

# Securing Cloudera Edge Management

Date published: 2019-04-15

Date modified: 2024-03-08



# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>CEM security overview.....</b>	<b>4</b>
<b>FIPS 140-2 compliance.....</b>	<b>4</b>
Configuring EFM to use GCM ciphers.....	5
<b>Identity mapping.....</b>	<b>6</b>
<b>TLS configuration for CEM.....</b>	<b>7</b>
TLS keys and certificates.....	7
TLS configuration for EFM.....	8
Agent authentication.....	10
<b>User authentication.....</b>	<b>10</b>
Configuring OpenID Connect SSO.....	11
Configuring SAML 2.0 SSO.....	12
Configuring Knox SSO.....	14
Configuring mutual TLS authentication.....	15
Configuring proxy authentication.....	16
Configuring LDAP authentication.....	17
<b>Access control policies.....</b>	<b>19</b>
Access control bootstrapping.....	19
Initial admin identities.....	19
First login with a new user.....	20
User management.....	21
Creating users.....	22
Editing users.....	22
User group management.....	23
Creating user groups.....	23
Editing user groups.....	24
Managing user groups using IdP.....	26
Managing user groups using LDAP.....	27
Agent class roles policies.....	30
Assigning policies to a user.....	33
Assigning policies to a user group.....	34
Assigning policies to agent classes.....	36
<b>SSO identity provider setup.....</b>	<b>39</b>
Okta integration using OIDC.....	39
Okta EFM integration using SAML.....	40

## CEM security overview

Cloudera Edge Management (CEM) is not secure by default. Cloudera recommends that you must always enable security for production environment. To secure CEM, you must secure both Edge Flow Manager (EFM) and MiNiFi agents.

By default, EFM runs in an unsecured mode where the web endpoints are accessible over HTTP on all network interfaces and clients are not authenticated. When unsecured, all clients are anonymous and have full access to the application. For this reason, insecure mode should only be used for test or development purposes and when EFM is not accessible through the public Internet.

Limiting the network interfaces that the web server binds to is configurable in the `efm.properties` file.

```
efm.web.host=localhost
```

For production environments, security should always be enabled by configuring a TLS context and method of user authentication.

Securing CEM involves securing both the EFM server and MiNiFi agents.

The EFM server provides centralized control of MiNiFi agents. Starting with version 1.3.0, EFM provides robust options for authentication and authorization.

The high-level steps for securing a CEM system are:

1. Generating or obtaining keys and certificates for EFM, MiNiFi agents, and optionally service user accounts.
2. Configuring the EFM TLS context.
3. Configuring MiNiFi agent TLS contexts, which allows MiNiFi agents to authenticate to a secured EFM server.
4. Configuring end-user authentication for the EFM web application UI, typically as an integration with a Single Sign On (SSO) identity provider.
5. Assigning access control policies to users and groups in the EFM web application UI.

For more information about the security aspects of EFM, check out the video on the Cloudera Edge Management YouTube playlist:

## FIPS 140-2 compliance

Federal Information Processing Standards (FIPS) are publicly available standards and guidelines issued by the National Institute of Standards and Technology for use in computer systems by non-military American government agencies and government contractors. You can configure CDP Private Cloud Base to use FIPS-compliant cryptography.

To install and configure a CDP cluster that is FIPS-compliant, see *Installing and Configuring CDP with FIPS*. In combination with AutoTLS, the cluster will use BouncyCastle FIPS Keystore (BCFKS) across all the components.

Note the following points about FIPS compliance in CEM:

- CEM is compatible with a FIPS 140-2 compliant environment.
- CEM can run on an OS with FIPS turned on and can use FIPS-compliant crypto libraries.
- By default, the KeyStore and TrustStore are in Java KeyStore (JKS) format. This format is not FIPS compliant.
- By default, Edge Flow Manager dataflows are not FIPS compliant. You must specifically design a dataflow to be FIPS compliant.
- If you want to implement FIPS mode on your CEM cluster, you should use the `bctls.jar`, which uses Galois/Counter Mode (GCM) ciphers. GCM ciphers are not allowed by the EFM Java process by default. For configuration information, see *Configuring EFM to use GCM ciphers*.
- EFM sensitive properties can be encrypted. See the *Encrypting sensitive properties* guide for more information.

- For additional details on the FIPS 140-2 Publication, see *FIPS 140-2 Security Requirements for Cryptographic Modules*.

For the National Institute of Standards and Technology publication, see *FIPS 140-2 Security Requirements for Cryptographic Modules*.

## Configuring EFM to use GCM ciphers

If you implement FIPS mode on your Cloudera Edge Management (CEM) cluster, you have to use the bctls.jar, which uses Galois/Counter Mode (GCM) ciphers. These ciphers are not allowed by the NiFi Java process by default. Learn how you can configure EFM to use the GCM ciphers.

### About this task

The Bouncy Castle TLS library bctls.jar includes an implementation of the TLS protocol that takes precedence over the standard Java implementation when configuring the BouncyCastleJsseProvider as a provider in java.security. The default configuration of the BCTLS library does not enable GCM-based ciphers, which results in TLS server components attempting to negotiate weak cipher suites based on AES-CBC. Modern web browsers such as Google Chrome and Mozilla Firefox disable weak cipher suites, causing cipher mismatch errors when attempting to connect to a FIPS-enabled deployment of CEM. Enabling GCM-based ciphers allows clients to negotiate modern TLS cipher suites, avoiding connection issues related to weak algorithms.

Follow these steps to configure EFM:

### Procedure

1. Download the FIPS JAR files from the [Cloudera repository](#).
2. Copy the bctls.jar and ccj-3.0.2.1.jar files from the downloaded files to the CEM library.

```
cp bctls.jar ccj-3.0.2.1.jar [***EFM_DIRECTORY***]/lib
```

3. Make sure there is a JDK 17 installed and configured for FIPS.

- The following snippet is present in the java.policy file.

```
// CCJ Java Permissions
permission java.lang.RuntimePermission "getProtectionDomain";
permission java.lang.RuntimePermission "accessDeclaredMembers";
permission java.util.PropertyPermission "java.runtime.name", "read";
permission java.security.SecurityPermission "putProviderProperty.CCJ";

// CCJ Key Export and Translation
permission com.safelogic.cryptocomply.crypto.CryptoServicesPermission "
exportKeys";

// CCJ SSL
permission com.safelogic.cryptocomply.crypto.CryptoServicesPermission "
tlsAlgorithmsEnabled";

// CCJ Setting of Default SecureRandom
permission com.safelogic.cryptocomply.crypto.CryptoServicesPermission
"defaultRandomConfig";
// CCJ Setting CryptoServicesRegistrar Properties
permission com.safelogic.cryptocomply.crypto.CryptoServicesPermission
"globalConfig";

// CCJ Enable JKS
```

```
permission com.safelogic.cryptocomply.jca.enable_jks "true";
```

**Note:**

You can find the java.policy file in [\*\*\*JDK\_17\_DIR\*\*\*/conf/security.

- The java.security security providers section contains the following:

```
security.provider.1=CryptoComplyFipsProvider
security.provider.2=BouncyCastleJsseProvider fips:CCJ
security.provider.3=SUN
security.provider.4=SunRsaSign
security.provider.5=SunEC
security.provider.6=SunJSSE
security.provider.7=SunJCE
security.provider.8=SunJGSS
security.provider.9=SunSASL
security.provider.10=XMLDSig
#security.provider.11=SunPCSC
security.provider.12=JdkLDAP
security.provider.13=JdkSASL
security.provider.14=SunPKCS11
```

**Note:**

You can find the java.security file in [\*\*\*JDK\_17\_DIR\*\*\*/conf/security.

- Set the org.bouncycastle.jsse.fips.allowGCMCiphers=true Java system property in the efm.conf configuration file to enable support for GCM-based ciphers using the BCTLS library.

```
JAVA_OPTS="-Xms2048m -Xmx2048m -XX:+UseG1GC -Dorg.bouncycastle.jsse.fips
.allowGCMCiphers=true --module-path=[***EFM_DIR***/lib/ccj-3.0.2.1.jar:/
[***EFM_DIR***/lib/bctls.jar"
```

If it is not configured properly, the following error message is displayed:

```
ERR_SSL_VERSION_OR_CIPHER_MISMATCH
Unsupported protocol
The client and server don't support a common SSL protocol version or cipher suite
```

- If you are using Cloudera Manager to manage EFM, add the following parameter to the JAVA\_OPTS property.
  - In Cloudera Manager, go to the NiFi service.
  - Select Configuration.
  - Locate the JAVA\_OPTS property.
 

You can search for it by typing its name in the Search box.
  - Append the following to the existing property.

```
-Dorg.bouncycastle.jsse.fips.allowGCMCiphers=true --module-path=[***EFM_DIR***/lib/ccj-3.0.2.1.jar:/[***EFM_DIR***/lib/bctls.jar
```

- Click Save Changes to commit the changes.
- Restart the service.

## Identity mapping

Identity mapping allows you to normalize user and group identities from different identity providers.

For example, you can combine mTLS authentication with an LDAP user group manager, even if there is no exact match in the Distinguished Name (DN) from the user certificate and LDAP.

The following examples demonstrate normalizing DN from certificates and LDAP:

### Certificate DN mapping

- `efm.security.identity.mapping.pattern.dn=^CN=(.*?), OU=(.*?), O=(.*?), L=(.*?), ST=(.*?), C=(.*?)$`
- `efm.security.identity.mapping.value.dn=$1`
- `efm.security.identity.mapping.transform.dn=LOWER`

### LDAP DN mapping

- `efm.security.identity.mapping.pattern.ldap=^cn=(.*?), ou=(.*?), o=(.*?)$`
- `efm.security.identity.mapping.value.ldap=$1`
- `efm.security.identity.mapping.transform.ldap=LOWER`

Each property's final segment serves as an identifier, linking the pattern to the replacement value. When a user initiates a request in EFM, their identity is verified against these patterns in lexicographical order. When encountering the first matching pattern, the replacement specified in the `efm.security.identity.mapping.value.xxxx` property is applied.

For instance, when a login with the DN "CN=admin, OU=EFM, O=Cloudera, L=Santa Clara, ST=CA, C=US" corresponds to the above DN mapping pattern, the DN mapping value of "\$1" is used, normalizing the user to localhost.

The LDAP DN is slightly different so it would not match with the certificate identity. To resolve this issue, a pattern is applied, to extract the CN part of the LDAP's DN and transform the value to lowercase.

In addition to mapping, there is an option to apply transformations. Supported transformation options include:

- **NONE:** no transformation is applied
- **LOWER:** identity is converted to lowercase
- **UPPER:** identity is converted to uppercase

If not specified, the default value is **NONE**.

Group names can also be mapped. The following example accepts the existing group name but converts it to uppercase:

### Group name mapping

- `efm.security.group.mapping.pattern.dn=^(.*)$`
- `efm.security.group.mapping.value.dn=$1`
- `efm.security.group.mapping.transform.dn=UPPER`



**Note:** These mappings are also applied to the User and Group Initial Admin Identities.

## TLS configuration for CEM

To secure Cloudera Edge Management (CEM), learn how to secure your Edge Flow Manager (EFM) server and MiNiFi agents through TLS. Also learn about the TLS keys and certificates needed for these tasks.

### TLS keys and certificates

When using Cloudera Edge Management (CEM), sensitive information are sent over the network between Edge Flow Manager (EFM) and MiNiFi agents including configuration files that contain passwords. To secure this transfer, you must configure Transport Layer Security (TLS) encryption.

TLS is an industry standard set of cryptographic protocols for securing communications over a network.

Configuring TLS involves creating a private key and a public key for use by server and client processes to negotiate an encrypted connection. In addition, TLS can use certificates to verify the trustworthiness of keys presented during the negotiation to prevent spoofing and mitigate other potential security issues.

Because TLS keys and certificates are usually specific to each enterprise organization, CEM needs these details manually configured.

To secure CEM through TLS, you need the following:

- A Certificate Authority (CA) or intermediate signing authority public certificate, used to sign certificates and verify trust. EFM uses this certificate in its truststore in order to verify the identity of MiNiFi agents by validating their certificates. MiNiFi agents use this certificate in their truststores in order to verify the identity of the EFM servers by validating their certificates.
- A public and private keypair for EFM signed by the CA, recommended one per host when doing a cluster deployment.
- A public and private keypair for MiNiFi agents signed by the CA, recommended one per agent.

Before configuring TLS, ensure that the certificates used meet the requirements.

### Certificate requirements

- The EFM KeyStore must be in Java KeyStore (JKS) or PKCS #12 (PKCS12 aka .p12) format.
- The EFM KeyStore must contain only one private key entry.
- The X509v3 KeyUsage section of the certificate must include the following attributes:
  - DigitalSignature
  - Key\_Encipherment
- The X509v3 ExtendedKeyUsages section of EFM and MiNiFi agent certificates must include the following attributes:
  - clientAuth
  - serverAuth
- The signature algorithm used for certificates must be sha256WithRSAEncryption (SHA-256).
- For EFM certificates, Subject Alternate Names (SANs) are mandatory and should at least include the FQDN of the host.
- Additional names for the certificate and host can be added to the certificate as SANs.
  - Add the FQDN used for the Common Name (CN) as a DNS SAN entry.
  - If you are planning to use a load balancer for an EFM cluster, include the FQDN for the load balancer as a DNS SAN entry.

Cloudera recommends the following security protocols:

- Use certificates that are signed by a CA. Do not issue self-signed certificates.
- Generate a unique certificate per host, including MiNiFi agents. Do not use wildcard certificates.

## TLS configuration for EFM

Learn about the properties that you need to set to configure TLS for Edge Flow Manager (EFM).

When TLS is disabled, EFM runs in an unsecured mode where the web endpoints are accessible over HTTP on all network interfaces and clients are not authenticated. When unsecured, all clients are anonymous and have full access to the application. For this reason, insecure mode should only be used for test or development purposes and when EFM is not accessible through the public Internet.

Limiting the network interfaces that the web server binds to is configurable in the `efm.properties` file.

```
efm.web.host=localhost
```

For production environments, security should always be enabled by configuring a TLS context and method of user authentication.

Securing an EFM instance starts with configuring a TLS context, also known as an SSL context. Enabling TLS enforces that all usage of the EFM web application as well as agent communication is authenticated and that the authenticated user or agent is allowed to perform requested actions based on policies.

Settings for the EFM TLS context are configured using the `efm.server.ssl.*` prefixed properties in the `efm.properties` file:

```
efm.server.ssl.enabled=true
efm.server.ssl.keyStore=/path/to/keystore.jks
efm.server.ssl.keyStoreType=jks
efm.server.ssl.keyStorePassword=keyStorePassword
efm.server.ssl.keyPassword=keyPassword
efm.server.ssl.trustStore=/path/to/truststore.jks
efm.server.ssl.trustStoreType=jks
efm.server.ssl.trustStorePassword=trustStorePassword
efm.server.ssl.clientAuth=WANT
```

Where,

- `efm.server.ssl.enabled`  
Set to true to enable TLS and secure EFM.
- `efm.server.ssl.keyStore`  
The file path to the keystore containing the EFM TLS keypair.
- `efm.server.ssl.keyStoreType`  
Set to jks or pkcs12 depending on the format of the keystore. Other formats are not supported.
- `efm.server.ssl.keyStorePassword`  
The passphrase for the keystore.
- `efm.server.ssl.keyPassword`  
The passphrase for the key in the keystore.
- `efm.server.ssl.trustStore`  
The file path to the truststore containing the public certificates of the Certificate Authorities (CA) trusted by EFM.
- `efm.server.ssl.trustStoreType`  
Set to jks or pkcs12 depending on the format of the truststore. Other formats are not supported.
- `efm.server.ssl.trustStorePassword`  
The passphrase for the truststore.
- `efm.server.ssl.clientAuth`  
Set to WANT to allow MiNiFi agents to authenticate to EFM with TLS client certificates, but allow users to use an alternative form of authentication (such as SSO). If users are also using mutual TLS with client certificates in browsers, then this can be set to NEED.



