

# Authentication

This section explains how to authenticate users in KumoScale.

The KumoScale Cluster Manager CLI and REST APIs are accessed using **JavaScript™ Object Notation (JSON) Web Tokens (JWT)** for both Appliance and Managed modes. The token will function until its expiration if the session has not terminated. Any user may generate a token for self-use, which can be entered in each REST command or once per CLI session. There are two ways to manage tokens for storage access and they apply to both Appliance and Managed Mode:

- [Self-generated tokens for local users](#)
- [OpenID Connect \(OIDC\)](#)

## Self-Generated Tokens for Local Users

Any user may generate a token for self-use, which can be entered in each REST command (or once per CLI session). The REST API **Generate Token** command has an expiration parameter (in seconds):

- Default (and minimum) value – one hour.
- Maximum – 30 days.

The token is a JWT and must be properly validated before it can be used thereafter as an access token in KumoScale Provisioner REST API calls.

## Open ID Connect Authentication

To configure KumoScale to support OpenID Connect (OIDC) authentication, you can use the Provisioner REST API, Cluster Manager CLI, or custom resource files to complete the following steps:

1. Define the authentication server. KumoScale supports both Keycloak™ and Active Directory Federation Services (ADFS).
2. Update the KumoScale Provisioner secret with details on the authentication server.
3. Create the secret.

**Note:** When you configure OIDC, you must confirm that the clock is synchronized across all servers involved in provisioning and storage. For example, servers supporting the KumoScale Provisioner, storage nodes, and Keycloak. Tokens have an expiration time and if there is a time difference between initiators, OIDC authentication may fail with a *cannot decrypt token* error.

## ADFS Users

You will need to configure KumoScale with Azure™ AD by creating new clients and rules:

1. Create a new client and configure it as a server application. For example: This may be the storage client of the Provisioner API.
- 2.. Add the Client ID, for example **storage**
3. In all applications, set the redirect URL as <https://localhost> since it is not used for client credential flows.
4. Generate a shared secret and copy it to the clipboard. Complete the process of creating a new application. Map the new application server to the Provisioner WEB API
5. Add a new rule by selecting **Send claims using custom rule** You can use the following rule:

```
c:[Type == "http://schemas.microsoft.com/2014/01/clientcontext/claims/appid", Value == "storage"]
p:issue(Type = "http://schemas.microsoft.com/ws/2008/06/identity/claims/role", Value = "STORAGE");
```

6. Define client permissions and add the client application .
7. Finally, use the new clientID and secret to perform operations as ‘storage’ user.
8. Generate a token and use it to operate as a storage user.

## Keycloak Users

Gather information on your authorization server from Keycloak.

1. Log in with a Keycloak account.
2. Find the values of the:
  - **Authorization server public key**, under **RealmSettings > keys**

- Provisioner Resource ID name, Provisioner Client ID, Storage node Resource ID and Storage node Client ID, under the client tab

Complete information on how to use Keycloak is available at <https://www.keycloak.org>.

### OIDC Authentication using Custom Resource Files

In this section we will show you how to set up the authorization server using custom resource files. You will need to configure two files. An example of each is provided with KumoScale software in **KumoScale\_Operator**.

- Authorization Secret CR, example file `/ks-config-operator/deploy/authorization-secret.yaml`.
- Authorization Server CR, example file `/ks-config-operator/samples/kumoscale_v1_authorizationserver.yaml`.

