Configuring user authentication and access controls for users and services
Abstract

This document provides instructions for defining identity providers in OpenShift Container Platform. It also discusses how to configure role-based access control to secure the cluster.
Table of Contents

CHAPTER 1. UNDERSTANDING AUTHENTICATION .......................................................... 7
1.1. USERS ............................................................................................................. 7
1.2. GROUPS ........................................................................................................ 7
1.3. API AUTHENTICATION
   1.3.1. OpenShift Container Platform OAuth server ............................................. 8
       1.3.1.1. OAuth token requests ........................................................................ 8
       1.3.1.2. API impersonation ........................................................................... 9
       1.3.1.3. Authentication metrics for Prometheus ............................................. 9

CHAPTER 2. CONFIGURING THE INTERNAL OAUTH SERVER ...................................... 11
2.1. OPENSHIFT CONTAINER PLATFORM OAUTH SERVER ..................................... 11
2.2. OAUTH TOKEN REQUEST FLOWS AND RESPONSES ...................................... 11
2.3. OPTIONS FOR THE INTERNAL OAUTH SERVER
   2.3.1. OAuth token duration options .................................................................. 12
   2.3.2. OAuth grant options .............................................................................. 12
2.4. CONFIGURING THE INTERNAL OAUTH SERVER’S TOKEN DURATION ............... 12
2.5. CONFIGURING TOKEN INACTIVITY TIMEOUT FOR THE INTERNAL OAUTH SERVER 13
2.6. OAUTH SERVER METADATA .......................................................................... 15
2.7. TROUBLESHOOTING OAUTH API EVENTS ....................................................... 16

CHAPTER 3. CONFIGURING OAUTH CLIENTS ............................................................... 18
3.1. DEFAULT OAUTH CLIENTS .............................................................................. 18
3.2. REGISTERING AN ADDITIONAL OAUTH CLIENT ............................................ 18
3.3. CONFIGURING TOKEN INACTIVITY TIMEOUT FOR AN OAUTH CLIENT .......... 19
3.4. ADDITIONAL RESOURCES ............................................................................ 20

CHAPTER 4. MANAGING USER-OWNED OAUTH ACCESS TOKENS ................................. 21
4.1. LISTING USER-OWNED OAUTH ACCESS TOKENS .......................................... 21
4.2. VIEWING THE DETAILS OF A USER-OWNED OAUTH ACCESS TOKEN ........... 21
4.3. DELETING USER-OWNED OAUTH ACCESS TOKENS ...................................... 22

CHAPTER 5. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION ........................ 24
5.1. ABOUT IDENTITY PROVIDERS IN OPENSHIFT CONTAINER PLATFORM ............ 24
5.2. SUPPORTED IDENTITY PROVIDERS ............................................................... 24
5.3. REMOVING THE KUBEADMIN USER .............................................................. 25
5.4. IDENTITY PROVIDER PARAMETERS .............................................................. 25
5.5. SAMPLE IDENTITY PROVIDER CR ................................................................. 26

CHAPTER 6. CONFIGURING IDENTITY PROVIDERS ..................................................... 28
6.1. CONFIGURING AN HTPASSWD IDENTITY PROVIDER .................................... 28
   6.1.1. About identity providers in OpenShift Container Platform ....................... 28
   6.1.2. Creating an HTPasswd file using Linux .................................................... 28
   6.1.3. Creating an HTPasswd file using Windows .............................................. 29
   6.1.4. Creating the HTPasswd secret ................................................................ 29
   6.1.5. Sample HTPasswd CR .......................................................................... 30
   6.1.6. Adding an identity provider to your clusters ............................................ 30
   6.1.7. Updating users for an HTPasswd identity provider .................................. 31
   6.1.8. Configuring identity providers using the web console .............................. 33
6.2. CONFIGURING A KEYSTONE IDENTITY PROVIDER ........................................ 33
   6.2.1. About identity providers in OpenShift Container Platform ....................... 33
   6.2.2. Creating the secret .................................................................................. 33
   6.2.3. Creating a config map .......................................................................... 34
6.2.4. Sample Keystone CR
6.2.5. Adding an identity provider to your clusters
6.3. CONFIGURING AN LDAP IDENTITY PROVIDER
  6.3.1. About identity providers in OpenShift Container Platform
  6.3.2. About LDAP authentication
  6.3.3. Creating the LDAP secret
  6.3.4. Creating a config map
  6.3.5. Sample LDAP CR
  6.3.6. Adding an identity provider to your clusters
6.4. CONFIGURING A BASIC AUTHENTICATION IDENTITY PROVIDER
  6.4.1. About identity providers in OpenShift Container Platform
  6.4.2. About basic authentication
  6.4.3. Creating the secret
  6.4.4. Creating a config map
  6.4.5. Sample basic authentication CR
  6.4.6. Adding an identity provider to your clusters
  6.4.7. Example Apache HTTPD configuration for basic identity providers
    6.4.7.1. File requirements
  6.4.8. Basic authentication troubleshooting
6.5. CONFIGURING A REQUEST HEADER IDENTITY PROVIDER
  6.5.1. About identity providers in OpenShift Container Platform
  6.5.2. About request header authentication
    6.5.2.1. SSPI connection support on Microsoft Windows
  6.5.3. Creating a config map
  6.5.4. Sample request header CR
  6.5.5. Adding an identity provider to your clusters
  6.5.6. Example Apache authentication configuration using request header
    Custom proxy configuration
    Configuring Apache authentication using request header
6.6. CONFIGURING A GITHUB OR GITHUB ENTERPRISE IDENTITY PROVIDER
  6.6.1. About identity providers in OpenShift Container Platform
  6.6.2. Registering a GitHub application
  6.6.3. Creating the secret
  6.6.4. Creating a config map
  6.6.5. Sample GitHub CR
  6.6.6. Adding an identity provider to your clusters
6.7. CONFIGURING A GITLAB IDENTITY PROVIDER
  6.7.1. About identity providers in OpenShift Container Platform
  6.7.2. Creating the secret
  6.7.3. Creating a config map
  6.7.4. Sample GitLab CR
  6.7.5. Adding an identity provider to your clusters
6.8. CONFIGURING A GOOGLE IDENTITY PROVIDER
  6.8.1. About identity providers in OpenShift Container Platform
  6.8.2. Creating the secret
  6.8.3. Sample Google CR
  6.8.4. Adding an identity provider to your clusters
6.9. CONFIGURING A OPENID CONNECT IDENTITY PROVIDER
  6.9.1. About identity providers in OpenShift Container Platform
  6.9.2. Creating the secret
  6.9.3. Creating a config map
  6.9.4. Sample OpenID Connect CRs
  6.9.5. Adding an identity provider to your clusters
CHAPTER 7. USING RBAC TO DEFINE AND APPLY PERMISSIONS ............................................. 73
  7.1. RBAC OVERVIEW ........................................................................................................... 73
    7.1.1. Default cluster roles ............................................................................................... 74
    7.1.2. Evaluating authorization ......................................................................................... 75
      7.1.2.1. Cluster role aggregation ..................................................................................... 76
    7.2. PROJECTS AND NAMESPACES .................................................................................. 76
    7.3. DEFAULT PROJECTS .................................................................................................. 77
    7.4. VIEWING CLUSTER ROLES AND BINDINGS ............................................................... 77
    7.5. VIEWING LOCAL ROLES AND BINDINGS ................................................................. 84
    7.6. ADDING ROLES TO USERS ......................................................................................... 86
    7.7. CREATING A LOCAL ROLE ......................................................................................... 88
    7.8. CREATING A CLUSTER ROLE ..................................................................................... 88
    7.9. LOCAL ROLE BINDING COMMANDS ......................................................................... 89
    7.10. CLUSTER ROLE BINDING COMMANDS .................................................................. 89
    7.11. CREATING A CLUSTER ADMIN ............................................................................... 90

CHAPTER 8. REMOVING THE KUBEADMIN USER ................................................................. 91
  8.1. THE KUBEADMIN USER ................................................................................................ 91
  8.2. REMOVING THE KUBEADMIN USER .......................................................................... 91

CHAPTER 9. UNDERSTANDING AND CREATING SERVICE ACCOUNTS .............................. 92
  9.1. SERVICE ACCOUNTS OVERVIEW ................................................................................. 92
  9.2. CREATING SERVICE ACCOUNTS ................................................................................ 92
  9.3. EXAMPLES OF GRANTING ROLES TO SERVICE ACCOUNTS ..................................... 93

CHAPTER 10. USING SERVICE ACCOUNTS IN APPLICATIONS ............................................ 96
  10.1. SERVICE ACCOUNTS OVERVIEW .............................................................................. 96
  10.2. DEFAULT SERVICE ACCOUNTS ............................................................................... 96
    10.2.1. Default cluster service accounts ........................................................................... 96
    10.2.2. Default project service accounts and roles ......................................................... 97
  10.3. CREATING SERVICE ACCOUNTS .............................................................................. 97
  10.4. USING A SERVICE ACCOUNT’S CREDENTIALS EXTERNALLY .................................. 98

CHAPTER 11. USING A SERVICE ACCOUNT AS AN OAUTH CLIENT .................................... 100
  11.1. SERVICE ACCOUNTS AS OAUTH CLIENTS .............................................................. 100
    11.1.1. Redirect URLs for service accounts as OAuth clients ......................................... 100

CHAPTER 12. SCOPING TOKENS ........................................................................................... 103
  12.1. ABOUT SCOPING TOKENS ........................................................................................ 103
    12.1.1. User scopes ........................................................................................................... 103
    12.1.2. Role scope ............................................................................................................. 103

CHAPTER 13. USING BOUND SERVICE ACCOUNT TOKENS ............................................... 104
  13.1. ABOUT BOUND SERVICE ACCOUNT TOKENS ......................................................... 104
  13.2. CONFIGURING BOUND SERVICE ACCOUNT TOKENS USING VOLUME PROJECTION .. 104

CHAPTER 14. MANAGING SECURITY CONTEXT CONSTRAINTS ........................................... 107
  14.1. ABOUT SECURITY CONTEXT CONSTRAINTS .......................................................... 107
    14.1.1. Default security context constraints ...................................................................... 107
    14.1.2. Security context constraints settings ..................................................................... 111
    14.1.3. Security context constraints strategies ............................................................... 112
    14.1.4. Controlling volumes ............................................................................................. 112
    14.1.5. Admission control ............................................................................................... 114
CHAPTER 18. MANAGING CLOUD PROVIDER CREDENTIALS ........................................ 164
18.1. ABOUT THE CLOUD CREDENTIAL OPERATOR ....................................... 164
  18.1.1. Modes .................................................................................................. 164
  18.1.2. Default behavior .................................................................................. 165
  18.1.3. Additional resources .......................................................................... 165
18.2. USING MINT MODE .................................................................................. 165
  18.2.1. Mint mode permissions requirements .................................................. 166
    18.2.1.1. Amazon Web Services (AWS) permissions ........................................ 166
    18.2.1.2. Microsoft Azure permissions .......................................................... 166
    18.2.1.3. Google Cloud Platform (GCP) permissions ......................................... 166
  18.2.2. Mint mode with removal or rotation of the administrator-level credential 167
    18.2.2.1. Rotating cloud provider credentials manually .................................... 167
    18.2.2.2. Removing cloud provider credentials ............................................... 169
  18.2.3. Additional resources .......................................................................... 170
18.3. USING PASSTHROUGH MODE ................................................................. 170
  18.3.1. Passthrough mode permissions requirements ....................................... 170
    18.3.1.1. Amazon Web Services (AWS) permissions ........................................ 170
    18.3.1.2. Microsoft Azure permissions .......................................................... 170
    18.3.1.3. Google Cloud Platform (GCP) permissions ......................................... 171
    18.3.1.4. Red Hat OpenStack Platform (RHOSP) permissions ......................... 171
    18.3.1.5. Red Hat Virtualization (RHV) permissions ......................................... 171
    18.3.1.6. VMware vSphere permissions .......................................................... 171
  18.3.2. Passthrough mode credential maintenance .......................................... 172
  18.3.3. Reducing permissions after installation .............................................. 172
  18.3.4. Additional resources .......................................................................... 172
18.4. USING MANUAL MODE ............................................................................ 172
  18.4.1. Manual mode with AWS STS ............................................................... 173
  18.4.2. Upgrading clusters with manually maintained credentials .................... 173
  18.4.3. Additional resources .......................................................................... 174
18.5. USING MANUAL MODE WITH STS ......................................................... 174
  18.5.1. Installing an OpenShift Container Platform cluster configured for manual mode with STS 175
    18.5.1.1. Configuring the Cloud Credential Operator utility .............................. 176
    18.5.1.2. Creating AWS resources with the Cloud Credential Operator utility 177
      18.5.1.2.1. Creating AWS resources individually .......................................... 177
      18.5.1.2.2. Creating AWS resources with a single command ......................... 179
    18.5.1.3. Running the installer ...................................................................... 180
    18.5.1.4. Verifying the installation ................................................................ 181
For users to interact with OpenShift Container Platform, they must first authenticate to the cluster. The authentication layer identifies the user associated with requests to the OpenShift Container Platform API. The authorization layer then uses information about the requesting user to determine if the request is allowed.

As an administrator, you can configure authentication for OpenShift Container Platform.

1.1. USERS

A user in OpenShift Container Platform is an entity that can make requests to the OpenShift Container Platform API. An OpenShift Container Platform 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 Container Platform.

Several types of users can exist:

<table>
<thead>
<tr>
<th>User type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular users</td>
<td>This is the way most interactive OpenShift Container Platform 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</td>
</tr>
<tr>
<td>System users</td>
<td>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</td>
</tr>
<tr>
<td>Service accounts</td>
<td>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</td>
</tr>
</tbody>
</table>

Each user must authenticate in some way to access OpenShift Container Platform. API requests with no authentication or invalid authentication are authenticated as requests by the anonymous system user. After 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 Container Platform API are authenticated using the following methods:

**OAuth access tokens**
- Obtained from the OpenShift Container Platform 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 Container Platform OAuth server

The OpenShift Container Platform 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 Container Platform API:

<table>
<thead>
<tr>
<th>OAuth client</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>openshift-browser-client</td>
<td>Requests tokens at &lt;namespace_route&gt;/oauth/token/request with a user-agent that can handle interactive logins. [1]</td>
</tr>
<tr>
<td>openshift-challenging-client</td>
<td>Requests tokens with a user-agent that can handle WWW-Authenticate challenges.</td>
</tr>
</tbody>
</table>

1. <namespace_route> refers to the namespace 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 Container Platform 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 Container Platform 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 Container Platform 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.

### 1.3.1.2. API impersonation

You can configure a request to the OpenShift Container Platform API to act as though it originated from another user. For more information, see User impersonation in the Kubernetes documentation.

### 1.3.1.3. Authentication metrics for Prometheus

OpenShift Container Platform captures the following Prometheus system metrics during authentication attempts:

- `openshift_auth_basic_password_count` counts the number of `oc login` user name and password attempts.
- `openshift_auth_basic_password_count_result` counts the number of `oc login` user name and password attempts by result, `success` or `error`.

- `openshift_auth_form_password_count` counts the number of web console login attempts.

- `openshift_auth_form_password_count_result` counts the number of web console login attempts by result, `success` or `error`.

- `openshift_auth_password_total` counts the total number of `oc login` and web console login attempts.
CHAPTER 2. CONFIGURING THE INTERNAL OAUTH SERVER

2.1. OPENSHIFT CONTAINER PLATFORM OAUTH SERVER

The OpenShift Container Platform 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.

2.2. OAUTH TOKEN REQUEST FLOWS AND RESPONSES

The OAuth server supports standard authorization code grant and the implicit grant OAuth authorization flows.

When requesting an OAuth token using the implicit grant flow (response_type=token) with a client_id configured to request WWW-Authenticate challenges (like openshift-challenging-client), these are the possible server responses from /oauth/authorize, and how they should be handled:

<table>
<thead>
<tr>
<th>Status</th>
<th>Content</th>
<th>Client response</th>
</tr>
</thead>
<tbody>
<tr>
<td>302</td>
<td>Location header containing an access_token parameter in the URL fragment (RFC 6749 section 4.2.2)</td>
<td>Use the access_token value as the OAuth token.</td>
</tr>
<tr>
<td>302</td>
<td>Location header containing an error query parameter (RFC 6749 section 4.1.2.1)</td>
<td>Fail, optionally surfacing the error (and optional error_description) query values to the user.</td>
</tr>
<tr>
<td>302</td>
<td>Other Location header</td>
<td>Follow the redirect, and process the result using these rules.</td>
</tr>
<tr>
<td>401</td>
<td>WWW-Authenticate header present</td>
<td>Respond to challenge if type is recognized (e.g. Basic, Negotiate, etc), resubmit request, and process the result using these rules.</td>
</tr>
<tr>
<td>401</td>
<td>WWW-Authenticate header missing</td>
<td>No challenge authentication is possible. Fail and show response body (which might contain links or details on alternate methods to obtain an OAuth token).</td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
<td>Fail, optionally surfacing response body to the user.</td>
</tr>
</tbody>
</table>

2.3. OPTIONS FOR THE INTERNAL OAUTH SERVER

Several configuration options are available for the internal OAuth server.
2.3.1. OAuth token duration options

The internal OAuth server generates two kinds of tokens:

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access tokens</td>
<td>Longer-lived tokens that grant access to the API.</td>
</tr>
<tr>
<td>Authorize codes</td>
<td>Short-lived tokens whose only use is to be exchanged for an access token.</td>
</tr>
</tbody>
</table>

You can configure the default duration for both types of token. If necessary, you can override the duration of the access token by using an `OAuthClient` object definition.

2.3.2. OAuth grant options

When the OAuth server receives token requests for a client to which the user has not previously granted permission, the action that the OAuth server takes is dependent on the OAuth client’s grant strategy.

The OAuth client requesting token must provide its own grant strategy.

You can apply the following default methods:

<table>
<thead>
<tr>
<th>Grant option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Auto-approve the grant and retry the request.</td>
</tr>
<tr>
<td>prompt</td>
<td>Prompt the user to approve or deny the grant.</td>
</tr>
</tbody>
</table>

2.4. CONFIGURING THE INTERNAL OAUTH SERVER’S TOKEN DURATION

You can configure default options for the internal OAuth server’s token duration.

**IMPORTANT**

By default, tokens are only valid for 24 hours. Existing sessions expire after this time elapses.

If the default time is insufficient, then this can be modified using the following procedure.

**Procedure**

1. Create a configuration file that contains the token duration options. The following file sets this to 48 hours, twice the default.

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
```
Set `accessTokenMaxAgeSeconds` to control the lifetime of access tokens. The default lifetime is 24 hours, or 86400 seconds. This attribute cannot be negative. If set to zero, the default lifetime is used.

2. Apply the new configuration file:

   ```
   NOTE
   Because you update the existing OAuth server, you must use the `oc apply` command to apply the change.
   ```

   ```
   $ oc apply -f </path/to/file.yaml>
   ```

3. Confirm that the changes are in effect:

   ```
   $ oc describe oauth.config.openshift.io/cluster
   ```

   **Example output**

   ```
   ... Spec:
   Token Config:
   Access Token Max Age Seconds: 172800 ...
   ```

2.5. Configuring Token Inactivity Timeout for the Internal OAuth Server

You can configure OAuth tokens to expire after a set period of inactivity. By default, no token inactivity timeout is set.

**NOTE**

If the token inactivity timeout is also configured in your OAuth client, that value overrides the timeout that is set in the internal OAuth server configuration.

**Prerequisites**

- You have access to the cluster as a user with the `cluster-admin` role.
- You have configured an identity provider (IDP).

**Procedure**

1. Update the OAuth configuration to set a token inactivity timeout.
   a. Edit the OAuth object:
Add the `spec.tokenConfig.accessTokenInactivityTimeout` field and set your timeout value:

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  ...
spec:
  tokenConfig:
    accessTokenInactivityTimeout: 400s

1 Set a value with the appropriate units, for example `400s` for 400 seconds, or `30m` for 30 minutes. The minimum allowed timeout value is `300s`.

b. Save the file to apply the changes.

2. Check that the OAuth server pods have restarted:

   ```bash
   $ oc get clusteroperators authentication
   NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
   authentication   4.8.0     True        False         False      145m
   ```

   Example output

   Do not continue to the next step until `PROGRESSING` is listed as `False`, as shown in the following output:

   3. Check that a new revision of the Kubernetes API server pods has rolled out. This will take several minutes.

   ```bash
   $ oc get clusteroperators kube-apiserver
   NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
   kube-apiserver   4.8.0     True        False         False      145m
   ```

   Example output

   Do not continue to the next step until `PROGRESSING` is listed as `False`, as shown in the following output:

   If `PROGRESSING` is showing `True`, wait a few minutes and try again.

Verification

1. Log in to the cluster with an identity from your IDP.

2. Execute a command and verify that it was successful.

3. Wait longer than the configured timeout without using the identity. In this procedure’s example, wait longer than 400 seconds.
4. Try to execute a command from the same identity’s session. This command should fail because the token should have expired due to inactivity longer than the configured timeout.

Example output

```
error: You must be logged in to the server (Unauthorized)
```

### 2.6. OAUTH SERVER METADATA

Applications running in OpenShift Container Platform might have to discover information about the built-in OAuth server. For example, they might have to discover what the address of the `<namespace_route>` is without manual configuration. To aid in this, OpenShift Container Platform implements the IETF OAuth 2.0 Authorization Server Metadata draft specification.

Thus, any application running inside the cluster can issue a GET request to `https://openshift.default.svc/.well-known/oauth-authorization-server` to fetch the following information:

```json
{
  "issuer": "https://<namespace_route>“, 1
  "authorization_endpoint": "https://<namespace_route>/oauth/authorize", 2
  "token_endpoint": "https://<namespace_route>/oauth/token", 3
  "scopes_supported": [ 4
    "user:full",
    "user:info",
    "user:check-access",
    "user:list-scoped-projects",
    "user:list-projects"
  ],
  "response_types_supported": [ 5
    "code",
    "token"
  ],
  "grant_types_supported": [ 6
    "authorization_code",
    "implicit"
  ],
  "code_challenge_methods_supported": [ 7
    "plain",
    "S256"
  ]
}
```

1. The authorization server’s issuer identifier, which is a URL that uses the https scheme and has no query or fragment components. This is the location where .well-known RFC 5785 resources containing information about the authorization server are published.

2. URL of the authorization server’s authorization endpoint. See RFC 6749.

3. URL of the authorization server’s token endpoint. See RFC 6749.

4. JSON array containing a list of the OAuth 2.0 RFC 6749 scope values that this authorization server supports. Note that not all supported scope values are advertised.
5. JSON array containing a list of the OAuth 2.0 `response_type` values that this authorization server supports. The array values used are the same as those used with the `response_types` parameter.

6. JSON array containing a list of the OAuth 2.0 grant type values that this authorization server supports. The array values used are the same as those used with the `grant_types` parameter defined by OAuth 2.0 Dynamic Client Registration Protocol in RFC 7591.

7. JSON array containing a list of PKCE RFC 7636 code challenge methods supported by this authorization server. Code challenge method values are used in the `code_challenge_method` parameter defined in Section 4.3 of RFC 7636. The valid code challenge method values are those registered in the IANA PKCE Code Challenge Methods registry. See IANA OAuth Parameters.

### 2.7. TROUBLESHOOTING OAUTH API EVENTS

In some cases the API server returns an unexpected condition error message that is difficult to debug without direct access to the API master log. The underlying reason for the error is purposely obscured in order to avoid providing an unauthenticated user with information about the server’s state.

A subset of these errors is related to service account OAuth configuration issues. These issues are captured in events that can be viewed by non-administrator users. When encountering an unexpected condition server error during OAuth, run `oc get events` to view these events under **ServiceAccount**.

The following example warns of a service account that is missing a proper OAuth redirect URI:

```
$ oc get events | grep ServiceAccount
```

**Example output**

```
1m 1m 1 proxy ServiceAccount Warning
NoSAOAuthRedirectURIs service-account-oauth-client-getter
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

Running `oc describe sa/<service_account_name>` reports any OAuth events associated with the given service account name.

```
$ oc describe sa/proxy | grep -A5 Events
```

**Example output**

```
Events:
FirstSeen LastSeen Count From SubObjectName Type Reason
Message ----------- -------- ----- ----- -------------- ------- ------
3m 3m 1 service-account-oauth-client-getter Warning
NoSAOAuthRedirectURIs system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

The following is a list of the possible event errors:

**No redirect URI annotations or an invalid URI is specified**
CHAPTER 2. CONFIGURING THE INTERNAL OAUTH SERVER

<table>
<thead>
<tr>
<th>Reason</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSAOAuthRedirectURIs</td>
<td>system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.&lt;some-value&gt;=&lt;redirect&gt; or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.&lt;some-value&gt;=&lt;reference&gt;</td>
</tr>
<tr>
<td>Invalid route specified</td>
<td></td>
</tr>
<tr>
<td>NoSAOAuthRedirectURIs</td>
<td>[routes.route.openshift.io &quot;&lt;name&gt;&quot; not found, system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.&lt;some-value&gt;=&lt;redirect&gt; or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.&lt;some-value&gt;=&lt;reference&gt;]</td>
</tr>
<tr>
<td>Invalid reference type specified</td>
<td></td>
</tr>
<tr>
<td>NoSAOAuthRedirectURIs</td>
<td>[no kind &quot;&lt;name&gt;&quot; is registered for version &quot;v1&quot;, system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.&lt;some-value&gt;=&lt;redirect&gt; or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.&lt;some-value&gt;=&lt;reference&gt;]</td>
</tr>
<tr>
<td>Missing SA tokens</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3. CONFIGURING OAUTH CLIENTS

Several OAuth clients are created by default in OpenShift Container Platform. You can also register and configure additional OAuth clients.

3.1. DEFAULT OAUTH CLIENTS

The following OAuth clients are automatically created when starting the OpenShift Container Platform API:

<table>
<thead>
<tr>
<th>OAuth client</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>openshift-browser-client</td>
<td>Requests tokens at &lt;namespace_route&gt;/oauth/token/request with a user-agent that can handle interactive logins. [1]</td>
</tr>
<tr>
<td>openshift-challenging-client</td>
<td>Requests tokens with a user-agent that can handle WWW-Authenticate challenges.</td>
</tr>
</tbody>
</table>

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

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

3.2. REGISTERING AN ADDITIONAL OAUTH CLIENT

If you need an additional OAuth client to manage authentication for your OpenShift Container Platform cluster, you can register one.

Procedure

- To register additional OAuth clients:

