identity configuration management for Kubernetes documentation (Technology Preview)

v0.4.0: identity configuration management for Kubernetes

Table of Contents

IMPORTANT	The identity configuration management for Kubernetes is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in 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 about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

The identity configuration management for Kubernetes is a software operator that enhances cluster fleet management. It enables you to quickly and easily propagate OAuth authentication configuration across your fleet of OpenShift Container Platform clusters regardless of cloud provider or data center.

Requirements and recommendations

Before you install identity configuration management for Kubernetes technology preview, review the following system configuration requirements and settings: * Supported operating systems and platforms * Sizing recommendations * Security requirements * Network configuration * Backup and restore recommendation

Note: There is no console for the Technology Preview version of this product.

Supported operating systems and platforms for hub clusters and managed clusters

The identity configuration management tech preview requires the multicluster engine for Kubernetes 2.0.x or Red Hat Advanced Cluster Management for Kubernetes 2.4.x or 2.5.x.

Both the hub and managed clusters must be running OpenShift Container Platform at the levels supported by your chosen multicluster solution, multicluster engine or Advanced Cluster Management. Red Hat OpenShift Managed cloud services are not supported for this technology preview.

Note: Only OpenShift Container Platform 4.10, 4.9 or 4.8.12 and higher is supported for the hub. For 4.8, a minimum of 4.8.12 is required due to Bugzilla 1969902.

Sizing recommendations

If you are using multicluster engine, you will need a minimum of 1 node with 8 CPU, 32Gb of memory, and 100Gb disk.

If you are using Advanced Cluster Management, follow the RHACM sizing requirements.

Security requirements

Securing OAuth requests to the OpenID Connect identity provider

The identity configuration management operator instantiates one or more OpenID Connect identity providers under the covers to enable fleet-wide authentication. OAuth requests from the managed clusters to these identity providers must be secured using valid, signed certificates. This can be accomplished by replacing the default ingress certificate on the hub cluster or by providing a certicate in an AuthRealm custom resource.

Network configuration

Configure your network settings to allow the connections in the following sections. These network configurations are in addition to those required by multicluster engine and Advanced Cluster Management.

Hub cluster networking requirements

For the hub cluster networking requirements, see the following table:

Direction	Connection	Port (if specified)
Outbound	GitHub or GitHub Enterprise API. This is only required if you are using GitHub as an identity provider.	
Outbound	LDAP server (Example: OpenLDAP or Azure Active Directory managed domain). This is only required if you are using LDAP as an identity provider.	
Inbound	The OpenID Connect issuer from the managed cluster.	443

Managed cluster networking requirements

For the managed cluster networking requirements, see the following table:

Direction	Connection	Port (if specified)
Outbound	OpenID Connect issuer running on the hub cluster	443

Backup and restore recommendation

Red Hat Advanced Cluster Management for Kubernetes 2.5.x added backup and restore capabilities. However these capabilities are not enabled for identity configuration management for Kubernetes technology preview. The recommended approach for backup and restore is using a GitOps based architecture where the identity configuration can be reapplied in the case of a restore scenario.

Installing while connected online

The identity configuration management for Kubernetes operator is installed with Operator Lifecycle Manager, which manages the installation, upgrade, and removal of the components that encompass identity configuration management for Kubernetes.

Required access: Cluster administrator

- By default, the identity configuration management components are installed on worker nodes of your OpenShift Container Platform cluster without any additional configuration. You can install the identity configuration management operator onto worker nodes by using the OpenShift Container Platform OperatorHub web console interface, or by using the OpenShift Container Platform CLI.
- If you have configured your OpenShift Container Platform cluster with infrastructure nodes, you can install the identity configuration management onto those infrastructure nodes.

NOTE	Upgrades are not supported from identity configuration management for Kubernetes v0.1.0. If you have v0.1.0 installed, you will need to uninstall it before installing v0.2.0 or higher.

Prerequisites

Before you install identity configuration management for Kubernetes, see the following requirements:

- Your RedHat OpenShift Container Platform cluster must have access to the identity configuration management for Kubernetes operator in the OperatorHub catalog from the OpenShift Container Platform console.
- You need access to the catalog.redhat.com.
- OpenShift Container Platform version 4.10, 4.9 or 4.8.12 and higher. For 4.8, a minimum of 4.8.12 is required, due to Bugzilla 1969902, must be deployed in your environment, and you must be logged into with the OpenShift Container Platform CLI. See the following install documentation for OpenShift Container Platform:
  - OpenShift Container Platform version 4.8
- Your OpenShift Container Platform command line interface (CLI) must be configured to run `oc` commands. See Getting started with the CLI for information about installing and configuring the OpenShift Container Platform CLI.
- Your OpenShift Container Platform permissions must allow you to create a namespace.
- You must have an Internet connection to access the dependencies for the operator.
- You must have one of the following installed:
  - multicluster engine for Kubernetes 2.0.x
  - Red Hat Advanced Cluster Management for Kubernetes 2.4.x or 2.5.x installed.

Confirm your OpenShift Container Platform installation

You must have a supported OpenShift Container Platform version, including the registry and storage services, installed and working. For more information about installing OpenShift Container Platform, see the OpenShift Container Platform documentation.

