OpenShift Dedicated 4

Authentication

Configuring user authentication, encryption, and access controls for users and services

Last Updated: 2020-05-06

OpenShift Dedicated 4 Authentication

Configuring user authentication, encryption, and access controls for users and services

Legal Notice

Abstract

This document provides instructions for defining identity providers in OpenShift Dedicated 4.

Table of Contents

CHAPTER 1. UNDERSTANDING AUTHENTICATION

For users to interact with OpenShift Dedicated, they must first authenticate to the cluster. The authentication layer identifies the user associated with requests to the OpenShift Dedicated API. The authorization layer then uses information about the requesting user to determine if the request is allowed.

1.1. USERS

A *user* in OpenShift Dedicated is an entity that can make requests to the OpenShift Dedicated API. An OpenShift Dedicated user object represents an actor which can be granted permissions in the system by adding roles to them or to their groups. Typically, this represents the account of a developer or administrator that is interacting with OpenShift Dedicated.

Several types of users can exist:

Regular users	This is the way most interactive OpenShift Dedicated users are represented. Regular users are created automatically in the system upon first login or can be created via the API. Regular users are represented with the **User** object. Examples: **joe alice**
System users	Many of these are created automatically when the infrastructure is defined, mainly for the purpose of enabling the infrastructure to interact with the API securely. They include a cluster administrator (with access to everything), a per-node user, users for use by routers and registries, and various others. Finally, there is an **anonymous** system user that is used by default for unauthenticated requests. Examples: **system:admin system:openshift-registry system:node:node1.example.com**
Service accounts	These are special system users associated with projects; some are created automatically when the project is first created, while project administrators can create more for the purpose of defining access to the contents of each project. Service accounts are represented with the **ServiceAccount** object. Examples: **system:serviceaccount:default:deployer system:serviceaccount:foo:builder**

Each user must authenticate in some way in order to access OpenShift Dedicated. API requests with no authentication or invalid authentication are authenticated as requests by the **anonymous** system user. Once authenticated, policy determines what the user is authorized to do.

1.2. GROUPS

A user can be assigned to one or more *groups*, each of which represent a certain set of users. Groups are useful when managing authorization policies to grant permissions to multiple users at once, for example allowing access to objects within a project, versus granting them to users individually.

In addition to explicitly defined groups, there are also system groups, or *virtual groups*, that are automatically provisioned by the cluster.

The following default virtual groups are most important:

Virtual group	Description
system:authenticated	Automatically associated with all authenticated users.
system:authenticated:oauth	Automatically associated with all users authenticated with an OAuth access token.
system:unauthenticated	Automatically associated with all unauthenticated users.

1.3. API AUTHENTICATION

Requests to the OpenShift Dedicated API are authenticated using the following methods:

OAuth Access Tokens

- Obtained from the OpenShift Dedicated OAuth server using the ***<namespace_route>*/oauth/authorize** and ***<namespace_route>*/oauth/token** endpoints.
- Sent as an **Authorization: Bearer…** header.
- Sent as a websocket subprotocol header in the form **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** for websocket requests.

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.

1.3.1. OpenShift Dedicated OAuth server

The OpenShift Dedicated master includes a built-in OAuth server. Users obtain OAuth access tokens to authenticate themselves to the API.

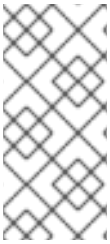
When a person requests a new OAuth token, the OAuth server uses the configured identity provider to determine the identity of the person making the request.

It then determines what user that identity maps to, creates an access token for that user, and returns the token for use.

1.3.1.1. OAuth token requests

Every request for an OAuth token must specify the OAuth client that will receive and use the token. The following OAuth clients are automatically created when starting the OpenShift Dedicated API:

OAuth Client	Usage
openshift-browser-client	Requests tokens at ***<namespace_route>*/oauth/token/request** with a user-agent that can handle interactive logins.
openshift-challenging-client	Requests tokens with a user-agent that can handle **WWW-Authenticate** challenges.



NOTE

<namespace_route> refers to the namespace's route. This is found by running the following command.

```
oc get route oauth-openshift -n openshift-authentication -o json | jq .spec.host
```

All requests for OAuth tokens involve a request to ***<namespace_route>*/oauth/authorize**. Most authentication integrations place an authenticating proxy in front of this endpoint, or configure OpenShift Dedicated to validate credentials against a backing identity provider. Requests to ***<namespace_route>*/oauth/authorize** can come from user-agents that cannot display interactive login pages, such as the CLI. Therefore, OpenShift Dedicated supports authenticating using a **WWW-Authenticate** challenge in addition to interactive login flows.

If an authenticating proxy is placed in front of the ***<namespace_route>*/oauth/authorize** endpoint, it sends unauthenticated, non-browser user-agents **WWW-Authenticate** challenges rather than displaying an interactive login page or redirecting to an interactive login flow.

NOTE

To prevent cross-site request forgery (CSRF) attacks against browser clients, only send Basic authentication challenges with if a **X-CSRF-Token** header is on the request. Clients that expect to receive Basic **WWW-Authenticate** challenges must set this header to a non-empty value.

If the authenticating proxy cannot support **WWW-Authenticate** challenges, or if OpenShift Dedicated is configured to use an identity provider that does not support WWW-Authenticate challenges, you must use a browser to manually obtain a token from ***<namespace_route>*/oauth/token/request**.