```
$ oc create -f "<echo \\
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: demo
secret: "...
redirectURIs:
  - "http://www.example.com/"
grantMethod: prompt
")"
```

1. The name of the OAuth client is used as the client_id parameter when making requests to <namespace_route>/oauth/authorize and <namespace_route>/oauth/token.

2. The secret is used as the client_secret parameter when making requests to <namespace_route>/oauth/token.
The `redirect_uri` parameter specified in requests to `<namespace_route>/oauth/authorize` and `<namespace_route>/oauth/token` must be equal to or prefixed by one of the URIs listed in the `redirectURIs` parameter value.

The `grantMethod` is used to determine what action to take when this client requests tokens and has not yet been granted access by the user. Specify `auto` to automatically approve the grant and retry the request, or `prompt` to prompt the user to approve or deny the grant.

### 3.3. CONFIGURING TOKEN INACTIVITY TIMEOUT FOR AN OAUTH CLIENT

You can configure OAuth clients to expire OAuth tokens after a set period of inactivity. By default, no token inactivity timeout is set.

**NOTE**

If the token inactivity timeout is also configured in the internal OAuth server configuration, the timeout that is set in the OAuth client overrides that value.

**Prerequisites**

- You have access to the cluster as a user with the `cluster-admin` role.
- You have configured an identity provider (IDP).

**Procedure**

- Update the `OAuthClient` configuration to set a token inactivity timeout.
  
  a. Edit the `OAuthClient` object:

     ```shell
     $ oc edit oauthclient <oauth_client>  
     
     Replace `<oauth_client>` with the OAuth client to configure, for example, `console`.
     
     Add the `accessTokenInactivityTimeoutSeconds` field and set your timeout value:
     
     ```yaml
     apiVersion: oauth.openshift.io/v1
     grantMethod: auto
     kind: OAuthClient
     metadata:
     ...
     accessTokenInactivityTimeoutSeconds: 600  
     ```
     
     The minimum allowed timeout value in seconds is 300.
     
     b. Save the file to apply the changes.

**Verification**
1. Log in to the cluster with an identity from your IDP. Be sure to use the OAuth client that you just configured.

2. Perform an action and verify that it was successful.

3. Wait longer than the configured timeout without using the identity. In this procedure’s example, wait longer than 600 seconds.

4. Try to perform an action from the same identity’s session. This attempt should fail because the token should have expired due to inactivity longer than the configured timeout.

3.4. ADDITIONAL RESOURCES

- OAuthClient [oauth.openshift.io/v1]
CHAPTER 4. MANAGING USER-OWNED OAUTH ACCESS TOKENS

Users can review their own OAuth access tokens and delete any that are no longer needed.

4.1. LISTING USER-OWNED OAUTH ACCESS TOKENS

You can list your user-owned OAuth access tokens. Token names are not sensitive and cannot be used to log in.

Procedure

- List all user-owned OAuth access tokens:

  $ oc get useroauthaccesstokens

  **Example output**

  NAME       CLIENT NAME                    CREATED                EXPIRES
  REDIRECT URI                                                       SCOPES

- List user-owned OAuth access tokens for a particular OAuth client:

  $ oc get useroauthaccesstokens --field-selector=clientName="console"

  **Example output**

  NAME       CLIENT NAME                    CREATED                EXPIRES
  REDIRECT URI                                                       SCOPES

4.2. VIEWING THE DETAILS OF A USER-OWNED OAUTH ACCESS TOKEN

You can view the details of a user-owned OAuth access token.

Procedure

- Describe the details of a user-owned OAuth access token:

  $ oc describe useroauthaccesstokens <token_name>

  **Example output**

  -
<table>
<thead>
<tr>
<th>Name:</th>
<th>&lt;token_name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace:</td>
<td></td>
</tr>
<tr>
<td>Labels:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Annotations:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>API Version:</td>
<td>oauth.openshift.io/v1</td>
</tr>
<tr>
<td>Authorize Token:</td>
<td>sha256~Ksckkug-9Fg_RWn_AUysPolg-_HqmFl9zUL_CgD8wr8</td>
</tr>
<tr>
<td>Client Name:</td>
<td>openshift-browser-client</td>
</tr>
<tr>
<td>Expires In:</td>
<td>86400</td>
</tr>
<tr>
<td>Inactivity Timeout Seconds:</td>
<td>317</td>
</tr>
<tr>
<td>Kind:</td>
<td>UserOAuthAccessToken</td>
</tr>
<tr>
<td>Metadata:</td>
<td></td>
</tr>
<tr>
<td>Creation Timestamp:</td>
<td>2021-01-11T19:27:06Z</td>
</tr>
<tr>
<td>Managed Fields:</td>
<td></td>
</tr>
<tr>
<td>API Version:</td>
<td>oauth.openshift.io/v1</td>
</tr>
<tr>
<td>Fields Type:</td>
<td>FieldsV1</td>
</tr>
<tr>
<td>fieldsV1:</td>
<td></td>
</tr>
<tr>
<td>f:authorizeToken:</td>
<td></td>
</tr>
<tr>
<td>f:clientName:</td>
<td></td>
</tr>
<tr>
<td>f:expiresIn:</td>
<td></td>
</tr>
<tr>
<td>f:redirectURI:</td>
<td></td>
</tr>
<tr>
<td>f:scopes:</td>
<td></td>
</tr>
<tr>
<td>f:userName:</td>
<td></td>
</tr>
<tr>
<td>f:userUID:</td>
<td></td>
</tr>
<tr>
<td>Manager:</td>
<td>oauth-server</td>
</tr>
<tr>
<td>Operation:</td>
<td>Update</td>
</tr>
<tr>
<td>Time:</td>
<td>2021-01-11T19:27:06Z</td>
</tr>
<tr>
<td>Resource Version:</td>
<td>30535</td>
</tr>
<tr>
<td>Self Link:</td>
<td>/apis/oauth.openshift.io/v1/useroauthaccesstokens/&lt;token_name&gt;</td>
</tr>
<tr>
<td>UID:</td>
<td>f9d00b67-ab65-489b-8080-e427fa3c6181</td>
</tr>
<tr>
<td>Redirect URI:</td>
<td><a href="https://oauth-openshift.apps.example.com/oauth/token/display">https://oauth-openshift.apps.example.com/oauth/token/display</a></td>
</tr>
<tr>
<td>Scopes:</td>
<td>user:full</td>
</tr>
<tr>
<td>User Name:</td>
<td>&lt;user_name&gt;</td>
</tr>
<tr>
<td>User UID:</td>
<td>82356ab0-95f9-4fb3-9bc0-10f1d6a6a345</td>
</tr>
<tr>
<td>Events:</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

1. The token name, which is the sha256 hash of the token. Token names are not sensitive and cannot be used to log in.
2. The client name, which describes where the token originated from.
3. The value in seconds from the creation time before this token expires.
4. If there is a token inactivity timeout set for the OAuth server, this is the value in seconds from the creation time before this token can no longer be used.
5. The scopes for this token.
6. The user name associated with this token.

### 4.3. DELETING USER-OWNED OAUTH ACCESS TOKENS

The `oc logout` command only invalidates the OAuth token for the active session. You can use the following procedure to delete any user-owned OAuth tokens that are no longer needed.
Deleting an OAuth access token logs out the user from all sessions that use the token.

**Procedure**

- Delete the user-owned OAuth access token:

  ```bash
  $ oc delete useroauthaccesstokens <token_name>
  ``

**Example output**

```
useroauthaccessstoken.oauth.openshift.io "<token_name>" deleted
```
CHAPTER 5. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION

The OpenShift Container Platform master includes a built-in OAuth server. Developers and administrators obtain OAuth access tokens to authenticate themselves to the API.

As an administrator, you can configure OAuth to specify an identity provider after you install your cluster.

5.1. ABOUT IDENTITY PROVIDERS IN OPENSHIFT CONTAINER PLATFORM

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

NOTE
OpenShift Container Platform user names containing /, :, and % are not supported.

5.2. SUPPORTED IDENTITY PROVIDERS

You can configure the following types of identity providers:

<table>
<thead>
<tr>
<th>Identity provider</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTPasswd</td>
<td>Configure the <code>htpasswd</code> identity provider to validate user names and passwords against a flat file generated using <code>htpasswd</code>.</td>
</tr>
<tr>
<td>Keystone</td>
<td>Configure the <code>keystone</code> identity provider to integrate your OpenShift Container Platform cluster with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database.</td>
</tr>
<tr>
<td>LDAP</td>
<td>Configure the <code>ldap</code> identity provider to validate user names and passwords against an LDAPv3 server, using simple bind authentication.</td>
</tr>
<tr>
<td>Basic authentication</td>
<td>Configure a basic-authentication identity provider for users to log in to OpenShift Container Platform with credentials validated against a remote identity provider. Basic authentication is a generic backend integration mechanism.</td>
</tr>
<tr>
<td>Request header</td>
<td>Configure a request-header identity provider to identify users from request header values, such as X-Remote-User. It is typically used in combination with an authenticating proxy, which sets the request header value.</td>
</tr>
<tr>
<td>GitHub or GitHub Enterprise</td>
<td>Configure a <code>github</code> identity provider to validate user names and passwords against GitHub or GitHub Enterprise’s OAuth authentication server.</td>
</tr>
<tr>
<td>GitLab</td>
<td>Configure a <code>gitlab</code> identity provider to use GitLab.com or any other GitLab instance as an identity provider.</td>
</tr>
</tbody>
</table>
Once an identity provider has been defined, you can use RBAC to define and apply permissions.

### 5.3. REMOVING THE KUBEADMIN USER

After you define an identity provider and create a new `cluster-admin` user, you can remove the `kubeadmin` to improve cluster security.

**WARNING**

If you follow this procedure before another user is a `cluster-admin`, then OpenShift Container Platform must be reinstalled. It is not possible to undo this command.

**Prerequisites**

- You must have configured at least one identity provider.
- You must have added the `cluster-admin` role to a user.
- You must be logged in as an administrator.

**Procedure**

- Remove the `kubeadmin` secrets:
  
  ```
  $ oc delete secrets kubeadmin -n kube-system
  ```

### 5.4. IDENTITY PROVIDER PARAMETERS

The following parameters are common to all identity providers:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The provider name is prefixed to provider user names to form an identity name.</td>
</tr>
</tbody>
</table>
1. **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 Container Platform 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**.

### 5.5. SAMPLE IDENTITY PROVIDER CR

The following custom resource (CR) shows the parameters and default values that you use to configure an identity provider. This example uses the HTPasswd identity provider.

**Sample identity provider CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_identity_provider
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: hpass-secret
```

1. This provider name is prefixed to provider user names to form an identity name.
2 Controls how mappings are established between this provider’s identities and User objects.

3 An existing secret containing a file generated using htpasswd.
CHAPTER 6. CONFIGURING IDENTITY PROVIDERS

6.1. CONFIGURING AN HTPASSWD IDENTITY PROVIDER

6.1.1. About identity providers in OpenShift Container Platform

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a Custom Resource (CR) that describes that identity provider and add it to the cluster.

NOTE

OpenShift Container Platform user names containing /, :, and % are not supported.

To define an HTPasswd identity provider you must perform the following steps:

1. Create an htpasswd file to store the user and password information. Instructions are provided for Linux and Windows.
2. Create an OpenShift Container Platform secret to represent the htpasswd file.
3. Define the HTPasswd identity provider resource.
4. Apply the resource to the default OAuth configuration.

6.1.2. Creating an HTPasswd file using Linux

To use the HTPasswd identity provider, you must generate a flat file that contains the user names and passwords for your cluster by using htpasswd.

Prerequisites

- Have access to the htpasswd utility. On Red Hat Enterprise Linux this is available by installing the httpd-tools package.

Procedure

1. Create or update your flat file with a user name and hashed password:

   $ htpasswd -c -B -b </path/to/users.htpasswd> <user_name> <password>

   The command generates a hashed version of the password.

   For example:

   $ htpasswd -c -B -b users.htpasswd user1 MyPassword!

   Example output

   Adding password for user user1

2. Continue to add or update credentials to the file:
6.1.3. Creating an HTPasswd file using Windows

To use the HTPasswd identity provider, you must generate a flat file that contains the user names and passwords for your cluster by using `htpasswd`.

**Prerequisites**

- Have access to `htpasswd.exe`. This file is included in the `\bin` directory of many Apache httpd distributions.

**Procedure**

1. Create or update your flat file with a user name and hashed password:

   ```sh
   htpasswd.exe -c -B -b \path\to\users.htpasswd <user_name> <password>
   
   The command generates a hashed version of the password.
   
   For example:
   
   ```sh
   htpasswd.exe -c -B -b users.htpasswd user1 MyPassword!
   ```

**Example output**

```
Adding password for user user1
```

2. Continue to add or update credentials to the file:

   ```sh
   htpasswd.exe -b \path\to\users.htpasswd <user_name> <password>
   ```

6.1.4. Creating the HTPasswd secret

To use the HTPasswd identity provider, you must define a secret that contains the HTPasswd user file.

**Prerequisites**

- Create an HTPasswd file.

**Procedure**

- Create a `Secret` object that contains the HTPasswd users file:

  ```sh
  $ oc create secret generic htpass-secret --from-file=htpasswd=\path\to\users.htpasswd -n openshift-config
  ```

**TIP**

The secret key containing the users file for the `--from-file` argument must be named `htpasswd`, as shown in the above command.
TIP

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
class: Secret
metadata:
  name: htpass-secret
  namespace: openshift-config
type: Opaque
data:
  htpasswd: <base64_encoded_hhtpasswd_file_contents>
```

6.1.5. Sample HTPasswd CR

The following custom resource (CR) shows the parameters and acceptable values for an HTPasswd identity provider.

**HTPasswd CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - name: my_htpasswd_provider
      mappingMethod: claim
      type: HTPasswd
      htpasswd:
        fileData:
          name: htpass-secret
```

1. This provider name is prefixed to provider user names to form an identity name.
2. Controls how mappings are established between this provider’s identities and **User** objects.
3. An existing secret containing a file generated using **htpasswd**.

Additional resources

- See **Identity provider parameters** for information on parameters, such as **mappingMethod**, that are common to all identity providers.

6.1.6. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

**Prerequisites**

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>

   **NOTE**
   
   If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: **Warning: oc apply should be used on resources created by either oc create --save-config or oc apply**. In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted:

   $ oc login -u <username>

3. Confirm that the user logged in successfully, and display the user name:

   $ oc whoami

6.1.7. Updating users for an HTPasswd identity provider

You can add or remove users from an existing HTPasswd identity provider.

Prerequisites

- You have created a **Secret** object that contains the HTPasswd user file. This procedure assumes that it is named **htpass-secret**.

- You have configured an HTPasswd identity provider. This procedure assumes that it is named **my_htpasswd_provider**.

- You have access to the **htpasswd** utility. On Red Hat Enterprise Linux this is available by installing the **httpd-tools** package.

- You have cluster administrator privileges.

Procedure

1. Retrieve the HTPasswd file from the **htpass-secret Secret** object and save the file to your file system:

   $ oc get secret htpass-secret -ojsonpath={.data.htpasswd} -n openshift-config | base64 --decode > users.htpasswd

2. Add or remove users from the **users.htpasswd** file.

   - To add a new user:
To remove an existing user:

Example output

```
Adding password for user <username>
```

- To remove an existing user:

```
$ htpasswd -D users.htpasswd <username>
```

Example output

```
Deleting password for user <username>
```

3. Replace the `htpass-secret Secret` object with the updated users in the `users.htpasswd` file:

```
$ oc create secret generic htpass-secret --from-file=htpasswd=users.htpasswd --dry-run=client -o yaml -n openshift-config | oc replace -f -
```

**TIP**

You can alternatively apply the following YAML to replace the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: htpass-secret
  namespace: openshift-config
  type: Opaque
data:
  htpasswd: <base64_encoded_htpasswd_file_contents>
```

4. If you removed one or more users, you must additionally remove existing resources for each user.

   a. Delete the `User` object:

```
$ oc delete user <username>
```

Example output

```
user.user.openshift.io "<username>" deleted
```

Be sure to remove the user, otherwise the user can continue using their token as long as it has not expired.

   b. Delete the `Identity` object for the user:

```
$ oc delete identity my_htpasswd_provider:<username>
```

Example output
6.1.8. Configuring identity providers using the web console

Configure your identity provider (IDP) through the web console instead of the CLI.

Prerequisites

- You must be logged in to the web console as a cluster administrator.

Procedure

1. Navigate to Administration → Cluster Settings.

2. Under the Global Configuration tab, click OAuth.

3. Under the Identity Providers section, select your identity provider from the Add drop-down menu.

**NOTE**

You can specify multiple IDPs through the web console without overwriting existing IDPs.

6.2. CONFIGURING A KEYSTONE IDENTITY PROVIDER

Configure the keystone identity provider to integrate your OpenShift Container Platform cluster with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database. This configuration allows users to log in to OpenShift Container Platform with their Keystone credentials.

**Keystone** is an OpenStack project that provides identity, token, catalog, and policy services.

You can configure the integration with Keystone so that the new OpenShift Container Platform users are based on either the Keystone user names or unique Keystone IDs. With both methods, users log in by entering their Keystone user name and password. Basing the OpenShift Container Platform users off of the Keystone ID is more secure. If you delete a Keystone user and create a new Keystone user with that user name, the new user might have access to the old user’s resources.

6.2.1. About identity providers in OpenShift Container Platform

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing /, :, and % are not supported.

6.2.2. Creating the secret

Identity providers use OpenShift Container Platform Secret objects in the openshift-config namespace to contain the client secret, client certificates, and keys.

**Procedure**

**identity.user.openshift.io "my_htpasswd_provider:<username>" deleted**
• Create a **Secret** object that contains the key and certificate by using the following command:

```
$ oc create secret tls <secret_name> --key=key.pem --cert=cert.pem -n openshift-config
```

**TIP**

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openshift-config
type: kubernetes.io/tls
data:
  tls.crt: <base64_encoded_cert>
  tls.key: <base64_encoded_key>
```

### 6.2.3. Creating a config map

Identity providers use OpenShift Container Platform **ConfigMap** objects in the *openshift-config* namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

**Procedure**

• Define an OpenShift Container Platform **ConfigMap** object containing the certificate authority by using the following command. The certificate authority must be stored in the `ca.crt` key of the **ConfigMap** object.

```
$ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
```

**TIP**

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
type: Opaque
data:
  ca.crt: |
    <CA_certificate_PEM>
```

### 6.2.4. Sample Keystone CR

The following custom resource (CR) shows the parameters and acceptable values for a Keystone identity provider.

**Keystone CR**
This provider name is prefixed to provider user names to form an identity name.

Controls how mappings are established between this provider’s identities and User objects.

Keystone domain name. In Keystone, usernames are domain-specific. Only a single domain is supported.

The URL to use to connect to the Keystone server (required). This must use https.

Optional: Reference to an OpenShift Container Platform ConfigMap object containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL.

Optional: Reference to an OpenShift Container Platform Secret object containing the client certificate to present when making requests to the configured URL.

Reference to an OpenShift Container Platform Secret object containing the key for the client certificate. Required if tlsClientCert is specified.

Additional resources

- See Identity provider parameters for information on parameters, such as mappingMethod, that are common to all identity providers.

6.2.5. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
- You must be logged in as an administrator.
Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>

   **NOTE**
   If a CR does not exist, oc apply creates a new CR and might trigger the following warning: **Warning: oc apply should be used on resources created by either oc create --save-config or oc apply.** In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted.

   $ oc login -u <username>

3. Confirm that the user logged in successfully, and display the user name.

   $ oc whoami

6.3. CONFIGURING AN LDAP IDENTITY PROVIDER

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

6.3.1. About identity providers in OpenShift Container Platform

By default, only a **kubeadmin** user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**
OpenShift Container Platform user names containing /, :, and % are not supported.

6.3.2. 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:

```
```

For this URL:

<table>
<thead>
<tr>
<th>URL component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldap</td>
<td>For regular LDAP, use the string <code>ldap</code>. For secure LDAP (LDAPS), use <code>ldaps</code> instead.</td>
</tr>
<tr>
<td>host:port</td>
<td>The name and port of the LDAP server. Defaults to <code>localhost:389</code> for ldap and <code>localhost:636</code> for LDAPS.</td>
</tr>
<tr>
<td>basedn</td>
<td>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.</td>
</tr>
<tr>
<td>attribute</td>
<td>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 <code>uid</code>. It is recommended to choose an attribute that will be unique across all entries in the subtree you will be using.</td>
</tr>
<tr>
<td>scope</td>
<td>The scope of the search. Can be either <code>one</code> or <code>sub</code>. If the scope is not provided, the default is to use a scope of <code>sub</code>.</td>
</tr>
<tr>
<td>filter</td>
<td>A valid LDAP search filter. If not provided, defaults to <code>(objectClass=*)</code>.</td>
</tr>
</tbody>
</table>

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.

### 6.3.3. Creating the LDAP secret


To use the identity provider, you must define an OpenShift Container Platform Secret object that contains the bindPassword field.

**Procedure**

- Create a Secret object that contains the bindPassword field:

  ```sh
  $ oc create secret generic ldap-secret --from-literal=bindPassword=<secret> -n openshift-config
  ``

  The secret key containing the bindPassword for the --from-literal argument must be called bindPassword.

**TIP**

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: ldap-secret
  namespace: openshift-config
  type: Opaque
data:
  bindPassword: <base64_encoded_bind_password>
```

### 6.3.4. Creating a config map

Identity providers use OpenShift Container Platform ConfigMap objects in the openshift-config namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

**Procedure**

- Define an OpenShift Container Platform ConfigMap object containing the certificate authority by using the following command. The certificate authority must be stored in the ca.crt key of the ConfigMap object.

  ```sh
  $ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
  ```
TIP

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
type: Opaque
data:
  ca.crt: |
    <CA_certificate_PEM>
```

6.3.5. Sample LDAP CR

The following custom resource (CR) shows the parameters and acceptable values for an LDAP identity provider.

**LDAP CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: ldapidp
    mappingMethod: claim
    type: LDAP
    ldap:
      attributes:
        id: 
        - dn
        email:
        - mail
        name:
        - cn
        preferredUsername:
        - uid
        bindDN: ""
      bindPassword: 
        name: ldap-secret
      ca:
        name: ca-config-map
      insecure: false
    url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid"
```

1. This provider name is prefixed to the returned user ID to form an identity name.
2. Controls how mappings are established between this provider’s identities and User objects.
3. List of attributes to use as the identity. First non-empty attribute is used. At least one attribute is required. If none of the listed attribute have a value, authentication fails. Defined attributes are
List of attributes to use as the email address. First non-empty attribute is used.

List of attributes to use as the display name. First non-empty attribute is used.

List of attributes to use as the preferred user name when provisioning a user for this identity. First non-empty attribute is used.

Optional DN to use to bind during the search phase. Must be set if bindPassword is defined.

Optional reference to an OpenShift Container Platform Secret object containing the bind password. Must be set if bindDN is defined.

Optional: Reference to an OpenShift Container Platform ConfigMap object containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL. Only used when insecure is false.

When true, no TLS connection is made to the server. When false, ldaps:// URLs connect using TLS, and ldap:// URLs are upgraded to TLS. This must be set to false when ldaps:// URLs are in use, as these URLs always attempt to connect using TLS.

An RFC 2255 URL which specifies the LDAP host and search parameters to use.

NOTE

To whitelist users for an LDAP integration, use the lookup mapping method. Before a login from LDAP would be allowed, a cluster administrator must create an Identity object and a User object for each LDAP user.

Additional resources

- See Identity provider parameters for information on parameters, such as mappingMethod, that are common to all identity providers.

6.3.6. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
- You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>
NOTE

If a CR does not exist, oc apply creates a new CR and might trigger the following warning: Warning: oc apply should be used on resources created by either oc create --save-config or oc apply. In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted.

   $ oc login -u <username>

3. Confirm that the user logged in successfully, and display the user name.

   $ oc whoami

6.4. CONFIGURING A BASIC AUTHENTICATION IDENTITY PROVIDER

Configure a basic-authentication identity provider for users to log in to OpenShift Container Platform with credentials validated against a remote identity provider. Basic authentication is a generic back-end integration mechanism.

6.4.1. About identity providers in OpenShift Container Platform

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

NOTE

OpenShift Container Platform user names containing /, :, and % are not supported.

6.4.2. About basic authentication

Basic authentication is a generic back-end integration mechanism that allows users to log in to OpenShift Container Platform with credentials validated against a remote identity provider.

Because basic authentication is generic, you can use this identity provider for advanced authentication configurations.

IMPORTANT

Basic authentication must use an HTTPS connection to the remote server to prevent potential snooping of the user ID and password and man-in-the-middle attacks.

With basic authentication configured, users send their user name and password to OpenShift Container Platform, which then validates those credentials against a remote server by making a server-to-server request, passing the credentials as a basic authentication header. This requires users to send their credentials to OpenShift Container Platform during login.
NOTE
This only works for user name/password login mechanisms, and OpenShift Container Platform must be able to make network requests to the remote authentication server.

User names and passwords are validated against a remote URL that is protected by basic authentication and returns JSON.

A 401 response indicates failed authentication.

A non-200 status, or the presence of a non-empty "error" key, indicates an error:

```
{"error":"Error message"}
```

A 200 status with a sub (subject) key indicates success:

```
{"sub":"userid"}
```

1. The subject must be unique to the authenticated user and must not be able to be modified.

A successful response can optionally provide additional data, such as:

- A display name using the name key. For example:

```
{"sub":"userid", "name": "User Name", ...}
```

- An email address using the email key. For example:

```
{"sub":"userid", "email":"user@example.com", ...}
```

- A preferred user name using the preferred_username key. This is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity. For example:

```
{"sub":"014fbff9a07c", "preferred_username":"bob", ...}
```

### 6.4.3. Creating the secret

Identity providers use OpenShift Container Platform Secret objects in the openshift-config namespace to contain the client secret, client certificates, and keys.

**Procedure**

- Create a Secret object that contains the key and certificate by using the following command:

```
$ oc create secret tls <secret_name> --key=key.pem --cert=cert.pem -n openshift-config
```
TIP

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openshift-config
type: kubernetes.io/tls
data:
  tls.crt: <base64_encoded_cert>
  tls.key: <base64_encoded_key>
```

6.4.4. Creating a config map

Identity providers use OpenShift Container Platform ConfigMap objects in the openshift-config namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

Procedure

- Define an OpenShift Container Platform ConfigMap object containing the certificate authority by using the following command. The certificate authority must be stored in the ca.crt key of the ConfigMap object.

  ```bash
  $ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
  ```

TIP

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
type: Opaque
data:
  ca.crt: |
    <CA_certificate_PEM>
```

6.4.5. Sample basic authentication CR

The following custom resource (CR) shows the parameters and acceptable values for a basic authentication identity provider.

Basic authentication CR

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
```
This provider name is prefixed to the returned user ID to form an identity name.