Complete the steps below:

1. Gather details on the Keycloak or ADFS authentication server as defined above.
2. Set up the authorization secret file. The authorization secret file must contain values for each of the following fields:

Secret Parameter	Description	Optional/Required
authenticationServerPublicKey	The public key used to decrypt tokens. Must be provided if authenticationServerJwkSetURL is not specified.	Optional
authenticationServerJwkSetURL	A URL for refreshing public keys used to decrypt tokens. Must be provided if authenticationServerJwkSetURL is not specified.	Optional
authenticationServerTokenURL	Authorization server URL for generating tokens.	Required
provisionerClientID	The client ID of a client which has a service account with KumoScale level role ADMIN.	Required
provisionerClientSecret	The client secret	Required
provisionerClientScope	The scope of the KumoScale Provisioner client.	Optional
provisionerResourceID	The ID of the provisioner resource; the client name as defined in the authorization server of the provisioner.	Required
storagenodeClientID	The client ID of a client which has a service account with Provisioner level role ADMIN.	Required
storagenodeClientSecret	The KumoScale client secret	Required
storagenodeClientScope	The scope of the KumoScale client secret in the Provisioner.	Optional
storagenodeResourceID	The ID of the KumoScale resource.	Required

For example, the file **authorization-secret.yaml** contains

```
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret
  namespace: default
type: Opaque
stringData:
  config.yaml: |-
    authorizationServerPublicKey: #####
    authServerTokenUrl: http://###.###.1:8080/auth/realms/Dev/protocol/openid-connect/token
    provisionerClientID: provisioner-adi
    provisionerClientSecret: #####-###-#####
    provisionerResourceID: provisioner-adi
    storagenodeClientID: kumoscale-adi
    storagenodeClientSecret: #####-###-#####
    storagenodeResourceID: kumoscale-adi
```

3. Specify the values of the authorization CR. Only one authorization server CR is allowed. The table below summarizes the parameters to specify.

Authorization server Parameter	Description	Optional/Required
secretName	The secret which provides details for configuring the authentication server. This has a string value.	Required
revision	Used to track the updates of the CR secret. It should be increased by 1 for each update to the secret. This has an integer value.	Required
status	Generated automatically after you create the CR. To verify the status, enter	N/A

	<code>kubect describe authorizationServer -A</code>	
	<div>The value will be<ul style="list-style-type: none"><li>• <b>Pending</b> if validation failed. The server is not active.</li><li>• <b>Active</b> , if validation was successful. The server is now active</li></ul></div>	

For example, using the secret file above which creates auth-secret, the file **authorization-server.yaml** contains

```
apiVersion: kumoscale.kioxia.com/v1
kind: AuthorizationServer
metadata:
  name: authorizationserver-sample
spec:
  secretName: auth-secret
revision: 1
```

4. Create the secret with:

```
kubectl create -f authorization-secret.yaml
```

**Note:** If you need to update the secret - use **kubectl replace** (not **apply**), otherwise sensitive data will be shown when using **kubectl get secret -o yaml**

## OIDC Authentication using the REST API

You may also use the REST API commands below. See the [KumoScale Provisioner REST API](#) documentation for complete details on how to set up OIDC authentication. Below is an example.

1. Define the Authorization server using the ***Set Authorization Server*** Provisioner REST API command to set the parameters specified in the Provisioner REST API documentation.

2. Set the server. For example:

```
curl -k --cert ./ssdtoolbox.pem -X POST -H 'Content-Type: application/json' -d '{"authorizationServerPublicKey":"<key
```

Example using ADFS:

```
curl -k --cert ./ssdtoolbox.pem -X POST -H 'Content-Type: application/json' -d '{ "authorizationServerJwkSetURL":"https://adfs.kioxia.com/keys,
"provisionerClientID":"provisioner",
"provisionerResourceID":"provisioner",
"provisionerClientSecret":"BIGLONGSTRING", "provisionerClientScope":"kumoscale/openid",
"backendsClientID":"kumoscale",
"backendsResourceID":"kumoscale",
"backendsClientSecret":"ceta9e3Z-1TAjAIpyTSQnRLcBVnSMh-df015nofK",
"backendsClientScope":"provisioner/openid"
}'
```

3. Get a token from the authorization server. For example:

```
curl -H 'Content-Type: application/x-www-form-urlencoded' --data 'grant_type=client_credentials&client_id=****&client_
```

Next: [Setting up Initiators](#)