CHAPTER 2. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION

2.1. IDENTITY PROVIDER PARAMETERS

The following parameters are common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
mappingMethod	Defines how new identities are mapped to users when they log in. Enter one of the following values: **claim** The default value. Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity. **lookup** Looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. This allows cluster administrators to set up identities and users manually, or using an external process. Using this method requires you to manually provision users. **generate** Provisions a user with the identity's preferred user name. If a user with the preferred user name is already mapped to an existing identity, a unique user name is generated. For example, **myuser2**. This method should not be used in combination with external processes that require exact matches between OpenShift Dedicated user names and identity provider user names, such as LDAP group sync. **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, adding to any existing identity mappings for the user. Required when multiple identity providers are configured that identify the same set of users and map to the same user names.

NOTE

When adding or changing identity providers, you can map identities from the new provider to existing users by setting the **mappingMethod** parameter to **add**.

2.2. SUPPORTED IDENTITY PROVIDERS

You can configure the following types of identity providers:

Identity provider	Description
LDAP	Configure the **ldap** identity provider to validate user names and passwords against an LDAPv3 server, using simple bind authentication.
GitHub or GitHub Enterprise	Configure a **github** identity provider to validate user names and passwords against GitHub or GitHub Enterprise's OAuth authentication server.

Identity provider	Description
Google	Configure a **google** identity provider using Google's OpenID Connect integration.
OpenID Connect	Configure an **oidc** identity provider to integrate with an OpenID Connect identity provider using an Authorization Code Flow.

CHAPTER 3. CONFIGURING IDENTITY PROVIDERS

3.1. CONFIGURING AN LDAP IDENTITY PROVIDER

Configure the **ldap** identity provider to validate user names and passwords against an LDAPv3 server, using simple bind authentication.

3.1.1. About LDAP authentication

During authentication, the LDAP directory is searched for an entry that matches the provided user name. If a single unique match is found, a simple bind is attempted using the distinguished name (DN) of the entry plus the provided password.

These are the steps taken:

1. Generate a search filter by combining the attribute and filter in the configured **url** with the user-provided user name.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny access.
3. Attempt to bind to the LDAP server using the DN of the entry retrieved from the search, and the user-provided password.
4. If the bind is unsuccessful, deny access.
5. If the bind is successful, build an identity using the configured attributes as the identity, email address, display name, and preferred user name.

The configured **url** is an RFC 2255 URL, which specifies the LDAP host and search parameters to use. The syntax of the URL is:

```
ldap://host:port/basedn?attribute?scope?filter
```

For this URL:

URL Component	Description
ldap	For regular LDAP, use the string **ldap**. For secure LDAP (LDAPS), use **ldaps** instead.
host:port	The name and port of the LDAP server. Defaults to **localhost:389** for ldap and **localhost:636** for LDAPS.
basedn	The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but it could also specify a subtree in the directory.
attribute	The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use **uid**. It is recommended to choose an attribute that will be unique across all entries in the subtree you will be using.

URL Component	Description
scope	The scope of the search. Can be either **one** or **sub**. If the scope is not provided, the default is to use a scope of **sub**.
filter	A valid LDAP search filter. If not provided, defaults to **(objectClass=*)**

When doing searches, the attribute, filter, and provided user name are combined to create a search filter that looks like:

```
(&(<filter>)(<attribute>=<username>))
```

For example, consider a URL of:

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

When a client attempts to connect using a user name of **bob**, the resulting search filter will be **(&(enabled=true)(cn=bob))**.

If the LDAP directory requires authentication to search, specify a **bindDN** and **bindPassword** to use to perform the entry search.

3.2. CONFIGURING A GITHUB OR GITHUB ENTERPRISE IDENTITY PROVIDER

Configure a **github** identity provider to validate user names and passwords against GitHub or GitHub Enterprise's OAuth authentication server. OAuth facilitates a token exchange flow between OpenShift Dedicated and GitHub or GitHub Enterprise.

You can use the GitHub integration to connect to either GitHub or GitHub Enterprise. For GitHub Enterprise integrations, you must provide the **hostname** of your instance and can optionally provide a **ca** certificate bundle to use in requests to the server.

NOTE

The following steps apply to both GitHub and GitHub Enterprise unless noted.

Configuring GitHub authentication allows users to log in to OpenShift Dedicated with their GitHub credentials. To prevent anyone with any GitHub user ID from logging in to your OpenShift Dedicated cluster, you can restrict access to only those in specific GitHub organizations.

3.2.1. Registering a GitHub application

To use GitHub or GitHub Enterprise as an identity provider, you must register an application to use.

Procedure

1. Register an application on GitHub:

- For GitHub, click Settings → Developer settings → OAuth Apps → Register a new OAuth application.
- For GitHub Enterprise, go to your GitHub Enterprise home page and then click **Settings → Developer settings → Register a new application**.