Controls how mappings are established between this provider’s identities and User objects.

URL accepting credentials in Basic authentication headers.

Optional: Reference to an OpenShift Container Platform ConfigMap object containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL.

Optional: Reference to an OpenShift Container Platform Secret object containing the client certificate to present when making requests to the configured URL.

Reference to an OpenShift Container Platform Secret object containing the key for the client certificate. Required if tlsClientCert is specified.

Additional resources

- See Identity provider parameters for information on parameters, such as mappingMethod, that are common to all identity providers.

6.4.6. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
- You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>
NOTE

If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: **Warning: oc apply should be used on resources created by either `oc create --save-config` or `oc apply`.** In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted.

```bash
$ oc login -u <username>
```

3. Confirm that the user logged in successfully, and display the user name.

```bash
$ oc whoami
```

### 6.4.7. Example Apache HTTPD configuration for basic identity providers

The basic identity provider (IDP) configuration in OpenShift Container Platform 4 requires that the IDP server respond with JSON for success and failures. You can use CGI scripting in Apache HTTPD to accomplish this. This section provides examples.

**Example /etc/httpd/conf.d/login.conf**

```xml
<VirtualHost *:443>
    # CGI Scripts in here
    DocumentRoot /var/www/cgi-bin

    # SSL Directives
    SSLSession on
    SSLCipherSuite PROFILE=SYSTEM
    SSLProxyCipherSuite PROFILE=SYSTEM
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key

    # Configure HTTPD to execute scripts
    ScriptAlias /basic /var/www/cgi-bin

    # Handles a failed login attempt
    ErrorDocument 401 /basic/fail.cgi

    # Handles authentication
    <Location /basic/login.cgi>
        AuthType Basic
        AuthName "Please Log In"
        AuthBasicProvider file
        AuthUserFile /etc/httpd/conf/passwords
        Require valid-user
    </Location>
</VirtualHost>
```

**Example /var/www/cgi-bin/login.cgi**

```bash
$ oc login -u <username>
$ oc whoami
```
6.4.7.1. File requirements

These are the requirements for the files you create on an Apache HTTPD web server:

- `login.cgi` and `fail.cgi` must be executable (`chmod +x`).
- `login.cgi` and `fail.cgi` must have proper SELinux contexts if SELinux is enabled: `restorecon -RFV /var/www/cgi-bin`, or ensure that the context is `httpd_sys_script_exec_t` using `ls -laZ`.
- `login.cgi` is only executed if your user successfully logs in per `Require` and `Auth` directives.
- `fail.cgi` is executed if the user fails to log in, resulting in an `HTTP 401` response.

6.4.8. Basic authentication troubleshooting

The most common issue relates to network connectivity to the backend server. For simple debugging, run `curl` commands on the master. To test for a successful login, replace the `<user>` and `<password>` in the following example command with valid credentials. To test an invalid login, replace them with false credentials.

```
$ curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key /path/to/client.key -u <user>:<password> -v https://www.example.com/remote-idp
```

Successful responses

A `200` status with a `sub` (subject) key indicates success:

```
{"sub":"userid"}
```

The subject must be unique to the authenticated user, and must not be able to be modified.

A successful response can optionally provide additional data, such as:

- A display name using the `name` key:
  
  ```
  {
  "sub":"userid", "name": "User Name", ...
  }
  ```

- An email address using the `email` key:
A preferred user name using the preferred_username key:

```json
{"sub":"014bf9a07c", "preferred_username":"bob", ...}
```

The preferred_username key is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity.

**Failed responses**

- A 401 response indicates failed authentication.
- A non-200 status or the presence of a non-empty "error" key indicates an error: 

```json
{"error":"Error message"}
```

### 6.5. CONFIGURING A REQUEST HEADER IDENTITY PROVIDER

Configure a request-header identity provider to identify users from request header values, such as X-Remote-User. It is typically used in combination with an authenticating proxy, which sets the request header value.

#### 6.5.1. About identity providers in OpenShift Container Platform

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing /, :, and % are not supported.

#### 6.5.2. About request header authentication

A request header identity provider identifies users from request header values, such as X-Remote-User. It is typically used in combination with an authenticating proxy, which sets the request header value. The request header identity provider cannot be combined with other identity providers that use direct password logins, such as HTPasswd, Keystone, LDAP or Basic authentication.

**NOTE**

You can also use the request header identity provider for advanced configurations such as the community-supported SAML authentication. Note that this solution is not supported by Red Hat.

For users to authenticate using this identity provider, they must access `https://<namespace_route>/oauth/authorize` (and subpaths) via an authenticating proxy. To accomplish this, configure the OAuth server to redirect unauthenticated requests for OAuth tokens to the proxy endpoint that proxies to `https://<namespace_route>/oauth/authorize`.

To redirect unauthenticated requests from clients expecting browser-based login flows:
• Set the `provider.loginURL` parameter to the authenticating proxy URL that will authenticate interactive clients and then proxy the request to https://<namespace_route>/oauth/authorize.

To redirect unauthenticated requests from clients expecting `WWW-Authenticate` challenges:

• Set the `provider.challengeURL` parameter to the authenticating proxy URL that will authenticate clients expecting `WWW-Authenticate` challenges and then proxy the request to https://<namespace_route>/oauth/authorize.

The `provider.challengeURL` and `provider.loginURL` parameters can include the following tokens in the query portion of the URL:

• `{url}` is replaced with the current URL, escaped to be safe in a query parameter.
  For example: https://www.example.com/sso-login?then={url}

• `{query}` is replaced with the current query string, unescaped.
  For example: https://www.example.com/auth-proxy/oauth/authorize?{query}

**IMPORTANT**
As of OpenShift Container Platform 4.1, your proxy must support mutual TLS.

### 6.5.2.1. SSPI connection support on Microsoft Windows

**IMPORTANT**
Using SSPI connection support on Microsoft Windows is a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see https://access.redhat.com/support/offerings/techpreview/.

The OpenShift CLI (oc) supports the Security Support Provider Interface (SSPI) to allow for SSO flows on Microsoft Windows. If you use the request header identity provider with a GSSAPI-enabled proxy to connect an Active Directory server to OpenShift Container Platform, users can automatically authenticate to OpenShift Container Platform by using the oc command line interface from a domain-joined Microsoft Windows computer.

### 6.5.3. Creating a config map

Identity providers use OpenShift Container Platform ConfigMap objects in the openshift-config namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

**Procedure**

• Define an OpenShift Container Platform ConfigMap object containing the certificate authority by using the following command. The certificate authority must be stored in the `ca.crt` key of the ConfigMap object.
$ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config

TIP

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
  type: Opaque
data:
  ca.crt:
    <CA_certificate_PEM>
```

6.5.4. Sample request header CR

The following custom resource (CR) shows the parameters and acceptable values for a request header identity provider.

Request header CR

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - name: requestheaderidp
      mappingMethod: claim
      type: RequestHeader
      requestHeader:
        challengeURL: "https://www.example.com/challenging-proxy/oauth/authorize?${query}"
        loginURL: "https://www.example.com/login-proxy/oauth/authorize?${query}"
  ca:
    name: ca-config-map
  clientCommonNames:
    - my-auth-proxy
  headers:
    - X-Remote-User
    - SSO-User
  emailHeaders:
    - X-Remote-User-Email
  nameHeaders:
    - X-Remote-User-Display-Name
  preferredUsernameHeaders:
    - X-Remote-User-Login
```

1. This provider name is prefixed to the user name in the request header to form an identity name.
2. Controls how mappings are established between this provider’s identities and User objects.
3 Optional: URL to redirect unauthenticated `/oauth/authorize` requests to, that will authenticate browser-based clients and then proxy their request to

4 Optional: URL to redirect unauthenticated `/oauth/authorize` requests to, that will authenticate clients which expect `WWW-Authenticate` challenges, and then proxy them to `https://<namespace_route>/oauth/authorize`. `${url}` is replaced with the current URL, escaped to be safe in a query parameter. `${query}` is replaced with the current query string. If this attribute is not defined, then `challengeURL` must be used.

5 Reference to an OpenShift Container Platform ConfigMap object containing a PEM-encoded certificate bundle. Used as a trust anchor to validate the TLS certificates presented by the remote server.

   IMPORTANT
   As of OpenShift Container Platform 4.1, the `ca` field is required for this identity provider. This means that your proxy must support mutual TLS.

6 Optional: list of common names (`cn`). If set, a valid client certificate with a Common Name (`cn`) in the specified list must be presented before the request headers are checked for user names. If empty, any Common Name is allowed. Can only be used in combination with `ca`.

7 Header names to check, in order, for the user identity. The first header containing a value is used as the identity. Required, case-insensitive.

8 Header names to check, in order, for an email address. The first header containing a value is used as the email address. Optional, case-insensitive.

9 Header names to check, in order, for a display name. The first header containing a value is used as the display name. Optional, case-insensitive.

10 Header names to check, in order, for a preferred user name, if different than the immutable identity determined from the headers specified in `headers`. The first header containing a value is used as the preferred user name when provisioning. Optional, case-insensitive.

Additional resources

- See Identity provider parameters for information on parameters, such as `mappingMethod`, that are common to all identity providers.

6.5.5. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.

- Create the custom resource (CR) for your identity providers.

- You must be logged in as an administrator.

Procedure
1. Apply the defined CR:

```
$ oc apply -f </path/to/CR>
```

**NOTE**

If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: **Warning: oc apply should be used on resources created by either oc create --save-config or oc apply.** In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted.

```
$ oc login -u <username>
```

3. Confirm that the user logged in successfully, and display the user name.

```
$ oc whoami
```

### 6.5.6. Example Apache authentication configuration using request header

This example configures an Apache authentication proxy for the OpenShift Container Platform using the request header identity provider.

**Custom proxy configuration**

Using the `mod_auth_gssapi` module is a popular way to configure the Apache authentication proxy using the request header identity provider; however, it is not required. Other proxies can easily be used if the following requirements are met:

- Block the `X-Remote-User` header from client requests to prevent spoofing.
- Enforce client certificate authentication in the `RequestHeaderIdentityProvider` configuration.
- Require the `X-Csrf-Token` header be set for all authentication requests using the challenge flow.
- Make sure only the `/oauth/authorize` endpoint and its subpaths are proxied; redirects must be rewritten to allow the backend server to send the client to the correct location.
- The URL that proxies to `https://<namespace_route>/oauth/authorize` must end with `/authorize` with no trailing slash. For example, `https://proxy.example.com/login-proxy/authorize?...` must proxy to `https://<namespace_route>/oauth/authorize?...`

- Subpaths of the URL that proxies to `https://<namespace_route>/oauth/authorize` must proxy to subpaths of `https://<namespace_route>/oauth/authorize`. For example, `https://proxy.example.com/login-proxy/authorize/approve?...` must proxy to `https://<namespace_route>/oauth/authorize/approve?...`

**NOTE**

The `https://<namespace_route>` address is the route to the OAuth server and can be obtained by running `oc get route -n openshift-authentication`. 
Configuring Apache authentication using request header
This example uses the `mod_auth_gssapi` module to configure an Apache authentication proxy using the request header identity provider.

Prerequisites

- Obtain the `mod_auth_gssapi` module from the Optional channel. You must have the following packages installed on your local machine:
  
  - `httpd`
  - `mod_ssl`
  - `mod_session`
  - `apr-util-openssl`
  - `mod_auth_gssapi`

- Generate a CA for validating requests that submit the trusted header. Define an OpenShift Container Platform ConfigMap object containing the CA. This is done by running:

  ```
  $ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
  ``

  The CA must be stored in the `ca.crt` key of the ConfigMap object.

  **TIP**

  You can alternatively apply the following YAML to create the config map:

  ```yaml
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: ca-config-map
    namespace: openshift-config
  type: Opaque
  data:
    ca.crt: |
    <CA_certificate_PEM>
  ```

- Generate a client certificate for the proxy. You can generate this certificate by using any x509 certificate tooling. The client certificate must be signed by the CA you generated for validating requests that submit the trusted header.

- Create the custom resource (CR) for your identity providers.

Procedure

This proxy uses a client certificate to connect to the OAuth server, which is configured to trust the `X-Remote-User` header.

1. Create the certificate for the Apache configuration. The certificate that you specify as the `SSLProxyMachineCertificateFile` parameter value is the proxy’s client certificate that is used to authenticate the proxy to the server. It must use TLS Web Client Authentication as the
2. Create the Apache configuration. Use the following template to provide your required settings and values:

**IMPORTANT**

Carefully review the template and customize its contents to fit your environment.

```apache
LoadModule request_module modules/mod_request.so
LoadModule auth_gssapi_module modules/mod_auth_gssapi.so
# Some Apache configurations might require these modules.
# LoadModule auth_form_module modules/mod_auth_form.so
# LoadModule session_module modules/mod_session.so

# Nothing needs to be served over HTTP. This virtual host simply redirects to HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine On
  RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated. See the CN and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
  ServerName www.example.com

  DocumentRoot /var/www/html
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It is critical to enforce client certificates. Otherwise, requests can
  # spoof the X-Remote-User header by accessing the /oauth/authorize endpoint
  # directly.
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

  # To use the challenging-proxy, an X-Csrft-Token must be present.
  RewriteCond %{REQUEST_URI} ^/oauth/authorize
  RewriteCond %{HTTP:X-Csrf-Token} ^"$ [NC]
  RewriteRule ^.* - [F,L]
  <Location /oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://<namespace_route>/oauth/authorize
    AuthName "SSO Login"
    # For Kerberos
    AuthType GSSAPI
    Require valid-user
    RequestHeader set X-Remote-User %{REMOTE_USER}s
  </Location>
```
GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
# Enable the following if you want to allow users to fallback
# to password based authentication when they do not have a client
# configured to perform kerberos authentication.
GssapiBasicAuth On

# For ldap:
# AuthBasicProvider ldap
# AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-domain,dc=com?uid?
sub?(objectClass=*)"
</Location>

<Location /login-proxy/oauth/authorize>
# Insert your backend server name/ip here.
ProxyPass https://<namespace_route>/oauth/authorize

AuthName "SSO Login"
AuthType GSSAPI
Require valid-user
RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
# Enable the following if you want to allow users to fallback
# to password based authentication when they do not have a client
# configured to perform kerberos authentication.
GssapiBasicAuth On

ErrorDocument 401 /login.html
</Location>
</VirtualHost>

RequestHeader unset X-Remote-User

NOTE
The \texttt{https://<namespace\_route>} address is the route to the OAuth server and can be obtained by running \texttt{oc get route -n openshift-authentication}.

3. Update the \texttt{identityProviders} stanza in the custom resource (CR):

```yaml
identityProviders:
  - name: requestheaderidp
type: RequestHeader
  requestHeader:
    challengeURL: "https://<namespace\_route>/challenging-proxy/oauth/authorize?${query}"
    loginURL: "https://<namespace\_route>/login-proxy/oauth/authorize?${query}"
    ca:
      name: ca-config-map
      clientCommonNames:
        - my-auth-proxy
      headers:
        - X-Remote-User
```
4. Verify the configuration.
   a. Confirm that you can bypass the proxy by requesting a token by supplying the correct client certificate and header:

```
# curl -L -k -H "X-Remote-User: joe" \
--cert /etc/pki/tls/certs/authproxy.pem \
https://<namespace_route>/oauth/token/request
```

b. Confirm that requests that do not supply the client certificate fail by requesting a token without the certificate:

```
# curl -L -k -H "X-Remote-User: joe" \
https://<namespace_route>/oauth/token/request
```

c. Confirm that the challengeURL redirect is active:

```
# curl -k -v -H 'X-Csrf-Token: 1' \
```

Copy the challengeURL redirect to use in the next step.

d. Run this command to show a 401 response with a WWW-Authenticate basic challenge, a negotiate challenge, or both challenges:

```
# curl -k -v -H 'X-Csrf-Token: 1' \
<challengeURL_redirect + query>
```

e. Test logging in to the OpenShift CLI (oc) with and without using a Kerberos ticket:
   i. If you generated a Kerberos ticket by using kinit, destroy it:

```
# kdestroy -c cache_name
```

1. Make sure to provide the name of your Kerberos cache.

   ii. Log in to the oc tool by using your Kerberos credentials:

```
# oc login -u <username>
```

Enter your Kerberos password at the prompt.

   iii. Log out of the oc tool:

```
# oc logout
```

   iv. Use your Kerberos credentials to get a ticket:

```
# kinit
```

Enter your Kerberos user name and password at the prompt.
v. Confirm that you can log in to the oc tool:

```
# oc login
```

If your configuration is correct, you are logged in without entering separate credentials.

### 6.6. 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 Container Platform 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 Container Platform with their GitHub credentials. To prevent anyone with any GitHub user ID from logging in to your OpenShift Container Platform cluster, you can restrict access to only those in specific GitHub organizations.

#### 6.6.1. About identity providers in OpenShift Container Platform

By default, only a kubeadmin user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing /, :, and % are not supported.

#### 6.6.2. 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.

### 6.6.3. Creating the secret

Identity providers use OpenShift Container Platform Secret objects in the openshift-config namespace to contain the client secret, client certificates, and keys.

**Procedure**

- Create a Secret object containing a string by using the following command:

  ```bash
  $ oc create secret generic <secret_name> --from-literal=clientSecret=<secret> -n openshift-config
  ```

  **TIP**

  You can alternatively apply the following YAML to create the secret:

  ```yaml
  apiVersion: v1
  kind: Secret
  metadata:
    name: <secret_name>
  namespace: openshift-config
  type: Opaque
  data:
    clientSecret: <base64_encoded_client_secret>
  ```

- You can define a Secret object containing the contents of a file, such as a certificate file, by using the following command:

  ```bash
  $ oc create secret generic <secret_name> --from-file=<path_to_file> -n openshift-config
  ```

### 6.6.4. Creating a config map

Identity providers use OpenShift Container Platform ConfigMap objects in the openshift-config namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.
NOTE
This procedure is only required for GitHub Enterprise.

Procedure

- Define an OpenShift Container Platform ConfigMap object containing the certificate authority by using the following command. The certificate authority must be stored in the `ca.crt` key of the ConfigMap object.

```bash
$ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
```

TIP
You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
type: Opaque
data:
  ca.crt: |
    <CA_certificate_PEM>
```

6.6.5. Sample GitHub CR

The following custom resource (CR) shows the parameters and acceptable values for a GitHub identity provider.

GitHub CR

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProducers:
  - name: githubidp
    mappingMethod: claim
    type: GitHub
github:
  ca: |
    name: ca-config-map
  clientID: {...}
  clientSecret: |
    name: github-secret
  hostname: ...
  organizations:
    - myorganization1
    - myorganization2
```
teams:
- myorganization1/team-a
- myorganization2/team-b

1. This provider name is prefixed to the GitHub numeric user ID to form an identity name. It is also used to build the callback URL.

2. Controls how mappings are established between this provider’s identities and User objects.

3. Optional: Reference to an OpenShift Container Platform ConfigMap object containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL. Only for use in GitHub Enterprise with a non-publicly trusted root certificate.

4. The client ID of a registered GitHub OAuth application. The application must be configured with a callback URL of https://oauth-openshift.apps.<cluster-name>.<cluster-domain>/oauth2callback/<idp-provider-name>.

5. Reference to an OpenShift Container Platform Secret object containing the client secret issued by GitHub.

6. For GitHub Enterprise, you must provide the hostname of your instance, such as example.com. This value must match the GitHub Enterprise hostname value in the /setup/settings file and cannot include a port number. If this value is not set, then either teams or organizations must be defined. For GitHub, omit this parameter.

7. The list of organizations. Either the organizations or teams field must be set unless the hostname field is set, or if mappingMethod is set to lookup. Cannot be used in combination with the teams field.

8. The list of teams. Either the teams or organizations field must be set unless the hostname field is set, or if mappingMethod is set to lookup. Cannot be used in combination with the organizations field.

NOTE

If organizations or teams is specified, only GitHub users that are members of at least one of the listed organizations will be allowed to log in. If the GitHub OAuth application configured in clientID is not owned by the organization, an organization owner must grant third-party access to use this option. This can be done during the first GitHub login by the organization’s administrator, or from the GitHub organization settings.

Additional resources

- See Identity provider parameters for information on parameters, such as mappingMethod, that are common to all identity providers.

6.6.6. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.
Create the custom resource (CR) for your identity providers.

You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   ```bash
   $ oc apply -f </path/to/CR>
   ```

   **NOTE**
   
   If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: *Warning: oc apply should be used on resources created by either* `oc create --save-config` *or* `oc apply`. In this case you can safely ignore this warning.

2. Obtain a token from the OAuth server.
   
   As long as the `kubeadm` user has been removed, the `oc login` command provides instructions on how to access a web page where you can retrieve the token.

   You can also access this page from the web console by navigating to (?) Help → Command Line Tools → Copy Login Command.

3. Log in to the cluster, passing in the token to authenticate.

   ```bash
   $ oc login --token=<token>
   ```

   **NOTE**
   
   This identity provider does not support logging in with a user name and password.

4. Confirm that the user logged in successfully, and display the user name.

   ```bash
   $ oc whoami
   ```

6.7. CONFIGURING A GITLAB IDENTITY PROVIDER

Configure a *gitlab* identity provider to use *GitLab.com* or any other GitLab instance as an identity provider. If you use GitLab version 7.7.0 to 11.0, you connect using the *OAuth integration*. If you use GitLab version 11.1 or later, you can use *OpenID Connect* (OIDC) to connect instead of OAuth.

6.7.1. About identity providers in OpenShift Container Platform

By default, only a *kubeadm* user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing /, :, and % are not supported.
6.7.2. Creating the secret

Identity providers use OpenShift Container Platform Secret objects in the openshift-config namespace to contain the client secret, client certificates, and keys.

**Procedure**

- Create a Secret object containing a string by using the following command:

  ```
  $ oc create secret generic <secret_name> --from-literal=clientSecret=<secret> -n openshift-config
  ```

  **TIP**

  You can alternatively apply the following YAML to create the secret:

  ```yaml
  apiVersion: v1
  kind: Secret
  metadata:
    name: <secret_name>
    namespace: openshift-config
  type: Opaque
  data:
    clientSecret: <base64_encoded_client_secret>
  ```

- You can define a Secret object containing the contents of a file, such as a certificate file, by using the following command:

  ```
  $ oc create secret generic <secret_name> --from-file=<path_to_file> -n openshift-config
  ```

6.7.3. Creating a config map

Identity providers use OpenShift Container Platform ConfigMap objects in the openshift-config namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

**NOTE**

This procedure is only required for GitHub Enterprise.

**Procedure**

- Define an OpenShift Container Platform ConfigMap object containing the certificate authority by using the following command. The certificate authority must be stored in the ca.crt key of the ConfigMap object.

  ```
  $ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
  ```
TIP

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
  type: Opaque
data:
  ca.crt: |
    <CA_certificate_PEM>
```

6.7.4. Sample GitLab CR

The following custom resource (CR) shows the parameters and acceptable values for a GitLab identity provider.

**GitLab CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - name: gitlabidp
      mappingMethod: claim
      type: GitLab
      gitlab:
        clientId: {...}
        clientSecret: 
        name: gitlab-secret
        url: https://gitlab.com
        ca:
        name: ca-config-map
```

1. This provider name is prefixed to the GitLab numeric user ID to form an identity name. It is also used to build the callback URL.

2. Controls how mappings are established between this provider’s identities and User objects.

3. The client ID of a registered GitLab OAuth application. The application must be configured with a callback URL of `https://oauth-openshift.apps.<cluster-name>.<cluster-domain>/oauth2callback/<idp-provider-name>`.

4. Reference to an OpenShift Container Platform Secret object containing the client secret issued by GitLab.

5. The host URL of a GitLab provider. This could either be `https://gitlab.com/` or any other self hosted instance of GitLab.

6. Optional: Reference to an OpenShift Container Platform ConfigMap object containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL.
Additional resources

- See Identity provider parameters for information on parameters, such as mappingMethod, that are common to all identity providers.

6.7.5. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

Prerequisites

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
- You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>

   **NOTE**
   If a CR does not exist, oc apply creates a new CR and might trigger the following warning: Warning: oc apply should be used on resources created by either oc create --save-config or oc apply. In this case you can safely ignore this warning.

2. Log in to the cluster as a user from your identity provider, entering the password when prompted.

   $ oc login -u <username>

3. Confirm that the user logged in successfully, and display the user name.

   $ oc whoami

6.8. 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.
6.8.1. About identity providers in OpenShift Container Platform

By default, only a `kubeadmin` user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing `/`, `;`, and `%` are not supported.

6.8.2. Creating the secret

Identity providers use OpenShift Container Platform `Secret` objects in the `openshift-config` namespace to contain the client secret, client certificates, and keys.

**Procedure**

- Create a `Secret` object containing a string by using the following command:

  ```
  $ oc create secret generic <secret_name> --from-literal=clientSecret=<secret> -n openshift-config
  ```

**TIP**

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
namespace: openshift-config
type: Opaque
data:
  clientSecret: <base64_encoded_client_secret>
```

- You can define a `Secret` object containing the contents of a file, such as a certificate file, by using the following command:

  ```
  $ oc create secret generic <secret_name> --from-file=<path_to_file> -n openshift-config
  ```

6.8.3. Sample Google CR

The following custom resource (CR) shows the parameters and acceptable values for a Google identity provider:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openshift-config
type: Opaque
data:
  clientSecret: <base64_encoded_client_secret>
```
The following custom resource (CR) shows the parameters and acceptable values for a Google identity provider.