**Note:** For a clustered deployment, these properties must be set on every node in the EFM cluster.

For more information regarding generating keystores and truststores for EFM, see *TLS keys and certificates*.

### Related Information

[TLS keys and certificates](#)

## Agent authentication

Learn about the properties that you need to set to configure TLS for MiNiFi agents.

When TLS is enabled to secure Edge Flow Manager (EFM), agents must be authenticated using client certificates. Client certificates are trusted if the Certificate Authority and any intermediate signing authorities (if applicable) are present in the EFM truststore configured as described in *TLS configuration for EFM*.

For more information on requirements for MiNiFi agent client certificates and instructions for generating new certificates, see *TLS keys and certificates*.

Cloudera recommends using a unique client certificate per agent. However, it is not a requirement.

For details about the properties that you need to set to configure TLS for MiNiFi agents, see *MiNiFi C++ agent authentication* and *MiNiFi Java agent authentication*.

### Related Information

[TLS configuration for EFM](#)

[TLS keys and certificates](#)

[Integrating with the Windows certificate store](#)

[MiNiFi C++ agent authentication](#)

[MiNiFi Java agent authentication](#)

## User authentication

You can secure Edge Flow Manager (EFM) by integrating with Single Sign-On (SSO) identity providers for login purposes. You can also manage user roles to control access to agent classes and authorized actions. Learn about the properties that you need to configure to enable user authentication within EFM.

Securing EFM requires a reliable authentication mechanism for users. EFM supports the following authentication methods:

- SSO integration with an identity provider using SAML 2.0
- SSO integration with an identity provider using OpenID Connect (OIDC)
- SSO with Apache Knox (using Knox as a gateway is not possible)
- Mutual TLS (mTLS) authentication with client certificates
- Proxy authentication
- LDAP authentication

Integrating with an SSO identity provider using OIDC or SAML is the preferred approach for authenticating users. If OIDC is available, Cloudera recommends it as it is usually the easiest to configure and administer.

For service accounts, such as those that programmatically interact with the EFM RESTful API for automation or monitoring purpose, mutual TLS authentication must be used, and can be enabled in addition to another mechanism such as one of the SSO login options.

MiNiFi agents use a different method of authenticating to EFM not controlled by these properties. For more information on MiNiFi agent authentication, see *Agent authentication*.

You need to set the following properties that are shared by all user authentication methods:

```
# User Authentication Properties
efm.security.user.auth.enabled=true
efm.security.user.auth.adminIdentities=admin
efm.security.user.auth.autoRegisterNewUsers=true
efm.security.user.auth.authTokenExpiration=12h
```

Where,

- `efm.security.user.auth.enabled`

Set to true to enable user authentication in EFM. You must also enable a specific method of authentication (`oidc|saml|knox|certificate|proxy`) when user authentication is enabled.

- `efm.security.user.auth.adminIdentities`

A comma separated list of identities needed for initial admins that can configure other user and group access policies in EFM. For more information on initial admins, see *Access control bootstrapping*.

If admin identities contain special characters such as a comma (,), then this alternative property key format can be used:

- `efm.security.user.auth.adminIdentities[0]=CN=admin1, OU=systems, O=cloudera`
- `efm.security.user.auth.adminIdentities[1]=CN=admin2, OU=systems, O=cloudera`
- ...
- `efm.security.user.auth.adminIdentities[n]=CN=adminN, OU=systems, O=cloudera`
- `efm.security.user.auth.autoRegisterNewUsers`

With SAML or OIDC, EFM can optionally create EFM user accounts for authenticated users automatically the first time that they log into EFM, rather than users needing to be explicitly created in EFM prior to first login. Set to true to enable this feature.



**Note:** Knox, mTLS, and Proxy authentication methods do not support this feature and ignore this property.

- `efm.security.user.auth.authTokenExpiration`

With SAML, OIDC, or LDAP, EFM issues access tokens once a user is authenticated. This property specifies the duration for which the token is valid.



**Note:** Knox, mTLS, and Proxy authentication methods do not issue access tokens and ignores this property.

### Related Information

[Agent authentication](#)

[Access control bootstrapping](#)

## Configuring OpenID Connect SSO

Edge Flow Manager (EFM) supports OpenID Connect (OIDC), an industry standard for using a third party as an identity provider for web applications such as EFM. Learn about the properties that you need to set for configuring OIDC SSO.

OIDC is sometimes referred to as OAuth 2 login because it is an authentication protocol that extends the authorization protocols in OAuth 2. In the terminology of OIDC, EFM is the Relying Party and the SSO identity provider is the OpenID Provider.

Before configuring EFM, you should first set up an OIDC client application configuration in your SSO identity provider. For more information about how to do this, see *SSO identity provider setup*.

Before configuring OIDC, EFM should already be configured to use TLS. OIDC authentication requires passing an access token to EFM that is vulnerable to compromise if TLS is not enabled, and therefore EFM does not allow using OIDC authentication without TLS enabled. For more information on enabling TLS in EFM, see *TLS configuration for EFM*.



**Note:** To use OIDC in a clustered EFM environment, sticky sessions should be enabled in the EFM load balancer.

To enable OIDC in EFM, configure the following properties in the `efm.properties` file:

```
efm.security.user.oidc.enabled=true
```

```
efm.security.user.oidc.issuerUri=https://example.okta.com/oauth2
efm.security.user.oidc.clientId=efm
efm.security.user.oidc.clientSecret=abc123...
efm.security.user.oidc.scopes=profile,email,groups
efm.security.user.oidc.usernameAttribute=email
efm.security.user.oidc.displayNameAttribute=name
efm.security.user.oidc.groupAttribute=groups
```

Where,

- `efm.security.user.oidc.enabled`  
Set to true to enable OIDC authentication.
- `efm.security.user.oidc.issuerUri`  
The OpenID Provider base URI. Consult your OpenID Provider documentation for the correct value.
- `efm.security.user.oidc.clientId`  
It must match the configured client application name from the OpenID Provider.
- `efm.security.user.oidc.clientSecret`  
It must match the configured client secret from the OpenID Provider.
- `efm.security.user.oidc.scopes`  
It controls what user attributes are provided from the OpenID Provider to EFM when the user authenticates.
- `efm.security.user.oidc.usernameAttribute`  
It controls which of the user attributes provided to EFM from the OpenID Provider is used as the EFM User identity. Typically this is the email address attribute.
- `efm.security.user.oidc.displayNameAttribute`  
If this property is provided and `efm.security.user.auth.autoRegisterNewUsers` is enabled, it controls which user attribute is mapped to the EFM User Display Name field when the user first logs into EFM.
- `efm.security.user.oidc.groupAttribute`  
It must match the configured Group Claim Name from the OpenID Provider.

EFM supports OpenID Connect Discovery as a way to discover additional details it needs directly from the OpenID Provider. For this to work, the OpenID Provider must host a discovery endpoint at `{issuerUri}/.well-known/openid-configuration`. This is supported by all major, modern SSO vendors offering OpenID Connect. If this endpoint is not available over the network at runtime, EFM fails to start. If this is not possible in your environment, EFM offers a set of staticConfig properties, that you need to configure, as an alternative to OpenID Connect Discovery over the network:

```
efm.security.user.oidc.staticConfig.enabled=false
efm.security.user.oidc.staticConfig.authorizationUri=
efm.security.user.oidc.staticConfig.tokenUri=
efm.security.user.oidc.staticConfig.userInfoUri=
efm.security.user.oidc.staticConfig.jwkSetUri=
```

When these properties are enabled, OpenID Connect Discovery is not performed. So, these act as an override for configuration that would normally be dynamically discovered. The correct values to use can be provided by the OpenID Provider.

### Related Information

[SSO identity provider setup](#)

[TLS configuration for EFM](#)

## Configuring SAML 2.0 SSO

Learn about the properties that you need to set for configuring SAML 2.0 SSO.

Before configuring Edge Flow Manager (EFM) to use SAML, you should first set up an SAML client application configuration in your SSO identity provider. For information about how to do this, see *SSO identity provider setup*.

Before configuring SAML, EFM should already be configured to use TLS. SAML authentication requires passing an access token to EFM that is vulnerable to compromise if TLS is not enabled, and therefore EFM does not allow using SAML authentication without TLS enabled. For more information on enabling TLS in EFM, see *TLS configuration for EFM*.

EFM supports Identity Provider (IdP) and Service Provider (SP) initiated flow.

To enable SAML in EFM, the following properties must be set in the `efm.properties` file:

```
efm.security.user.saml.enabled=true
efm.security.user.saml.metadataLocation=https://example.okta.com/app/example/sso/saml/metadata
efm.security.user.saml.groupAttribute=groups
```

Where,

- `efm.security.user.saml.enabled`  
Set to true to enable SAML authentication.
- `efm.security.user.saml.metadataLocation`  
Identity provider metadata location that is used for dynamic configuration.
- `efm.security.user.saml.groupAttribute`  
The name of a SAML assertion attribute containing group names the user belongs to.



**Note:** With default configuration, Audience URI (SP Entity ID) is `{baseUrl}/saml2/serviceProviderMetadata/efmSaml`, where `baseUrl` depends on the server configuration. Example Entity ID: `https://localhost:10090/efm/saml2/serviceProviderMetadata/efmSaml`.

Optional parameters for further configuration are as follows:

```
efm.security.user.saml.entityId=efmCustomEntityId
efm.security.user.saml.signingCredentials.privateKeyLocation=/path/to/private.key
efm.security.user.saml.signingCredentials.privateKeyPassword=password
efm.security.user.saml.signingCredentials.certificateLocation=/path/to/certificate.crt
efm.security.user.saml.displayNameAttribute=displayNameAttribute
```

Where,

- `efm.security.user.saml.entityId`  
The application-defined unique identifier that is the intended audience of the SAML assertion. It is advisable to use this property when the default `{baseUrl}` dependent Entity ID is undesirable.
- `efm.security.user.saml.signingCredentials.privateKeyLocation`  
Private key location for signing SAML request.
- `efm.security.user.saml.signingCredentials.privateKeyPassword`  
Private key password.
- `efm.security.user.saml.signingCredentials.certificateLocation`  
Certificate location for signing SAML request.
- `efm.security.user.saml.displayNameAttribute`  
Name of the attribute field of the SAML response which can be used to populate the display name of the user.

EFM supports SAML metadata based configuration. SAML metadata is an XML document which contains information necessary for interacting with SAML-enabled identity providers. To obtain such a metadata file, you need to create a new application integration at your identity provider. For more information, see [IdP documentation](#).

To configure the application at the IdP, the following EFM specific properties are needed:

- Single sign-on URL

This property is also known as SAML Assertion Consumer Service (ACS) URL. In case of localhost, it is `https://localhost:10090/efm/login/saml2/sso/efmSaml`.

- Audience URI (SP Entity ID)

By default, this is `https://localhost:10090/efm/saml2/serviceProviderMetadata/efmSaml`.



**Note:** The address, port, and contextPath can be different based on configuration. This value can be overridden by the `efm.security.user.saml.entityId` parameter.

After the successful IdP configuration, you can obtain the `metadata.xml` file. The location of that metadata file should be set in the `efm.security.user.saml.metadataLocation` property. The value can either be a local file system path or a location hosted by the identity provider.

If the identity provider supports SAML request signing and it is enabled, you can set up the required key and certificate in the `efm.security.user.saml.signingCredentials.*` properties. The values should be file system locations, except the `privateKeyPassword`.

Without additional configuration, the user name is equal to the display name in EFM. If you want to provide a different display name, set up an SAML attribute at the IdP and set the name of the SAML attribute in the `efm.security.user.saml.displayNameAttribute` property.

### Related Information

[SSO identity provider setup](#)

[TLS configuration for EFM](#)

## Configuring Knox SSO

Learn about the properties that you need to set for configuring Knox SSO.

### About this task

Edge Flow Manager (EFM) supports Knox SSO for end user authentication with the following limitations:

- Knox cannot be used as a gateway that forwards requests to EFM.
- Knox must be accessible using the same hostname as EFM (a different port is fine). Otherwise, the authentication token issued by Knox, which is stored in a browser cookie, cannot be passed to EFM and EFM authentication fails with a Credentials not present error. This means the following:
  - For EFM clusters, Knox must be accessible through the same load balancer or reverse proxy used to access EFM nodes.
  - For individual, non-clustered EFM instances, Knox must run on the same host as EFM on a different port.

### Before you begin

- You have installed Knox.
- You have installed and secured the EFM server to use TLS. For more information on enabling TLS in EFM, see *TLS configuration for EFM*.

## Procedure

1. Obtain the Knox SSO token signing certificate in PEM format.

This is sometimes the same as the public certificate used for TLS by Knox, and can therefore be obtained using OpenSSL against the Knox host:

```
openssl s_client -servername HOSTNAME -connect HOST:PORT | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > Knox_server_cert_in_pem_file_format.pem
```

Sometimes, however, a different signing keypair is configured specifically for Knox SSO signing and must be obtained from that server configuration. Once you have the Knox SSO token signing certificate, save the certificate on the EFM server and ensure that the file is readable by the user running the EFM process.

2. Update the EFM configuration file with the following properties:

```
efm.security.user.knox.enabled=true
efm.security.user.knox.url=https://knox.mycompany.com:8443/gateway/knoxsso/api/v1/webssso
efm.security.user.knox.publicKey=path/to/knox_server_cert_in_pem_file_format.pem
efm.security.user.knox.cookieName=hadoop-jwt
efm.security.user.knox.audiences=
```

3. In Cloudera Manager, that manages Knox, update the Knox topology for the Knox SSO service to add the EFM hostname (or EFM load balancer hostname when clustered) to the authorized redirect URLs.

For example:

```
<service>
  <role>KNOXSSO</role>
  ...
  <param>
    <name>knoxsso.redirect.whitelist.regex</name>
    <value>^https?:\/\/(efm\.hostname\.com|localhost|127\.0\.0\.1|::1):[0-9]\.*$</value>
  </param>
</service>
```

4. Restart EFM and Knox.

## Related Information

[TLS configuration for EFM](#)

## Configuring mutual TLS authentication

Edge Flow Manager (EFM) supports mutual TLS (mTLS) authentication in which the client provides the EFM server with a client certificate as part of the TLS handshake, and the client certificate provides the identity of the client. Learn about the properties that you need to set for configuring mTLS.