2. Enter an application name, for example **My OpenShift Install**.
3. Enter a homepage URL, such as **https://oauth-openshift.apps.<cluster-name>.<cluster-domain>**.
4. Optional: Enter an application description.
5. Enter the authorization callback URL, where the end of the URL contains the identity provider **name**:

```
https://oauth-openshift.apps.<cluster-name>.<cluster-domain>/oauth2callback/<idp-provider-name>
```

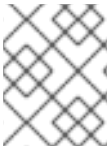
For example:

```
https://oauth-openshift.apps.example-openshift-cluster.com/oauth2callback/github/
```

6. Click **Register application**. GitHub provides a Client ID and a Client Secret. You need these values to complete the identity provider configuration.

3.3. CONFIGURING A GOOGLE IDENTITY PROVIDER

Configure a **google** identity provider using Google's OpenID Connect integration.



NOTE

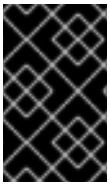
Using Google as an identity provider requires users to get a token using **<master>/oauth/token/request** to use with command-line tools.

WARNING

Using Google as an identity provider allows any Google user to authenticate to your server. You can limit authentication to members of a specific hosted domain with the **hostedDomain** configuration attribute.

3.4. CONFIGURING A OPENID CONNECT IDENTITY PROVIDER

Configure an **oidc** identity provider to integrate with an OpenID Connect identity provider using an Authorization Code Flow.



IMPORTANT

The Authentication Operator in OpenShift Dedicated requires that the configured OpenID Connect identity provider implements the OpenID Connect Discovery specification.

NOTE

ID Token and **UserInfo** decryptions are not supported.

By default, the **openid** scope is requested. If required, extra scopes can be specified in the **extraScopes** field.

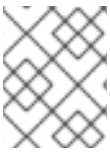
Claims are read from the JWT **id_token** returned from the OpenID identity provider and, if specified, from the JSON returned by the **UserInfo** URL.

At least one claim must be configured to use as the user's identity. The standard identity claim is **sub**.

You can also indicate which claims to use as the user's preferred user name, display name, and email address. If multiple claims are specified, the first one with a non-empty value is used. The standard claims are:

sub	Short for "subject identifier." The remote identity for the user at the issuer.
preferred_username	The preferred user name when provisioning a user. A shorthand name that the user wants to be referred to as, such as **janedoe**. Typically a value that corresponding to the user's login or username in the authentication system, such as username or email.
email	Email address.
name	Display name.

See the OpenID claims documentation for more information.



NOTE

Using an OpenID Connect identity provider requires users to get a token using **<master>/oauth/token/request** to use with command-line tools.

CHAPTER 4. USING RBAC TO DEFINE AND APPLY PERMISSIONS

4.1. RBAC OVERVIEW

Role-based access control (RBAC) objects determine whether a user is allowed to perform a given action within a project.

Developers can use local roles and bindings to control who has access to their projects. Note that authorization is a separate step from authentication, which is more about determining the identity of who is taking the action.

Authorization is managed using:

Rules	Sets of permitted verbs on a set of objects. For example, whether a user or service account can **create** pods.
Roles	Collections of rules. You can associate, or bind, users and groups to multiple roles.
Bindings	Associations between users and/or groups with a role.

There are two levels of RBAC roles and bindings that control authorization:

Cluster RBAC	Roles and bindings that are applicable across all projects. *Cluster roles* exist cluster-wide, and *cluster role bindings* can reference only cluster roles.
Local RBAC	Roles and bindings that are scoped to a given project. While *local roles* exist only in a single project, local role bindings can reference **both** cluster and local roles.

A cluster role binding is a binding that exists at the cluster level. A role binding exists at the project level. The cluster role *view* must be bound to a user using a local role binding for that user to view the project. Create local roles only if a cluster role does not provide the set of permissions needed for a particular situation.

This two-level hierarchy allows reuse across multiple projects through the cluster roles while allowing customization inside of individual projects through local roles.

During evaluation, both the cluster role bindings and the local role bindings are used. For example:

1. Cluster-wide "allow" rules are checked.
2. Locally-bound "allow" rules are checked.
3. Deny by default.

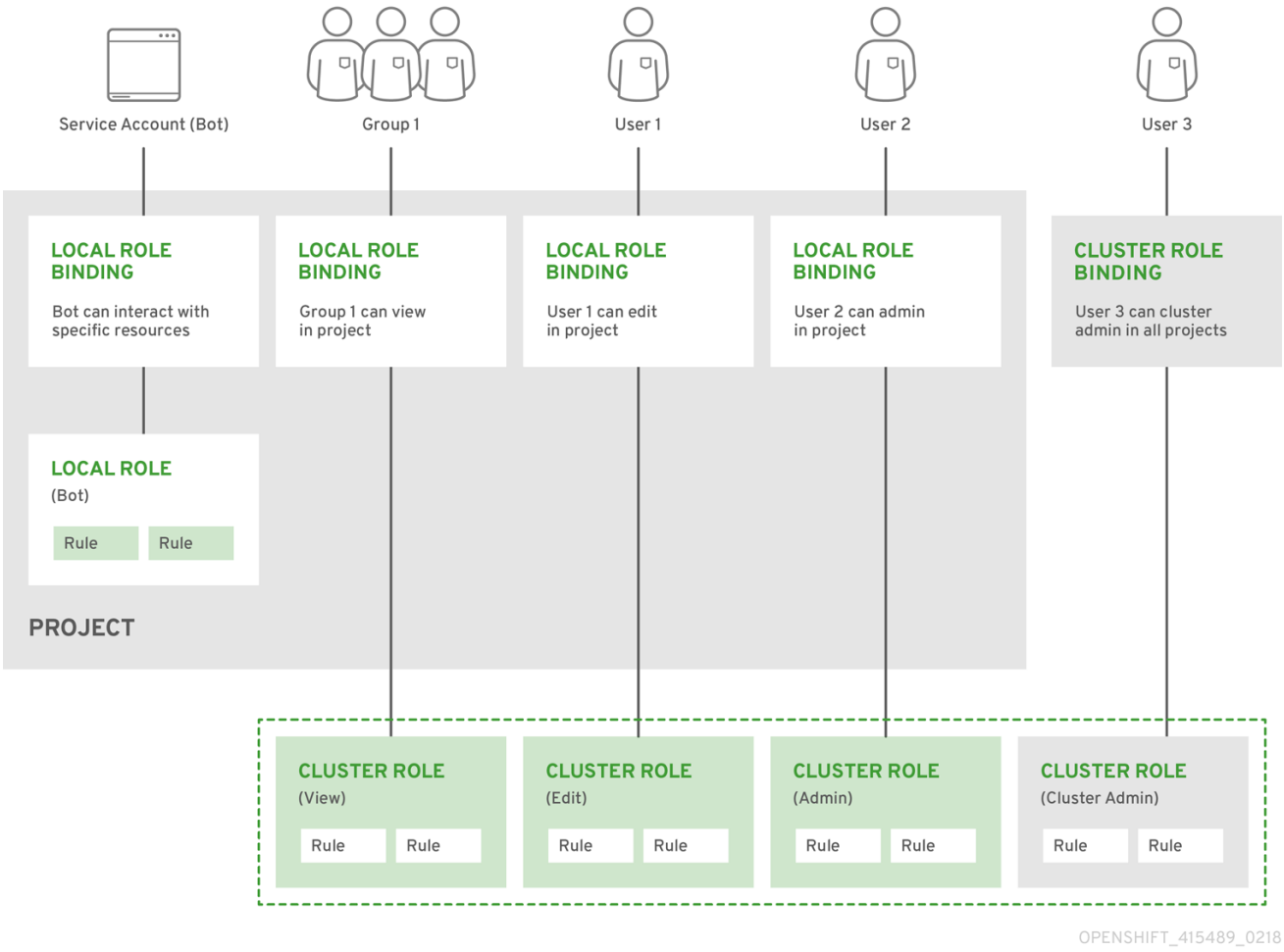
4.1.1. Default cluster roles

OpenShift Dedicated includes a set of default cluster roles that you can bind to users and groups cluster-wide or locally. You can manually modify the default cluster roles, if required, but you must take extra steps each time you restart a master node.

Default Cluster Role	Description
admin	A project manager. If used in a local binding, an **admin** has rights to view any resource in the project and modify any resource in the project except for quota.
basic-user	A user that can get basic information about projects and users.
cluster-admin	A super-user that can perform any action in any project. When bound to a user with a local binding, they have full control over quota and every action on every resource in the project.
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 the power to view or modify roles or bindings.
self-provisioner	A user that can create their own projects.
view	A user who cannot make any modifications, but can see most objects in a project. They cannot view or modify roles or bindings.

Be mindful of the difference between local and cluster bindings. For example, if you bind the **cluster-admin** role to a user by using a local role binding, it might appear that this user has the privileges of a cluster administrator. This is not the case. Binding the **cluster-admin** to a user in a project grants super administrator privileges for only that project to the user. That user has the permissions of the cluster role **admin**, plus a few additional permissions like the ability to edit rate limits, for that project. This binding can be confusing via the web console UI, which does not list cluster role bindings that are bound to true cluster administrators. However, it does list local role bindings that you can use to locally bind **cluster-admin**.

The relationships between cluster roles, local roles, cluster role bindings, local role bindings, users, groups and service accounts are illustrated below.



4.1.2. Evaluating authorization

OpenShift Dedicated evaluates authorization by using:

Identity

The user name and list of groups that the user belongs to.

Action

The action you perform. In most cases, this consists of:

- **Project**: The project you access. A project is a Kubernetes namespace with additional annotations that allows a community of users to organize and manage their content in isolation from other communities.
- **Verb** : The action itself: **get**, **list**, **create**, **update**, **delete**, **deletecollection**, or **watch**.
- **Resource Name**: The API endpoint that you access.

Bindings

The full list of bindings, the associations between users or groups with a role.

OpenShift Dedicated evaluates authorization by using the following steps:

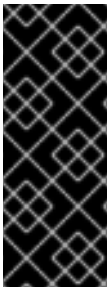
1. The identity and the project-scoped action is used to find all bindings that apply to the user or their groups.
2. Bindings are used to locate all the roles that apply.

3. Roles are used to find all the rules that apply.
4. The action is checked against each rule to find a match.
5. If no matching rule is found, the action is then denied by default.

TIP

Remember that users and groups can be associated with, or bound to, multiple roles at the same time.

Project administrators can use the CLI to view local bindings, including a matrix of the verbs and resources each are associated with.



IMPORTANT

The cluster role bound to the project administrator is limited in a project through a local binding. It is not bound cluster-wide like the cluster roles granted to the **cluster-admin** or **system:admin**.