**Google CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: googleidp
    mappingMethod: claim
    type: Google
    google:
      clientID: {...}
      clientSecret: name: google-secret
      hostedDomain: "example.com"
```

1. This provider name is prefixed to the Google numeric user ID to form an identity name. It is also used to build the redirect URL.

2. Controls how mappings are established between this provider’s identities and User objects.

3. The client ID of a registered Google project. The project must be configured with a redirect URI of `https://oauth-openshift.apps.<cluster-name>.<cluster-domain>/oauth2callback/<idp-provider-name>`.

4. Reference to an OpenShift Container Platform Secret object containing the client secret issued by Google.

5. A hosted domain used to restrict sign-in accounts. Optional if the lookup mappingMethod is used. If empty, any Google account is allowed to authenticate.

**Additional resources**

- See [Identity provider parameters](#) for information on parameters, such as `mappingMethod`, that are common to all identity providers.

### 6.8.4. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.

**Prerequisites**

- Create an OpenShift Container Platform cluster.

- Create the custom resource (CR) for your identity providers.

- You must be logged in as an administrator.

**Procedure**
1. Apply the defined CR:

   ```
   $ oc apply -f </path/to/CR>
   ```

   **NOTE**
   
   If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: `Warning: oc apply should be used on resources created by either oc create --save-config or oc apply`. In this case you can safely ignore this warning.

2. Obtain a token from the OAuth server.
   
   As long as the `kubeadmin` user has been removed, the `oc login` command provides instructions on how to access a web page where you can retrieve the token.

   You can also access this page from the web console by navigating to (? Help → Command Line Tools → Copy Login Command

3. Log in to the cluster, passing in the token to authenticate.

   ```
   $ oc login --token=<token>
   ```

   **NOTE**
   
   This identity provider does not support logging in with a user name and password.

4. Confirm that the user logged in successfully, and display the user name.

   ```
   $ oc whoami
   ```

### 6.9. CONFIGURING A OPENID CONNECT IDENTITY PROVIDER

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

You can configure Red Hat Single Sign-On as an OpenID Connect identity provider for OpenShift Container Platform.

**IMPORTANT**

The Authentication Operator in OpenShift Container Platform 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:

<table>
<thead>
<tr>
<th>Claim</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>Short for “subject identifier.” The remote identity for the user at the issuer.</td>
</tr>
<tr>
<td>preferred_username</td>
<td>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.</td>
</tr>
<tr>
<td>email</td>
<td>Email address.</td>
</tr>
<tr>
<td>name</td>
<td>Display name.</td>
</tr>
</tbody>
</table>

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.

### 6.9.1. About identity providers in OpenShift Container Platform

By default, only a kubeadm user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

**NOTE**

OpenShift Container Platform user names containing /, :, and % are not supported.

### 6.9.2. Creating the secret

Identity providers use OpenShift Container Platform Secret objects in the openshift-config namespace to contain the client secret, client certificates, and keys.

**Procedure**

- Create a Secret object containing a string by using the following command:

  ```
  $ oc create secret generic <secret_name> --from-literal=clientSecret=<secret> -n openshift-config
  ```
**TIP**

You can alternatively apply the following YAML to create the secret:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openshift-config
type: Opaque
data:
  clientSecret: <base64_encoded_client_secret>
```

- You can define a **Secret** object containing the contents of a file, such as a certificate file, by using the following command:

  ```bash
  $ oc create secret generic <secret_name> --from-file=<path_to_file> -n openshift-config
  ```

### 6.9.3. Creating a config map

Identity providers use OpenShift Container Platform **ConfigMap** objects in the **openshift-config** namespace to contain the certificate authority bundle. These are primarily used to contain certificate bundles needed by the identity provider.

**NOTE**

This procedure is only required for GitHub Enterprise.

**Procedure**

- Define an OpenShift Container Platform **ConfigMap** object containing the certificate authority by using the following command. The certificate authority must be stored in the **ca.crt** key of the **ConfigMap** object.

  ```bash
  $ oc create configmap ca-config-map --from-file=ca.crt=/path/to/ca -n openshift-config
  ```

**TIP**

You can alternatively apply the following YAML to create the config map:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ca-config-map
  namespace: openshift-config
type: Opaque
data:
  ca.crt: |
  <CA_certificate_PEM>
```

### 6.9.4. Sample OpenID Connect CRs
The following custom resources (CRs) show the parameters and acceptable values for an OpenID Connect identity provider.

If you must specify a custom certificate bundle, extra scopes, extra authorization request parameters, or a `userInfo` URL, use the full OpenID Connect CR.

**Standard OpenID Connect CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: oidcidp
    mappingMethod: claim
    type: OpenID
    openID:
      clientID: ...
      clientSecret:
        name: idp-secret
    claims:
      preferredUsername:
        - preferred_username
        name:
        email:
        - email
      issuer: https://www.idp-issuer.com
```

1. This provider name is prefixed to the value of the identity claim to form an identity name. It is also used to build the redirect URL.

2. Controls how mappings are established between this provider’s identities and User objects.

3. The client ID of a client registered with the OpenID provider. The client must be allowed to redirect to `https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>`.

4. Reference to an OpenShift Container Platform Secret object containing the client secret.

5. List of claims to use as the identity. First non-empty claim is used. At least one claim is required. If none of the listed claims have a value, authentication fails. For example, this uses the value of the `sub` claim in the returned `id_token` as the user’s identity.

6. Issuer Identifier described in the OpenID spec. Must use `https` without query or fragment component.

**Full OpenID Connect CR**

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
```
Optional: Reference to an OpenShift Container Platform config map containing the PEM-encoded certificate authority bundle to use in validating server certificates for the configured URL.

Optional list of scopes to request, in addition to the `openid` scope, during the authorization token request.

Optional map of extra parameters to add to the authorization token request.

List of claims to use as the preferred user name when provisioning a user for this identity. First non-empty claim is used.

List of claims to use as the display name. First non-empty claim is used.

List of claims to use as the email address. First non-empty claim is used.

Additional resources

- See Identity provider parameters for information on parameters, such as `mappingMethod`, that are common to all identity providers.

6.9.5. Adding an identity provider to your clusters

After you install your cluster, add an identity provider to it so your users can authenticate.
Prerequisites

- Create an OpenShift Container Platform cluster.
- Create the custom resource (CR) for your identity providers.
- You must be logged in as an administrator.

Procedure

1. Apply the defined CR:

   $ oc apply -f </path/to/CR>

   **NOTE**

   If a CR does not exist, `oc apply` creates a new CR and might trigger the following warning: **Warning: oc apply should be used on resources created by either `oc create --save-config` or `oc apply`.** In this case you can safely ignore this warning.

2. Obtain a token from the OAuth server.
   As long as the `kubeadmin` user has been removed, the `oc login` command provides instructions on how to access a web page where you can retrieve the token.

   You can also access this page from the web console by navigating to (?) Help → Command Line Tools → Copy Login Command

3. Log in to the cluster, passing in the token to authenticate.

   $ oc login --token=<token>

   **NOTE**

   If your OpenID Connect identity provider supports the resource owner password credentials (ROPC) grant flow, you can log in with a user name and password. You might need to take steps to enable the ROPC grant flow for your identity provider.

   After the OIDC identity provider is configured in OpenShift Container Platform, you can log in by using the following command, which prompts for your user name and password:

   $ oc login -u <identity_provider_username> --server= <api_server_url_and_port>

4. Confirm that the user logged in successfully, and display the user name.

   $ oc whoami

6.9.6. Configuring identity providers using the web console
Configure your identity provider (IDP) through the web console instead of the CLI.

**Prerequisites**

- You must be logged in to the web console as a cluster administrator.

**Procedure**

1. Navigate to Administration → Cluster Settings.
2. Under the Global Configuration tab, click OAuth.
3. Under the Identity Providers section, select your identity provider from the Add drop-down menu.

**NOTE**

You can specify multiple IDPs through the web console without overwriting existing IDPs.
CHAPTER 7. USING RBAC TO DEFINE AND APPLY PERMISSIONS

7.1. RBAC OVERVIEW

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

Cluster administrators can use the cluster roles and bindings to control who has various access levels to the OpenShift Container Platform platform itself and all projects.

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:

<table>
<thead>
<tr>
<th>Authorization object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>Sets of permitted verbs on a set of objects. For example, whether a user or service account can create pods.</td>
</tr>
<tr>
<td>Roles</td>
<td>Collections of rules. You can associate, or bind, users and groups to multiple roles.</td>
</tr>
<tr>
<td>Bindings</td>
<td>Associations between users and/or groups with a role.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>RBAC level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster RBAC</td>
<td>Roles and bindings that are applicable across all projects. Cluster roles exist cluster-wide, and cluster role bindings can reference only cluster roles.</td>
</tr>
<tr>
<td>Local RBAC</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

7.1.1. Default cluster roles

OpenShift Container Platform includes a set of default cluster roles that you can bind to users and groups cluster-wide or locally.

**IMPORTANT**

It is not recommended to manually modify the default cluster roles. Modifications to these system roles can prevent a cluster from functioning properly.

<table>
<thead>
<tr>
<th>Default cluster role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>A project manager. If used in a local binding, an <strong>admin</strong> has rights to view any resource in the project and modify any resource in the project except for quota.</td>
</tr>
<tr>
<td>basic-user</td>
<td>A user that can get basic information about projects and users.</td>
</tr>
<tr>
<td>cluster-admin</td>
<td>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.</td>
</tr>
<tr>
<td>cluster-status</td>
<td>A user that can get basic cluster status information.</td>
</tr>
<tr>
<td>edit</td>
<td>A user that can modify most objects in a project but does not have the power to view or modify roles or bindings.</td>
</tr>
<tr>
<td>self-provisioner</td>
<td>A user that can create their own projects.</td>
</tr>
<tr>
<td>view</td>
<td>A user who cannot make any modifications, but can see most objects in a project. They cannot view or modify roles or bindings.</td>
</tr>
</tbody>
</table>

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.
7.1.2. Evaluating authorization

OpenShift Container Platform 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 Container Platform 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 roles and 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.

### 7.1.2.1. Cluster role aggregation

The default admin, edit, view, and cluster-reader cluster roles support **cluster role aggregation**, where the cluster rules for each role are dynamically updated as new rules are created. This feature is relevant only if you extend the Kubernetes API by creating custom resources.

### 7.2. 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:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objects</strong></td>
<td>Pods, services, replication controllers, etc.</td>
</tr>
<tr>
<td><strong>Policies</strong></td>
<td>Rules for which users can or cannot perform actions on objects.</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Quotas for each kind of object that can be limited.</td>
</tr>
<tr>
<td><strong>Service accounts</strong></td>
<td>Service accounts act automatically with designated access to objects in the project.</td>
</tr>
</tbody>
</table>

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 by using the CLI or the web console.

### 7.3. DEFAULT PROJECTS

OpenShift Container Platform 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.

**NOTE**

You cannot assign an SCC to pods created in one of the default namespaces: `default`, `kube-system`, `kube-public`, `openshift-node`, `openshift-infra`, and `openshift`. You cannot use these namespaces for running pods or services.

### 7.4. 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 `cluster-admin` default cluster role bound cluster-wide can perform any action on any resource, including viewing cluster roles and bindings.
Procedure

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

   $ oc describe clusterrole.rbac

Example output

<table>
<thead>
<tr>
<th>Resources</th>
<th>Non-Resource URLs</th>
<th>Resource Names</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>.packages.apps.redhat.com</td>
<td>[]</td>
<td>[]</td>
<td>[* create update patch delete get list watch]</td>
</tr>
<tr>
<td>imagesstreams</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch create get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>imagesstreams.image.openshift.io</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch create get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>secrets</td>
<td>[]</td>
<td>[]</td>
<td>[create delete deletecollection get list patch update watch create delete deletecollection patch update]</td>
</tr>
<tr>
<td>buildconfigs/webhooks</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>buildconfigs</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>buildlogs</td>
<td>[]</td>
<td>[]</td>
<td>[create delete deletecollection get list patch update watch create delete deletecollection]</td>
</tr>
<tr>
<td>deploymentconfigs/scale</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deploymentconfigs</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>imagestreamimages</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>imagestreammappings</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>imagemap</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>processedtemplates</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>routes</td>
<td>[]</td>
<td>[]</td>
<td>[create delete deletecollection get list patch update watch create delete deletecollection]</td>
</tr>
<tr>
<td>templateconfigs</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>templateinstances</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>templates</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deploymentconfigs.apps.openshift.io/scale</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deploymentconfigs.apps.openshift.io</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
<tr>
<td>deletecollection get list patch update watch get list watch</td>
<td>[]</td>
<td>[]</td>
<td>[create delete]</td>
</tr>
</tbody>
</table>
CHAPTER 7. USING RBAC TO DEFINE AND APPLY PERMISSIONS

buildlogs.build.openshift.io

[create delete]

deletemcollection get list patch update watch get list watch

imagestreamimages.image.openshift.io

[create delete]

deletemcollection get list patch update watch get list watch

imagestreammappings.image.openshift.io

[create delete]

deletemcollection get list patch update watch get list watch

imagestreamtags.image.openshift.io

[create delete]

deletemcollection get list patch update watch get list watch

routes.route.openshift.io

[create delete]

serviceaccounts

[create delete]

deletemcollection get list patch update watch impersonate create delete deletemcollection patch update get list watch

imagestreams/secrets

[create delete]

rolebindings

[create delete]

roles

[create delete deletemcollection get list patch update watch]

rolebindings.authorization.openshift.io

[create delete]

roles.authorization.openshift.io

[create delete]

rolebindings.authorization.k8s.io

[create delete]

deletemcollection get list patch update watch

networkpolicies.extensions

[create delete]

deletemcollection patch update create delete deletemcollection get list patch update watch get list watch

networkpolicies.networking.k8s.io

[create delete]

configmaps

[create delete]

deletemcollection patch update get list watch

endpoints

[create delete]

repletioncontrollers/scale

[create delete]

replicationcontrollers

[create delete]

deletemcollection patch update get list watch

services

[create delete]

CHAPTER 7. USING RBAC TO DEFINE AND APPLY PERMISSIONS
### OpenShift Container Platform 4.8 Authentication and authorization

<table>
<thead>
<tr>
<th>Resource</th>
<th>Non-Resource URLs</th>
<th>Resource Names</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imagestreams/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limitranges</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>namespaces/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pods/log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pods/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replicationcontrollers/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resourcequotas/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resourcequotas</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resourcequotausages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rolebindingrestrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deploymentconfigs.apps.openshift.io/log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deploymentconfigs.apps.openshift.io/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>controllerrevisions.apps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rolebindingrestrictions.authorization.openshift.io</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>builds.build.openshift.io/log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imagestreams.image.openshift.io/status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imagestreams/image.openshift.io/layers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imagestreams/image.openshift.io/layers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>builds/details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>builds.build.openshift.io/details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>applications.clusters.resourcequotas.quota.openshift.io</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imageslayers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imageslayers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clusters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>users</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>projects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>projectrequests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>projectrequests</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Name:** basic-user  
**Labels:** <none>  
**Annotations:** openshift.io/description: A user that can get basic information about projects.  
rbac.authorization.kubernetes.io/autoupdate: true

**PolicyRule:**

<table>
<thead>
<tr>
<th>Resources</th>
<th>Non-Resource URLs</th>
<th>Resource Names</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>selfsubjectrulesreviews</td>
<td></td>
<td></td>
<td>[create]</td>
</tr>
<tr>
<td>selfsubjectaccessreviews.authorization.k8s.io</td>
<td></td>
<td></td>
<td>[create]</td>
</tr>
<tr>
<td>selfsubjectaccessreviews.authorization.openshift.io</td>
<td></td>
<td></td>
<td>[create]</td>
</tr>
<tr>
<td>clusterroles.rbac.authorization.k8s.io</td>
<td></td>
<td></td>
<td>[get list watch]</td>
</tr>
<tr>
<td>clusterroles</td>
<td></td>
<td></td>
<td>[get list]</td>
</tr>
<tr>
<td>storageclasses.storage.k8s.io</td>
<td></td>
<td></td>
<td>[get list]</td>
</tr>
<tr>
<td>users</td>
<td></td>
<td></td>
<td>[get]</td>
</tr>
<tr>
<td>users.user.openshift.io</td>
<td></td>
<td></td>
<td>[get]</td>
</tr>
<tr>
<td>projects</td>
<td></td>
<td></td>
<td>[list watch]</td>
</tr>
<tr>
<td>projectrequests</td>
<td></td>
<td></td>
<td>[list]</td>
</tr>
<tr>
<td>projectrequests</td>
<td></td>
<td></td>
<td>[list]</td>
</tr>
</tbody>
</table>

**Name:** cluster-admin  
**Labels:** kubernetes.io/bootstrapping=rbac-defaults 
**Annotations:** rbac.authorization.kubernetes.io/autoupdate: true  
**PolicyRule:**

<table>
<thead>
<tr>
<th>Resources</th>
<th>Non-Resource URLs</th>
<th>Resource Names</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>.</em></td>
<td></td>
<td></td>
<td>[*]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[*]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[*]</td>
</tr>
</tbody>
</table>

...
2. 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
```

**Example output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Labels</th>
<th>Annotations</th>
<th>Role</th>
<th>Subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertmanager-main</td>
<td>&lt;none&gt;</td>
<td></td>
<td>ClusterRole</td>
<td></td>
</tr>
<tr>
<td>basic-users</td>
<td>&lt;none&gt;</td>
<td>rbac.authorization.kubernetes.io/autoupdate: true</td>
<td>ClusterRole</td>
<td></td>
</tr>
<tr>
<td>cloud-credential-operator-rolebinding</td>
<td>&lt;none&gt;</td>
<td></td>
<td>ClusterRole</td>
<td></td>
</tr>
<tr>
<td>cluster-admin</td>
<td>kubernetes.io/bootstrapping=rbac-defaults</td>
<td>rbac.authorization.kubernetes.io/autoupdate: true</td>
<td>ClusterRole</td>
<td></td>
</tr>
</tbody>
</table>

```
7.5. 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 `cluster-admin` default cluster role bound cluster-wide can perform any action on any resource, including viewing 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```

**Example output**

```
Name:         cluster-admins
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name                   Namespace
  ----   ----                   ---------
  Group  system:cluster-admins
  User   system:admin

Name:         cluster-api-manager-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cluster-api-manager-role
Subjects:
  Kind            Name     Namespace
  ----            ----     ---------
  ServiceAccount  default  openshift-machine-api
```
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
7.6. ADDING ROLES TO USERS

You can use the `oc adm` administrator CLI to manage the 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:

   ```bash
   $ 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:

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

   **TIP**

   You can alternatively apply the following YAML to add the role to the user:

   ```yaml
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: admin-0
     namespace: joe
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: admin
   subjects:
     - apiGroup: rbac.authorization.k8s.io
       kind: User
       name: alice
   ```

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

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

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

   ```bash
   $ oc describe rolebinding.rbac -n joe
   ```

   **Example output**

   ```
   Name:         admin
   Labels:       <none>
   Annotations:  <none>
   Role:
     Kind:  ClusterRole
   ```
<table>
<thead>
<tr>
<th>Name</th>
<th>Subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kind  Name               Namespace</td>
</tr>
<tr>
<td></td>
<td>----  ----                ---------</td>
</tr>
<tr>
<td>User</td>
<td>kube:admin</td>
</tr>
<tr>
<td>Name:</td>
<td>admin-0</td>
</tr>
<tr>
<td>Labels:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Annotations:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Role:</td>
<td>ClusterRole</td>
</tr>
<tr>
<td></td>
<td>Name: admin</td>
</tr>
<tr>
<td>Subjects:</td>
<td>Kind  Name               Namespace</td>
</tr>
<tr>
<td></td>
<td>----  ----                ---------</td>
</tr>
<tr>
<td>User</td>
<td>alice</td>
</tr>
</tbody>
</table>

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:
The alice user has been added to the admins RoleBinding.

7.7. CREATING A LOCAL ROLE

You can create a local role for a project and then bind it to a user.

Procedure

1. To create a local role for a project, run the following command:

   ```bash
   $ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
   ```

   In this command, specify:
   - `<name>`, the local role’s name
   - `<verb>`, a comma-separated list of the verbs to apply to the role
   - `<resource>`, the resources that the role applies to
   - `<project>`, the project name

   For example, to create a local role that allows a user to view pods in the blue project, run the following command:

   ```bash
   $ oc create role podview --verb=get --resource=pod -n blue
   ```

2. To bind the new role to a user, run the following command:

   ```bash
   $ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
   ```

7.8. CREATING A CLUSTER ROLE

You can create a cluster role.

Procedure

1. To create a cluster role, run the following command:

   ```bash
   $ oc create clusterrole <name> --verb=<verb> --resource=<resource>
   ```

   In this command, specify:
   - `<name>`, the local role’s name
7.9. 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 7.1. Local role binding operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ oc adm policy who-can &lt;verb&gt; &lt;resource&gt;</code></td>
<td>Indicates which users can perform an action on a resource.</td>
</tr>
<tr>
<td><code>$ oc adm policy add-role-to-user &lt;role&gt; &lt;username&gt;</code></td>
<td>Binds a specified role to specified users in the current project.</td>
</tr>
<tr>
<td><code>$ oc adm policy remove-role-from-user &lt;role&gt; &lt;username&gt;</code></td>
<td>Removes a given role from specified users in the current project.</td>
</tr>
<tr>
<td><code>$ oc adm policy remove-user &lt;username&gt;</code></td>
<td>Removes specified users and all of their roles in the current project.</td>
</tr>
<tr>
<td><code>$ oc adm policy add-role-to-group &lt;role&gt; &lt;groupname&gt;</code></td>
<td>Binds a given role to specified groups in the current project.</td>
</tr>
<tr>
<td><code>$ oc adm policy remove-role-from-group &lt;role&gt; &lt;groupname&gt;</code></td>
<td>Removes a given role from specified groups in the current project.</td>
</tr>
<tr>
<td><code>$ oc adm policy remove-group &lt;groupname&gt;</code></td>
<td>Removes specified groups and all of their roles in the current project.</td>
</tr>
</tbody>
</table>

7.10. CLUSTER ROLE BINDING COMMANDS

You can also manage cluster role bindings using the following operations. The `-n` flag is not used for these operations because cluster role bindings use non-namespaced resources.

Table 7.2. Cluster role binding operations
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ oc adm policy add-cluster-role-to-user &lt;role&gt; &lt;username&gt;</td>
<td>Binds a given role to specified users for all projects in the cluster.</td>
</tr>
<tr>
<td>$ oc adm policy remove-cluster-role-from-user &lt;role&gt; &lt;username&gt;</td>
<td>Removes a given role from specified users for all projects in the cluster.</td>
</tr>
<tr>
<td>$ oc adm policy add-cluster-role-to-group &lt;role&gt; &lt;groupname&gt;</td>
<td>Binds a given role to specified groups for all projects in the cluster.</td>
</tr>
<tr>
<td>$ oc adm policy remove-cluster-role-from-group &lt;role&gt; &lt;groupname&gt;</td>
<td>Removes a given role from specified groups for all projects in the cluster.</td>
</tr>
</tbody>
</table>

### 7.11. CREATING A CLUSTER ADMIN

The `cluster-admin` role is required to perform administrator level tasks on the OpenShift Container Platform cluster, such as modifying cluster resources.

**Prerequisites**

- You must have created a user to define as the cluster admin.

**Procedure**

- Define the user as a cluster admin:

  ```bash
  $ oc adm policy add-cluster-role-to-user cluster-admin <user>
  ```
CHAPTER 8. REMOVING THE KUBEADMIN USER

8.1. THE KUBEADMIN USER

OpenShift Container Platform creates a cluster administrator, **kubeadmin**, after the installation process completes.

This user has the **cluster-admin** role automatically applied and is treated as the root user for the cluster. The password is dynamically generated and unique to your OpenShift Container Platform environment. After installation completes the password is provided in the installation program’s output. For example:

```text
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

8.2. REMOVING THE KUBEADMIN USER

After you define an identity provider and create a new **cluster-admin** user, you can remove the **kubeadmin** to improve cluster security.

**WARNING**

If you follow this procedure before another user is a **cluster-admin**, then OpenShift Container Platform must be reinstalled. It is not possible to undo this command.

Prerequisites

- You must have configured at least one identity provider.
- You must have added the **cluster-admin** role to a user.
- You must be logged in as an administrator.

Procedure

- Remove the **kubeadmin** secrets:

  ```bash
  $ oc delete secrets kubeadmin -n kube-system
  ```
CHAPTER 9. UNDERSTANDING AND CREATING SERVICE ACCOUNTS

9.1. SERVICE ACCOUNTS OVERVIEW

A service account is an OpenShift Container Platform 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 Container Platform 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:

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system:serviceaccounts</td>
<td>Includes all service accounts in the system.</td>
</tr>
<tr>
<td>system:serviceaccounts:&lt;project&gt;</td>
<td>Includes all service accounts in the specified project.</td>
</tr>
</tbody>
</table>

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.

9.2. 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
```

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

$ oc create sa <service_account_name>  

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

Example output

serviceaccount "robot" created

TIP

You can alternatively apply the following YAML to create the service account:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <service_account_name>
  namespace: <current_project>
```

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

$ oc describe sa robot

Example output

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

### 9.3. 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

**TIP**

You can alternatively apply the following YAML to add the role:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: view
  namespace: top-secret
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
  name: robot
  namespace: top-secret
```

- 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 `<service_account_name>`

$ oc policy add-role-to-user <role_name> -z <service_account_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.