EFM requires using mTLS for MiNiFi agent authentication as described in *Agent authentication*, and optionally offers using this form of authentication for end users as well. For example, web browsers support loading client certificates that can be passed to a web server when using a web application.

For service accounts, such as those that programmatically interact with the EFM RESTful API for automation or monitoring purpose, mutual TLS authentication must be used, and can be enabled in addition to another mechanism such as one of the SSO login options.

Before configuring mTLS authentication, EFM should already be configured to use TLS. For more information on enabling TLS in EFM, see *TLS configuration for EFM*.

To enable mTLS authentication in EFM, set the following property in the `efm.properties` file:

```
efm.security.user.certificate.enabled=true
```

Again, this can be enabled simultaneously with another method of authentication, such as OIDC or SAML SSO, in the case that RESTful API service accounts use mTLS but web users login with SSO using username and password credentials.

EFM trusts client certificates signed by a Certificate Authority (CA) or intermediate authority present in the EFM truststore. For information on generating certificates, see *TLS keys and certificates*. For more information about configuring the EFM truststore, see *TLS configuration for EFM*.

When a client authenticates using mTLS, the client certificate DN is the user identity. For example, `CN=monitoring-service, OU=systems, O=cloudera`. Therefore, an admin must add a user with an identity equal to the client certificate DN in order to grant this type of user access to parts of EFM. For more information on creating users and assigning policies, see *Access control policies*.

Here is an example of accessing the EFM RESTful API using curl when mTLS authentication is enabled:

```
curl \
  --cacert /path/to/ca-trust-cert.pem \
  --cert /path/to/user-cert.pem \
  --key /path/to/user-key.pem \
  --pass password \
  https://localhost:10090/efm/api/access

# JSON response
{"identity": "CN=user, OU=systems, O=cloudera", "anonymous": false, "globalPermissions": {"accessAdministration": false}}
```

### Related Information

[Agent authentication](#)

[TLS configuration for EFM](#)

[TLS keys and certificates](#)

[Access control policies](#)

## Configuring proxy authentication

Learn about the properties that you need to set for configuring proxy authentication.



**Important:** Proxy Authentication is being deprecated and will be removed in the next EFM release.

Edge Flow Manager (EFM) can be configured to trust an HTTP reverse proxy to authenticate users externally and pass the user details with each request. This is useful in some SSO environments in which OIDC or SAML is not an option, and some gateway, proxy, or central web service handles user authentication to multiple backend services.

You need to set the following configuration options for using proxy authentication:

```
efm.security.user.proxy.enabled=true
efm.security.user.proxy.headerName=x-webauth-user
efm.security.user.proxy.ipWhitelist=
efm.security.user.proxy.dnWhitelist[0]=
```

Where,

- `efm.security.user.proxy.enabled`  
Whether proxy authentication is enabled.

- **efm.security.user.proxy.headerName**

Case-insensitive header name set by the proxy holding the end user identity.

- **efm.security.user.proxy.ipWhitelist**

Limits trusted proxy IP addresses to prevent spoofing the user header. Comma-separated or multiple properties using the ipWhitelist[n] syntax:

- **efm.security.user.proxy.ipWhitelist[0]=**
- **efm.security.user.proxy.ipWhitelist[1]=**
- **efm.security.user.proxy.dnWhitelist[0]..[n]**

Limits trusted proxy client certificate DNs to prevent spoofing the user header. Use the dnWhitelist[n] syntax as it is common for certificate DNs to contain commas.

If you are using proxy authentication, Cloudera strongly recommends that you use either the DN whitelist or IP whitelist feature to specify trusted reverse proxies. If you are not using a whitelist, it is assumed that you are using some other networking mechanism to ensure that all authenticated requests are coming from a trusted client, such as only binding the EFM server to localhost and running the authenticating proxy on the same machine on a different network interface.

The following is a curl example of passing the proxy user header to the /api/access endpoint that returns the recognized current user:

```
curl -H "X-WEBAUTH-USER: alice"
      https://localhost:10090/efm/api/access
```

```
# Response:
{"identity": "alice", "anonymous": false}
```

As you can see, this header can be added to any request, which is why DN whitelisting, IP whitelisting, or localhost binding should be used with proxy authentication.

## Configuring LDAP authentication

Learn about the properties that you need to set for configuring LDAP authentication.

### Before you begin

Edge Flow Manager (EFM) supports username/password authentication using LDAP. Before configuring LDAP, ensure that EFM is already configured for TLS and user authentication is enabled with the following properties:

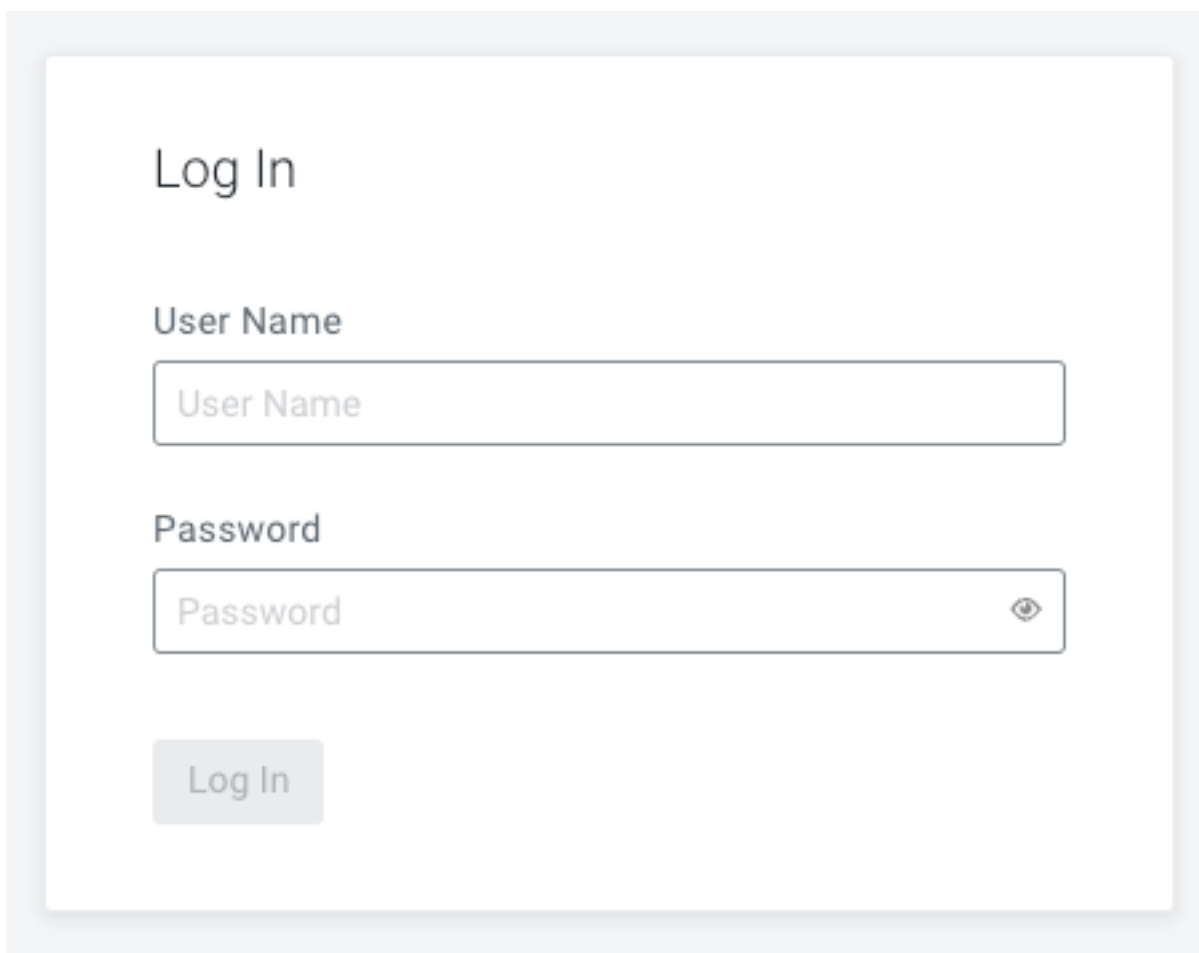
- **efm.security.user.auth.enabled**  
Set to true to enable EFM user authentication. You need to configure an authentication method such as MTLS, OIDC, SAML, KNOX, or LDAP.
- **efm.security.user.auth.authTokenExpiration**  
EFM issues access tokens when a user is authenticated. Specify the duration of the token's validity. The default value is 12 h.

### Procedure

1. Update the EFM configuration file with the following LDAP authentication properties.

- **efm.security.ldap.auth.enabled**  
Use it to enable LDAP authentication in EFM.
- **efm.security.ldap.auth.searchFilter**  
Use this filter to search for users against efm.security.ldap.user.searchBase (for example: uid={0}). The user's specified name is inserted into '{0}'.

2. Configure the following LDAP user search properties.
  - **efm.security.ldap.user.searchBase**  
Define the base DN to search for users (for example: ou=users,o=efm).
  - **efm.security.ldap.user.objectClass**  
Specify the object class used to identify users (for example: person).
  - **efm.security.ldap.user.searchScope**  
Set the search scope for user searches. Possible values are ONE\_LEVEL, OBJECT, or SUBTREE.
  - **efm.security.ldap.user.identityAttribute**  
Optionally, you can specify the attribute used to extract user identity (for example: cn). If not set, the entire DN is used.
3. Once EFM is started with the above configuration, open the site to get redirected to the login page.



On this page, you can log in with your username (as defined by the LDAP identity attribute value) and your password.

#### Related Information

[TLS configuration for CEM](#)

[Initial admin identities](#)

[Managing user groups using LDAP](#)

## Access control policies

Edge Flow Manager (EFM) uses role based access control at the agent class level. This means that everything in EFM is tied to agent classes. On top of restricting access to specific endpoints and functionality, all data returned by EFM is also filtered based on the agent classes the given user has access to. All entities, such as flows, events, and parameters are mapped to agent classes by EFM so access can be determined based on that.

EFM recognizes agent class viewer, designer, and operator roles along with the global administrator role. For more information, see *Policies for agent class roles*.

### Related Information

[Policies for agent class roles](#)

## Access control bootstrapping

After Edge Flow Manager (EFM) is secured, you have to set up and configure the access control policies and permissions in a system.

Access control bootstrapping involves two key aspects: authentication and authorization.

- Authentication ensures that EFM can verify the identity of the client or the user using configured credentials, as detailed in the *User authentication* documentation.
- Authorization determines whether the authenticated user is allowed to perform specific actions on the requested data. Authorization in EFM is driven by local access control policies. To grant these policies to users, you need to establish an initial admin account with the authority to set up user access. This section provides an overview of that process.

### Related Information

[User authentication](#)

## Initial admin identities

To be able to register new users and assign roles to them, first you need to set up an initial admin identity. This initial admin has comprehensive access rights and can assign roles and administrative privileges to others as needed. After the initial setup, you can remove this configuration if needed.

You can configure initial admin identities in the `efm.properties` file.

- `efm.security.user.auth.adminIdentities`

A comma separated list of identities needed for initial admins that can configure other user and group access policies in Edge Flow Manager (EFM). For example, `efm.security.user.auth.adminIdentities=admin@cloudera.com`. If admin identities contain special characters such as a comma (,), you can use the following alternative property key format:

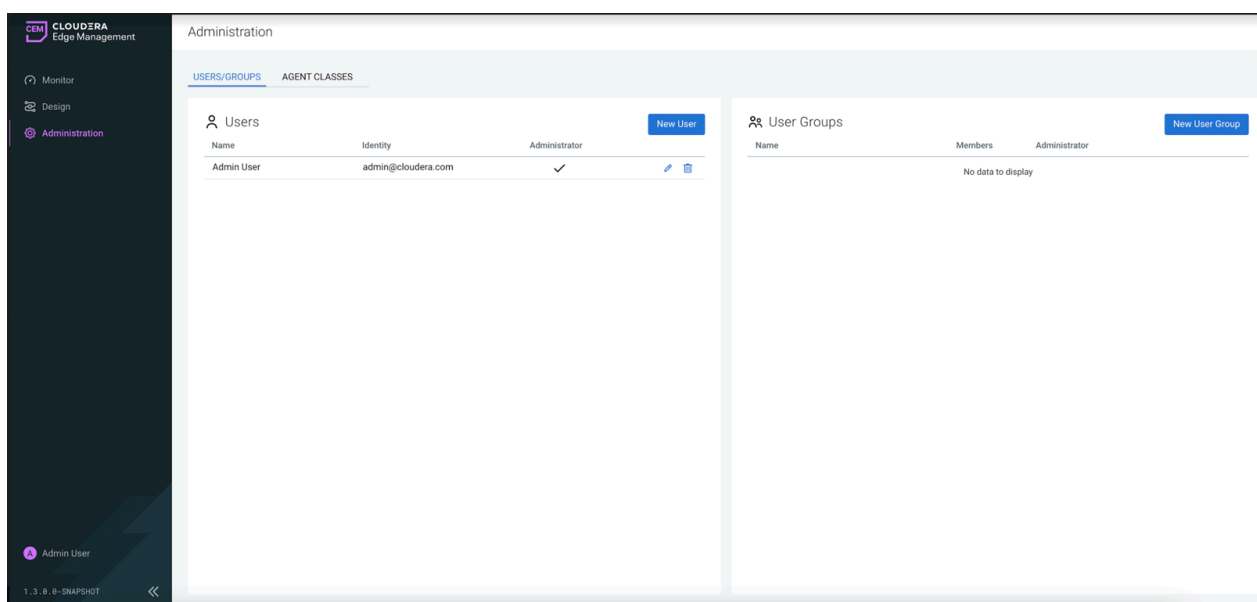
- `efm.security.user.auth.adminIdentities[0]=CN=admin1, OU=systems, O=cloudera`
- `efm.security.user.auth.adminIdentities[1]=CN=admin2, OU=systems, O=cloudera`
- ...
- `efm.security.user.auth.adminIdentities[n]=CN=adminN, OU=systems, O=cloudera`

Starting EFM with this property and logging in with the predefined identity grants you administrator access.

- `efm.security.user.auth.autoRegisterNewUsers`

It controls whether you create each user manually in the Administration page, or the system should do this automatically after the first login attempt for each user. This is a convenience functionality supported by the OIDC and SAML methods.

The following image shows the Administration page after initial admin login:



You can also configure initial admin permissions at group level. To do this, you need to set the following properties:

```
efm.security.user.auth.groups.manager=EXTERNAL
efm.security.user.auth.groups.adminIdentities=Admin-Group-Name
```

Where,

- `efm.security.user.auth.groups.manager`

It indicates that user group management is handled externally by the IdP or LDAP. Its value can be INTERNAL, EXTERNAL, or LDAP. The default value is INTERNAL.