Cluster roles are roles defined at the cluster level but can be bound either at the cluster level or at the project level.

4.2. PROJECTS AND NAMESPACES

A Kubernetes *namespace* provides a mechanism to scope resources in a cluster. The Kubernetes documentation has more information on namespaces.

Namespaces provide a unique scope for:

- Named resources to avoid basic naming collisions.
- Delegated management authority to trusted users.
- The ability to limit community resource consumption.

Most objects in the system are scoped by namespace, but some are excepted and have no namespace, including nodes and users.

A *project* is a Kubernetes namespace with additional annotations and is the central vehicle by which access to resources for regular users is managed. A project allows a community of users to organize and manage their content in isolation from other communities. Users must be given access to projects by administrators, or if allowed to create projects, automatically have access to their own projects.

Projects can have a separate **name**, **displayName**, and **description**.

- The mandatory **name** is a unique identifier for the project and is most visible when using the CLI tools or API. The maximum name length is 63 characters.
- The optional **displayName** is how the project is displayed in the web console (defaults to **name**).
- The optional **description** can be a more detailed description of the project and is also visible in the web console.

Each project scopes its own set of:

Objects	Pods, services, replication controllers, etc.
Policies	Rules for which users can or cannot perform actions on objects.
Constraints	Quotas for each kind of object that can be limited.
Service accounts	Service accounts act automatically with designated access to objects in the project.

Cluster administrators can create projects and delegate administrative rights for the project to any member of the user community. Cluster administrators can also allow developers to create their own projects.

Developers and administrators can interact with projects the CLI or the web console.

4.3. DEFAULT PROJECTS

OpenShift Dedicated comes with a number of default projects, and projects starting with **openshift-** are the most essential to users. These projects host master components that run as pods and other infrastructure components. The pods created in these namespaces that have a critical pod annotation are considered critical, and the have guaranteed admission by kubelet. Pods created for master components in these namespaces are already marked as critical.

4.4. VIEWING CLUSTER ROLES AND BINDINGS

You can use the **oc** CLI to view cluster roles and bindings by using the **oc describe** command.

Prerequisites

- Install the **oc** CLI.
- Obtain permission to view the cluster roles and bindings. Users with the **dedicated-admins-cluster** role can view cluster roles and bindings.

Procedure

1. To view the cluster roles and their associated rule sets:

```
$ oc describe clusterrole.rbac
```

1. To view the current set of cluster role bindings, which shows the users and groups that are bound to various roles:

```
$ oc describe clusterrolebinding.rbac
```

4.5. VIEWING LOCAL ROLES AND BINDINGS

You can use the **oc** CLI to view local roles and bindings by using the **oc describe** command.

Prerequisites

- Install the **oc** CLI.
- Obtain permission to view the local roles and bindings:
  - Users with the **dedicated-admins-cluster** role can view and manage local roles and bindings.
  - Users with the **admin** default cluster role bound locally can view and manage roles and bindings in that project.

Procedure

1. To view the current set of local role bindings, which show the users and groups that are bound to various roles for the current project:

```
$ oc describe rolebinding.rbac
```

2. To view the local role bindings for a different project, add the **-n** flag to the command:

```
$ oc describe rolebinding.rbac -n joe-project
Name:         admin
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name        Namespace
  ----  ----        ---------
  User  kube:admin


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind           Name      Namespace
  ----           ----      ---------
  ServiceAccount  deployer  joe-project


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
```

```
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  builder  joe-project


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                           Namespace
  ----   ----                           ---------
  Group  system:serviceaccounts:joe-project
```

4.6. ADDING ROLES TO USERS

You can use the **oc adm** administrator CLI to manage the roles and bindings.

Dedicated administrators cannot manage cluster roles. They can manage cluster role bindings and local roles and bindings.

Binding, or adding, a role to users or groups gives the user or group the access that is granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

You can bind any of the default cluster roles to local users or groups in your project.

Procedure

1. Add a role to a user in a specific project:

   ```
   $ oc adm policy add-role-to-user <role> <user> -n <project>
   ```

   For example, you can add the **admin** role to the **alice** user in **joe** project by running:

   ```
   $ oc adm policy add-role-to-user admin alice -n joe
   ```

2. View the local role bindings and verify the addition in the output:

   ```
   $ oc describe rolebinding.rbac -n <project>
   ```

   For example, to view the local role bindings for the **joe** project:

   ```
   $ oc describe rolebinding.rbac -n joe
   Name:         admin
   Labels:       <none>
   Annotations:  <none>
   Role:
     Kind:  ClusterRole
   ```

```
  Name:  admin
Subjects:
  Kind  Name       Namespace
  ----  ----       ---------
  User  kube:admin


Name:         admin-0
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice (1)


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind           Name      Namespace
  ----           ----      ---------
  ServiceAccount  deployer  joe


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  builder  joe


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
```

```
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:serviceaccounts:joe
```

1 The **alice** user has been added to the **admins RoleBinding**.

4.7. LOCAL ROLE BINDING COMMANDS

When you manage a user or group's associated roles for local role bindings using the following operations, a project may be specified with the **-n** flag. If it is not specified, then the current project is used.

You can use the following commands for local RBAC management.