**TIP**

You can alternatively apply the following YAML to add the role:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <rolebinding_name>
  namespace: <current_project_name>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: <role_name>
subjects:
- kind: ServiceAccount
  name: <service_account_name>
  namespace: <current_project_name>
```

- 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 **my-project** project:

$ oc policy add-role-to-group view system:serviceaccounts -n my-project

**TIP**

You can alternatively apply the following YAML to add the role:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: view
  namespace: my-project
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: system:serviceaccounts
```

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

$ oc policy add-role-to-group edit system:serviceaccounts:managers -n my-project

**TIP**

You can alternatively apply the following YAML to add the role:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: edit
  namespace: my-project
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: system:serviceaccounts:managers
```
CHAPTER 10. USING SERVICE ACCOUNTS IN APPLICATIONS

10.1. SERVICE ACCOUNTS OVERVIEW

A service account is an OpenShift Container Platform 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 Container Platform 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:

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system:serviceaccounts</td>
<td>Includes all service accounts in the system.</td>
</tr>
<tr>
<td>system:serviceaccounts:&lt;project&gt;</td>
<td>Includes all service accounts in the specified project.</td>
</tr>
</tbody>
</table>

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.

10.2. DEFAULT SERVICE ACCOUNTS

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

10.2.1. Default cluster service accounts

Several infrastructure controllers run using service account credentials. The following service accounts are created in the OpenShift Container Platform infrastructure project (`openshift-infra`) at server start, and given the following roles cluster-wide:
### Service Account Description

<table>
<thead>
<tr>
<th>Service Account</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replication-controller</td>
<td>Assigned the <code>system:replication-controller</code> role</td>
</tr>
<tr>
<td>deployment-controller</td>
<td>Assigned the <code>system:deployment-controller</code> role</td>
</tr>
<tr>
<td>build-controller</td>
<td>Assigned the <code>system:build-controller</code> role. Additionally, the <code>build-controller</code> service account is included in the privileged security context constraint to create privileged build pods.</td>
</tr>
</tbody>
</table>

### 10.2.2. Default project service accounts and roles

Three service accounts are automatically created in each project:

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

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.

### 10.3. 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```

   **Example output**

   ```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

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

Example output

```
serviceaccount "robot" created
```

TIP

You can alternatively apply the following YAML to create the service account:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <service_account_name>
  namespace: <current_project>
```

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

```
$ oc describe sa robot
```

Example output

```
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
```

10.4. USING A SERVICE ACCOUNT’S CREDENTIALS EXTERNALLY

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

To pull an image, the authenticated user must have **get** rights on the requested `imagestreams/layers`. 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:
CHAPTER 10. USING SERVICE ACCOUNTS IN APPLICATIONS

$ oc describe secret <secret_name>

For example:

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

Example output

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...

Example output


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

Example output

system:serviceaccount:top-secret:robot
CHAPTER 11. USING A SERVICE ACCOUNT AS AN OAUTH CLIENT

11.1. 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>:<service_account_namespace>**
- **role:<any_role>:<service_account_namespace>:!**

When using a service account as an OAuth client:

- **client_id** is `system:serviceaccount:<service_account_namespace>:<service_account_name>`.
- **client_secret** can be any of the API tokens for that service account. For example:
  ```bash
  $ oc sa get-token <service_account_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.

11.1.1. 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": "Route",
"name": ...
"group": ...
}
}
```

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](https://example.com), now [https://example.com/custompath](https://example.com/custompath) is considered valid, but [https://example.com](https://example.com) is not. The format for partially supplying override data is as follows:
<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme</td>
<td>&quot;https://&quot;</td>
</tr>
<tr>
<td>Hostname</td>
<td>&quot;//website.com&quot;</td>
</tr>
<tr>
<td>Port</td>
<td>&quot;//:8000&quot;</td>
</tr>
<tr>
<td>Path</td>
<td>&quot;examplepath&quot;</td>
</tr>
</tbody>
</table>

**NOTE**

Specifying a hostname override will replace the hostname 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"
```
12.1. ABOUT SCOPING TOKENS

You can create 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.

12.1.1. 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.

12.1.2. 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.
CHAPTER 13. USING BOUND SERVICE ACCOUNT TOKENS

You can use bound service account tokens, which improves the ability to integrate with cloud provider identity access management (IAM) services, such as AWS IAM.

13.1. ABOUT BOUND SERVICE ACCOUNT TOKENS

You can use bound service account tokens to limit the scope of permissions for a given service account token. These tokens are audience and time-bound. This facilitates the authentication of a service account to an IAM role and the generation of temporary credentials mounted to a pod. You can request bound service account tokens by using volume projection and the TokenRequest API.

13.2. CONFIGURING BOUND SERVICE ACCOUNT TOKENS USING VOLUME PROJECTION

You can configure pods to request bound service account tokens by using volume projection.

Prerequisites

- You have access to the cluster as a user with the `cluster-admin` role.
- You have created a service account. This procedure assumes that the service account is named `build-robot`.

Procedure

1. Optional: Set the service account issuer.
   This step is typically not required if the bound tokens are used only within the cluster.

   ![WARNING]
   
   If you update the `serviceAccountIssuer` field and there are bound tokens already in use, all bound tokens with the previous issuer value will be invalidated. Unless the holder of a bound token has explicit support for a change in issuer, the holder will not request a new bound token until pods have been restarted.

   If necessary, you can use the following command to manually restart all pods in the cluster. Be aware that running this command causes a service interruption, because it deletes every running pod in every namespace. These pods will automatically restart after they are deleted.

   ```bash
   $ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}
   {"n"} {end}'); \
   do oc delete pods --all -n $I; \
   sleep 1; \
   done
   ```
a. Edit the `cluster Authentication` object:

```bash
$ oc edit authentications cluster
```

b. Set the `spec.serviceAccountIssuer` field to the desired service account issuer value:

```yaml
spec:
  serviceAccountIssuer: https://test.default.svc
```

This value should be a URL from which the recipient of a bound token can source the public keys necessary to verify the signature of the token. The default is `https://kubernetes.default.svc`.

2. Configure a pod to use a bound service account token by using volume projection.

a. Create a file called `pod-projected-svc-token.yaml` with the following contents:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    volumeMounts:
    - mountPath: /var/run/secrets/tokens
      name: vault-token
  serviceAccountName: build-robot
  volumes:
  - name: vault-token
    projected:
      sources:
      - serviceAccountToken:
        path: vault-token
        expirationSeconds: 7200
        audience: vault
```

1. A reference to an existing service account.

2. The path relative to the mount point of the file to project the token into.

3. Optionally set the expiration of the service account token, in seconds. The default is 3600 seconds (1 hour) and must be at least 600 seconds (10 minutes). The kubelet will start trying to rotate the token if the token is older than 80 percent of its time to live or if the token is older than 24 hours.

4. Optionally set the intended audience of the token. The recipient of a token should verify that the recipient identity matches the audience claim of the token, and should otherwise reject the token. The audience defaults to the identifier of the API server.

b. Create the pod:
$ oc create -f pod-projected-svc-token.yaml

The kubelet requests and stores the token on behalf of the pod, makes the token available to the pod at a configurable file path, and refreshes the token as it approaches expiration.

3. The application that uses the bound token must handle reloading the token when it rotates. The kubelet rotates the token if it is older than 80 percent of its time to live, or if the token is older than 24 hours.
CHAPTER 14. MANAGING SECURITY CONTEXT CONSTRAINTS

14.1. ABOUT SECURITY CONTEXT CONSTRAINTS

Similar to the way that RBAC resources control user access, administrators can use security context constraints (SCCs) to control permissions for pods. These permissions include actions that a pod can perform and what resources it can access. You can use SCCs to define a set of conditions that a pod must run with to be accepted into the system.

Security context constraints allow an administrator to control:

- Whether a pod can run privileged containers
- The capabilities that a container can request
- The use of host directories as volumes
- The SELinux context of the container
- The container user ID
- The use of host namespaces and networking
- The allocation of an FSGroup that owns the pod volumes
- The configuration of allowable supplemental groups
- Whether a container requires write access to its root file system
- The usage of volume types
- The configuration of allowable seccomp profiles

**IMPORTANT**

Do not set the openshift.io/run-level label on any namespaces in OpenShift Container Platform. This label is for use by internal OpenShift Container Platform components to manage the startup of major API groups, such as the Kubernetes API server and OpenShift API server. If the openshift.io/run-level label is set, no SCCs are applied to pods in that namespace, causing any workloads running in that namespace to be highly privileged.

14.1.1. Default security context constraints

The cluster contains several default security context constraints (SCCs) as described in the table below. Additional SCCs might be installed when you install Operators or other components to OpenShift Container Platform.
IMPORTANT

Do not modify the default SCCs. Customizing the default SCCs can lead to issues when some of the platform pods deploy or OpenShift Container Platform is upgraded. During upgrades between some versions of OpenShift Container Platform, the values of the default SCCs are reset to the default values, which discards all customizations to those SCCs. Instead, create new SCCs as needed.

Table 14.1. Default security context constraints

<table>
<thead>
<tr>
<th>Security context constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyuid</td>
<td>Provides all features of the restricted SCC, but allows users to run with any UID and any GID.</td>
</tr>
<tr>
<td>hostaccess</td>
<td>Allows access to all host namespaces but still requires pods to be run with a UID and SELinux context that are allocated to the namespace.</td>
</tr>
<tr>
<td>hostmount-anyuid</td>
<td>Provides all the features of the restricted SCC, but allows host mounts and running as any UID and any GID on the system.</td>
</tr>
</tbody>
</table>

WARNING

This SCC allows host access to namespaces, file systems, and PIDS. It should only be used by trusted pods. Grant with caution.

WARNING

This SCC allows host file system access as any UID, including UID 0. Grant with caution.
<table>
<thead>
<tr>
<th>Security context constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hostnetwork</strong></td>
<td>Allows using host networking and host ports but still requires pods to be run with a UID and SELinux context that are allocated to the namespace.</td>
</tr>
<tr>
<td></td>
<td><strong>WARNING</strong></td>
</tr>
<tr>
<td></td>
<td>If additional workloads are run on control plane hosts (also known as the master hosts), use caution when providing access to <strong>hostnetwork</strong>. A workload that runs <strong>hostnetwork</strong> on a control plane host is effectively root on the cluster and must be trusted accordingly.</td>
</tr>
<tr>
<td><strong>node-exporter</strong></td>
<td>Used for the Prometheus node exporter.</td>
</tr>
<tr>
<td></td>
<td><strong>WARNING</strong></td>
</tr>
<tr>
<td></td>
<td>This SCC allows host file system access as any UID, including UID 0. Grant with caution.</td>
</tr>
<tr>
<td><strong>nonroot</strong></td>
<td>Provides all features of the <strong>restricted</strong> SCC, but allows users to run with any non-root UID. The user must specify the UID or it must be specified in the manifest of the container runtime.</td>
</tr>
</tbody>
</table>
Table: Security context constraint

<table>
<thead>
<tr>
<th>Security context constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privileged</td>
<td>Allows access to all privileged and host features and the ability to run as any user, any group, any FSGroup, and with any SELinux context.</td>
</tr>
</tbody>
</table>

**WARNING**

This is the most relaxed SCC and should be used only for cluster administration. Grant with caution.

The **privileged** SCC allows:

- Users to run privileged pods
- Pods to mount host directories as volumes
- Pods to run as any user
- Pods to run with any MCS label
- Pods to use the host’s IPC namespace
- Pods to use the host’s PID namespace
- Pods to use any FSGroup
- Pods to use any supplemental group
- Pods to use any seccomp profiles
- Pods to request any capabilities

**NOTE**

Setting `privileged: true` in the pod specification does not select the **privileged** SCC. Setting `privileged: true` in the pod specification matches on the `allowPrivilegedContainer` field of an SCC.
### Security context constraints

Security context constraints (SCCs) are composed of settings and strategies that control the security features a pod has access to. These settings fall into three categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlled by a boolean</td>
<td>Fields of this type default to the most restrictive value. For example, <strong>AllowPrivilegedContainer</strong> is always set to <strong>false</strong> if unspecified.</td>
</tr>
<tr>
<td>Controlled by an allowable set</td>
<td>Fields of this type are checked against the set to ensure their value is allowed.</td>
</tr>
<tr>
<td>Controlled by a strategy</td>
<td>Items that have a strategy to generate a value provide:</td>
</tr>
<tr>
<td></td>
<td>- A mechanism to generate the value, and</td>
</tr>
<tr>
<td></td>
<td>- A mechanism to ensure that a specified value falls into the set of allowable values.</td>
</tr>
</tbody>
</table>

CRIO has the following default list of capabilities that are allowed for each container of a pod:

- **CHOWN**
- **DAC_OVERRIDE**
- **FSETID**
- **FOWNER**
- **SETGID**

---

**The restricted SCC:**

- Ensures that pods cannot run as privileged
- Ensures that pods cannot mount host directory volumes
- Requires that a pod is run as a user in a pre-allocated range of UIDs
- Requires that a pod is run with a pre-allocated MCS label
- Allows pods to use any FSGroup
- Allows pods to use any supplemental group

---

**CHAPTER 14. MANAGING SECURITY CONTEXT CONSTRAINTS**

---

**Security context constraints settings**

Security context constraints (SCCs) are composed of settings and strategies that control the security features a pod has access to. These settings fall into three categories:
- SETUID
- SETPCAP
- NET_BIND_SERVICE
- KILL

The containers use the capabilities from this default list, but pod manifest authors can alter it by requesting additional capabilities or removing some of the default behaviors. Use the `allowedCapabilities`, `defaultAddCapabilities`, and `requiredDropCapabilities` parameters to control such requests from the pods and to dictate which capabilities can be requested, which ones must be added to each container, and which ones must be forbidden.

### 14.1.3. Security context constraints strategies

**RunAsUser**

- **MustRunAs** - Requires a `runAsUser` to be configured. Uses the configured `runAsUser` as the default. Validates against the configured `runAsUser`.
- **MustRunAsRange** - Requires minimum and maximum values to be defined if not using pre-allocated values. Uses the minimum as the default. Validates against the entire allowable range.
- **MustRunAsNonRoot** - Requires that the pod be submitted with a non-zero `runAsUser` or have the `USER` directive defined in the image. No default provided.
- **RunAsAny** - No default provided. Allows any `runAsUser` to be specified.

**SELinuxContext**

- **MustRunAs** - Requires `seLinuxOptions` to be configured if not using pre-allocated values. Uses `seLinuxOptions` as the default. Validates against `seLinuxOptions`.
- **RunAsAny** - No default provided. Allows any `seLinuxOptions` to be specified.

**SupplementalGroups**

- **MustRunAs** - Requires at least one range to be specified if not using pre-allocated values. Uses the minimum value of the first range as the default. Validates against all ranges.
- **RunAsAny** - No default provided. Allows any `supplementalGroups` to be specified.

**FSGroup**

- **MustRunAs** - Requires at least one range to be specified if not using pre-allocated values. Uses the minimum value of the first range as the default. Validates against the first ID in the first range.
- **RunAsAny** - No default provided. Allows any `fsGroup` ID to be specified.

### 14.1.4. Controlling volumes

The usage of specific volume types can be controlled by setting the `volumes` field of the SCC. The allowable values of this field correspond to the volume sources that are defined when creating a volume:
- awsElasticBlockStore
- azureDisk
- azureFile
- cephFS
- cinder
- configMap
- downwardAPI
- emptyDir
- fc
- flexVolume
- flocker
- gcePersistentDisk
- gitRepo
- glusterfs
- hostPath
- iscsi
- nfs
- persistentVolumeClaim
- photonPersistentDisk
- portworxVolume
- projected
- quobyte
- rbd
- scaleIO
- secret
- storageos
- vsphereVolume

- * (A special value to allow the use of all volume types.)

- none (A special value to disallow the use of all volumes types. Exists only for backwards compatibility.)
The recommended minimum set of allowed volumes for new SCCs are `configMap`, `downwardAPI`, `emptyDir`, `persistentVolumeClaim`, `secret`, and `projected`.

**NOTE**

This list of allowable volume types is not exhaustive because new types are added with each release of OpenShift Container Platform.

**NOTE**

For backwards compatibility, the usage of `allowHostDirVolumePlugin` overrides settings in the `volumes` field. For example, if `allowHostDirVolumePlugin` is set to false but allowed in the `volumes` field, then the `hostPath` value will be removed from `volumes`.

### 14.1.5. Admission control

Admission control with SCCs allows for control over the creation of resources based on the capabilities granted to a user.

In terms of the SCCs, this means that an admission controller can inspect the user information made available in the context to retrieve an appropriate set of SCCs. Doing so ensures the pod is authorized to make requests about its operating environment or to generate a set of constraints to apply to the pod.

The set of SCCs that admission uses to authorize a pod are determined by the user identity and groups that the user belongs to. Additionally, if the pod specifies a service account, the set of allowable SCCs includes any constraints accessible to the service account.

Admission uses the following approach to create the final security context for the pod:

1. Retrieve all SCCs available for use.
2. Generate field values for security context settings that were not specified on the request.
3. Validate the final settings against the available constraints.

If a matching set of constraints is found, then the pod is accepted. If the request cannot be matched to an SCC, the pod is rejected.

A pod must validate every field against the SCC. The following are examples for just two of the fields that must be validated:

**NOTE**

These examples are in the context of a strategy using the pre-allocated values.

**An FSGroup SCC strategy of MustRunAs**

If the pod defines a `fsGroup` ID, then that ID must equal the default `fsGroup` ID. Otherwise, the pod is not validated by that SCC and the next SCC is evaluated.

If the `SecurityContextConstraints.fsGroup` field has value `RunAsAny` and the pod specification omits the `Pod.spec.securityContext.fsGroup`, then this field is considered valid. Note that it is possible that during validation, other SCC settings will reject other pod fields and thus cause the pod to fail.
A SupplementalGroups SCC strategy of MustRunAs

If the pod specification defines one or more supplementalGroups IDs, then the pod’s IDs must equal one of the IDs in the namespace’s openshift.io/sa.scc.supplemental-groups annotation. Otherwise, the pod is not validated by that SCC and the next SCC is evaluated.

If the SecurityContextConstraints.supplementalGroups field has value RunAsAny and the pod specification omits the Pod.spec.securityContext.supplementalGroups, then this field is considered valid. Note that it is possible that during validation, other SCC settings will reject other pod fields and thus cause the pod to fail.

14.1.6. Security context constraints prioritization

Security context constraints (SCCs) have a priority field that affects the ordering when attempting to validate a request by the admission controller. A higher priority SCC is moved to the front of the set when sorting. When the complete set of available SCCs are determined they are ordered by:

1. Highest priority first, nil is considered a 0 priority
2. If priorities are equal, the SCCs will be sorted from most restrictive to least restrictive
3. If both priorities and restrictions are equal the SCCs will be sorted by name

By default, the anyuid SCC granted to cluster administrators is given priority in their SCC set. This allows cluster administrators to run pods as any user by without specifying a RunAsUser on the pod’s SecurityContext. The administrator may still specify a RunAsUser if they wish.

14.2. ABOUT PRE-ALLOCATED SECURITY CONTEXT CONSTRAINTS VALUES

The admission controller is aware of certain conditions in the security context constraints (SCCs) that trigger it to look up pre-allocated values from a namespace and populate the SCC before processing the pod. Each SCC strategy is evaluated independently of other strategies, with the pre-allocated values, where allowed, for each policy aggregated with pod specification values to make the final values for the various IDs defined in the running pod.

The following SCCs cause the admission controller to look for pre-allocated values when no ranges are defined in the pod specification:

1. A RunAsUser strategy of MustRunAsRange with no minimum or maximum set. Admission looks for the openshift.io/sa.scc.uid-range annotation to populate range fields.
2. An SELinuxContext strategy of MustRunAs with no level set. Admission looks for the openshift.io/sa.scc.mcs annotation to populate the level.
3. A FSGroup strategy of MustRunAs. Admission looks for the openshift.io/sa.scc.supplemental-groups annotation.

During the generation phase, the security context provider uses default values for any parameter values that are not specifically set in the pod. Default values are based on the selected strategy:

1. RunAsAny and MustRunAsNonRoot strategies do not provide default values. If the pod needs a parameter value, such as a group ID, you must define the value in the pod specification.
2. **MustRunAs** (single value) strategies provide a default value that is always used. For example, for group IDs, even if the pod specification defines its own ID value, the namespace’s default parameter value also appears in the pod’s groups.

3. **MustRunAsRange** and **MustRunAs** (range-based) strategies provide the minimum value of the range. As with a single value **MustRunAs** strategy, the namespace’s default parameter value appears in the running pod. If a range-based strategy is configurable with multiple ranges, it provides the minimum value of the first configured range.

**NOTE**

**FSGroup** and **SupplementalGroups** strategies fall back to the `openshift.io/sa.scc.uid-range` annotation if the `openshift.io/sa.scc.supplemental-groups` annotation does not exist on the namespace. If neither exists, the SCC is not created.

**NOTE**

By default, the annotation-based **FSGroup** strategy configures itself with a single range based on the minimum value for the annotation. For example, if your annotation reads `1/3`, the **FSGroup** strategy configures itself with a minimum and maximum value of `1`. If you want to allow more groups to be accepted for the **FSGroup** field, you can configure a custom SCC that does not use the annotation.

**NOTE**

The `openshift.io/sa.scc.supplemental-groups` annotation accepts a comma-delimited list of blocks in the format of `<start>/<length` or `<start>`-<end>. The `openshift.io/sa.scc.uid-range` annotation accepts only a single block.

### 14.3. EXAMPLE SECURITY CONTEXT CONSTRAINTS

The following examples show the security context constraints (SCC) format and annotations:

**Annotated privileged SCC**

```yaml
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: 1
- '*'
apiVersion: security.openshift.io/v1
defaultAddCapabilities: []
fsGroup: 3
type: RunAsAny
groups: 4
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: privileged allows access to all privileged and host
```
features and the ability to run as any user, any group, any fsGroup, and with any SELinux context. WARNING: this is the most relaxed SCC and should be used only for cluster administration. Grant with caution.'

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 6
      type: RunAsAny
    seLinuxContext: 7
      type: RunAsAny
    seccompProfiles:
      - '*'
    supplementalGroups: 8
      type: RunAsAny
    users:
      - system: serviceaccount:default:registry
      - system: serviceaccount:default:router
      - system: serviceaccount:openshift-infra:build-controller
volumes:
  - '*'
```

1. A list of capabilities that a pod can request. An empty list means that none of capabilities can be requested while the special symbol * allows any capabilities.

2. A list of additional capabilities that are added to any pod.

3. The **FSGroup** strategy, which dictates the allowable values for the security context.

4. The groups that can access this SCC.

5. A list of capabilities that are be dropped from a pod.

6. The **runAsUser** strategy type, which dictates the allowable values for the Security Context.

7. The **seLinuxContext** strategy type, which dictates the allowable values for the Security Context.

8. The **supplementalGroups** strategy, which dictates the allowable supplemental groups for the Security Context.

9. The users who can access this SCC.

The **users** and **groups** fields on the SCC control which users can access the SCC. By default, cluster administrators, nodes, and the build controller are granted access to the privileged SCC. All authenticated users are granted access to the restricted SCC.

**Without explicit runAsUser setting**
When a container or pod does not request a user ID under which it should be run, the effective UID depends on the SCC that emits this pod. Because restricted SCC is granted to all authenticated users by default, it will be available to all users and service accounts and used in most cases. The restricted SCC uses MustRunAsRange strategy for constraining and defaulting the possible values of the securityContext.runAsUser field. The admission plug-in will look for the openshift.io/sa.scc.uid-range annotation on the current project to populate range fields, as it does not provide this range. In the end, a container will have runAsUser equal to the first value of the range that is hard to predict because every project has different ranges.

With explicit runAsUser setting

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
  containers:
    - name: sec-ctx-demo
      image: gcr.io/google-samples/node-hello:1.0
```

A container or pod that requests a specific user ID will be accepted by OpenShift Container Platform only when a service account or a user is granted access to a SCC that allows such a user ID. The SCC can allow arbitrary IDs, an ID that falls into a range, or the exact user ID specific to the request. This configuration is valid for SELinux, fsGroup, and Supplemental Groups.

### 14.4. Creating Security Context Constraints

You can create security context constraints (SCCs) by using the OpenShift CLI (oc).

**Prerequisites**

- Install the OpenShift CLI (oc).
- Log in to the cluster as a user with the cluster-admin role.

**Procedure**

1. Define the SCC in a YAML file named `scc_admin.yaml`:

   ```yaml
   kind: SecurityContextConstraints
   apiVersion: security.openshift.io/v1
   metadata:
   ```
name: scc-admin
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
  - my-admin-user
groups:
  - my-admin-group

Optionally, you can specify drop capabilities for an SCC by setting the `requiredDropCapabilities` field with the desired values. Any specified capabilities are dropped from the container. For example, to create an SCC with the `KILL`, `MKNOD`, and `SYS_CHROOT` required drop capabilities, add the following to the SCC object:

```
requiredDropCapabilities:
  - KILL
  - MKNOD
  - SYS_CHROOT
```

CRI-O supports the same list of capability values that are found in the Docker documentation.

2. Create the SCC by passing in the file:

```
$ oc create -f scc_admin.yaml
```

Example output

```
securitycontextconstraints "scc-admin" created
```

Verification

- Verify that the SCC was created:

```
$ oc get scc scc-admin
```

Example output

```
NAME PRIV CAPS SELINUX RUNASUSER FSGROUP SUPGROUP PRIORITY READONLYROOTFS VOLUMES
scc-admin true [] RunAsAny RunAsAny RunAsAny RunAsAny <none> false [awsElasticBlockStore azureDisk azureFile cephFS cinder configMap downwardAPI emptyDir fc flexVolume folder gcePersistentDisk gitRepo glusterfs iscsi nfs persistentVolumeClaim photonPersistentDisk quobyte rbd secret vsphere]
```

14.5. ROLE-BASED ACCESS TO SECURITY CONTEXT CONSTRAINTS
You can specify SCCs as resources that are handled by RBAC. This allows you to scope access to your SCCs to a certain project or to the entire cluster. Assigning users, groups, or service accounts directly to an SCC retains cluster-wide scope.

**NOTE**

You cannot assign a SCC to pods created in one of the default namespaces: `default`, `kube-system`, `kube-public`, `openshift-node`, `openshift-infra`, `openshift`. These namespaces should not be used for running pods or services.

To include access to SCCs for your role, specify the `scc` resource when creating a role.

```
$ oc create role <role-name> --verb=use --resource=scc --resource-name=<scc-name> -n <namespace>
```

This results in the following role definition:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: role-name
  namespace: namespace
...
rules:
  - apiGroups:
    - security.openshift.io
    resourceNames:
      - scc-name
    resources:
      - securitycontextconstraints
    verbs:
      - use
```

1. The role's name.
2. Namespace of the defined role. Defaults to `default` if not specified.
3. The API group that includes the `SecurityContextConstraints` resource. Automatically defined when `scc` is specified as a resource.
4. An example name for an SCC you want to have access.
5. Name of the resource group that allows users to specify SCC names in the `resourceNames` field.
6. A list of verbs to apply to the role.

A local or cluster role with such a rule allows the subjects that are bound to it with a role binding or a cluster role binding to use the user-defined SCC called `scc-name`. 

OpenShift Container Platform 4.8 Authentication and authorization
Because RBAC is designed to prevent escalation, even project administrators are unable to grant access to an SCC. By default, they are not allowed to use the verb **use** on SCC resources, including the **restricted** SCC.

