DIR}
      git:
        ref: ${SOURCE_REPOSITORY_REF}
        uri: ${SOURCE_REPOSITORY_URL}
      type: Git
    strategy:
      sourceStrategy:
        from:
          kind: ImageStreamTag
          name: python:3.6
          namespace: openshift
      type: Source
    triggers:
    - type: ConfigChange
    - github:
        secret: ${GITHUB_WEBHOOK_SECRET}
      type: GitHub
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
```

```
    annotations:
      description: Defines how to deploy the
application server
    name: podcool-example
  spec:
    replicas: 1
    selector:
      name: podcool-example
    strategy:
      type: Rolling
    template:
      metadata:
        labels:
          name: podcool-example
        name: podcool-example
      spec:
        containers:
        - image: podcool-example
          name: podcool-example
          ports:
          - containerPort: 8080
          env:
          - name: APP_VERSION
            value: v1
          - name: APP_MESSAGE
            value: Deployment from Template
    triggers:
    - imageChangeParams:
        automatic: true
        containerNames:
        - podcool-example
        from:
          kind: ImageStreamTag
          name: podcool-example:latest
      type: ImageChange
    - type: ConfigChange
parameters:
- description: The URL of the repository with your
application source code
```

```
  name: SOURCE_REPOSITORY_URL
  value: https://github.com/williamcaban/podcool.git
- description: Set this to a branch name, tag or other
ref of your repository if you
    are not using the default branch
  name: SOURCE_REPOSITORY_REF
- description: Set this to the relative path to your
project if it is not in the root
    of your repository
  name: CONTEXT_DIR
- description: Github trigger secret.  A difficult to
guess string encoded as part
    of the webhook URL.  Not encrypted.
  from: '[a-zA-Z0-9]{40}'
  generate: expression
  name: GITHUB_WEBHOOK_SECRET
```

Listing 7-9 OpenShift Template example

An *OpenShift Template*[14] can use or create any *OpenShift* and *Kubernetes* object the user executing it has privileges to create in a *Project*. That is a wide range of options and possible objects to create with a *Template*. As such, the process of writing *OpenShift Templates* is beyond the scope of this book.

Summary

This chapter focused on the main tasks of user management, security, quotas, and Templates. With respect to user management, this chapter covered basic user management, groups, virtual users, and service accounts. The security topics covered setting secure profiles, quotas, and limits. Finally, this chapter described using OpenShift Templates with the service catalog as a mechanism to provide self-service capabilities to the users.

The administration of OpenShift Clusters involves much more than what is covered in the chapter, and the reader should explore additional topics that will enhance the experience for the users while facilitating sustainable operations of the platform.

One of the OpenShift features designed to enhance the developer experience is the native capability to support CI/CD pipelines. The OpenShift Pipelines are covered in Chapter 8 .

Footnotes

1 OpenShift OAuth Server: https://docs.openshift.com/container-platform/3.11/architecture/additional_concepts/authentication.html#oa

2 Using Service Account as OAuth client: https://docs.openshift.com/container-platform/3.11/architecture/additional_concepts/authentication.html#se accounts-as-oauth-clients

3 To define additional OAuth clients, refer to https://docs.openshift.com/container-platform/3.11/architecture/additional_concepts/authentication.html#oa clients

4 The FSGroup defines Pod's "file system group" ID, for more information refer to the documentation at https://docs.openshift.com/container-platform/3.11/install_config/persistent_storage/pod_security_context

5 Details about the SCC Strategies: https://docs.openshift.com/container-platform/3.11/architecture/additional_concepts/authorization.html#au SCC-strategies

6 To check if SECCOMP is enabled, consult the documentation at https://docs.openshift.com/container-platform/3.11/admin_guide/seccomp.html#seccomp-enabling-seccomp

7 For additional information of safe vs. unsafe sysctls, refer to https://docs.openshift.com/container-platform/3.11/admin_guide/sysctls.html#safe-vs-unsafe-sysclts

8 Additional details on configuring identity providers: https://docs.openshift.com/container-platform/3.11/install_config/configuring_authentication.html#identit providers-configuring

9 Additional information about Service Accounts https://docs.openshift.com/container-platform/3.11/dev_guide/service_accounts.html

10 Additional information about creating LimitRange: https://docs.openshift.com/container-platform/3.11/admin_guide/limits.html#creating-a-limit-range

11 Details about the Open Service Broker API are available at the project home page: www.openservicebrokerapi.org

12 Additional information on using Instant App and Quickstart Templates is available at https://docs.openshift.com/container-

platform/3.11/dev_guide/templates.html#using-the-instantapp-templates

13 Additional details about the Ansible Service Broker is available at https://docs.openshift.com/container-platform/3.11/architecture/service_catalog/ansible_service_broker.htm ansible-service-broker

14 Additional information about writing OpenShift Templates: https://docs.openshift.com/container-platform/3.11/dev_guide/templates.html