Table 4.1. Local role binding operations

Command	Description
$ oc adm policy who-can *<verb> <resource>*	Indicates which users can perform an action on a resource.
$ oc adm policy add-role-to-user *<role> <username>*	Binds a specified role to specified users in the current project.
$ oc adm policy remove-role-from-user *<role> <username>*	Removes a given role from specified users in the current project.
$ oc adm policy remove-user *<username>*	Removes specified users and all of their roles in the current project.
$ oc adm policy add-role-to-group *<role> <groupname>*	Binds a given role to specified groups in the current project.
$ oc adm policy remove-role-from-group *<role> <groupname>*	Removes a given role from specified groups in the current project.
$ oc adm policy remove-group *<groupname>*	Removes specified groups and all of their roles in the current project.

CHAPTER 5. UNDERSTANDING AND CREATING SERVICE ACCOUNTS

5.1. SERVICE ACCOUNTS OVERVIEW

A service account is an OpenShift Dedicated account that allows a component to directly access the API. Service accounts are API objects that exist within each project. Service accounts provide a flexible way to control API access without sharing a regular user's credentials.

When you use the OpenShift Dedicated CLI or web console, your API token authenticates you to the API. You can associate a component with a service account so that they can access the API without using a regular user's credentials. For example, service accounts can allow:

- Replication controllers to make API calls to create or delete pods.
- Applications inside containers to make API calls for discovery purposes.
- External applications to make API calls for monitoring or integration purposes.

Each service account's user name is derived from its project and name:

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

Every service account is also a member of two groups:

system:serviceaccounts
Includes all service accounts in the system.

system:serviceaccounts:<project>
Includes all service accounts in the specified project.

Each service account automatically contains two secrets:

- An API token
- Credentials for the OpenShift Container Registry

The generated API token and registry credentials do not expire, but you can revoke them by deleting the secret. When you delete the secret, a new one is automatically generated to take its place.

5.2. SERVICE ACCOUNTS IN OPENSHIFT DEDICATED

As an administrator, you can use service accounts to perform any actions that require OpenShift Dedicated **admin** roles.

The **dedicated-admin** service creates the **dedicated-admins** group. This group is granted the roles at the cluster or individual project level. Users can be assigned to this group, and group membership defines who has OpenShift Dedicated administrator access. However, by design, service accounts cannot be added to regular groups.

Instead, the **dedicated-admin** service creates a special project for this purpose named **dedicated-admin**. The service account group for this project is granted OpenShift Dedicated **admin** roles, which grants OpenShift Dedicated administrator access to all service accounts within the **dedicated-admin**

project. You can then use these service accounts to perform any actions that require OpenShift Dedicated administrator access.

Users that are members of the **dedicated-admins** group are granted the **dedicated-admin** role permissions and have **edit** access to the **dedicated-admin** project. These users can manage the service accounts in this project and create new ones as needed.

Users with a **dedicated-reader** role are granted edit and view access to the **dedicated-reader** project and view-only access to the other projects.

5.3. ABOUT THE DEDICATED ADMINISTRATOR ROLE

The accounts for dedicated administrators of an OpenShift Dedicated cluster, have increased permissions and access to all user-created projects.

When your account has the **dedicated-admins-cluster** authorization role bound to it, you are automatically bound to the **dedicated-admins-project** role for any new projects that users in the cluster create.

You can perform actions that are associated with a set of verbs, such as **create**, to operate on a set of resource names, such as **templates**. To view the details of these roles and their sets of verbs and resources, run the following commands:

```
$ oc describe clusterrole/dedicated-admins-cluster
$ oc describe clusterrole/dedicated-admins-project
```

The verb names do not necessarily all map directly to **oc** commands but rather equate more generally to the types of CLI operations you can perform. For example, having the **list** verb means that you can display a list of all objects of a given resource name by running the **oc get** command, but the **get** verb means that you can display the details of a specific object if you know its name by runnign the **oc describe** command.

OpenShift Dedicated administrators can grant users a **dedicated-reader** role, which provides view-only access at the cluster level and view access for all user projects.

5.4. CREATING SERVICE ACCOUNTS

You can create a service account in a project and grant it permissions by binding it to a role.

Procedure

1. Optional: To view the service accounts in the current project:

```
$ oc get sa

NAME      SECRETS   AGE
builder   2         2d
default   2         2d
deployer  2         2d
```

2. To create a new service account in the current project:

```
$ oc create sa <service_account_name> 1

serviceaccount "robot" created
```

1 To create a service account in a different project, specify **-n <project_name>**.

3. Optional: View the secrets for the service account:

```
$ oc describe sa robot
Name:  robot
Namespace: project1
Labels:  <none>
Annotations: <none>

Image pull secrets: robot-dockercfg-qzbhb

Mountable secrets:  robot-token-f4khf
                    robot-dockercfg-qzbhb

Tokens:             robot-token-f4khf
                    robot-token-z8h44
```

5.5. EXAMPLES OF GRANTING ROLES TO SERVICE ACCOUNTS

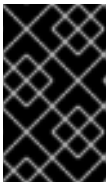
You can grant roles to service accounts in the same way that you grant roles to a regular user account.