### 14.6. REFERENCE OF SECURITY CONTEXT CONSTRAINTS COMMANDS

You can manage security context constraints (SCCs) in your instance as normal API objects using the OpenShift CLI (**oc**).

**NOTE**

You must have **cluster-admin** privileges to manage SCCs.

#### 14.6.1. Listing security context constraints

To get a current list of SCCs:

```bash
$ oc get scc
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PRIV</th>
<th>CAPS</th>
<th>SELINUX</th>
<th>RUNASUSER</th>
<th>FSGROUP</th>
<th>SUPGROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyuid</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hostaccess</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>MustRunAsRange</td>
<td>RunAsAny</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hostmount-anyuid</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hostnetwork</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>MustRunAsRange</td>
<td>MustRunAs</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node-exporter</td>
<td>false</td>
<td>[]</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nonroot</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>MustRunAsNonRoot</td>
<td>RunAsAny</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>privileged</td>
<td>true</td>
<td>[*]</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>RunAsAny</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restricted</td>
<td>false</td>
<td>[]</td>
<td>MustRunAs</td>
<td>MustRunAsRange</td>
<td>MustRunAs</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 14.6.2. Examining security context constraints

You can view information about a particular SCC, including which users, service accounts, and groups the SCC is applied to.

For example, to examine the **restricted** SCC:

```bash
$ oc describe scc restricted
```
Example output

| Name:     restricted |
| Priority:  <none>   |
| Access:    |
| Users:     <none>|
| Groups:    system:authenticated |
| Settings:  |
| Allow Privileged:  false  |
| Default Add Capabilities:  <none>   |
| Required Drop Capabilities:  KILL,MKNOD,SYS_CHROOT,SETUID,SETGID |
| Allowed Capabilities:  <none> |
| Allowed Seccomp Profiles:  <none>   |
| Allowed Volume Types:   configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret |
| Allow Host Network:   false |
| Allow Host Ports:    false |
| Allow Host PID:      false |
| Allow Host IPC:      false |
| Read Only Root Filesystem:  false |
| Run As User Strategy: MustRunAsRange |
| UID:    <none> |
| UID Range Min:      <none> |
| UID Range Max:      <none> |
| SELinux Context Strategy: MustRunAs |
| User:      <none>  |
| Role:      <none>   |
| Type:      <none>   |
| Level:     <none>   |
| FSGroup Strategy: MustRunAs |
| Ranges:    <none> |
| Supplemental Groups Strategy: RunAsAny |
| Ranges:    <none> |

1 Lists which users and service accounts the SCC is applied to.
2 Lists which groups the SCC is applied to.

NOTE

To preserve customized SCCs during upgrades, do not edit settings on the default SCCs.

14.6.3. Deleting security context constraints

To delete an SCC:

```
$ oc delete scc <scc_name>
```

NOTE

If you delete a default SCC, it will regenerate when you restart the cluster.
14.6.4. Updating security context constraints

To update an existing SCC:

```bash
$ oc edit scc <scc_name>
```

**NOTE**

To preserve customized SCCs during upgrades, do not edit settings on the default SCCs.
CHAPTER 15. IMPERSONATING THE SYSTEM:ADMIN USER

15.1. API IMPERSONATION

You can configure a request to the OpenShift Container Platform API to act as though it originated from another user. For more information, see User impersonation in the Kubernetes documentation.

15.2. IMPERSONATING THE SYSTEM:ADMIN USER

You can grant a user permission to impersonate system:admin, which grants them cluster administrator permissions.

Procedure

- To grant a user permission to impersonate system:admin, run the following command:

  $ oc create clusterrolebinding <any_valid_name> --clusterrole=sudoer --user=<username>

  TIP

  You can alternatively apply the following YAML to grant permission to impersonate system:admin:

  ```yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: <any_valid_name>
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: sudoer
  subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: <username>
  ```

15.3. IMPERSONATING THE SYSTEM:ADMIN GROUP

When a system:admin user is granted cluster administration permissions through a group, you must include the --as=<user> --as-group=<group1> --as-group=<group2> parameters in the command to impersonate the associated groups.

Procedure

- To grant a user permission to impersonate a system:admin by impersonating the associated cluster administration groups, run the following command:

  $ oc create clusterrolebinding <any_valid_name> --clusterrole=sudoer --as=<user>
  --as-group=<group1> --as-group=<group2>
CHAPTER 16. SYNCING LDAP GROUPS

As an administrator, you can use groups to manage users, change their permissions, and enhance collaboration. Your organization may have already created user groups and stored them in an LDAP server. OpenShift Container Platform can sync those LDAP records with internal OpenShift Container Platform records, enabling you to manage your groups in one place. OpenShift Container Platform currently supports group sync with LDAP servers using three common schemas for defining group membership: RFC 2307, Active Directory, and augmented Active Directory.

For more information on configuring LDAP, see Configuring an LDAP identity provider.

NOTE

You must have cluster-admin privileges to sync groups.

16.1. ABOUT CONFIGURING LDAP SYNC

Before you can run LDAP sync, you need a sync configuration file. This file contains the following LDAP client configuration details:

- Configuration for connecting to your LDAP server.
- Sync configuration options that are dependent on the schema used in your LDAP server.
- An administrator-defined list of name mappings that maps OpenShift Container Platform group names to groups in your LDAP server.

The format of the configuration file depends upon the schema you are using: RFC 2307, Active Directory, or augmented Active Directory.

LDAP client configuration

The LDAP client configuration section of the configuration defines the connections to your LDAP server.

The LDAP client configuration section of the configuration defines the connections to your LDAP server.

LDAP client configuration

```yaml
url: ldap://10.0.0.0:389
bindDN: cn=admin,dc=example,dc=com
bindPassword: password
insecure: false
ca: my-ldap-ca-bundle.crt
```

1. The connection protocol, IP address of the LDAP server hosting your database, and the port to connect to, formatted as **scheme://host:port**.

2. Optional distinguished name (DN) to use as the Bind DN. OpenShift Container Platform uses this if elevated privilege is required to retrieve entries for the sync operation.

3. Optional password to use to bind. OpenShift Container Platform uses this if elevated privilege is necessary to retrieve entries for the sync operation. This value may also be provided in an environment variable, external file, or encrypted file.
When false, secure LDAP (ldaps://) URLs connect using TLS, and insecure LDAP (ldap://) URLs are upgraded to TLS. When true, no TLS connection is made to the server and you cannot use the certificate bundle to use for validating server certificates for the configured URL. If empty, OpenShift Container Platform uses system-trusted roots. This only applies if insecure is set to false.

LDAP query definition

Sync configurations consist of LDAP query definitions for the entries that are required for synchronization. The specific definition of an LDAP query depends on the schema used to store membership information in the LDAP server.

```
baseDN: ou=users,dc=example,dc=com
scope: sub
derefAliases: never
timeout: 0
filter: (objectClass=person)
pageSize: 0
```

1. The distinguished name (DN) of the branch of the directory where all searches will start from. It is required that you specify the top of your directory tree, but you can also specify a subtree in the directory.

2. The scope of the search. Valid values are base, one, or sub. If this is left undefined, then a scope of sub is assumed. Descriptions of the scope options can be found in the table below.

3. The behavior of the search with respect to aliases in the LDAP tree. Valid values are never, search, base, or always. If this is left undefined, then the default is to always dereference aliases. Descriptions of the dereferencing behaviors can be found in the table below.

4. The time limit allowed for the search by the client, in seconds. A value of 0 imposes no client-side limit.

5. A valid LDAP search filter. If this is left undefined, then the default is (objectClass=*).

6. The optional maximum size of response pages from the server, measured in LDAP entries. If set to 0, no size restrictions will be made on pages of responses. Setting paging sizes is necessary when queries return more entries than the client or server allow by default.

Table 16.1. LDAP search scope options

<table>
<thead>
<tr>
<th>LDAP search scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>Only consider the object specified by the base DN given for the query.</td>
</tr>
<tr>
<td>one</td>
<td>Consider all of the objects on the same level in the tree as the base DN for the query.</td>
</tr>
<tr>
<td>sub</td>
<td>Consider the entire subtree rooted at the base DN given for the query.</td>
</tr>
</tbody>
</table>
### Table 16.2. LDAP dereferencing behaviors

<table>
<thead>
<tr>
<th>Dereferencing behavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>never</td>
<td>Never dereference any aliases found in the LDAP tree.</td>
</tr>
<tr>
<td>search</td>
<td>Only dereference aliases found while searching.</td>
</tr>
<tr>
<td>base</td>
<td>Only dereference aliases while finding the base object.</td>
</tr>
<tr>
<td>always</td>
<td>Always dereference all aliases found in the LDAP tree.</td>
</tr>
</tbody>
</table>

### User-defined name mapping

A user-defined name mapping explicitly maps the names of OpenShift Container Platform groups to unique identifiers that find groups on your LDAP server. The mapping uses normal YAML syntax. A user-defined mapping can contain an entry for every group in your LDAP server or only a subset of those groups. If there are groups on the LDAP server that do not have a user-defined name mapping, the default behavior during sync is to use the attribute specified as the OpenShift Container Platform group’s name.

#### User-defined name mapping

```yaml
  groupUIDNameMapping:
    "cn=group1,ou=groups,dc=example,dc=com": firstgroup
    "cn=group2,ou=groups,dc=example,dc=com": secondgroup
    "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

### 16.1.1. About the RFC 2307 configuration file

The RFC 2307 schema requires you to provide an LDAP query definition for both user and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships:

#### NOTE

If using user-defined name mappings, your configuration file will differ.

### LDAP sync configuration that uses RFC 2307 schema: rfc2307_config.yaml

```yaml
  kind: LDAPSyncConfig
  apiVersion: v1
  url: ldap://LDAP_SERVICE_IP:389
  insecure: false
  rfc2307:
    groupsQuery:
```
The IP address and host of the LDAP server where this group's record is stored.

When **false**, secure LDAP (ldaps://) URLs connect using TLS, and insecure LDAP (ldap://) URLs are upgraded to TLS. When **true**, no TLS connection is made to the server and you cannot use ldaps:// URL schemes.

The attribute that uniquely identifies a group on the LDAP server. You cannot specify **groupsQuery** filters when using DN for **groupUIDAttribute**. For fine-grained filtering, use the whitelist / blacklist method.

The attribute to use as the name of the group.

The attribute on the group that stores the membership information.

The attribute that uniquely identifies a user on the LDAP server. You cannot specify **usersQuery** filters when using DN for userUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

The attribute to use as the name of the user in the OpenShift Container Platform group record.

### 16.1.2. About the Active Directory configuration file

The Active Directory schema requires you to provide an LDAP query definition for user entries, as well as the attributes to represent them within in the internal OpenShift Container Platform group records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, but define the name of the group by the name of the group on the LDAP server. The following configuration file creates these relationships:

**LDAP sync configuration that uses Active Directory schema: active_directory_config.yaml**

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldaps://LDAP_SERVICE_IP:389
activeDirectory:
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
    userUIDAttribute: dn
    userNameAttributes: [ mail ]
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfRangeErrors: false
```
The attribute to use as the name of the user in the OpenShift Container Platform group record.

The attribute on the user that stores the membership information.

16.1.3. About the augmented Active Directory configuration file

The augmented Active Directory schema requires you to provide an LDAP query definition for both user entries and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform group records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships.

LDAP sync configuration that uses augmented Active Directory schema:
*augmented_active_directory_config.yaml*

```yaml
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=person)
    pageSize: 0
    userNameAttributes: [ mail ]
    groupMembershipAttributes: [ memberOf ]
```

1 The attribute that uniquely identifies a group on the LDAP server. You cannot specify `groupsQuery` filters when using DN for groupUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

2 The attribute to use as the name of the group.

3 The attribute to use as the name of the user in the OpenShift Container Platform group record.
The attribute on the user that stores the membership information.

16.2. RUNNING LDAP SYNC

Once you have created a sync configuration file, you can begin to sync. OpenShift Container Platform allows administrators to perform a number of different sync types with the same server.

16.2.1. Syncing the LDAP server with OpenShift Container Platform

You can sync all groups from the LDAP server with OpenShift Container Platform.

Prerequisites

- Create a sync configuration file.

Procedure

- To sync all groups from the LDAP server with OpenShift Container Platform:

  ```bash
  $ oc adm groups sync --sync-config=config.yaml --confirm
  ```

  **NOTE**
  
  By default, all group synchronization operations are dry-run, so you must set the `-confirm` flag on the `oc adm groups sync` command to make changes to OpenShift Container Platform group records.

16.2.2. Syncing OpenShift Container Platform groups with the LDAP server

You can sync all groups already in OpenShift Container Platform that correspond to groups in the LDAP server specified in the configuration file.

Prerequisites

- Create a sync configuration file.

Procedure

- To sync OpenShift Container Platform groups with the LDAP server:

  ```bash
  $ oc adm groups sync --type=openshift --sync-config=config.yaml --confirm
  ```

  **NOTE**
  
  By default, all group synchronization operations are dry-run, so you must set the `-confirm` flag on the `oc adm groups sync` command to make changes to OpenShift Container Platform group records.

16.2.3. Syncing subgroups from the LDAP server with OpenShift Container Platform

OpenShift Container Platform 4.8 Authentication and authorization
You can sync a subset of LDAP groups with OpenShift Container Platform using whitelist files, blacklist files, or both.

**NOTE**

You can use any combination of blacklist files, whitelist files, or whitelist literals. Whitelist and blacklist files must contain one unique group identifier per line, and you can include whitelist literals directly in the command itself. These guidelines apply to groups found on LDAP servers as well as groups already present in OpenShift Container Platform.

**Prerequisites**

- Create a sync configuration file.

**Procedure**

- To sync a subset of LDAP groups with OpenShift Container Platform, use any of the following commands:

  ```
  $ oc adm groups sync --whitelist=<whitelist_file> \ 
  --sync-config=config.yaml \ 
  --confirm
  
  $ oc adm groups sync --blacklist=<blacklist_file> \ 
  --sync-config=config.yaml \ 
  --confirm
  
  $ oc adm groups sync <group_unique_identifier> \ 
  --sync-config=config.yaml \ 
  --confirm
  
  $ oc adm groups sync <group_unique_identifier> \ 
  --whitelist=<whitelist_file> \ 
  --blacklist=<blacklist_file> \ 
  --sync-config=config.yaml \ 
  --confirm
  
  $ oc adm groups sync --type=openshift \ 
  --whitelist=<whitelist_file> \ 
  --sync-config=config.yaml \ 
  --confirm
  ```

**NOTE**

By default, all group synchronization operations are dry-run, so you must set the `-confirm` flag on the `oc adm groups sync` command to make changes to OpenShift Container Platform group records.

### 16.3. RUNNING A GROUP PRUNING JOB
An administrator can also choose to remove groups from OpenShift Container Platform records if the records on the LDAP server that created them are no longer present. The prune job will accept the same sync configuration file and whitelists or blacklists as used for the sync job.

For example:

```bash
$ oc adm prune groups --sync-config=/path/to/ldap-sync-config.yaml --confirm
$ oc adm prune groups --whitelist=/path/to/whitelist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm
$ oc adm prune groups --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm
```

### 16.4. LDAP GROUP SYNC EXAMPLES

This section contains examples for the RFC 2307, Active Directory, and augmented Active Directory schemas.

**NOTE**

These examples assume that all users are direct members of their respective groups. Specifically, no groups have other groups as members. See the Nested Membership Sync Example for information on how to sync nested groups.

#### 16.4.1. Syncing groups using the RFC 2307 schema

For the RFC 2307 schema, the following examples synchronize a group named `admins` that has two members: Jane and Jim. The examples explain:

- How the group and users are added to the LDAP server.
- What the resulting group record in OpenShift Container Platform will be after synchronization.

**NOTE**

These examples assume that all users are direct members of their respective groups. Specifically, no groups have other groups as members. See the Nested Membership Sync Example for information on how to sync nested groups.

In the RFC 2307 schema, both users (Jane and Jim) and groups exist on the LDAP server as first-class entries, and group membership is stored in attributes on the group. The following snippet of `ldif` defines the users and group for this schema:

**LDAP entries that use RFC 2307 schema: rfc2307.ldif**

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users
dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
```
The group is a first-class entry in the LDAP server.

Members of a group are listed with an identifying reference as attributes on the group.

Prerequisites

- Create the configuration file.

Procedure

- Run the sync with the `rfc2307_config.yaml` file:

  ```
  $ oc adm groups sync --sync-config=rfc2307_config.yaml --confirm
  ```

  OpenShift Container Platform creates the following group record as a result of the above sync operation:

**OpenShift Container Platform group created by using the `rfc2307_config.yaml` file**

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp: name: admins
```
The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.

The unique identifier for the group on the LDAP server.

The IP address and host of the LDAP server where this group’s record is stored.

The name of the group as specified by the sync file.

The users that are members of the group, named as specified by the sync file.

16.4.2. Syncing groups using the RFC2307 schema with user-defined name mappings

When syncing groups with user-defined name mappings, the configuration file changes to contain these mappings as shown below.

LDAP sync configuration that uses RFC 2307 schema with user-defined name mappings:

```
---
kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators
rfc2307:
groupsQuery:
  baseDN: "ou=groups,dc=example,dc=com"
  scope: sub
  derefAliases: never
  pageSize: 0
groupUIDAttribute: dn
groupNameAttributes: [ cn ]
groupMembershipAttributes: [ member ]
usersQuery:
  baseDN: "ou=users,dc=example,dc=com"
  scope: sub
  derefAliases: never
  pageSize: 0
userUIDAttribute: dn
userNameAttributes: [ mail ]
tolerateMemberNotFoundErrors: false
tolerateMemberOutOfScopeErrors: false
---
```

1. The user-defined name mapping.

2. The unique identifier attribute that is used for the keys in the user-defined name mapping. You cannot specify `groupsQuery` filters when using DN for groupUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.
The attribute to name OpenShift Container Platform groups with if their unique identifier is not in the user-defined name mapping.

The attribute that uniquely identifies a user on the LDAP server. You cannot specify `usersQuery` filters when using DN for `userUIDAttribute`. For fine-grained filtering, use the whitelist / blacklist method.

Prerequisites

- Create the configuration file.

Procedure

- Run the sync with the `rfc2307_config_user_defined.yaml` file:

  ```bash
  $ oc adm groups sync --sync-config=rfc2307_config_user_defined.yaml --confirm
  
  OpenShift Container Platform creates the following group record as a result of the above sync operation:

  **OpenShift Container Platform group created by using the rfc2307_config_user_defined.yaml file**

  ```yaml
  apiVersion: user.openshift.io/v1
  kind: Group
  metadata:
    annotations:
      openshift.io/ldap.uid: cn=admin,ou=groups,dc=example,dc=com
      openshift.io/ldap.url: LDAP_SERVER_IP:389
    creationTimestamp:
    name: Administrators
  users:
    - jane.smith@example.com
    - jim.adams@example.com
  ```

  1 The name of the group as specified by the user-defined name mapping.

### 16.4.3. Syncing groups using RFC 2307 with user-defined error tolerances

By default, if the groups being synced contain members whose entries are outside of the scope defined in the member query, the group sync fails with an error:

```
Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with dn="<user-dn>" would search outside of the base dn specified (dn="<base-dn>")",
```

This often indicates a misconfigured `baseDN` in the `usersQuery` field. However, in cases where the `baseDN` intentionally does not contain some of the members of the group, setting `tolerateMemberOutOfScopeErrors: true` allows the group sync to continue. Out of scope members will be ignored.
Similarly, when the group sync process fails to locate a member for a group, it fails outright with errors:

```
Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn="<user-dn>" refers to a non-existent entry".
Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn="<user-dn>" and filter "<filter>" did not return any results".
```

This often indicates a misconfigured `usersQuery` field. However, in cases where the group contains member entries that are known to be missing, setting `tolerateMemberNotFoundErrors: true` allows the group sync to continue. Problematic members will be ignored.

**WARNING**

Enabling error tolerances for the LDAP group sync causes the sync process to ignore problematic member entries. If the LDAP group sync is not configured correctly, this could result in synced OpenShift Container Platform groups missing members.

LDAP entries that use RFC 2307 schema with problematic group membership:
`rfc2307_problematic_users.ldif`

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users
dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups
dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
```
A member that does not exist on the LDAP server.

A member that may exist, but is not under the `baseDN` in the user query for the sync job.

To tolerate the errors in the above example, the following additions to your sync configuration file must be made:

**LDAP sync configuration that uses RFC 2307 schema tolerating errors:**

*rfc2307_config_tolerating.yaml*

```yaml
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
rfc2307:
groupsQuery:
  baseDN: "ou=groups,dc=example,dc=com"
  scope: sub
  derefAliases: never
 groupeUIDAttribute: dn
 groupNameAttributes: [ cn ]
groupMembershipAttributes: [ member ]
usersQuery:
  baseDN: "ou=users,dc=example,dc=com"
  scope: sub
  derefAliases: never
  userUIDAttribute: dn
  userNameAttributes: [ mail ]
tolerateMemberNotFoundErrors: true
 toleratMemberOutOfScopeErrors: true
```

1. The attribute that uniquely identifies a user on the LDAP server. You cannot specify `usersQuery` filters when using DN for `userUIDAttribute`. For fine-grained filtering, use the whitelist / blacklist method.

2. When `true`, the sync job tolerates groups for which some members were not found, and members whose LDAP entries are not found are ignored. The default behavior for the sync job is to fail if a member of a group is not found.

3. When `true`, the sync job tolerates groups for which some members are outside the user scope given in the `usersQuery` base DN, and members outside the member query scope are ignored. The default behavior for the sync job is to fail if a member of a group is out of scope.

**Prerequisites**

- Create the configuration file.

**Procedure**
Run the sync with the `rfc2307_config_tolerating.yaml` file:

```
$ oc adm groups sync --sync-config=rfc2307_config_tolerating.yaml --confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

**OpenShift Container Platform group created by using the `rfc2307_config.yaml` file**

```yaml
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp: null
name: admins
users: 1
- jane.smith@example.com
- jim.adams@example.com
```

The users that are members of the group, as specified by the sync file. Members for which lookup encountered tolerated errors are absent.

### 16.4.4. Syncing groups using the Active Directory schema

In the Active Directory schema, both users (Jane and Jim) exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of `ldif` defines the users and group for this schema:

**LDAP entries that use Active Directory schema: active_directory.ldif**

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: admins


dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
```
The user’s group memberships are listed as attributes on the user, and the group does not exist as an entry on the server. The `memberOf` attribute does not have to be a literal attribute on the user; in some LDAP servers, it is created during search and returned to the client, but not committed to the database.

Prerequisites

- Create the configuration file.

Procedure

- Run the sync with the `active_directory_config.yaml` file:

  ```bash
  $ oc adm groups sync --sync-config=active_directory_config.yaml --confirm
  ```

  OpenShift Container Platform creates the following group record as a result of the above sync operation:

  **OpenShift Container Platform group created by using the `active_directory_config.yaml` file**

  ```yaml
  apiVersion: user.openshift.io/v1
  kind: Group
  metadata:
    annotations:
      openshift.io/ldap.uid: admins
      openshift.io/ldap.url: LDAP_SERVER_IP:389
    creationTimestamp: name:admins
  users:
  - jane.smith@example.com
  - jim.adams@example.com
  ```

  1. The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
  2. The unique identifier for the group on the LDAP server.
  3. The IP address and host of the LDAP server where this group’s record is stored.
  4. The name of the group as listed in the LDAP server.
  5. The users that are members of the group, named as specified by the sync file.

16.4.5. Syncing groups using the augmented Active Directory schema
In the augmented Active Directory schema, both users (Jane and Jim) and groups exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of ldif defines the users and group for this schema:

**LDAP entries that use augmented Active Directory schema: augmented_active_directory.ldif**

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberof: cn=admins,ou=groups,dc=example,dc=com


dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberof: cn=admins,ou=groups,dc=example,dc=com


dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
memberof: cn=Jane,ou=users,dc=example,dc=com
cn=Jim,ou=users,dc=example,dc=com
```

**Notes**

1. The user’s group memberships are listed as attributes on the user.
2. The group is a first-class entry on the LDAP server.

**Prerequisites**

- Create the configuration file.

**Procedure**
Run the sync with the `augmented_active_directory_config.yaml` file:

```
$ oc adm groups sync --sync-config=augmented_active_directory_config.yaml --confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

**OpenShift Container Platform group created by using the augmented_active_directory_config.yaml file**

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
creationTimestamp:
  name: admins
users:
  - jane.smith@example.com
  - jim.adams@example.com
```

1. The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
2. The unique identifier for the group on the LDAP server.
3. The IP address and host of the LDAP server where this group’s record is stored.
4. The name of the group as specified by the sync file.
5. The users that are members of the group, named as specified by the sync file.

### 16.4.5.1. LDAP nested membership sync example

Groups in OpenShift Container Platform do not nest. The LDAP server must flatten group membership before the data can be consumed. Microsoft’s Active Directory Server supports this feature via the `LDAP_MATCHING_RULE_IN_CHAIN` rule, which has the OID `1.2.840.113556.1.4.1941`. Furthermore, only explicitly whitelisted groups can be synced when using this matching rule.

This section has an example for the augmented Active Directory schema, which synchronizes a group named `admins` that has one user `Jane` and one group `otheradmins` as members. The `otheradmins` group has one user member: `Jim`. This example explains:

- How the group and users are added to the LDAP server.
- What the LDAP sync configuration file looks like.
- What the resulting group record in OpenShift Container Platform will be after synchronization.