- `efm.security.user.auth.groups.adminIdentities`

A comma separated list of group names needed for initial admins that can configure other user and group access policies in EFM. If group names contain special characters such as a comma (,), you can use the following alternative property key format:

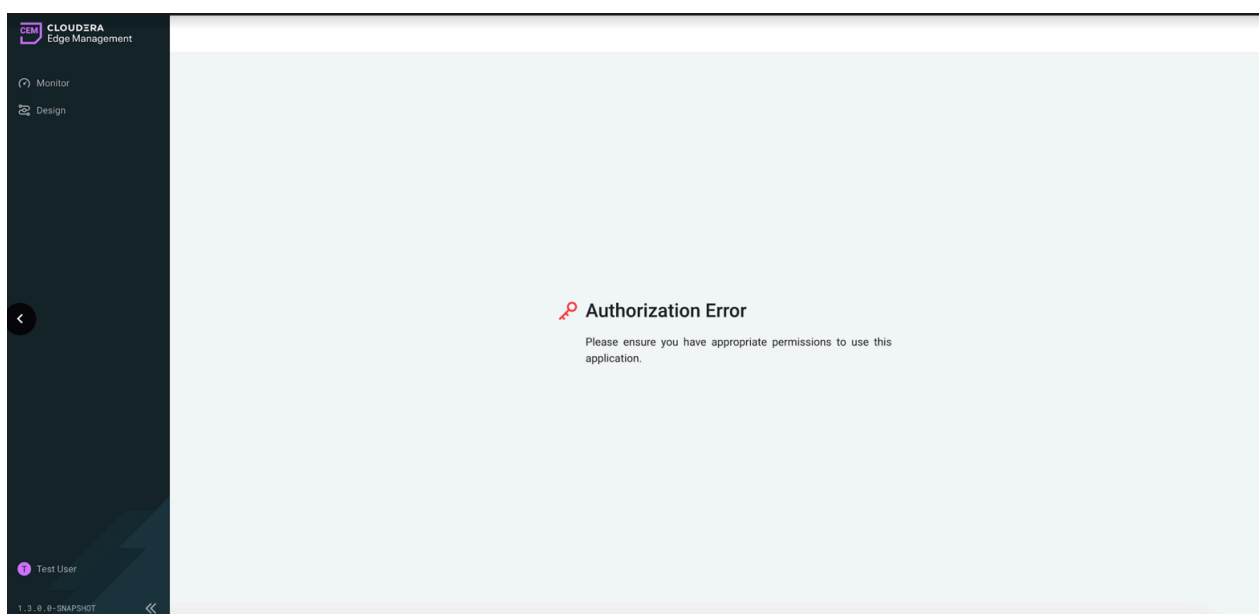
- `efm.security.user.auth.groups.adminIdentities[0]=Group,1`
- `efm.security.user.auth.groups.adminIdentities[1]=Group,2`
- `efm.security.user.auth.groups.adminIdentities[N]=Group,N`

## First login with a new user

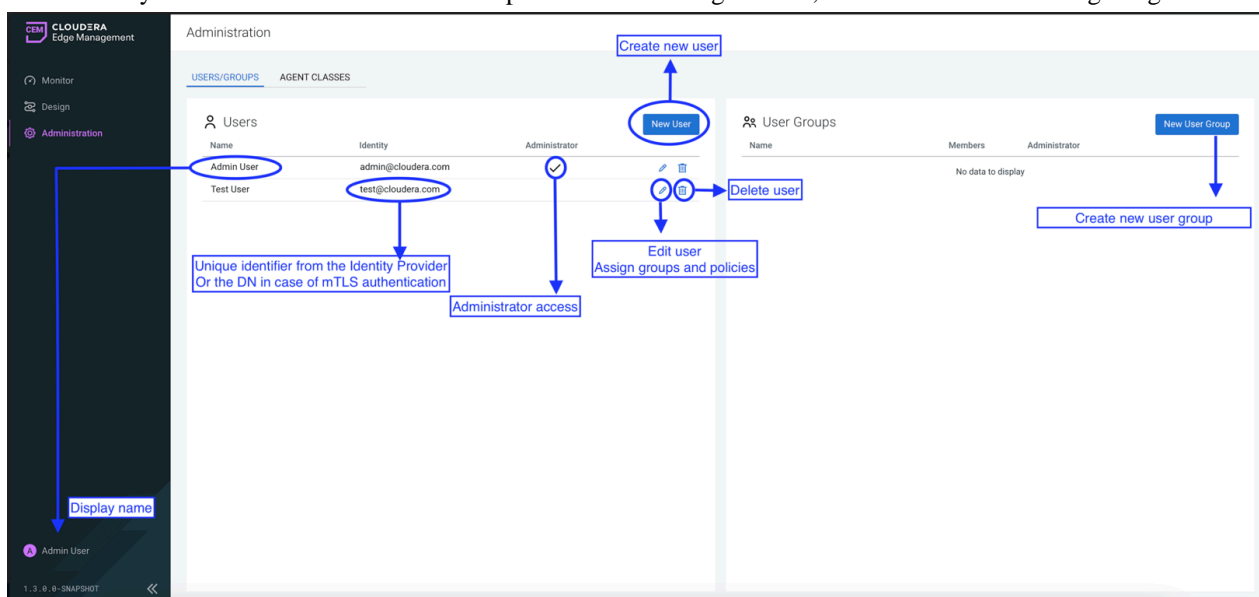
When a new user logs in for the first time, even though they are recognized by the identity provider, Edge Flow Manager (EFM) treats them as an unknown user. While the login itself may be successful, an authorization error occurs because the user does not yet have access to any EFM resources or functionalities.

An administrator needs to create an EFM user with a matching identity and grant that user access to an agent class(s).

The following image shows the Administration page after initial admin login:



With the `efm.security.user.auth.autoRegisterNewUsers` property set to true, after the first login, the EFM user is automatically available in the admin view and policies can be assigned to it, as shown in the following image:



The user rights are assigned to the users based on the configured state when the login happens. This means that if policies are changed by an administrator, a re-login is required to issue a new token based on the new policies. After this, the user has access to filtered data based on the assigned policies. Everything is assigned based on agent class, so if users have no assigned role for a given agent class, they do not see anything related to it.

## User management

Authorization in Edge Flow Manager (EFM) is linked to roles assigned to agent classes. These roles must be associated with authenticated users. While the identity provider can recognize these users to support the authentication process, EFM remains responsible for directly managing the permissions of each valid user.

The following sections explain how to manage users.

## Creating users

Learn how to create a new user and define a unique identity and display name for that user.

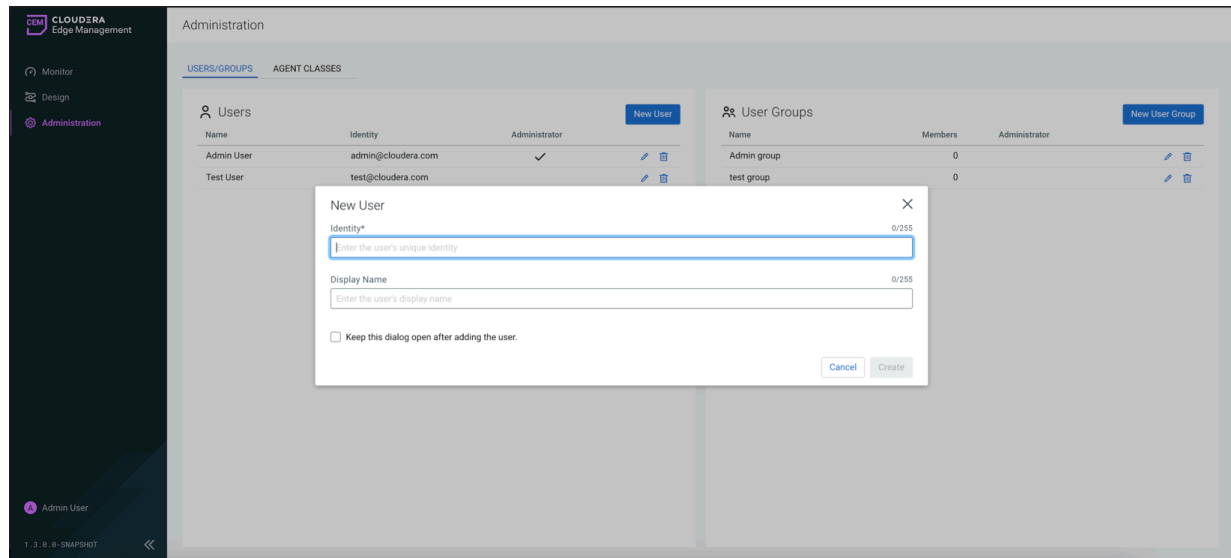
### Procedure

1. Click Administration from the left navigation in the Cloudera Edge Management UI.

The Administration page appears.

2. Click New User in the Users/Groups tab to create a new user.

The New User dialog appears.



3. Enter the unique identity of the user.

The identity should be the unique identifier for the user provided by the identity provider, typically an email address. When using mutual TLS authentication, the identity should be the DN of the client certificate.

4. Enter the display name of the user.

The display name can be set to anything and represents the user in other places where the user name appears in the application. If you create a new user before the initial login, you can prevent the authorization error described above.

5. Click Create to create the new user.

## Editing users

Learn how you can assign policies and roles to a user you created.

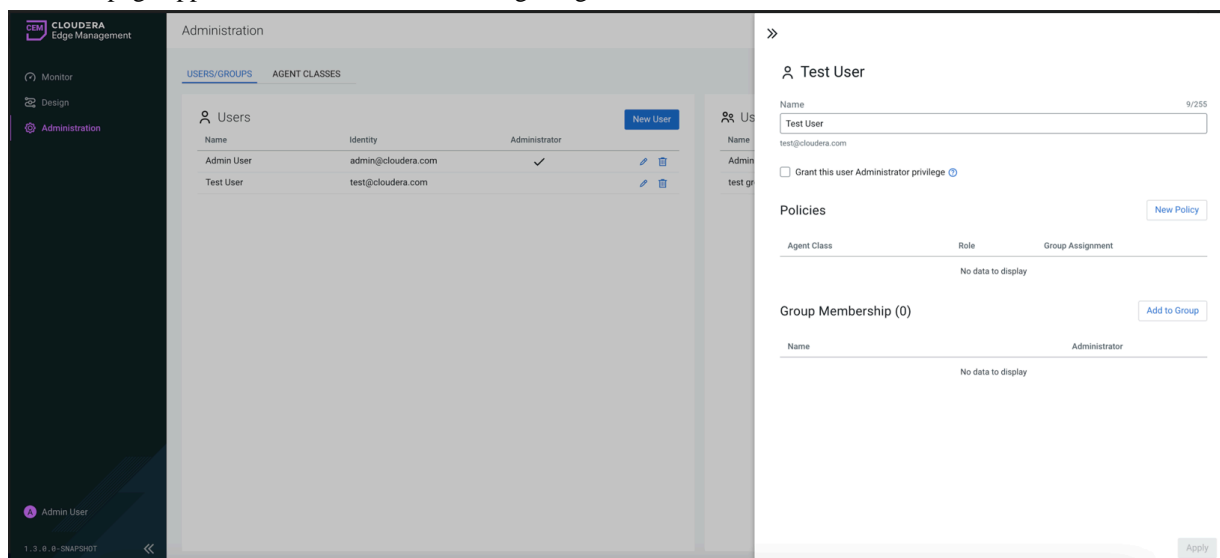
### Procedure

1. Click Administration from the left navigation in the Cloudera Edge Management UI.

The Administration page appears.

2. In the Users/Groups tab, click the [edit] icon next to a user to edit the user.

The user page appears as shown in the following image:



3. Edit the name of the user.
4. Click the Grant this user Administrator privilege checkbox to grant administrator privileges to the given user.
5. Click New Policy to assign policies to the given user (specific role to the given agent class).
6. Click Add to Group to add a given user to a group (all of the roles from the group are inherited by the user).



**Note:**

Adding and removing members from groups are disabled when the `efm.security.user.auth.groups.manager` EFM property is set to *EXTERNAL* or *LDAP*.

7. Click Apply.

## User group management

Managing access through groups is very similar to direct user access management. This is preferred when you need to manage access for multiple users. Users inherit changes made to the group, so no direct changes are needed on the individual user level.

The following sections explain how to manage user groups.

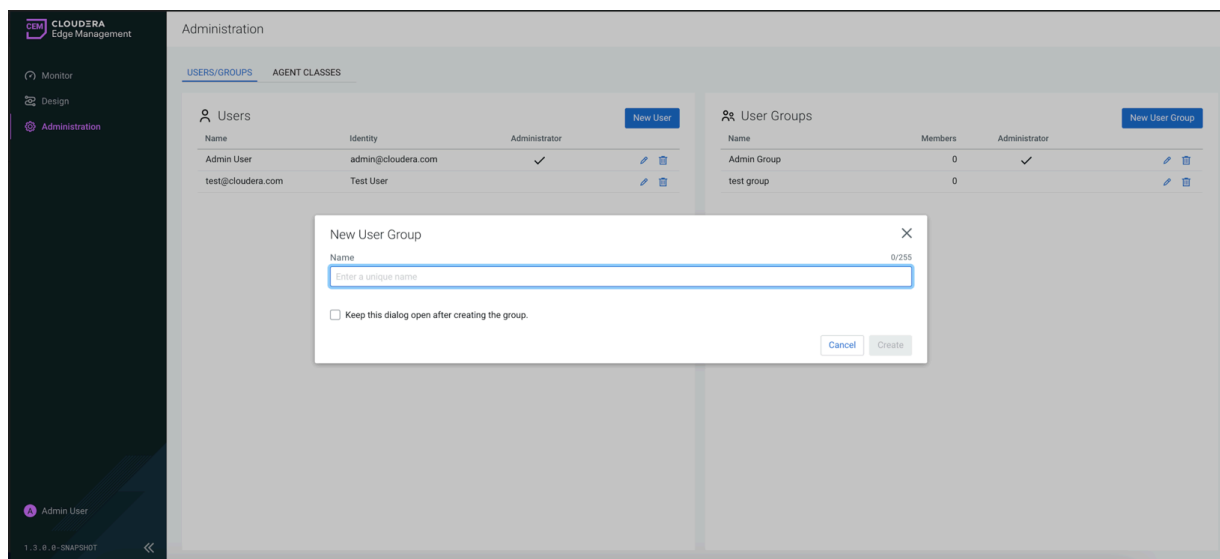
### Creating user groups

Learn how to create a new user group.

#### Procedure

1. Click Administration from the left navigation in the Cloudera Edge Management UI.  
The Administration page appears.

- Click New User Group, in the Users/Groups tab to create a new group.



- Enter a name for the user group.



**Note:** The name must be unique because it represents the group in other places where the group name appears in the application.

- Click Create.



**Note:** Group creation is disabled when the `efm.security.user.auth.groups.manager` EFM property is set to *EXTERNAL* or *LDAP*.