- You can modify the service accounts for the current project. For example, to add the **view** role to the **robot** service account in the **top-secret** project:

```
$ oc policy add-role-to-user view system:serviceaccount:top-secret:robot
```

- You can also grant access to a specific service account in a project. For example, from the project to which the service account belongs, use the **-z** flag and specify the **<serviceaccount_name>**

```
$ oc policy add-role-to-user <role_name> -z <serviceaccount_name>
```



IMPORTANT

If you want to grant access to a specific service account in a project, use the **-z** flag. Using this flag helps prevent typos and ensures that access is granted to only the specified service account.

- To modify a different namespace, you can use the **-n** option to indicate the project namespace it applies to, as shown in the following examples.
  - For example, to allow all service accounts in all projects to view resources in the **top-secret** project:

```
$ oc policy add-role-to-group view system:serviceaccounts -n top-secret
```

- To allow all service accounts in the **managers** project to edit resources in the **top-secret** project:

```
$ oc policy add-role-to-group edit system:serviceaccounts:managers -n top-secret
```

CHAPTER 6. USING SERVICE ACCOUNTS IN APPLICATIONS

6.1. SERVICE ACCOUNTS OVERVIEW

A service account is an OpenShift Dedicated account that allows a component to directly access the API. Service accounts are API objects that exist within each project. Service accounts provide a flexible way to control API access without sharing a regular user's credentials.

When you use the OpenShift Dedicated CLI or web console, your API token authenticates you to the API. You can associate a component with a service account so that they can access the API without using a regular user's credentials. For example, service accounts can allow:

- Replication controllers to make API calls to create or delete pods.
- Applications inside containers to make API calls for discovery purposes.
- External applications to make API calls for monitoring or integration purposes.

Each service account's user name is derived from its project and name:

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

Every service account is also a member of two groups:

system:serviceaccounts
Includes all service accounts in the system.

system:serviceaccounts:<project>
Includes all service accounts in the specified project.

Each service account automatically contains two secrets:

- An API token
- Credentials for the OpenShift Container Registry

The generated API token and registry credentials do not expire, but you can revoke them by deleting the secret. When you delete the secret, a new one is automatically generated to take its place.

6.2. DEFAULT SERVICE ACCOUNTS

Your OpenShift Dedicated cluster contains default service accounts for cluster management and generates more service accounts for each project.

6.2.1. Default cluster service accounts

Several infrastructure controllers run using service account credentials. The following service accounts are created in the OpenShift Dedicated infrastructure project (**openshift-infra**) at server start, and given the following roles cluster-wide:

Service Account	Description
replication-controller	Assigned the **system:replication-controller** role

Service Account	Description
deployment-controller	Assigned the **system:deployment-controller** role
build-controller	Assigned the **system:build-controller** role. Additionally, the **build-controller** service account is included in the privileged Security Context Constraint in order to create privileged build pods.

6.2.2. Default project service accounts and roles

Three service accounts are automatically created in each project:

Service Account	Usage
builder	Used by build pods. It is given the **system:image-builder** role, which allows pushing images to any imagestream in the project using the internal Docker registry.
deployer	Used by deployment pods and given the **system:deployer** role, which allows viewing and modifying replication controllers and pods in the project.
default	Used to run all other pods unless they specify a different service account.

All service accounts in a project are given the **system:image-puller** role, which allows pulling images from any imagestream in the project using the internal container image registry.

6.3. CREATING SERVICE ACCOUNTS

You can create a service account in a project and grant it permissions by binding it to a role.

Procedure

1. Optional: To view the service accounts in the current project:

```
$ oc get sa

NAME       SECRETS   AGE
builder    2         2d
default    2         2d
deployer   2         2d
```

2. To create a new service account in the current project:

```
$ oc create sa <service_account_name> 1

serviceaccount "robot" created
```

1 To create a service account in a different project, specify **-n <project_name>**.

3. Optional: View the secrets for the service account:

```
$ oc describe sa robot
Name:  robot
Namespace: project1
Labels:  <none>
Annotations: <none>

Image pull secrets: robot-dockercfg-qzbhb

Mountable secrets:  robot-token-f4khf
                    robot-dockercfg-qzbhb

Tokens:             robot-token-f4khf
                    robot-token-z8h44
```

6.4. USING A SERVICE ACCOUNT'S CREDENTIALS EXTERNALLY

You can distribute a service account's token to external applications that must authenticate to the API.

In order to pull an image, the authenticated user must have **get** rights on the requested **imagestreams/layers**. In order to push an image, the authenticated user must have **update** rights on the requested **imagestreams/layers**.

By default, all service accounts in a project have rights to pull any image in the same project, and the **builder** service account has rights to push any image in the same project.

Procedure

1. View the service account's API token:

```
$ oc describe secret <secret-name>
```

For example:

```
$ oc describe secret robot-token-uzkbh -n top-secret

Name:  robot-token-uzkbh
Labels:  <none>
Annotations: kubernetes.io/service-account.name=robot,kubernetes.io/service-
account.uid=49f19e2e-16c6-11e5-afdc-3c970e4b7ffe

Type: kubernetes.io/service-account-token

Data

token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

2. Log in using the token that you obtained:

```
$ oc login --token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...