In the augmented Active Directory schema, both users (`Jane` and `Jim`) and groups exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user or the group. The following snippet of `ldif` defines the users and groups for this schema:
LDAP entries that use augmented Active Directory schema with nested members: 
*augmented_active_directory_nested.ldif*

```ldif
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com


dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com


dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: group
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
memberOf: cn=admins,ou=groups,dc=example,dc=com


dn: cn=otheradmins,ou=groups,dc=example,dc=com
objectClass: group
cn: otheradmins
owner: cn=admin,dc=example,dc=com
description: Other System Administrators
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com
```

1. The user’s and group’s memberships are listed as attributes on the object.
2. The groups are first-class entries on the LDAP server.
3. The otheradmins group is a member of the admins group.
When syncing nested groups with Active Directory, you must provide an LDAP query definition for both user entries and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform group records. Furthermore, certain changes are required in this configuration:

- The `oc adm groups sync` command must explicitly whitelist groups.
- The user’s `groupMembershipAttributes` must include "memberOf:1.2.840.113556.1.4.1941:" to comply with the `LDAP_MATCHING_RULE_IN_CHAIN` rule.
- The `groupUIDAttribute` must be set to `dn`.
- The `groupsQuery`:
  - Must not set `filter`.
  - Must set a valid `derefAliases`.
  - Should not set `baseDN` as that value is ignored.
  - Should not set `scope` as that value is ignored.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships:

**LDAP sync configuration that uses augmented Active Directory schema with nested members: augmented_active_directory_config_nested.yaml**

```yaml
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
groupsQuery: 1
derefAliases: never
pageSize: 0
groupUIDAttribute: dn 2
groupNameAttributes: [ cn ] 3
usersQuery:
  baseDN: "ou=users,dc=example,dc=com"
  scope: sub
derefAliases: never
  filter: (objectclass=person)
  pageSize: 0
userNameAttributes: [ mail ] 4
groupMembershipAttributes: [ "memberOf:1.2.840.113556.1.4.1941:" ] 5
```

1. **groupsQuery** filters cannot be specified. The **groupsQuery** base DN and scope values are ignored. **groupsQuery** must set a valid **derefAliases**.

2. The attribute that uniquely identifies a group on the LDAP server. It must be set to **dn**.

3. The attribute to use as the name of the group.
The attribute to use as the name of the user in the OpenShift Container Platform group record. `mail` or `sAMAccountName` are preferred choices in most installations.

The attribute on the user that stores the membership information. Note the use of `LDAP_MATCHING_RULE_IN_CHAIN`.

Prerequisites
- Create the configuration file.

Procedure
- Run the sync with the `augmented_active_directory_config_nested.yaml` file:

```bash
$ oc adm groups sync \
  'cn=admins,ou=groups,dc=example,dc=com' \
  --sync-config=augmented_active_directory_config_nested.yaml \
  --confirm

NOTE
You must explicitly whitelist the `cn=admins,ou=groups,dc=example,dc=com` group.

OpenShift Container Platform creates the following group record as a result of the above sync operation:

**OpenShift Container Platform group created by using the `augmented_active_directory_config_nested.yaml` file**

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp: name: admins
users:
- jane.smith@example.com
- jim.adams@example.com
```

1. The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
2. The unique identifier for the group on the LDAP server.
3. The IP address and host of the LDAP server where this group's record is stored.
4. The name of the group as specified by the sync file.
The users that are members of the group, named as specified by the sync file. Note that members of nested groups are included since the group membership was flattened by the

16.5. LDAP SYNC CONFIGURATION SPECIFICATION

The object specification for the configuration file is below. Note that the different schema objects have different fields. For example, v1.ActiveDirectoryConfig has no `groupsQuery` field whereas v1.RFC2307Config and v1.AugmentedActiveDirectoryConfig both do.

**IMPORTANT**

There is no support for binary attributes. All attribute data coming from the LDAP server must be in the format of a UTF-8 encoded string. For example, never use a binary attribute, such as `objectGUID`, as an ID attribute. You must use string attributes, such as `sAMAccountName` or `userPrincipalName`, instead.

16.5.1. v1.LDAPSyncConfig

`LDAPSyncConfig` holds the necessary configuration options to define an LDAP group sync.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>kind</strong></td>
<td>String value representing the REST resource this object represents. Servers may infer this from</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-">https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>conventions.md#types-kinds</td>
<td></td>
</tr>
<tr>
<td><strong>apiVersion</strong></td>
<td>Defines the versioned schema of this representation of an object. Servers should convert</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>recognized schemas to the latest internal value, and may reject unrecognized values. More info:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-">https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>conventions.md#resources</td>
<td></td>
</tr>
<tr>
<td><strong>url</strong></td>
<td>Host is the scheme, host and port of the LDAP server to connect to:</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>scheme://host:port</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>bindDN</strong> Optional DN to bind to the LDAP server with.</td>
<td>string</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>bindPassword</td>
<td>Optional password to bind with during the search phase.</td>
<td>v1.StringSource</td>
</tr>
<tr>
<td>insecure</td>
<td>If true, indicates the connection should not use TLS. If false, ldaps:// URLs connect using TLS, and ldap:// URLs are upgraded to a TLS connection using StartTLS as specified in <a href="https://tools.ietf.org/html/rfc2830">https://tools.ietf.org/html/rfc2830</a>. If you set insecure to true, you cannot use ldaps:// URL schemes.</td>
<td>boolean</td>
</tr>
<tr>
<td>ca</td>
<td>Optional trusted certificate authority bundle to use when making requests to the server. If empty, the default system roots are used.</td>
<td>string</td>
</tr>
<tr>
<td>groupUIDNameMapping</td>
<td>Optional direct mapping of LDAP group UIDs to OpenShift Container Platform group names.</td>
<td>object</td>
</tr>
<tr>
<td>rfc2307</td>
<td>Holds the configuration for extracting data from an LDAP server set up in a fashion similar to RFC2307: first-class group and user entries, with group membership determined by a multi-valued attribute on the group entry listing its members.</td>
<td>v1.RFC2307Config</td>
</tr>
<tr>
<td>activeDirectory</td>
<td>Holds the configuration for extracting data from an LDAP server set up in a fashion similar to that used in Active Directory: first-class user entries, with group membership determined by a multi-valued attribute on members listing groups they are a member of.</td>
<td>v1.ActiveDirectoryConfig</td>
</tr>
</tbody>
</table>
augmentedActiveDirectory
Holds the configuration for extracting data from an LDAP server set up in a fashion similar to that used in Active Directory as described above, with one addition: first-class group entries exist and are used to hold metadata but not group membership.

16.5.2. v1.StringSource
StringSource allows specifying a string inline, or externally via environment variable or file. When it contains only a string value, it marshals to a simple JSON string.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the cleartext value, or an encrypted value if <strong>keyFile</strong> is specified.</td>
<td>string</td>
</tr>
<tr>
<td>env</td>
<td>Specifies an environment variable containing the cleartext value, or an encrypted value if the <strong>keyFile</strong> is specified.</td>
<td>string</td>
</tr>
<tr>
<td>file</td>
<td>References a file containing the cleartext value, or an encrypted value if a <strong>keyFile</strong> is specified.</td>
<td>string</td>
</tr>
<tr>
<td>keyFile</td>
<td>References a file containing the key to use to decrypt the value.</td>
<td>string</td>
</tr>
</tbody>
</table>

16.5.3. v1.LDAPQuery
LDAPQuery holds the options necessary to build an LDAP query.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseDN</td>
<td>DN of the branch of the directory where all searches should start from.</td>
<td>string</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>scope</td>
<td>The optional scope of the search. Can be <strong>base</strong>: only the base object, <strong>one</strong>: all objects on the base level, <strong>sub</strong>: the entire subtree. Defaults to <strong>sub</strong> if not set.</td>
<td>string</td>
</tr>
<tr>
<td>derefAliases</td>
<td>The optional behavior of the search with regards to aliases. Can be <strong>never</strong>: never dereference aliases, <strong>search</strong>: only dereference in searching, <strong>base</strong>: only dereference in finding the base object, <strong>always</strong>: always dereference. Defaults to <strong>always</strong> if not set.</td>
<td>string</td>
</tr>
<tr>
<td>timeout</td>
<td>Holds the limit of time in seconds that any request to the server can remain outstanding before the wait for a response is given up. If this is 0, no client-side limit is imposed.</td>
<td>integer</td>
</tr>
<tr>
<td>filter</td>
<td>A valid LDAP search filter that retrieves all relevant entries from the LDAP server with the base DN.</td>
<td>string</td>
</tr>
<tr>
<td>pageSize</td>
<td>Maximum preferred page size, measured in LDAP entries. A page size of 0 means no paging will be done.</td>
<td>integer</td>
</tr>
</tbody>
</table>

**16.5.4. v1.RFC2307Config**

**RFC2307Config** holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the RFC2307 schema.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupsQuery</td>
<td>Holds the template for an LDAP query that returns group entries.</td>
<td>v1.LDAPQuery</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>groupUIDAttribute</td>
<td>Defines which attribute on an LDAP group entry will be interpreted as its unique identifier. <em>(ldapGroupUID)</em></td>
<td>string</td>
</tr>
<tr>
<td>groupNameAttributes</td>
<td>Defines which attributes on an LDAP group entry will be interpreted as its name to use for an OpenShift Container Platform group.</td>
<td>string array</td>
</tr>
<tr>
<td>groupMembershipAttributes</td>
<td>Defines which attributes on an LDAP group entry will be interpreted as its members. The values contained in those attributes must be queryable by your UserUIDAttribute.</td>
<td>string array</td>
</tr>
<tr>
<td>usersQuery</td>
<td>Holds the template for an LDAP query that returns user entries.</td>
<td>v1.LDAPQuery</td>
</tr>
<tr>
<td>userUIDAttribute</td>
<td>Defines which attribute on an LDAP user entry will be interpreted as its unique identifier. It must correspond to values that will be found from the GroupMembershipAttributes.</td>
<td>string</td>
</tr>
<tr>
<td>userNameAttributes</td>
<td>Defines which attributes on an LDAP user entry will be used, in order, as its OpenShift Container Platform user name. The first attribute with a non-empty value is used. This should match your PreferredUsername setting for your LDAPPasswordIdentityProvider. The attribute to use as the name of the user in the OpenShift Container Platform group record. mail or sAMAccountName are preferred choices in most installations.</td>
<td>string array</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>tolerateMemberNotFoundErrors</td>
<td>Determines the behavior of the LDAP sync job when missing user entries are encountered. If true, an LDAP query for users that does not find any will be tolerated and an only and error will be logged. If false, the LDAP sync job will fail if a query for users doesn't find any. The default value is false. Misconfigured LDAP sync jobs with this flag set to true can cause group membership to be removed, so it is recommended to use this flag with caution.</td>
<td>boolean</td>
</tr>
<tr>
<td>tolerateMemberOutOfScopeErrors</td>
<td>Determines the behavior of the LDAP sync job when out-of-scope user entries are encountered. If true, an LDAP query for a user that falls outside of the base DN given for the all user query will be tolerated and only an error will be logged. If false, the LDAP sync job will fail if a user query would search outside of the base DN specified by the all user query. Misconfigured LDAP sync jobs with this flag set to true can result in groups missing users, so it is recommended to use this flag with caution.</td>
<td>boolean</td>
</tr>
</tbody>
</table>

### 16.5.5. v1.ActiveDirectoryConfig

**ActiveDirectoryConfig** holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the Active Directory schema.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>usersQuery</td>
<td>Holds the template for an LDAP query that returns user entries.</td>
<td>v1.LDAPQuery</td>
</tr>
</tbody>
</table>
### 16.5.6. v1.AugmentedActiveDirectoryConfig

**AugmentedActiveDirectoryConfig** holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the augmented Active Directory schema.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usersQuery</strong></td>
<td>Holds the template for an LDAP query that returns user entries.</td>
<td>v1.LDAPQuery</td>
</tr>
<tr>
<td><strong>userNameAttributes</strong></td>
<td>Defines which attributes on an LDAP user entry will be interpreted as its OpenShift Container Platform user name. The attribute to use as the name of the user in the OpenShift Container Platform group record. <em>mail</em> or <em>sAMAccountName</em> are preferred choices in most installations.</td>
<td>string array</td>
</tr>
<tr>
<td><strong>groupMembershipAttributes</strong></td>
<td>Defines which attributes on an LDAP user entry will be interpreted as the groups it is a member of.</td>
<td>string array</td>
</tr>
<tr>
<td><strong>groupsQuery</strong></td>
<td>Holds the template for an LDAP query that returns group entries.</td>
<td>v1.LDAPQuery</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>groupUIDAttribute</td>
<td>Defines which attribute on an LDAP group entry will be interpreted as its unique identifier. (\texttt{ldapGroupUID})</td>
<td>string</td>
</tr>
<tr>
<td>groupNameAttributes</td>
<td>Defines which attributes on an LDAP group entry will be interpreted as its name to use for an OpenShift Container Platform group.</td>
<td>string array</td>
</tr>
</tbody>
</table>
CHAPTER 17. CREATING AND USING CONFIG MAPS

The following sections define config maps and how to create and use them.

17.1. UNDERSTANDING CONFIGMAPS

Many applications require configuration using some combination of configuration files, command line arguments, and environment variables. In OpenShift Container Platform, these configuration artifacts are decoupled from image content to keep containerized applications portable.

The ConfigMap object provides mechanisms to inject containers with configuration data while keeping containers agnostic of OpenShift Container Platform. A config map can be used to store fine-grained information like individual properties or coarse-grained information like entire configuration files or JSON blobs.

The ConfigMap API object holds key-value pairs of configuration data that can be consumed in pods or used to store configuration data for system components such as controllers. For example:

**ConfigMap Object Definition**

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  example.property.1: hello
  example.property.2: world
  example.property.file: |
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw
```

1. Contains the configuration data.
2. Points to a file that contains non-UTF8 data, for example, a binary Java keystore file. Enter the file data in Base 64.

**NOTE**

You can use the binaryData field when you create a config map from a binary file, such as an image.

Configuration data can be consumed in pods in a variety of ways. A config map can be used to:

- Populate environment variable values in containers
- Set command-line arguments in a container
- Populate configuration files in a volume
Users and system components can store configuration data in a config map.

A config map is similar to a secret, but designed to more conveniently support working with strings that do not contain sensitive information.

**Config map restrictions**
A config map must be created before its contents can be consumed in pods.

Controllers can be written to tolerate missing configuration data. Consult individual components configured by using config maps on a case-by-case basis.

**ConfigMap objects reside in a project.**

They can only be referenced by pods in the same project.

**The Kubelet only supports the use of a config map for pods it gets from the API server.**

This includes any pods created by using the CLI, or indirectly from a replication controller. It does not include pods created by using the OpenShift Container Platform node’s `--manifest-url` flag, its `--config` flag, or its REST API because these are not common ways to create pods.

### 17.2. CREATING A CONFIG MAP IN THE OPENSHIFT CONTAINER PLATFORM WEB CONSOLE

You can create a config map in the OpenShift Container Platform web console.

**Procedure**

- To create a config map as a cluster administrator:
  1. In the Administrator perspective, select **Workloads → Config Maps**.
  2. At the top right side of the page, select **Create Config Map**.
  3. Enter the contents of your config map.
  4. Select **Create**.

- To create a config map as a developer:
  1. In the Developer perspective, select **Config Maps**.
  2. At the top right side of the page, select **Create Config Map**.
  3. Enter the contents of your config map.
  4. Select **Create**.

### 17.3. CREATING A CONFIG MAP

You can use the following command to create a config map from directories, specific files, or literal values.

**Procedure**

- Create a config map:
17.3.1. Creating a config map from a directory

You can create a config map from a directory. This method allows you to use multiple files within a directory to create a config map.

Procedure

The following example procedure outlines how to create a config map from a directory.

1. Start with a directory with some files that already contain the data with which you want to populate a config map:

   $ ls example-files

Example output

   game.properties
   ui.properties

   $ cat example-files/game.properties

Example output

   enemies=aliens
   lives=3
   enemies.cheat=true
   enemies.cheat.level=noGoodRotten
   secret.code.passphrase=UUDDLRLRBABAS
   secret.code.allowed=true
   secret.code.lives=30

   $ cat example-files/ui.properties

Example output

   color.good=purple
   color.bad=yellow
   allow.textmode=true
   how.nice.to.look=fairlyNice

2. Create a ConfigMap holding the content of each file in this directory by entering the following command:

   $ oc create configmap game-config
     --from-file=example-files/

When the --from-file option points to a directory, each file directly in that directory is used to populate a key in the ConfigMap, where the name of the key is the file name, and the value of the key is the content of the file.
For example, the previous command creates the following **ConfigMap**:

```bash
$ oc describe configmaps game-config
```

**Example output**

```
Name:           game-config
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data

game.properties: 158 bytes
ui.properties:   83 bytes
```

You can see that the two keys in the map are created from the file names in the directory specified in the command. Because the content of those keys might be large, the output of `oc describe` only shows the names of the keys and their sizes.

3. Enter the **oc get** command for the object with the `-o` option to see the values of the keys:

```bash
$ oc get configmaps game-config -o yaml
```

**Example output**

```yaml
apiVersion: v1
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"
  selflink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

### 17.3.2. Creating a ConfigMap from a file

You can create a config map from a file.
Procedure

The following example procedure outlines how to create a config map from a file.

NOTE

If you create a config map from a file, you can include files containing non-UTF8 data that are placed in this field without corrupting the non-UTF8 data. OpenShift Container Platform detects binary files and transparently encodes the file as MIME. On the server, the MIME payload is decoded and stored without corrupting the data.

You can pass the `--from-file` option multiple times to the CLI. The following example yields equivalent results to the creating from directories example.

1. Create a ConfigMap specifying a specific file:

   ```
   $ oc create configmap game-config-2 \
      --from-file=example-files/game.properties \
      --from-file=example-files/ui.properties
   ```

2. Verify the results:

   ```
   $ oc get configmaps game-config-2 -o yaml
   ```

   **Example output**

   ```yaml
   apiVersion: v1
data:
   game.properties: |
      enemies=aliens
      lives=3
      enemies.cheat=true
      enemies.cheat.level=noGoodRotten
      secret.code.passphrase=UUDDLRLRBABAS
      secret.code.allowed=true
      secret.code.lives=30
   ui.properties: |
      color.good=purple
      color.bad=yellow
      allow.textmode=true
      how.nice.to.look=fairlyNice
   kind: ConfigMap
metadata:
.creationTimestamp: 2016-02-18T18:52:05Z
name: game-config-2
namespace: default
resourceVersion: "516"
selflink: /api/v1/namespaces/default/configmaps/game-config-2
uid: b4952dc3-d670-11e5-8cd0-68f728db1985
   ```

You can specify the key to set in a ConfigMap for content imported from a file. This can be set by passing a `key=value` expression to the `--from-file` option. For example:

1. Create a ConfigMap specifying a key-value pair:

   ```
   -
   ```
17.3.3. Creating a config map from literal values

You can supply literal values for a config map.

Procedure

The `--from-literal` option takes a **key=value** syntax that allows literal values to be supplied directly on the command line.

1. Create a ConfigMap specifying a literal value:

   ```
   $ oc create configmap special-config \
   --from-literal=special.how=very \
   --from-literal=special.type=charm
   ```

2. Verify the results:

   ```
   $ oc get configmaps special-config -o yaml
   ```

   **Example output**

   ```yaml
   apiVersion: v1
   data:
     special.how: very
     special.type: charm
   kind: ConfigMap
   metadata:
     creationTimestamp: 2016-02-18T18:54:22Z
     name: special-config
     namespace: default
     resourceVersion: "530"
     selflink: /api/v1/namespaces/default/configmaps/special-config
     uid: 05f8da22-d671-11e5-8c0-68f728db1985
   ```

This is the key that you set in the preceding step.
17.4. USE CASES: CONSUMING CONFIGMAPS IN PODS

The following sections describe some use cases when consuming ConfigMap objects in pods.

17.4.1. Populating environment variables in containers by using config maps

Config maps can be used to populate individual environment variables in containers or to populate environment variables in containers from all keys that form valid environment variable names.

As an example, consider the following config map:

**ConfigMap with two environment variables**

```yaml
apiVersion: v1
type: ConfigMap
metadata:
  name: special-config
namespace: default
data:
  special.how: very
  special.type: charm
```

1. Name of the **ConfigMap**.
2. The project in which the **ConfigMap** resides. Config maps can only be referenced by pods in the same project.
3, 4. Environment variables to inject.

**ConfigMap with one environment variable**

```yaml
apiVersion: v1
type: ConfigMap
metadata:
  name: env-config
namespace: default
data:
  log_level: INFO
```

1. Name of the **ConfigMap**.
Environment variable to inject.

Procedure

- You can consume the keys of this ConfigMap in a pod using configMapKeyRef sections.

Sample Pod specification configured to inject specific environment variables

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: ["/bin/sh", "-c", "env"]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
              optional: true
      envFrom:
        - configMapRef:
            name: env-config
  restartPolicy: Never
```

1. Stanza to pull the specified environment variables from a ConfigMap.
2. Name of a pod environment variable that you are injecting a key’s value into.
3. Name of the ConfigMap to pull specific environment variables from.
4. Environment variable to pull from the ConfigMap.
5. Makes the environment variable optional. As optional, the pod will be started even if the specified ConfigMap and keys do not exist.
6. Stanza to pull all environment variables from a ConfigMap.
7. Name of the ConfigMap to pull all environment variables from.

When this pod is run, the pod logs will include the following output:
17.4.2. Setting command-line arguments for container commands with ConfigMaps

A ConfigMap can also be used to set the value of the commands or arguments in a container. This is accomplished by using the Kubernetes substitution syntax $(VAR_NAME). Consider the following ConfigMaps:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
namespace: default
data:
  special.how: very
  special.type: charm
```

**Procedure**

- To inject values into a command in a container, you must consume the keys you want to use as environment variables, as in the consuming ConfigMaps in environment variables use case. Then you can refer to them in a container’s command using the $(VAR_NAME) syntax.

**Sample Pod specification configured to inject specific environment variables**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: ["/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)"

  env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: special.how
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: special.type

restartPolicy: Never
```
1. Inject the values into a command in a container using the keys you want to use as environment variables.

When this Pod is run, the output from the echo command run in the test-container container is as follows:

```
very charm
```

17.4.3. Injecting content into a volume by using config maps

You can inject content into a volume by using config maps.

**Example ConfigMap**

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

**Procedure**

You have a couple different options for injecting content into a volume by using config maps.

- The most basic way to inject content into a volume by using a config map is to populate the volume with files where the key is the file name and the content of the file is the value of the key:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox
    command: ["/bin/sh", "cat", "/etc/config/special.how"
  volumeMounts:
  - name: config-volume
    mountPath: /etc/config
  volumes:
  - name: config-volume
cfgMap:
    name: special-config
  restartPolicy: Never
```

1. File containing key.

When this pod is run, the output of the cat command will be:
You can also control the paths within the volume where ConfigMap keys are projected:

```yaml
apiVersion: v1
class: Pod
metadata:
  name: dapi-test-pod
class: spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [/bin/sh, "cat", "/etc/config/path/to/special-key"]
  volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
          - key: special.how
            path: path/to/special-key
        restartPolicy: Never
```

Path to ConfigMap key.

When this pod is run, the output of the cat command will be:

very
CHAPTER 18. MANAGING CLOUD PROVIDER CREDENTIALS

18.1. ABOUT THE CLOUD CREDENTIAL OPERATOR

The Cloud Credential Operator (CCO) manages cloud provider credentials as custom resource definitions (CRDs). The CCO syncs on CredentialsRequest custom resources (CRs) to allow OpenShift Container Platform components to request cloud provider credentials with the specific permissions that are required for the cluster to run.

By setting different values for the credentialsMode parameter in the install-config.yaml file, the CCO can be configured to operate in several different modes. If no mode is specified, or the credentialsMode parameter is set to an empty string (""), the CCO operates in its default mode.

18.1.1. Modes

By setting different values for the credentialsMode parameter in the install-config.yaml file, the CCO can be configured to operate in mint, passthrough, or manual mode. These options provide transparency and flexibility in how the CCO uses cloud credentials to process CredentialsRequest CRs in the cluster, and allow the CCO to be configured to suit the security requirements of your organization. Not all CCO modes are supported for all cloud providers.

- **Mint**: In mint mode, the CCO uses the provided admin-level cloud credential to create new credentials for components in the cluster with only the specific permissions that are required.

  NOTE

  Mint mode is the default and recommended best practice setting for the CCO to use.

- **Passthrough**: In passthrough mode, the CCO passes the provided cloud credential to the components that request cloud credentials.

- **Manual**: In manual mode, a user manages cloud credentials instead of the CCO.
  - **Manual with AWS STS**: In manual mode, you can configure an AWS cluster to use Amazon Web Services Secure Token Service (AWS STS). With this configuration, the CCO uses temporary credentials for different components.

Table 18.1. CCO mode support matrix

<table>
<thead>
<tr>
<th>Cloud provider</th>
<th>Mint</th>
<th>Passthrough</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Web Services (AWS)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Azure</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Google Cloud Platform (GCP)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Red Hat OpenStack Platform (RHOSP)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Red Hat Virtualization (RHV)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
18.1.2. Default behavior

For platforms on which multiple modes are supported (AWS, Azure, and GCP), when the CCO operates in its default mode, it checks the provided credentials dynamically to determine for which mode they are sufficient to process `CredentialsRequest` CRs.

By default, the CCO determines whether the credentials are sufficient for mint mode, which is the preferred mode of operation, and uses those credentials to create appropriate credentials for components in the cluster. If the credentials are not sufficient for mint mode, it determines whether they are sufficient for passthrough mode. If the credentials are not sufficient for passthrough mode, the CCO cannot adequately process `CredentialsRequest` CRs.

**NOTE**
The CCO cannot verify whether Azure credentials are sufficient for passthrough mode. If Azure credentials are insufficient for mint mode, the CCO operates with the assumption that the credentials are sufficient for passthrough mode.

If the provided credentials are determined to be insufficient during installation, the installation fails. For AWS, the installer fails early in the process and indicates which required permissions are missing. Other providers might not provide specific information about the cause of the error until errors are encountered.

If the credentials are changed after a successful installation and the CCO determines that the new credentials are insufficient, the CCO puts conditions on any new `CredentialsRequest` CRs to indicate that it cannot process them because of the insufficient credentials.

To resolve insufficient credentials issues, provide a credential with sufficient permissions. If an error occurred during installation, try installing again. For issues with new `CredentialsRequest` CRs, wait for the CCO to try to process the CR again. As an alternative, you can manually create IAM for AWS, Azure, and GCP.

18.1.3. Additional resources

- Red Hat Operators reference page for the Cloud Credential Operator

18.2. USING MINT MODE

Mint mode is supported for Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

Mint mode is the default and recommended best practice setting for the Cloud Credential Operator (CCO) to use on the platforms for which it is supported. In this mode, the CCO uses the provided administrator-level cloud credential to create new credentials for components in the cluster with only the specific permissions that are required.

If the credential is not removed after installation, it is stored and used by the CCO to process `CredentialsRequest` CRs for components in the cluster and create new credentials for each with only the specific permissions that are required. The continuous reconciliation of cloud credentials in mint
mode allows actions that require additional credentials or permissions, such as upgrading, to proceed.