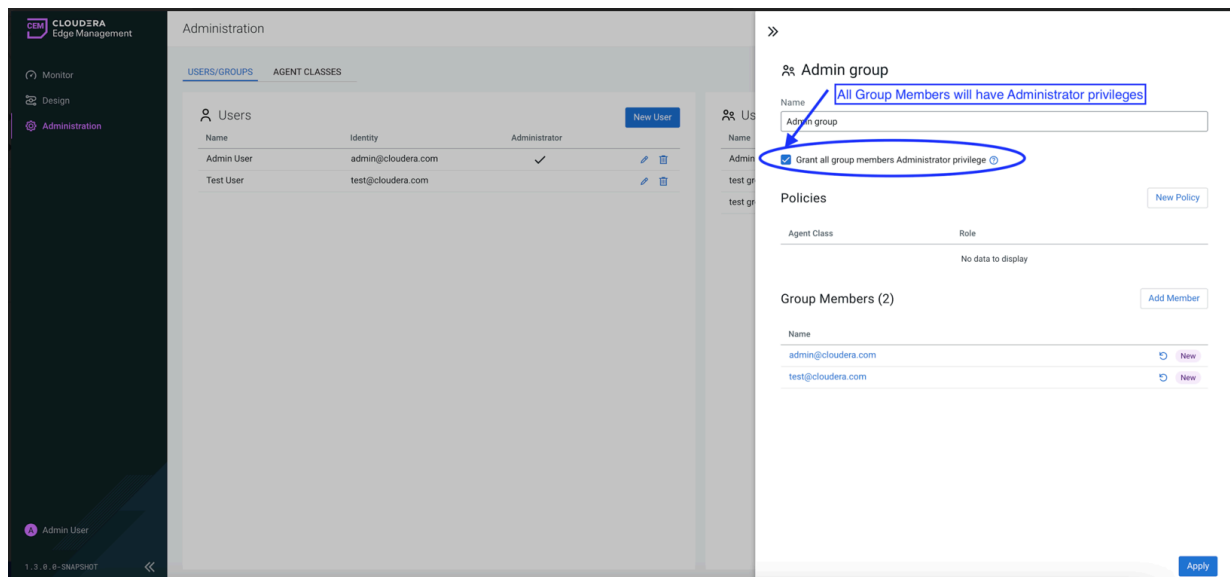
## Editing user groups

Learn how to assign policies and roles to user groups and add users these groups.

### Procedure

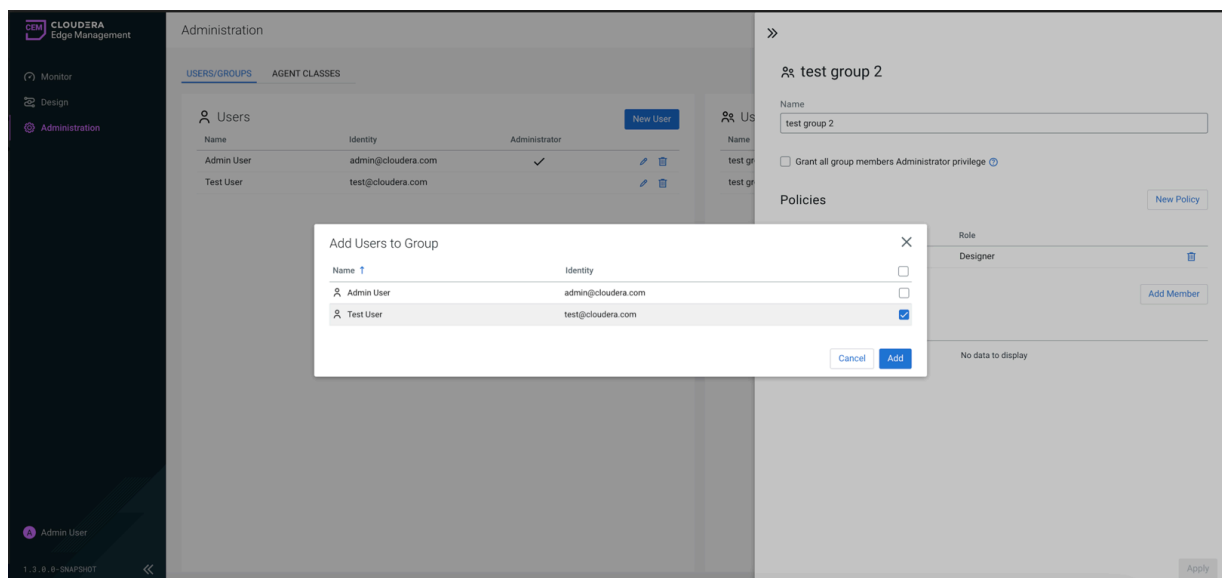
- Click Administration from the left navigation in the Cloudera Edge Management UI.  
The Administration page appears.
- In the Users/Groups tab, click the [edit] icon next to a user group to edit the user group.
- Edit the name of the user group.

- Click the Grant all group members Administrator privilege checkbox to grant administrator privileges to all group members.



- Click New Policy to assign policies to the group (specific role to the given agent class).

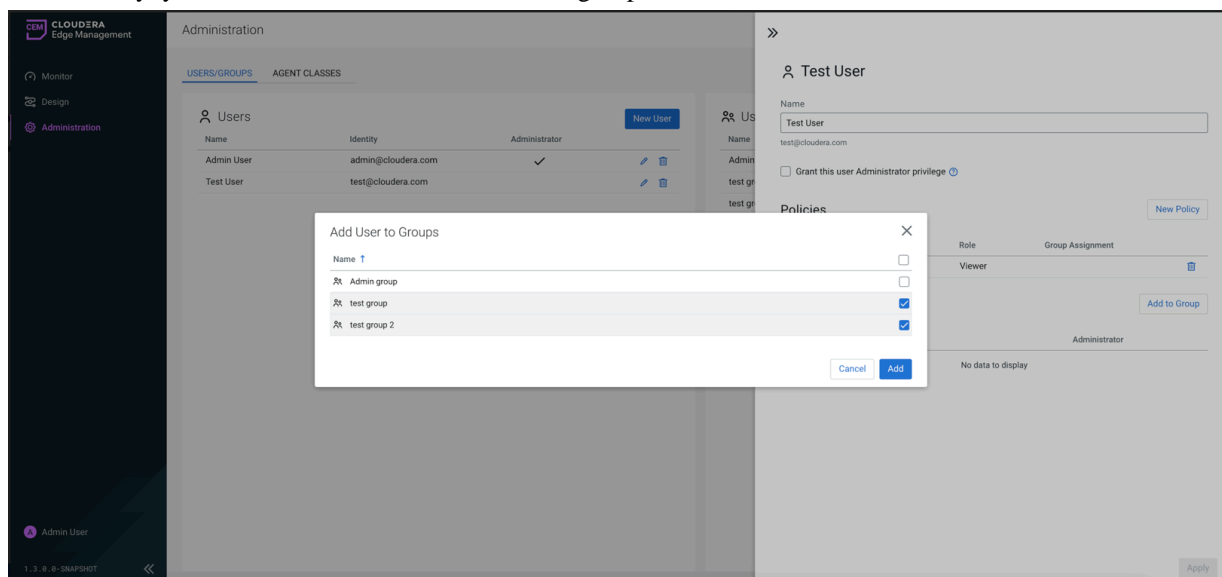
6. Click Add Members to add users to the group (all of the roles from the group are inherited by the members)



#### Note:

Adding and removing members from groups are disabled when the `efm.security.user.auth.groups.manager` EFM property is set to *EXTERNAL* or *LDAP*.

Alternatively, you can edit a user to add the user to a group.



7. Click Apply.

## Managing user groups using IdP

Learn how to enable user group management through your Identity Provider (IdP).

### About this task

In the case of OIDC / SAML providers, user groups and assignments can be exclusively managed externally using the IdP. This means that user groups can only be added or modified by the IdP, and the assignment of users to groups can solely be performed through the IdP. However, you still retain the capability to assign policies to users and groups within Edge Flow Manager (EFM).

## Before you begin

Ensure that you have configured the following parameters appropriately:

### For OIDC:

- `efm.security.user.oidc.scopes`
- `efm.security.user.oidc.groupAttribute`

For more information, see *Configuration of OpenID Connect SSO*.

### For SAML:

- `efm.security.user.saml.groupAttribute`

For more information, see *Configuration of SAML 2.0 SSO*.

To enable Group Management by IdP, set the following properties in the `efm.properties` file:

```
efm.security.user.auth.groups.manager=EXTERNAL
efm.security.user.auth.groups.filter=.*
```

Where,

- `efm.security.user.auth.groups.manager`

It indicates how user group management is handled: either internally or externally. When the value is set to `EXTERNAL`, user group management is handled by IdP. When the value is set to `INTERNAL`, EFM disregards any group information received from the IdP. The default value is `INTERNAL`.

- `efm.security.user.auth.groups.filter`

Users might be assigned to multiple groups on the IdP side, but EFM may only need to consider a few of these groups. You can use a regular expression to define which groups should be registered in EFM. The regex allows you to specify a pattern, and only the groups that match the defined pattern will be registered in EFM.



### Note:

User groups and individual user-to-group assignments are updated when the user logs in to EFM. As a result, any changes to group-related data in the IdP will only become visible in EFM after the user logs in.

## Related Information

[Configuration of OpenID Connect SSO](#)

[Configuration of SAML 2.0 SSO](#)

[Initial admin identities](#)

## Managing user groups using LDAP

Learn how to enable user group management using the Lightweight Directory Access Protocol (LDAP).

### About this task

Edge Flow Manager (EFM) can be configured to retrieve group assignments from LDAP. This implies that user groups can only be added or modified using LDAP, and user-group assignments can exclusively be managed through LDAP. However, you still retain the ability to assign policies to users and groups within EFM.

### Procedure

1. Enable user group management by LDAP.

To enable user group management using LDAP, set the following property in the `efm.properties` file:

```
efm.security.user.auth.groups.manager=LDAP
```

## 2. Configure LDAP integration.

For a successful integration, make sure that the following properties are accurately configured.

### Basic properties:

#### **efm.security.ldap.url**

Enter a comma-separated list of LDAP server URLs (for example: ldap://<hostname>:<port>)

#### **efm.security.ldap.syncInterval**

Specify the duration of time between synchronizing users and groups (for example: 30 m). The minimum allowable value is 30 seconds and the default value is 30 minutes.

#### **efm.security.ldap.managerDn**

Provide the Distinguished Name (DN) of the manager used to bind to the LDAP server to search for users and groups.

#### **efm.security.ldap.managerPassword**

Enter the password of the manager used to bind to the LDAP server to search for users and groups.

#### **efm.security.ldap.authenticationStrategy**

Select an authentication strategy for connecting to the LDAP server.

Possible values:

- ANONYMOUS
- SIMPLE
- LDAPS
- START\_TLS

#### **efm.security.ldap.referralStrategy**

Select a strategy for handling LDAP referrals.

Possible values:

- FOLLOW (default value, automatically follow any referrals)
- IGNORE (ignore referrals)
- THROW (throw an error if a referral occurs)

#### **efm.security.ldap.connectTimeout**

Set the duration of the connection timeout. The default value is 10 seconds.

#### **efm.security.ldap.readTimeout**

Set the duration of the read timeout. The default value is 10 seconds.

#### **efm.security.ldap.pageSize**

Optionally, you can set the page size when retrieving users and groups. If not specified, no paging is performed.

#### **efm.security.ldap.groupMembershipCaseSensitive**

Indicate whether group membership decisions should be case-sensitive. The default value is false.

### TLS properties

These properties are required when efm.security.ldap.authenticationStrategy is set to LDAPS or START\_TLS.

#### **efm.security.ldap.tls.keyStore**

Specify the path to the keystore used when connecting to LDAP.

#### **efm.security.ldap.tls.keyStorePassword**

Set the password for the keystore used when connecting to LDAP.

#### **efm.security.ldap.tls.keyStoreType**

Define the type of the keystore used when connecting to LDAP. Possible values are BCFKS, PKCS12, or JKS.

**efm.security.ldap.tls.trustStore**

Provide the path to the truststore used when connecting to LDAP.

**efm.security.ldap.tls.trustStorePassword**

Set the password for the truststore used when connecting to LDAP.

**efm.security.ldap.tls.trustStoreType**

Specify the type of the truststore used when connecting to LDAP. Possible values are BCFKS, PKCS12, or JKS.

**efm.security.ldap.tls.tlsProtocol**

Select the protocol to use when establishing LDAP connections. Possible values are TLS, TLSv1\_1, or TLSv1\_2.

**efm.security.ldap.tls.shutdownGracefully**

Determine whether TLS should be shut down gracefully before the target context is closed. The default setting is false and applies exclusively to the START\_TLS authentication strategy.

**Search properties**

Synchronizing users and groups is possible through user, group, or both user and group searches, depending on your LDAP scheme and configuration. However, it is essential to configure at least one of these properties to enable user and group synchronization.

User search

**efm.security.ldap.user.searchBase**

Specify the base DN for searching users (for example: ou=users, o=efm).

**efm.security.ldap.user.objectClass**

Define the object class for identifying users (for example: person).

**efm.security.ldap.user.searchScope**

Set the search scope for searching users. Possible values are ONE\_LEVEL, OBJECT, or SUBTREE.

**efm.security.ldap.user.searchFilter**

Apply a filter for searching users in the User Search Base (for example: (memberof=cn=team1,ou=groups,o=efm)).

**efm.security.ldap.user.identityAttribute**

Optionally, you can specify the attribute used to extract user identity (for example: cn). If not set, the entire DN is used.

**efm.security.ldap.user.groupNameAttribute**

Optionally, you can define the attribute used to establish group membership (for example: memberof). If not set, group memberships are calculated based on efm.security.ldap.group.memberAttribute. If set, the value of this property is the name of the attribute in the user LDAP entry that associates them with a group. The value of this user attribute could be a DN or group name for instance. The expected value is configured in efm.security.ldap.user.referencedGroupAttribute.

**efm.security.ldap.user.referencedGroupAttribute**

If not set, the value of the attribute defined in efm.security.ldap.user.groupNameAttribute is expected to be the full DN of the group. If set, this property defines the attribute of the group LDAP entry to which the value of the attribute defined in

`efm.security.ldap.user.groupNameAttribute` refers (for example: `name`). The use of this property requires that `efm.security.ldap.group.searchBase` is also configured.

Group search

**`efm.security.ldap.group.searchBase`**

Specify the base DN for group searches (for example: `ou=users,o=efm`).

**`efm.security.ldap.group.objectClass`**

Identify the object class for groups (for example: `groupOfNames`).

**`efm.security.ldap.group.searchScope`**

Define the search scope for group searches. Possible values are `ONE_LEVEL`, `OBJECT`, or `SUBTREE`.

**`efm.security.ldap.group.searchFilter`**

Optionally, you can filter for group searches against `efm.security.ldap.group.searchBase`.

**`efm.security.ldap.group.nameAttribute`**

Optionally, you can extract group names using the specified attribute (for example: `cn`). If not set, the entire DN is used.

**`efm.security.ldap.group.memberAttribute`**

Optionally, you can specify an attribute for defining group membership (for example: `member`). If not set, group membership will not be calculated through the groups. Instead, it relies on group membership defined using `efm.security.ldap.user.groupNameAttribute`. If set, the value of this property is the name of the attribute in the group LDAP entry that associates users with groups. The value of this group attribute can be a DN or `memberUid`, depending on your configuration (for example: `member: cn=User 1,ou=users,o=efm` vs. `memberUid: user1`).

**`efm.security.ldap.group.referencedUserAttribute`**

If not set, the value of the attribute defined in `efm.security.ldap.group.memberAttribute` is expected to be the full DN of the user. If set, this property defines the attribute of the user LDAP entry referenced by `efm.security.ldap.group.memberAttribute` (for example: `uid`). Using this property requires that `efm.security.ldap.user.searchBase` is also configured (for example: `member: cn=User 1,ou=users,o=efm` vs. `memberUid: user1`).