Logged into "https://server:8443" as "system:serviceaccount:top-secret:robot" using the
token provided.
```

```
You don't have any projects. You can try to create a new project, by running

    $ oc new-project <projectname>
```

3. Confirm that you logged in as the service account:

```
$ oc whoami

system:serviceaccount:top-secret:robot
```

CHAPTER 7. USING A SERVICE ACCOUNT AS AN OAUTH CLIENT

7.1. SERVICE ACCOUNTS AS OAUTH CLIENTS

You can use a service account as a constrained form of OAuth client. Service accounts can request only a subset of scopes that allow access to some basic user information and role-based power inside of the service account's own namespace:

- **user:info**
- **user:check-access**
- **role:<any_role>:<serviceaccount_namespace>**
- **role:<any_role>:<serviceaccount_namespace>:!**

When using a service account as an OAuth client:

- **client_id** is **system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>**.
- **client_secret** can be any of the API tokens for that service account. For example:

```
$ oc sa get-token <serviceaccount_name>
```

- To get **WWW-Authenticate** challenges, set an **serviceaccounts.openshift.io/oauth-want-challenges** annotation on the service account to **true**.
- **redirect_uri** must match an annotation on the service account.

7.1.1. Redirect URIs for Service Accounts as OAuth Clients

Annotation keys must have the prefix **serviceaccounts.openshift.io/oauth-redirecturi.** or **serviceaccounts.openshift.io/oauth-redirectreference.** such as:

```
serviceaccounts.openshift.io/oauth-redirecturi.<name>
```

In its simplest form, the annotation can be used to directly specify valid redirect URIs. For example:

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "https://example.com"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

The **first** and **second** postfixes in the above example are used to separate the two valid redirect URIs.

In more complex configurations, static redirect URIs may not be enough. For example, perhaps you want all Ingresses for a route to be considered valid. This is where dynamic redirect URIs via the **serviceaccounts.openshift.io/oauth-redirectreference.** prefix come into play.

For example:

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

Since the value for this annotation contains serialized JSON data, it is easier to see in an expanded format:

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": "Route",
    "name": "jenkins"
  }
}
```

Now you can see that an **OAuthRedirectReference** allows us to reference the route named **jenkins**. Thus, all Ingresses for that route will now be considered valid. The full specification for an **OAuthRedirectReference** is:

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": ..., 1
    "name": ..., 2
    "group": ... 3
  }
}
```

1. **kind** refers to the type of the object being referenced. Currently, only **route** is supported.
2. **name** refers to the name of the object. The object must be in the same namespace as the service account.
3. **group** refers to the group of the object. Leave this blank, as the group for a route is the empty string.

Both annotation prefixes can be combined to override the data provided by the reference object. For example:

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

The **first** postfix is used to tie the annotations together. Assuming that the **jenkins** route had an Ingress of ***https://example.com***, now ***https://example.com/custompath*** is considered valid, but ***https://example.com*** is not. The format for partially supplying override data is as follows:

Type	Syntax
Scheme	"https://"
Hostname	"//website.com"
Port	"//:8000"
Path	"examplepath"

NOTE

Specifying a host name override will replace the host name data from the referenced object, which is not likely to be desired behavior.

Any combination of the above syntax can be combined using the following format:

<scheme:>//<hostname><:port>/<path>

The same object can be referenced more than once for more flexibility:

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second":  "//:8000"
"serviceaccounts.openshift.io/oauth-redirectreference.second": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

Assuming that the route named **jenkins** has an Ingress of ***https://example.com***, then both ***https://example.com:8000*** and ***https://example.com/custompath*** are considered valid.

Static and dynamic annotations can be used at the same time to achieve the desired behavior:

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

CHAPTER 8. SCOPING TOKENS

8.1. ABOUT SCOPING TOKENS

You can created scoped tokens to delegate some of your permissions to another user or service account. For example, a project administrator might want to delegate the power to create pods.

A scoped token is a token that identifies as a given user but is limited to certain actions by its scope. Only a user with the **cluster-admin** role can create scoped tokens.

Scopes are evaluated by converting the set of scopes for a token into a set of **PolicyRules**. Then, the request is matched against those rules. The request attributes must match at least one of the scope rules to be passed to the "normal" authorizer for further authorization checks.

8.1.1. User scopes

User scopes are focused on getting information about a given user. They are intent-based, so the rules are automatically created for you:

- **user:full** - Allows full read/write access to the API with all of the user's permissions.
- **user:info** - Allows read-only access to information about the user, such as name and groups.
- **user:check-access** - Allows access to **self-localsubjectaccessreviews** and **self-subjectaccessreviews**. These are the variables where you pass an empty user and groups in your request object.
- **user:list-projects** - Allows read-only access to list the projects the user has access to.

8.1.2. Role scope

The role scope allows you to have the same level of access as a given role filtered by namespace.

- **role:<cluster-role name>:<namespace or * for all>** - Limits the scope to the rules specified by the cluster-role, but only in the specified namespace .

NOTE

Caveat: This prevents escalating access. Even if the role allows access to resources like secrets, rolebindings, and roles, this scope will deny access to those resources. This helps prevent unexpected escalations. Many people do not think of a role like **edit** as being an escalating role, but with access to a secret it is.

- **role:<cluster-role name>:<namespace or * for all>:!** - This is similar to the example above, except that including the bang causes this scope to allow escalating access.