If the requirement that mint mode stores the administrator-level credential in the cluster `kube-system` namespace does not suit the security requirements of your organization, see *Alternatives to storing administrator-level secrets in the kube-system project* for AWS, Azure, or GCP.

### 18.2.1. Mint mode permissions requirements

When using the CCO in mint mode, ensure that the credential you provide meets the requirements of the cloud on which you are running or installing OpenShift Container Platform. If the provided credentials are not sufficient for mint mode, the CCO cannot create an IAM user.

#### 18.2.1.1. Amazon Web Services (AWS) permissions

The credential you provide for mint mode in AWS must have the following permissions:

- `iam:CreateAccessKey`
- `iam:CreateUser`
- `iam:DeleteAccessKey`
- `iam:DeleteUser`
- `iam:DeleteUserPolicy`
- `iam:GetUser`
- `iam:GetUserPolicy`
- `iam:ListAccessKeys`
- `iam:PutUserPolicy`
- `iam:TagUser`
- `iam:SimulatePrincipalPolicy`

#### 18.2.1.2. Microsoft Azure permissions

The credential you provide for mint mode in Azure must have a service principal with the permissions specified in *Creating a service principal*.

#### 18.2.1.3. Google Cloud Platform (GCP) permissions

The credential you provide for mint mode in GCP must have the following permissions:

- `resourcemanager.projects.get`
- `serviceusage.services.list`
- `iam.serviceAccountKeys.create`
- `iam.serviceAccountKeys.delete`
- `iam.serviceAccounts.create`
18.2.2. Mint mode with removal or rotation of the administrator-level credential

Currently, this mode is only supported on AWS and GCP.

In this mode, a user installs OpenShift Container Platform with an administrator-level credential just like the normal mint mode. However, this process removes the administrator-level credential secret from the cluster post-installation.

The administrator can have the Cloud Credential Operator make its own request for a read-only credential that allows it to verify if all `CredentialsRequest` objects have their required permissions, thus the administrator-level credential is not required unless something needs to be changed. After the associated credential is removed, it can be deleted or deactivated on the underlying cloud, if desired.

**NOTE**

Prior to a non z-stream upgrade, you must reinstate the credential secret with the administrator-level credential. If the credential is not present, the upgrade might be blocked.

The administrator-level credential is not stored in the cluster permanently.

Following these steps still requires the administrator-level credential in the cluster for brief periods of time. It also requires manually re-instating the secret with administrator-level credentials for each upgrade.

18.2.2.1. Rotating cloud provider credentials manually

If your cloud provider credentials are changed for any reason, you must manually update the secret that the Cloud Credential Operator (CCO) uses to manage cloud provider credentials.

The process for rotating cloud credentials depends on the mode that the CCO is configured to use. After you rotate credentials for a cluster that is using mint mode, you must manually remove the component credentials that were created by the removed credential.

**Prerequisites**

- Your cluster is installed on a platform that supports rotating cloud credentials manually with the CCO mode that you are using:
  - For mint mode, AWS, Azure, and GCP are supported.
  - For passthrough mode, AWS, Azure, GCP, Red Hat OpenStack Platform (RHOSP), Red Hat Virtualization (RHV), and VMware vSphere are supported.
- You have changed the credentials that are used to interface with your cloud provider.
The new credentials have sufficient permissions for the mode CCO is configured to use in your cluster.

**NOTE**

When rotating the credentials for an Azure cluster that is using mint mode, do not delete or replace the service principal that was used during installation. Instead, generate new Azure service principal client secrets and update the OpenShift Container Platform secrets accordingly.

**Procedure**

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

2. In the table on the **Secrets** page, find the root secret for your cloud provider.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Secret name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>aws-creds</td>
</tr>
<tr>
<td>Azure</td>
<td>azure-credentials</td>
</tr>
<tr>
<td>GCP</td>
<td>gcp-credentials</td>
</tr>
</tbody>
</table>

3. Click the **Options** menu in the same row as the secret and select **Edit Secret**.

4. Record the contents of the **Value** field or fields. You can use this information to verify that the value is different after updating the credentials.

5. Update the text in the **Value** field or fields with the new authentication information for your cloud provider, and then click **Save**.

6. If the CCO for your cluster is configured to use mint mode, delete each component secret that is referenced by the individual **CredentialsRequest** objects.

   a. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

   b. Get the names and namespaces of all referenced component secrets:

   ```bash
   $ oc -n openshift-cloud-credential-operator get CredentialsRequest -o json | jq -r '.items[] | select (.spec[].kind="<provider_spec>") | .spec.secretRef'
   ```

   Where `<provider_spec>` is the corresponding value for your cloud provider: **AWSProviderSpec** for AWS, **AzureProviderSpec** for Azure, or **GCPProviderSpec** for GCP.

   **Partial example output for AWS**

   ```json
   {
     "name": "ebs-cloud-credentials",
     "namespace": "openshift-cluster-csi-drivers"
   }
   ```
Delete each of the referenced component secrets:

```bash
$ oc delete secret <secret_name> -n <secret_namespace>
```

Where `<secret_name>` is the name of a secret and `<secret_namespace>` is the namespace that contains the secret.

**Example deletion of an AWS secret**

```bash
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

You do not need to manually delete the credentials from your provider console. Deleting the referenced component secrets will cause the CCO to delete the existing credentials from the platform and create new ones.

7. To verify that the credentials have changed:

   a. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

   b. Verify that the contents of the **Value** field or fields are different than the previously recorded information.

18.2.2.2. Removing cloud provider credentials

After installing an OpenShift Container Platform cluster with the Cloud Credential Operator (CCO) in mint mode, you can remove the administrator-level credential secret from the `kube-system` namespace in the cluster. The administrator-level credential is required only during changes that require its elevated permissions, such as upgrades.

**NOTE**

Prior to a non z-stream upgrade, you must reinstate the credential secret with the administrator-level credential. If the credential is not present, the upgrade might be blocked.

**Prerequisites**

- Your cluster is installed on a platform that supports removing cloud credentials from the CCO. Supported platforms are AWS and GCP.

**Procedure**

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

2. In the table on the **Secrets** page, find the root secret for your cloud provider.
<table>
<thead>
<tr>
<th>Platform</th>
<th>Secret name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>aws-creds</td>
</tr>
<tr>
<td>GCP</td>
<td>gcp-credentials</td>
</tr>
</tbody>
</table>

3. Click the **Options** menu in the same row as the secret and select **Delete Secret**

### 18.2.3. Additional resources

- Alternatives to storing administrator-level secrets in the kube-system project for AWS
- Alternatives to storing administrator-level secrets in the kube-system project for Azure
- Alternatives to storing administrator-level secrets in the kube-system project for GCP
- Creating a service principal in Azure

### 18.3. USING PASSTHROUGH MODE

Passthrough mode is supported for Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Red Hat OpenStack Platform (RHOSP), Red Hat Virtualization (RHV), and VMware vSphere.

In passthrough mode, the Cloud Credential Operator (CCO) passes the provided cloud credential to the components that request cloud credentials. The credential must have permissions to perform the installation and complete the operations that are required by components in the cluster, but does not need to be able to create new credentials. The CCO does not attempt to create additional limited-scoped credentials in passthrough mode.

#### 18.3.1. Passthrough mode permissions requirements

When using the CCO in passthrough mode, ensure that the credential you provide meets the requirements of the cloud on which you are running or installing OpenShift Container Platform. If the provided credentials the CCO passes to a component that creates a `CredentialsRequest` CR are not sufficient, that component will report an error when it tries to call an API that it does not have permissions for.

#### 18.3.1.1. Amazon Web Services (AWS) permissions

The credential you provide for passthrough mode in AWS must have all the requested permissions for all `CredentialsRequest` CRs that are required by the version of OpenShift Container Platform you are running or installing.

To locate the `CredentialsRequest` CRs that are required, see [Manually creating IAM for AWS](#).

#### 18.3.1.2. Microsoft Azure permissions

The credential you provide for passthrough mode in Azure must have all the requested permissions for all `CredentialsRequest` CRs that are required by the version of OpenShift Container Platform you are running or installing.
To locate the CredentialsRequest CRs that are required, see Manually creating IAM for Azure.

18.3.1.3. Google Cloud Platform (GCP) permissions

The credential you provide for passthrough mode in GCP must have all the requested permissions for all CredentialsRequest CRs that are required by the version of OpenShift Container Platform you are running or installing.

To locate the CredentialsRequest CRs that are required, see Manually creating IAM for GCP.

18.3.1.4. Red Hat OpenStack Platform (RHOSP) permissions

To install an OpenShift Container Platform cluster on RHOSP, the CCO requires a credential with the permissions of a member user role.

18.3.1.5. Red Hat Virtualization (RHV) permissions

To install an OpenShift Container Platform cluster on RHV, the CCO requires a credential with the following privileges:

- DiskOperator
- DiskCreator
- UserTemplateBasedVm
- TemplateOwner
- TemplateCreator
- ClusterAdmin on the specific cluster that is targeted for OpenShift Container Platform deployment

18.3.1.6. VMware vSphere permissions

To install an OpenShift Container Platform cluster on VMware vSphere, the CCO requires a credential with the following vSphere privileges:

Table 18.2. Required vSphere privileges

<table>
<thead>
<tr>
<th>Category</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datastore</td>
<td>Allocate space</td>
</tr>
<tr>
<td>Folder</td>
<td>Create folder, Delete folder</td>
</tr>
<tr>
<td>vSphere Tagging</td>
<td>All privileges</td>
</tr>
<tr>
<td>Network</td>
<td>Assign network</td>
</tr>
<tr>
<td>Resource</td>
<td>Assign virtual machine to resource pool</td>
</tr>
<tr>
<td>Profile-driven storage</td>
<td>All privileges</td>
</tr>
</tbody>
</table>
18.3.2. Passthrough mode credential maintenance

If **CredentialsRequest** CRs change over time as the cluster is upgraded, you must manually update the passthrough mode credential to meet the requirements. To avoid credentials issues during an upgrade, check the **CredentialsRequest** CRs in the release image for the new version of OpenShift Container Platform before upgrading. To locate the **CredentialsRequest** CRs that are required for your cloud provider, see *Manually creating IAM for AWS, Azure, or GCP*.

18.3.3. Reducing permissions after installation

When using passthrough mode, each component has the same permissions used by all other components. If you do not reduce the permissions after installing, all components have the broad permissions that are required to run the installer.

After installation, you can reduce the permissions on your credential to only those that are required to run the cluster, as defined by the **CredentialsRequest** CRs in the release image for the version of OpenShift Container Platform that you are using.

To locate the **CredentialsRequest** CRs that are required for AWS, Azure, or GCP and learn how to change the permissions the CCO uses, see *Manually creating IAM for AWS, Azure, or GCP*.

18.3.4. Additional resources

- Manually creating IAM for AWS
- Manually creating IAM for Azure
- Manually creating IAM for GCP

18.4. USING MANUAL MODE

Manual mode is supported for Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

In manual mode, a user manages cloud credentials instead of the Cloud Credential Operator (CCO). To use this mode, you must examine the **CredentialsRequest** CRs in the release image for the version of OpenShift Container Platform that you are running or installing, create corresponding credentials in the underlying cloud provider, and create Kubernetes Secrets in the correct namespaces to satisfy all **CredentialsRequest** CRs for the cluster’s cloud provider.

Using manual mode allows each cluster component to have only the permissions it requires, without storing an administrator-level credential in the cluster. This mode also does not require connectivity to the AWS public IAM endpoint. However, you must manually reconcile permissions with new release images for every upgrade.

For information about configuring your cloud provider to use manual mode, see *Manually creating IAM for AWS, Azure, or GCP*.
18.4.1. Manual mode with AWS STS

You can configure an AWS cluster in manual mode to use Amazon Web Services Secure Token Service (AWS STS). With this configuration, the CCO uses temporary credentials for different components.

18.4.2. Upgrading clusters with manually maintained credentials

The Cloud Credential Operator (CCO) **Upgradable** status for a cluster with manually maintained credentials is **False** by default.

- For minor releases, for example, from 4.7 to 4.8, this status prevents you from upgrading until you have addressed any updated permissions and annotated the CloudCredential resource to indicate that the permissions are updated as needed for the next version. This annotation changes the **Upgradable** status to **True**.

- For z-stream releases, for example, from 4.8.9 to 4.8.10, no permissions are added or changed, so the upgrade is not blocked.

Before upgrading a cluster with manually maintained credentials, you must create any new credentials for the release image that you are upgrading to. Additionally, you must review the required permissions for existing credentials and accommodate any new permissions requirements in the new release for those components.

**Procedure**

1. Extract and examine the **CredentialsRequest** custom resource for the new release.
   The *Manually creating IAM* section of the installation content for your cloud provider explains how to obtain and use the credentials required for your cloud.

2. Update the manually maintained credentials on your cluster:
   - Create new secrets for any **CredentialsRequest** custom resources that are added by the new release image.
   - If the **CredentialsRequest** custom resources for any existing credentials that are stored in secrets have changed their permissions requirements, update the permissions as required.

3. When all of the secrets are correct for the new release, indicate that the cluster is ready to upgrade:
   a. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.
   b. Edit the **CloudCredential** resource to add an **upgradeable-to** annotation within the metadata field:

   ```bash
   $ oc edit cloudcredential cluster
   ```

   **Text to add**

   ```yaml
   metadata:
   annotations:
   cloudcredential.openshift.io/upgradeable-to: <version_number>
   ```
Where `<version_number>` is the version you are upgrading to, in the format `x.y.z`. For example, `4.8.2` for OpenShift Container Platform 4.8.2.

It may take several minutes after adding the annotation for the upgradeable status to change.

4. Verify that the CCO is upgradeable:

   a. In the the **Administrator** perspective of the web console, navigate to **Administration → Cluster Settings**.

   b. To view the CCO status details, click **cloud-credential** in the **Cluster Operators** list.

   c. If the **Upgradeable** status in the **Conditions** section is **False**, verify that the **upgradeable-to** annotation is free of typographical errors.

When the **Upgradeable** status in the **Conditions** section is **True**, you can begin the OpenShift Container Platform upgrade.

### 18.4.3. Additional resources

- Manually creating IAM for AWS
- Manually creating IAM for Azure
- Manually creating IAM for GCP
- Using manual mode with AWS STS

### 18.5. USING MANUAL MODE WITH STS

Manual mode with STS is supported for Amazon Web Services (AWS).

In manual mode with STS, the individual OpenShift Container Platform cluster components use AWS Secure Token Service (STS) to assign components IAM roles that provide short-term, limited-privilege security credentials. These credentials are associated with IAM roles that are specific to each component that makes AWS API calls.

Requests for new and refreshed credentials are automated by using an appropriately configured AWS IAM OpenID Connect (OIDC) identity provider, combined with AWS IAM roles. OpenShift Container Platform signs service account tokens that are trusted by AWS IAM, and can be projected into a pod and used for authentication. Tokens are refreshed after one hour.

**Figure 18.1. STS authentication flow**  

![STS Authentication Flow Diagram](image)
Using manual mode with STS changes the content of the AWS credentials that are provided to individual OpenShift Container Platform components.

**AWS secret format using long-lived credentials**

```yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: <target-namespace>  
  name: <target-secret-name>
  data:
    aws_access_key_id: <base64-encoded-access-key-id>
    aws_secret_access_key: <base64-encoded-secret-access-key>
```

1. The namespace for the component.
2. The name of the component secret.

**AWS secret format with STS**

```yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: <target-namespace>  
  name: <target-secret-name>
  data:
    role_name: <operator-role-name>
    web_identity_token_file: <path-to-token>
```

1. The namespace for the component.
2. The name of the component secret.
3. The IAM role for the component.
4. The path to the service account token inside the pod. By convention, this is `/var/run/secrets/openshift/serviceaccount/token` for OpenShift Container Platform components.

### 18.5.1. Installing an OpenShift Container Platform cluster configured for manual mode with STS

To install a cluster that is configured to use the Cloud Credential Operator (CCO) in manual mode with STS:

1. Configure the Cloud Credential Operator utility.
2. Create the required AWS resources individually, or with a single command.
3. Run the OpenShift Container Platform installer.
4. Verify that the cluster is using short-lived credentials.
NOTE

Because the cluster is operating in manual mode when using STS, it is not able to create new credentials for components with the permissions that they require. When upgrading to a different minor version of OpenShift Container Platform, there are often new AWS permission requirements. Before upgrading a cluster that is using STS, the cluster administrator must manually ensure that the AWS permissions are sufficient for existing components and available to any new components.

18.5.1.1. Configuring the Cloud Credential Operator utility

To create and manage cloud credentials from outside of the cluster when the Cloud Credential Operator (CCO) is operating in manual mode with STS, extract and prepare the CCO utility (ccoctl) binary.

NOTE

The ccoctl is a Linux binary that must run in a Linux environment.

Prerequisites

- Obtain the OpenShift Container Platform release image.

Procedure

1. Get the CCO container image from the OpenShift Container Platform release image:

   ```bash
   $ CCO_IMAGE=$(oc adm release info --image-for='cloud-credential-operator' $RELEASE_IMAGE)
   ```

2. Extract the ccoctl binary from the CCO container image within the OpenShift Container Platform release image:

   ```bash
   $ oc image extract $CCO_IMAGE --file="/usr/bin/ccoctl" -a ~/.pull-secret
   ```

3. Change the permissions to make ccoctl executable:

   ```bash
   $ chmod 775 ccoctl
   ```

Verification

- To verify that ccoctl is ready to use, display the help file:

  ```bash
  $ ccoctl aws --help
  ```

Output of ccoctl aws --help:

Creating/updating/deleting cloud credentials objects for AWS cloud

Usage:

  ccoctl aws [command]
18.5.1.2. Creating AWS resources with the Cloud Credential Operator utility

You can use the CCO utility (ccoctl) to create the required AWS resources individually, or with a single command.

18.5.1.2.1. Creating AWS resources individually

If you need to review the JSON files that the ccoctl tool creates before modifying AWS resources, or if the process the ccoctl tool uses to create AWS resources automatically does not meet the requirements of your organization, you can create the AWS resources individually. For example, this option might be useful for an organization that shares the responsibility for creating these resources among different users or departments.

Otherwise, you can use the ccoctl aws create-all command to create the AWS resources automatically.

NOTE

By default, ccoctl creates objects in the directory in which the commands are run. To specify a directory, use the --output-dir flag. This procedure uses <path_to_ccoctl_output_dir> to refer to this location.

Some ccoctl commands make AWS API calls to create or modify AWS resources. To place JSON files on the local file system instead, use the --dry-run flag. These JSON files can be reviewed or modified and then applied with the AWS CLI tool using the --cli-input-json parameters.

Prerequisites

- Extract and prepare the ccoctl binary.

Procedure

1. Generate the public and private RSA key files that are used to set up the OpenID Connect provider for the cluster:

   $ ccoctl aws create-key-pair

   Example output:

   2021/04/13 11:01:02 Generating RSA keypair
   2021/04/13 11:01:03 Writing private key to /<path_to_ccoctl_output_dir>/serviceaccount-
signer.private
2021/04/13 11:01:03 Writing public key to /<path_to_ccoctl_output_dir>/serviceaccount-signer.public
2021/04/13 11:01:03 Copying signing key for use by installer

where **serviceaccount-signer.private** and **serviceaccount-signer.public** are the generated key files.

This command also creates a private key that the cluster requires during installation in
/<path_to_ccoctl_output_dir>/tls/bound-service-account-signing-key.key.

2. Create an OpenID Connect identity provider and S3 bucket on AWS:

```bash
$ ccoctl aws create-identity-provider --name=<name> --region=<aws_region> --public-key-file=<path_to_ccoctl_output_dir>/serviceaccount-signer.public
```

where:

- **<name>** is the name used to tag any cloud resources that are created for tracking.
- **<aws-region>** is the AWS region in which cloud resources will be created.
- **<path_to_ccoctl_output_dir>** is the path to the public key file that the `ccoctl aws create-key-pair` command generated.

**Example output:**

```
2021/04/13 11:16:09 Bucket <name>-oidc created
2021/04/13 11:16:10 OpenID Connect discovery document in the S3 bucket <name>-oidc at .well-known/openid-configuration updated
2021/04/13 11:16:10 JSON web key set (JWKS) in the S3 bucket <name>-oidc at keys.json updated
```

where **02-openid-configuration** is a discovery document and **03-keys.json** is a JSON web key set file.

This command also creates a YAML configuration file in
/`<path_to_ccoctl_output_dir>/manifests/cluster-authentication-02-config.yaml`. This file sets the issuer URL field for the service account tokens that the cluster generates, so that the AWS IAM identity provider trusts the tokens.

3. Create IAM roles for each component in the cluster.

   a. Extract the list of **CredentialsRequest** objects from the OpenShift Container Platform release image:

```bash
$ oc adm release extract --credentials-requests --cloud=aws --
to=<path_to_directory_with_list_of_credentials_requests>/credrequests
quay.io/<path_to>/ocp-release:<version>
```

---

OpenShift Container Platform 4.8 Authentication and authorization

178
b. Use the `ccoctl` tool to process all `CredentialsRequest` objects in the `credrequests` directory:

```
```

**NOTE**

For AWS environments that use alternative IAM API endpoints, such as GovCloud, you must also specify your region with the `--region` parameter.

For each `CredentialsRequest` object, `ccoctl` creates an IAM role with a trust policy that is tied to the specified OIDC identity provider, and a permissions policy as defined in each `CredentialsRequest` object from the OpenShift Container Platform release image.

**Verification**

- To verify that the OpenShift Container Platform secrets are created, list the files in the `<path_to_ccoctl_output_dir>/manifests` directory:

  ```
  $ ll <path_to_ccoctl_output_dir>/manifests
  total 24
  -rw-------. 1 <user> <user> 161 Apr 13 11:42 cluster-authentication-02-config.yaml
  -rw-------. 1 <user> <user> 379 Apr 13 11:59 openshift-cloud-credential-operator-cloud-credential-operator-iam-ro-creds-credentials.yaml
  -rw-------. 1 <user> <user> 353 Apr 13 11:59 openshift-cluster-csi-drivers-ebs-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 355 Apr 13 11:59 openshift-image-registry-installer-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 339 Apr 13 11:59 openshift-ingress-operator-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 337 Apr 13 11:59 openshift-machine-api-aws-cloud-credentials-credentials.yaml
  ```

You can verify that the IAM roles are created by querying AWS. For more information, refer to AWS documentation on listing IAM roles.

### 18.5.1.2.2. Creating AWS resources with a single command

If you do not need to review the JSON files that the `ccoctl` tool creates before modifying AWS resources, and if the process the `ccoctl` tool uses to create AWS resources automatically meets the requirements of your organization, you can use the `ccoctl aws create-all` command to automate the creation of AWS resources.

Otherwise, you can create the AWS resources individually.
NOTE

By default, `ccoctl` creates objects in the directory in which the commands are run. To specify a directory, use the `--output-dir` flag. This procedure uses `<path_to_ccoctl_output_dir>` to refer to this location.

Prerequisites

- Extract and prepare the `ccoctl` binary.

Procedure

1. Extract the list of `CredentialsRequest` objects from the OpenShift Container Platform release image:

   ```bash
   $ oc adm release extract --credentials-requests --cloud=aws --
   to=<path_to_directory_with_list_of_credentials_requests>/credrequests
   quay.io/<path_to>/ocp-release:<version>
   ```

2. Use the `ccoctl` tool to process all `CredentialsRequest` objects in the `credrequests` directory:

   ```bash
   $ ccoctl aws create-all --name=<name> --region=<aws_region> --credentials-requests-dir=<path_to_directory_with_list_of_credentials_requests>/credrequests
   ```

Verification

- To verify that the OpenShift Container Platform secrets are created, list the files in the `<path_to_ccoctl_output_dir>/manifests` directory:

  ```bash
  $ ll <path_to_ccoctl_output_dir>/manifests
  ```

  Example output:

  ```
  total 24
  -rw-------. 1 <user> <user> 161 Apr 13 11:42 cluster-authentication-02-config.yaml
  -rw-------. 1 <user> <user> 379 Apr 13 11:59 openshift-cloud-credential-operator-cloud-credential-operator-iam-ro-creds-credentials.yaml
  -rw-------. 1 <user> <user> 353 Apr 13 11:59 openshift-cluster-csi-drivers-ebs-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 355 Apr 13 11:59 openshift-image-registry-installer-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 339 Apr 13 11:59 openshift-ingress-operator-cloud-credentials-credentials.yaml
  -rw-------. 1 <user> <user> 337 Apr 13 11:59 openshift-machine-api-aws-cloud-credentials-credentials.yaml
  ```

  You can verify that the IAM roles are created by querying AWS. For more information, refer to AWS documentation on listing IAM roles.

18.5.1.3. Running the installer

Prerequisites
Create the `install-config.yaml` file for your installation.

**Procedure**

1. From the installation directory that contains your `install-config.yaml` file, configure the Cloud Credential Operator (CCO) to run in manual mode:
   
   ```
   $ echo "credentialsMode: Manual" >> install-config.yaml
   ```

2. Create the required OpenShift Container Platform installation manifests:
   
   ```
   $ openshift-install create manifests
   ```

3. Copy the manifests that `ccoctl` generated to the manifests directory that the installation program created:
   
   ```
   $ cp /<path_to_ccoctl_output_dir>/manifests/* ./manifests/
   ```

4. Copy the private key that the `ccoctl` generated in the `tls` directory to the installation directory:
   
   ```
   $ cp -a /<path_to_ccoctl_output_dir>/tls .
   ```

5. Run the OpenShift Container Platform installer:
   
   ```
   $ ./openshift-install create cluster
   ```

**18.5.1.4. Verifying the installation**

1. Connect to the OpenShift Container Platform cluster.

2. Verify that the cluster does not have root credentials:

   ```
   $ oc get secrets -n kube-system aws-creds
   
   The output should look similar to:
   
   Error from server (NotFound): secrets "aws-creds" not found
   ```

3. Verify that the components are assuming the IAM roles that are specified in the secret manifests, instead of using credentials that are created by the CCO:

   **Example command with the Image Registry Operator**

   ```
   $ oc get secrets -n openshift-image-registry installer-cloud-credentials -o json | jq -r .data.credentials | base64 --decode
   
   The output should show the role and web identity token that are used by the component and look similar to:
   
   **Example output with the Image Registry Operator**
[default]
role_arn = arn:aws:iam::123456789:role/openshift-image-registry-installer-cloud-credentials
web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token