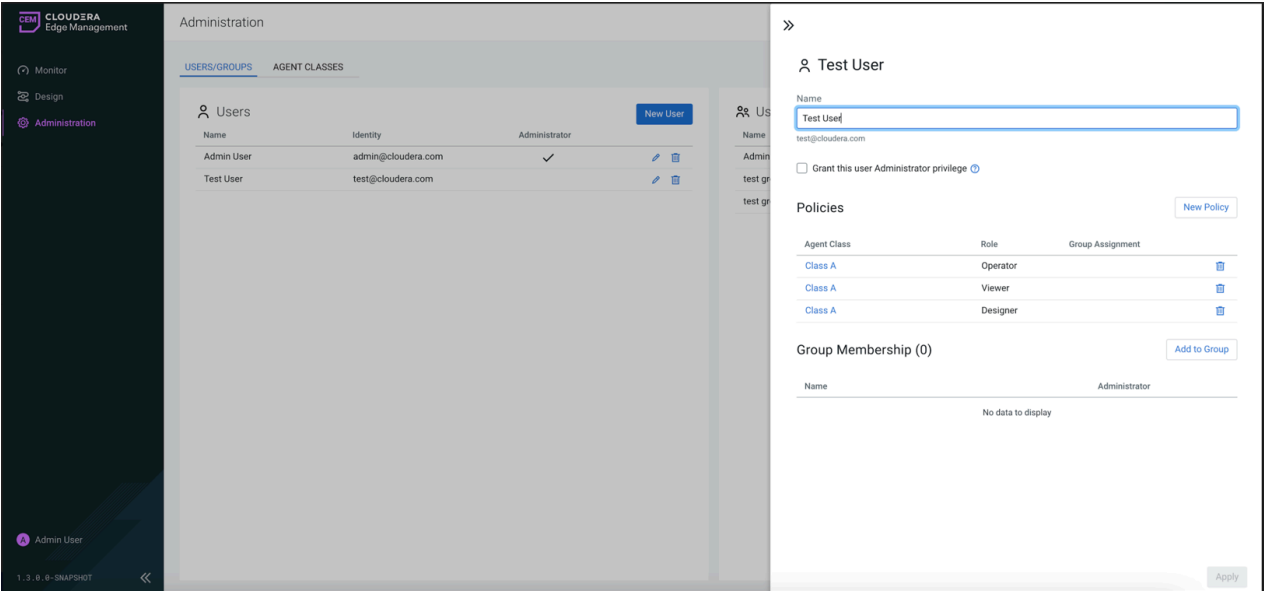
## Agent class roles policies

Learn about the different policies of an agent class that you can assign to users, and understand the specific functions of each policy.

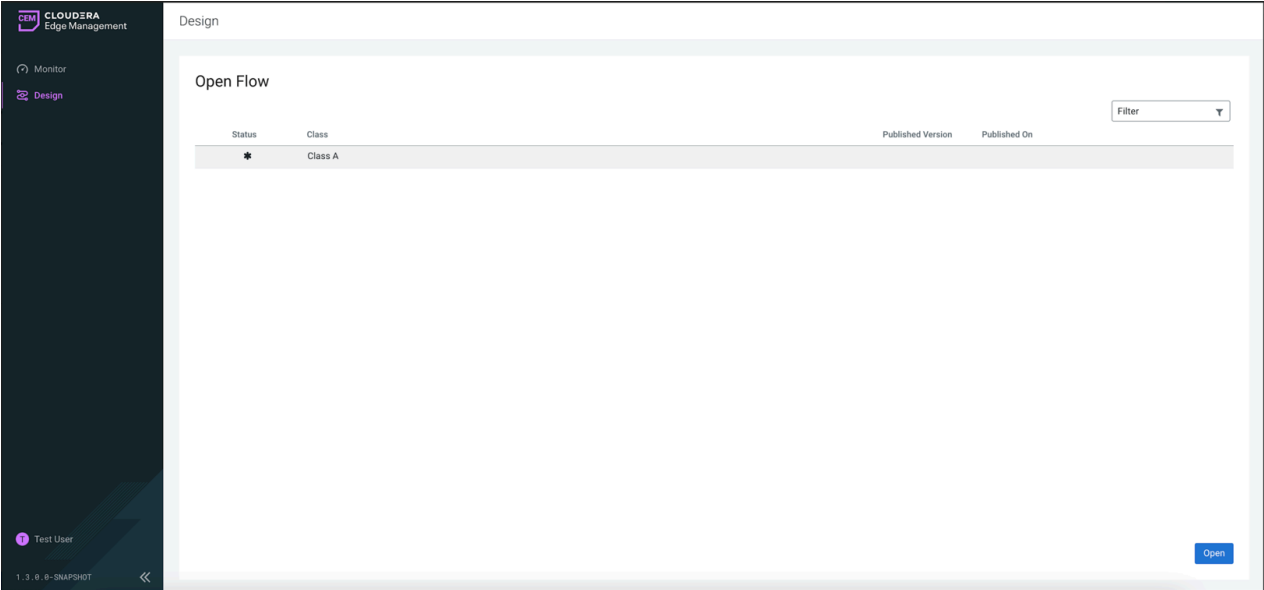
Policies are linked to agent classes, and EFM currently supports three roles:

- A Viewer can view designs and monitor events for the assigned agent class, but can not edit or publish the flows. Essentially, viewers cannot make any modifications in the system.
- A Designer can view and edit designs, monitor events for the assigned agent class, but cannot publish flows.
- An Operator can monitor events, publish designs for the assigned agent class (without editing), and can modify flow and parameter mappings.

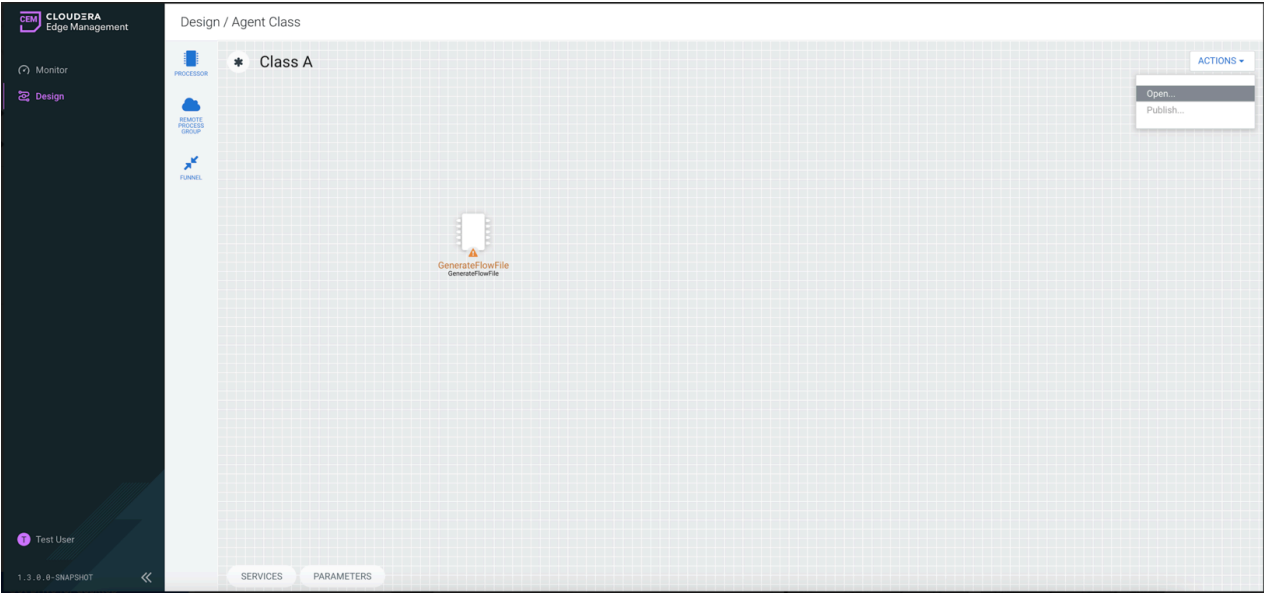
Depending on the use case, it is possible to have multiple roles for the same user. For example, if you do not want to differentiate between a designer and an operator for a given agent class (you want the same person to edit and publish the flow), you can do that by assigning both roles to the user. The following image shows that one user can have multiple roles on the same agent class:



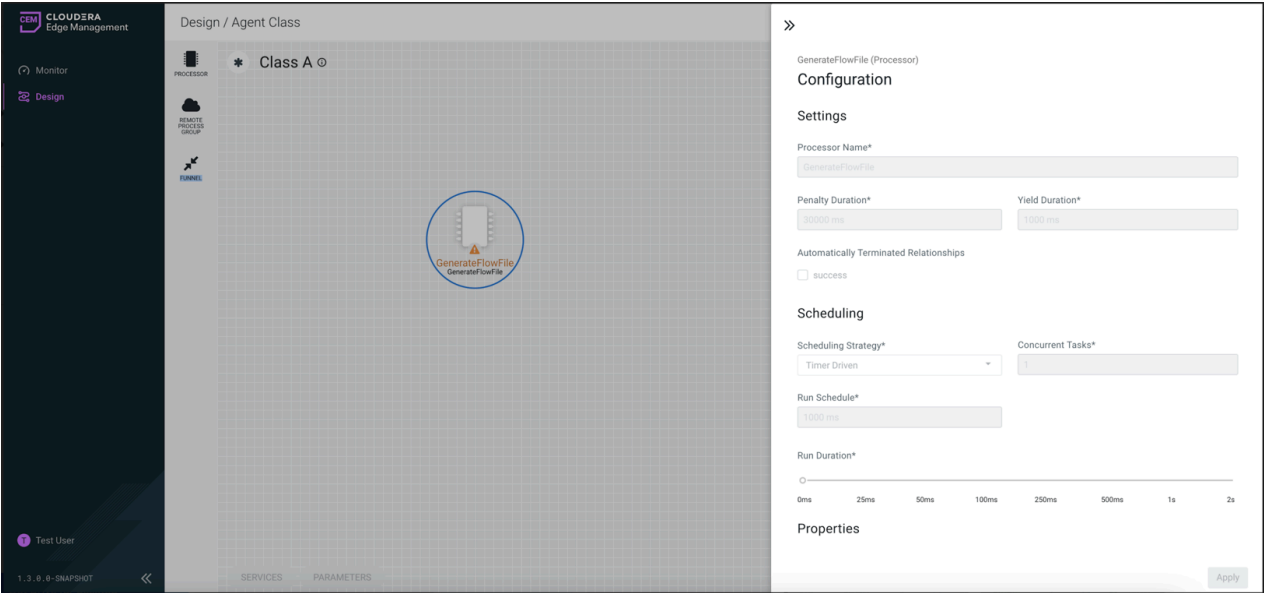
The flow designs and event lists are filtered based on authorization. So, only those items are visible, which are related to the agent class the user is assigned to. In the following case, only Class A is visible as roles are assigned only to Class A for this user (though Class B and Class C exist as well for this user).



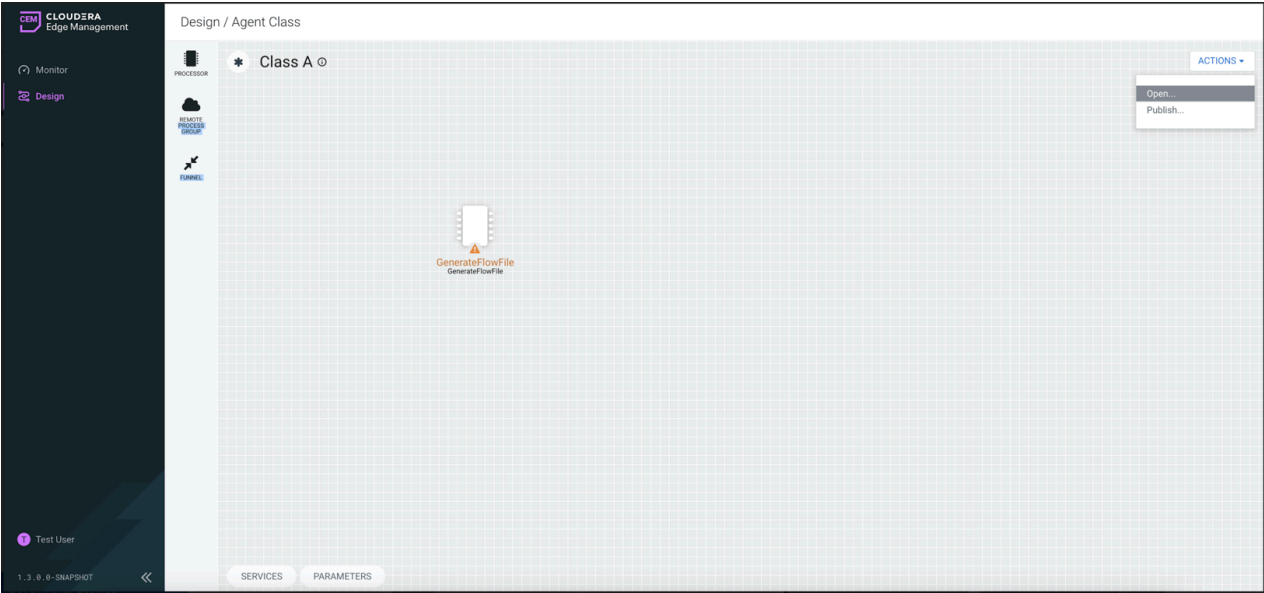
The following image shows that the users with viewer or designer roles can not publish a flow, as the Publish option is disabled:



The following image shows that users with viewer or operator roles can not edit the flow, as all configurations are disabled;



The following image shows that users with operator role can view and publish the flow but not edit:

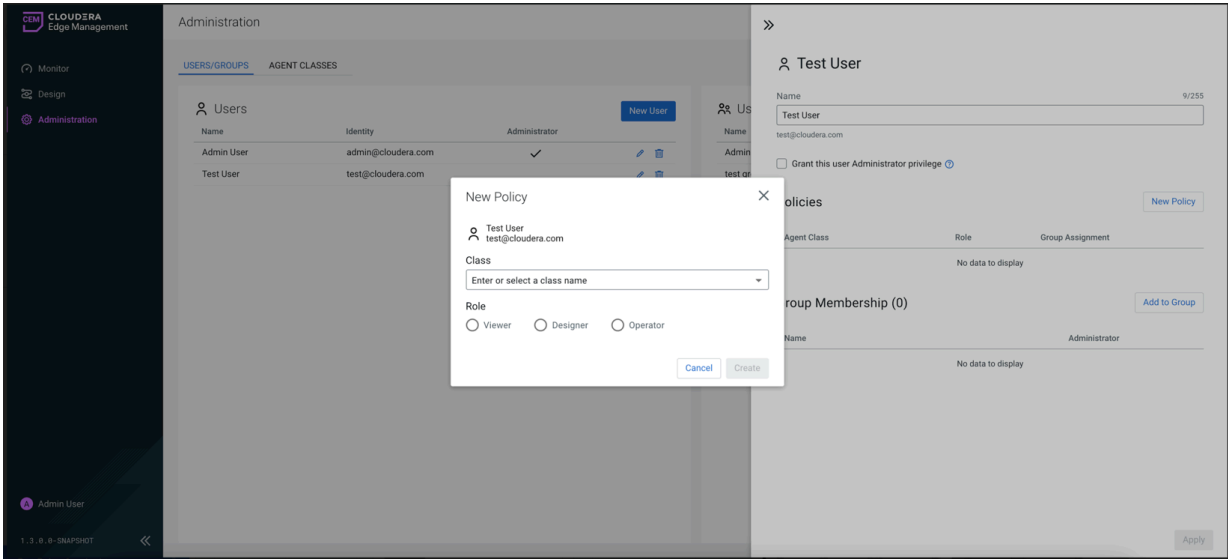


Assigning policies to a user

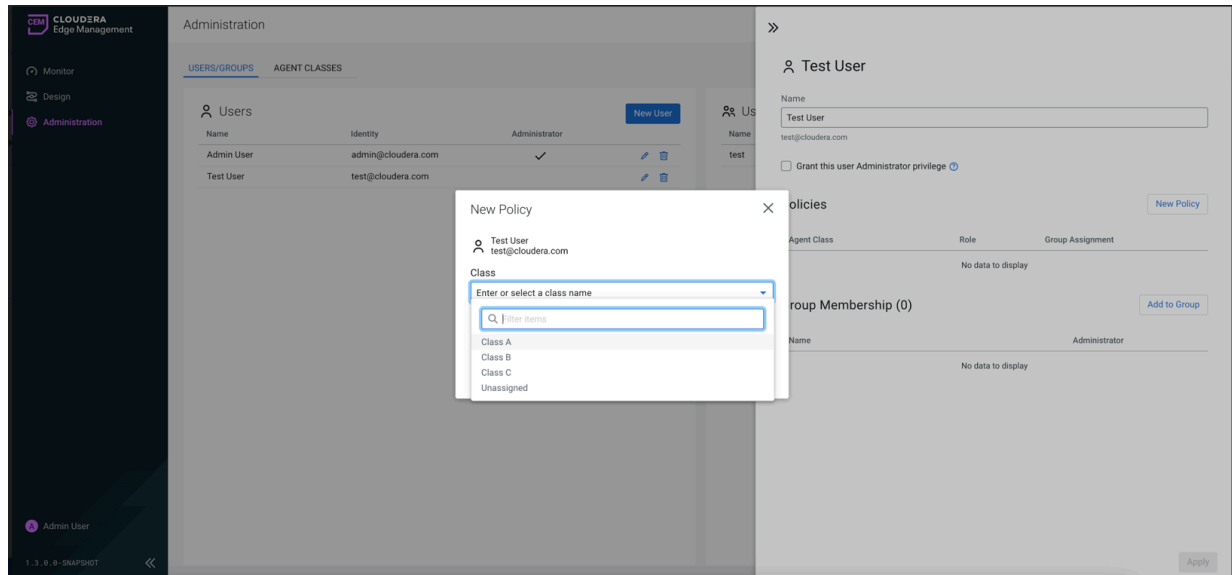
Learn how to select an agent class for a user and assign roles or policies to that user.

Procedure

- 1. Click Administration from the left navigation in the Cloudera Edge Management UI.  
The Administration page appears.
- 2. In the Users/Groups tab, click New User.  
Alternatively, you can click the edit icon for an existing user.  
The user page appears.
- 3. Click New Policy.  
The New Policy page appears.



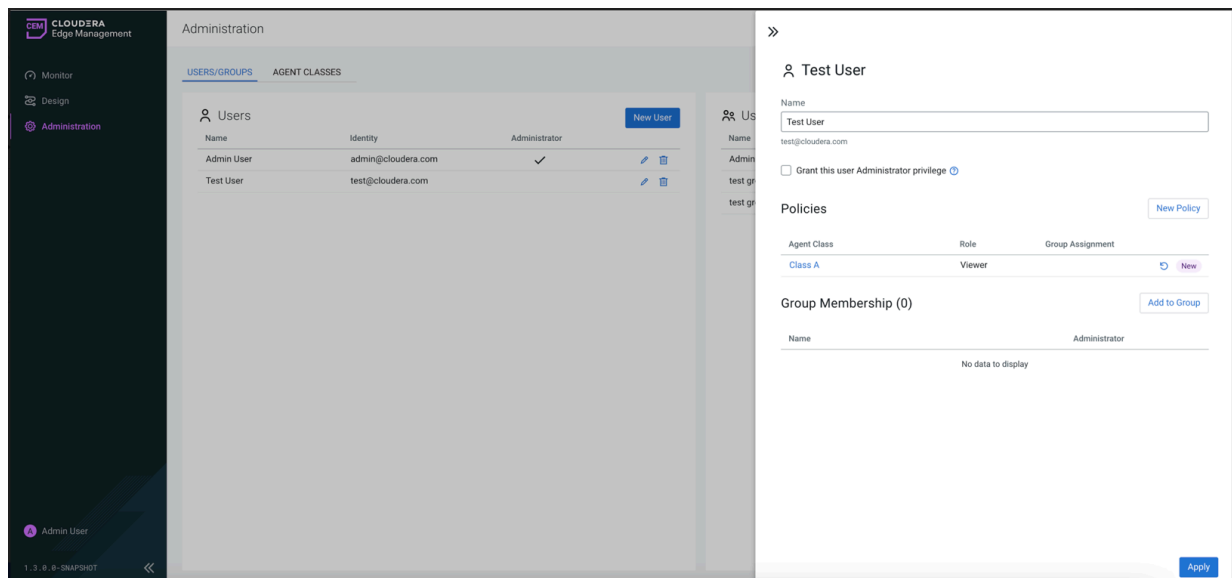
#### 4. Select a class for the user.



#### 5. Select a role for the user.

You can assign multiple roles to a user for the same class.

#### 6. Click Apply to save the changes.



**Note:** Logging in with a prepared user (matching happens based on identity) gives the user the predefined roles, so there is no initial authorization error.

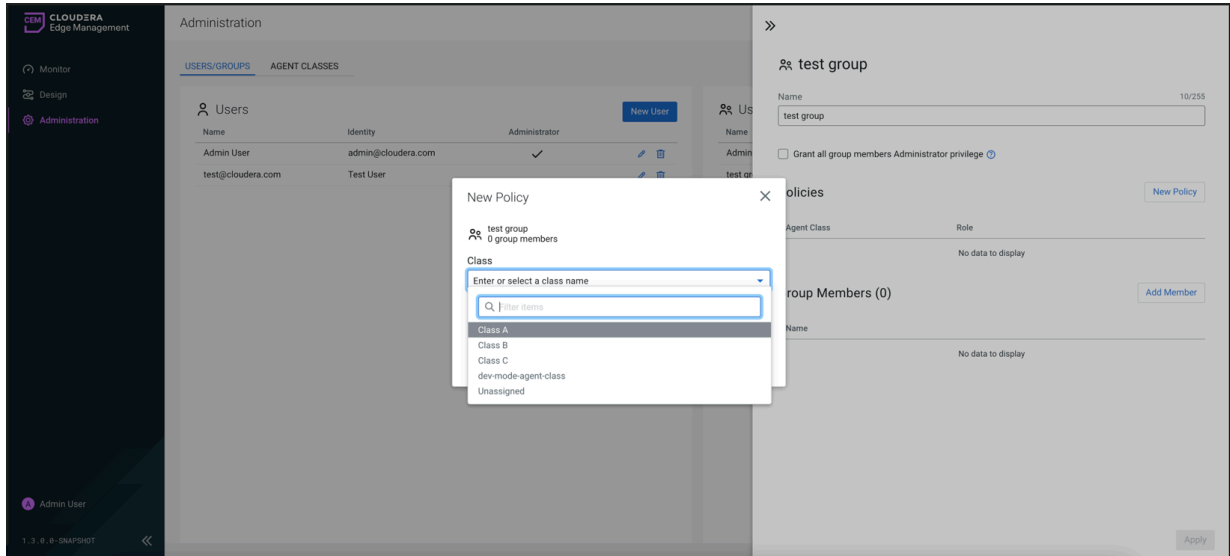
## Assigning policies to a user group

Learn how to select an agent class for a user group and assign roles or policies to that user group.

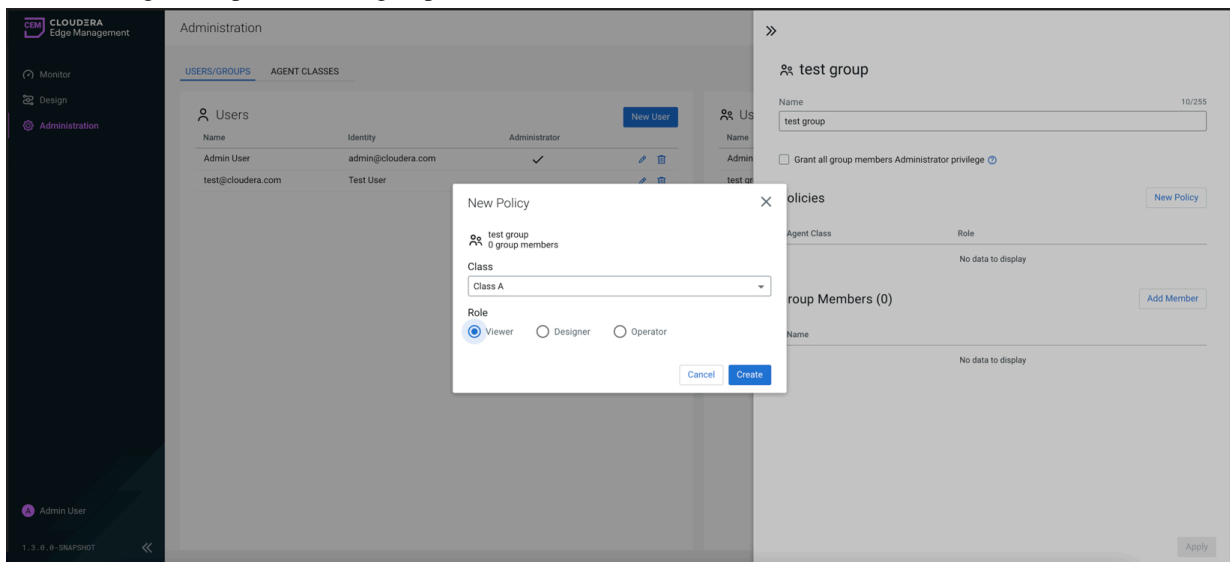
### Procedure

1. Click Administration from the left navigation in the Cloudera Edge Management UI.  
The Administration page appears.

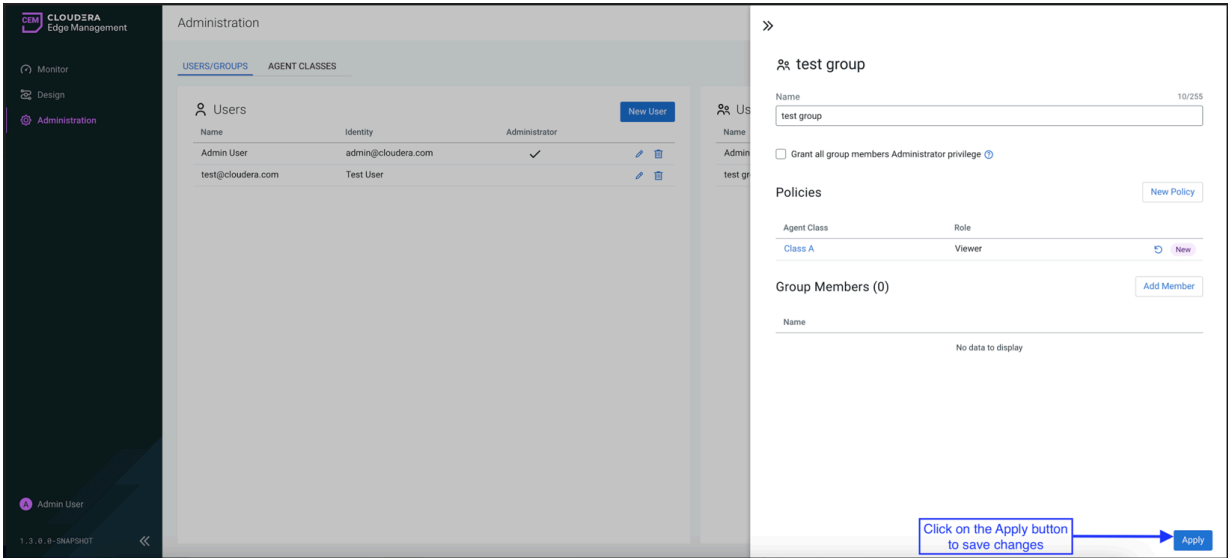
2. In the Users/Groups tab, click New User Group.  
Alternatively, you can click the edit icon for an existing user group.  
The user group page appears.
3. Click New Policy.
- The New Policy page appears.
4. Select a class for the user group.



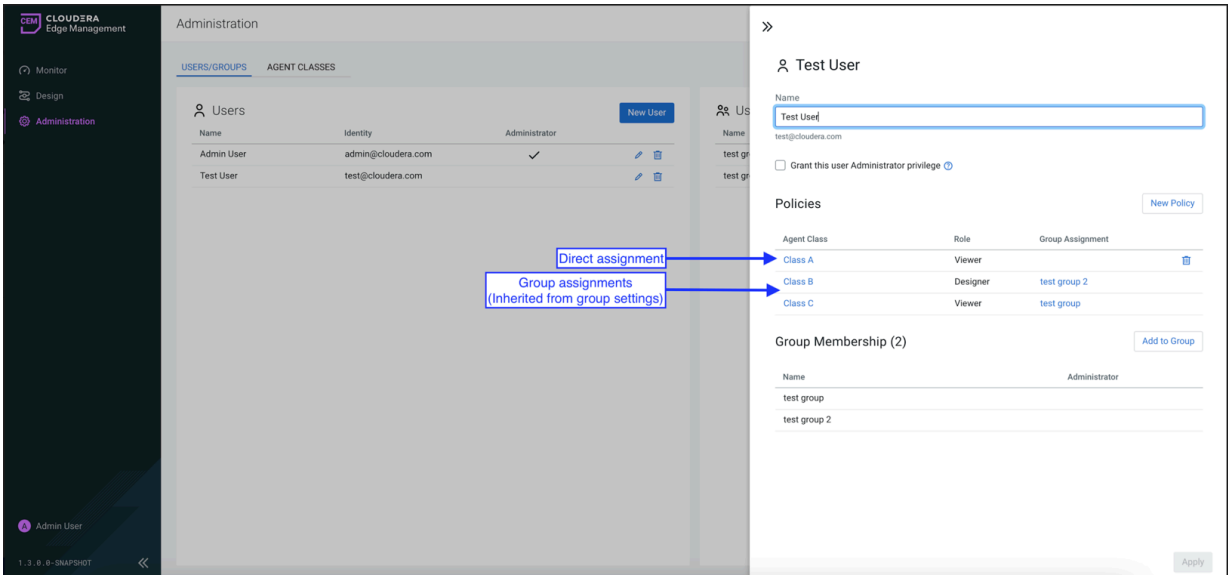
5. Select a role for the user group.  
You can assign multiple roles to a group for the same class.



6. Click Apply to save the changes.



In the user edit view in the policies section, you can see which policies are direct and which are inherited through group assignment.



### Assigning policies to agent classes

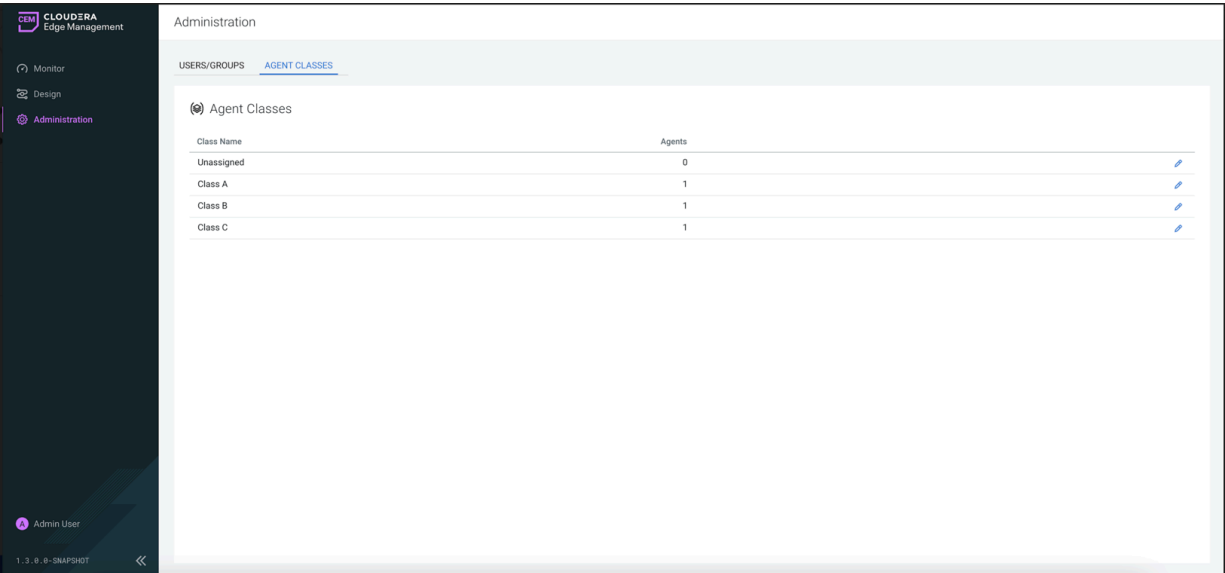
Learn how you can manage access control on the agent class level.

#### Procedure

1. Click Administration in the Cloudera Edge Management UI.  
The Administration page appears.

- 2. Go to Agent Classes tab.

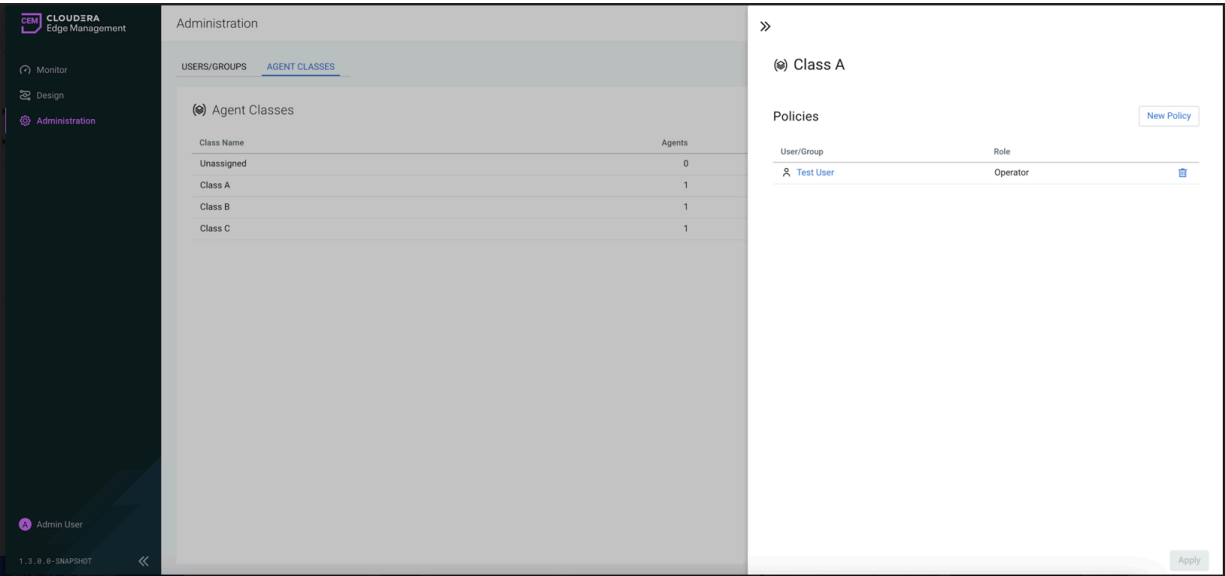
The list of agent classes with number of deployed agents appears.



You can manage user or group level assignments for a specific agent class.

- 3. Click the edit icon beside the class you want to modify.

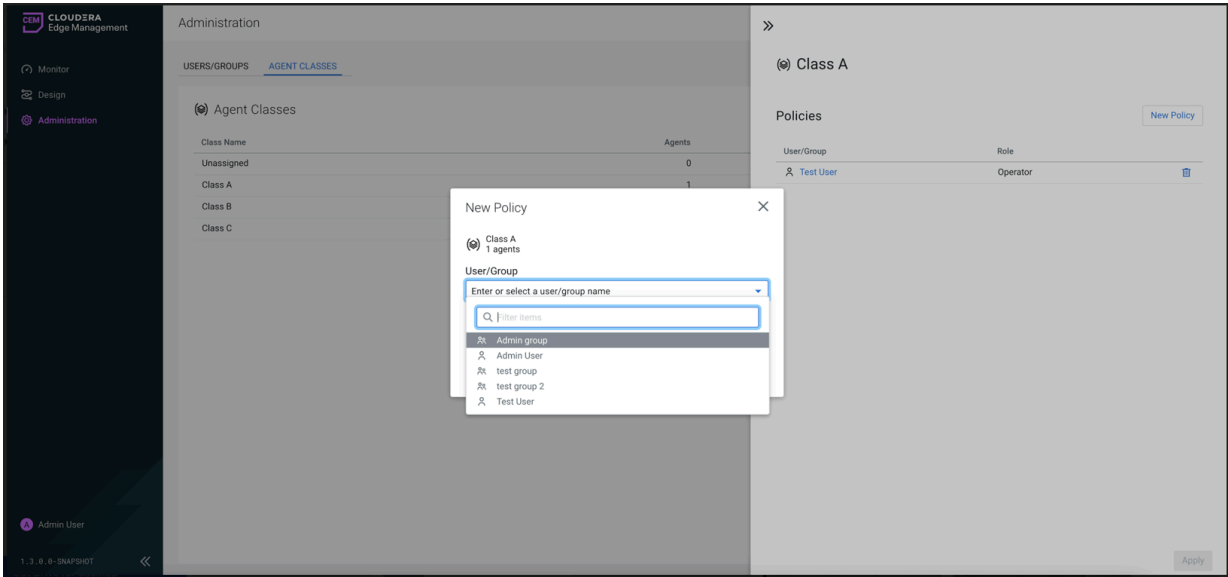
The class dialog appears.



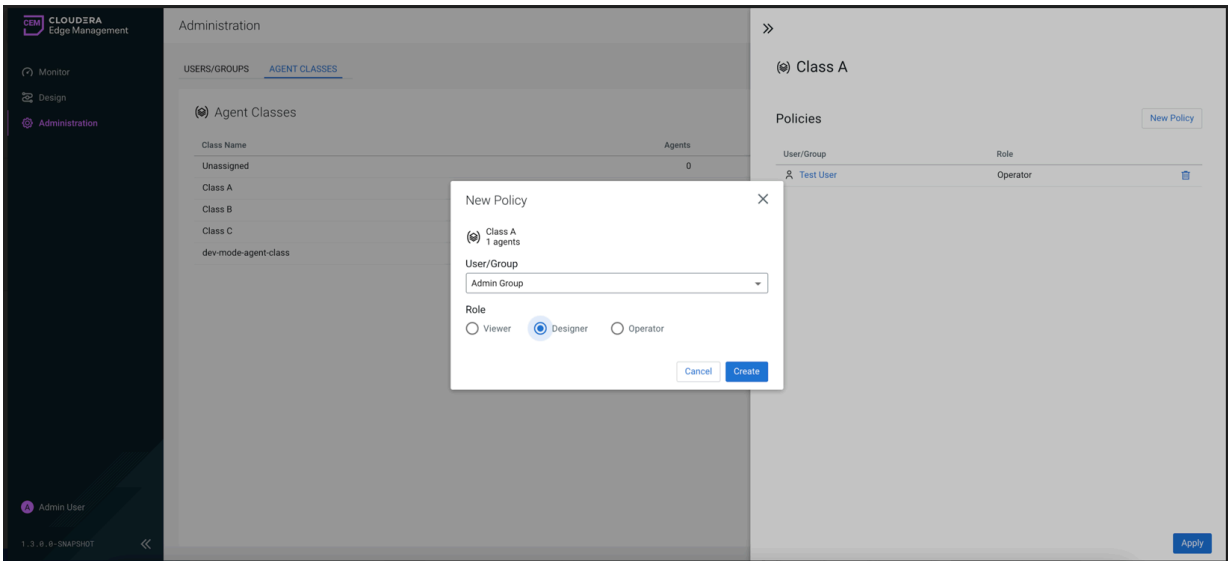
- 4. Click New Policy.

The New Policy page appears.

5. Select a user or user group for the agent class.

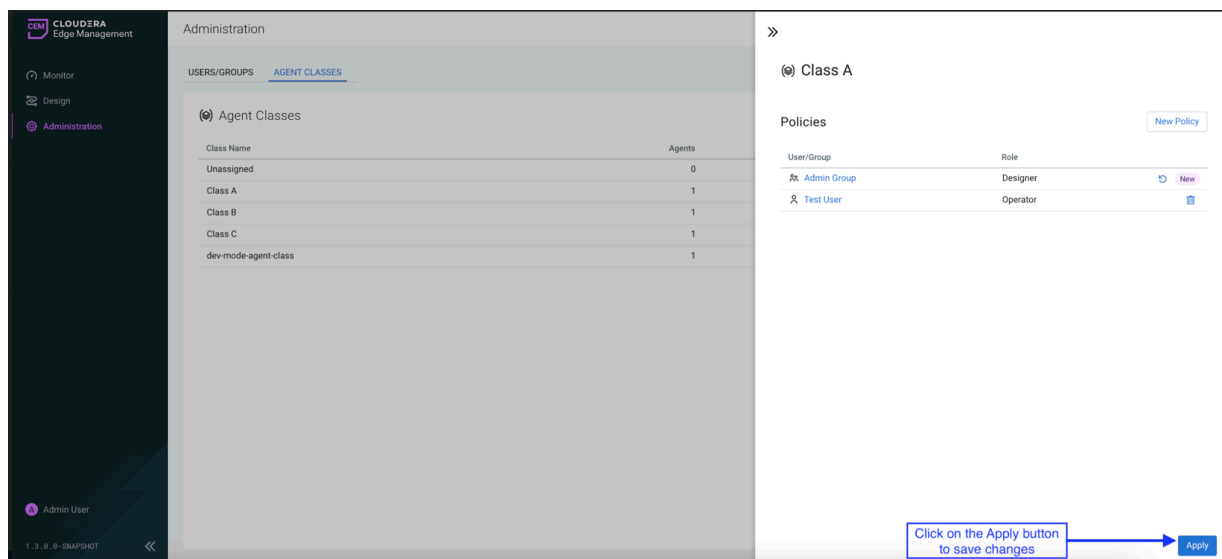


6. Select a role for the user or user group.



You can assign multiple roles to a user or group for the same class.

- Click Apply to save changes.



## SSO identity provider setup

To integrate Edge Flow Manager (EFM) with a third-party single sign-on (SSO) identity provider, you need to add EFM as an application client to the identity provider. This configuration allows EFM facilitate SSO logins by redirecting users to the identity provider and retrieving user details from the provider's user directory for use within EFM.

The supported protocols for identity provider integration are OpenID Connect (OIDC) and SAML 2.0. OIDC is sometimes referred to as OAuth2 login because it extends the authorization protocols in OAuth2. OIDC and SAML 2.0 are industry standard protocols supported by the majority of modern enterprise identity providers including Okta, Google, GitHub, Azure Active Directory and others.

For the latest guidance on configuring EFM as an application client for your SSO identity provider, see the *Identity Provider documentation*.

## Okta integration using OIDC

To integrate Edge Flow Manager (EFM) with Okta, Cloudera recommends using the Okta Application Integration Wizard. This wizard simplifies the process of adding EFM as a custom application using the OIDC protocol for integration.

For the latest guidance on how to add EFM as a custom application using OIDC, see [Create OIDC app integrations using AIW](#).

You need to enter few pieces of information about EFM, asked by Okta. The following table specifies the values to provide:

Property	Value
App Integration Name	Cloudera Edge Flow Manager (or whichever name you prefer)
Grant Type	Authorization Code
Sign-in redirect URIs	https://{efm-host:efm-port}/efm/login/oauth2/code/efm-oidc, https://{efm-host:efm-port}/efm/*

Property	Value
Sign-out redirect URIs	https://{efm-host:efm-port}/efm/ui/#/logged-out, https://{efm-host:efm-port}/efm/*
Client Credentials > Client ID	Set in Okta to any value, for example, efm, and then copy into the efm.properties file as efm.security.user.oidc.clientId
Client Credentials > Client Secret	Generate in Okta and copy the value into the efm.properties file as efm.security.user.oidc.clientSecret

After completion, configure EFM by following the instructions in *Configuration of OpenID Connect SSO*.

### Related Information

[Configuration of OpenID Connect SSO](#)

## Okta EFM integration using SAML

To integrate Edge Flow Manager (EFM) with Okta, Cloudera recommends using the Okta Application Integration Wizard. This wizard simplifies the process of adding EFM as a custom application using SAML 2.0.

For the latest guidance on how to add EFM as a custom application using SAML, see [Create SAML app integrations using AIW](#).

You need to enter few pieces of information about EFM, asked by Okta. The following table specifies the values to provide:

Property	Value
App Integration Name	Cloudera Edge Flow Manager (or whichever name you prefer)
Single sign on URL	https://localhost:10090/efm/login/saml2/sso/efm-saml
Audience URI (SP Entity ID)	https://localhost:10090/efm/saml2/service-provider-metadata/efm-saml
Attribute Statements (optional)	EFM contains an optional property in the efm.properties file called efm.security.user.saml.displayNameAttribute. This controls what name gets set in EFM the first time a user logs in using SAML SSO. In order to use this, set an attribute in SAML settings in Okta to hold the user's display name and then update EFM to set the displayNameAttribute to the same attribute key/name. For example:  In Okta:  Name: first_name Value: user.firstName  In the efm.properties file:  efm.security.user.saml.displayNameAttribute=first_name

After completion, configure EFM by following the instructions in *Configuration of SAML 2.0 SSO*.

### Related Information

[Configuration of SAML 2.0 SSO](#)