1. Verify that the identity configuration management for Kubernetes operator is not already installed on your OpenShift Container Platform cluster. The identity configuration management for Kubernetes operator allows only one single installation on each OpenShift Container Platform cluster. Continue with the following steps if there is no installation.
2. To ensure that the OpenShift Container Platform cluster is set up correctly, access the OpenShift Container Platform web console with the following command:

   ```
   kubectl -n openshift-console get route
   ```

   See the following example output:

   ```
   openshift-console console console-openshift-console.apps.new-coral.purple-
   chesterfield.com
   console   https   reencrypt/Redirect     None
   ```

3. Open the URL in your browser and check the result. If the console URL displays `console-openshift-console.router.default.svc.cluster.local`, set the value for `openshift_master_default_subdomain` when you install OpenShift Container Platform. See the following example of a URL: `https://console-openshift-console.apps.new-coral.purple-chesterfield.com`.

You can proceed to install identity configuration management for Kubernetes.

Installing from the OperatorHub web console interface

Best practice: From the *Administrator* view in your OpenShift Container Platform navigation, install the OperatorHub web console interface that is provided with OpenShift Container Platform.

1. Select **Operators** > **OperatorHub** to access the list of available operators, and select *identity configuration management for Kubernetes* operator.
2. On the *Operator subscription* page, select the options for your installation:
    - Namespace:
        - The identity configuration management for Kubernetes must be installed in its own namespace, or project.
        - Under **Installed Namespace**, choose **Create Namespace** and specify `idp-mgmt-config`. If there is already a namespace `idp-mgmt-config` or you prefer another namespace, choose a different namespace.
    - Channel: The channel that you select corresponds to the release that you are installing. When you select the channel, it installs the identified release, and establishes that the future errata updates within that release are obtained.
    - Approval strategy: The approval strategy identifies the human interaction that is required for applying updates to the channel or release to which you subscribed.
        - Select **Automatic** to ensure any updates within that release are automatically applied.
        - Select **Manual** to receive a notification when an update is available. If you have concerns about when the updates are applied, this might be best practice for you.

    Note: To upgrade to the next minor release, you must return to the *OperatorHub* page and select a new channel for the more current release.
3. Select **Install** to apply your changes and create the operator.

    After the identity configuration management for Kubernetes is created, the status for the operator is *Running* on the *Installed Operators* page.
4. After installing the operator, create an instance of the `IDPConfig` resource to install the necessary components that provide the IDP configuration management APIs. Here's an example:

    ```
    apiVersion: identityconfig.identitatem.io/v1alpha1
    kind: IDPConfig
    metadata:
      name: idp-config
      namespace: idp-mgmt-config
    spec:
    ```

    You can now use the services provided by the identity configuration management for

Kubernetes.

Installing from the OpenShift Container Platform CLI

1. Create a identity configuration management for Kubernetes namespace where the operator requirements are contained. Run the following command, where `namespace` is the name for your identity configuration management for Kubernetes namespace. A value of `idp-mgmt-config` is recommended. The value for `namespace` might be referred to as *Project* in the OpenShift Container Platform environment:

```
oc create namespace <namespace>
```

2. Switch your project namespace to the one that you created. Replace `namespace` with the name of the identity configuration management for Kubernetes namespace that you created in step 1.

```
oc project <namespace>
```

3. Create a YAML file to configure an `OperatorGroup` resource. Each namespace can have only one operator group. Replace `default` with the name of your operator group. Replace `namespace` with the name of your project namespace. See the following example:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <default>
spec:
  targetNamespaces:
  - <namespace>
```

4. Run the following command to create the `OperatorGroup` resource. Replace `operator-group` with the name of the operator group YAML file that you created:

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

5. Create a YAML file to configure an OpenShift Container Platform Subscription. Your file should look similar to the following example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: idp-mgmt-operator-subscription
spec:
  sourceNamespace: openshift-marketplace
  source: redhat-operators
```

```
  channel: alpha
  installPlanApproval: Automatic
  name: idp-mgmt-operator
```

Note: For installing the identity configuration management for Kubernetes on infrastructure nodes, the see Operator Lifecycle Manager Subscription additional configuration section.

6. Run the following command to create the OpenShift Container Platform Subscription. Replace `subscription` with the name of the subscription file that you created:

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

7. Create an instance of the `IDPConfig` resource to install the necessary components that provide the IDP configuration management APIs. Here's an example:

```
apiVersion: identityconfig.identitatem.io/v1alpha1
kind: IDPConfig
metadata:
  name: idp-config
  namespace: idp-mgmt-config
spec:
```

Notes:

- A `ServiceAccount` with a `ClusterRoleBinding` automatically gives cluster administrator privileges to identity configuration management for Kubernetes and to any user credentials with access to the namespace where you install identity configuration management for Kubernetes.

Installing on infrastructure nodes

An OpenShift Container Platform cluster can be configured to contain infrastructure nodes for running approved management components. Running components on infrastructure nodes avoids allocating OpenShift Container Platform subscription quota for the nodes that are running those management components.

After adding infrastructure nodes to your OpenShift Container Platform cluster, follow the Installing from the OpenShift Container Platform CLI instructions and add the following configurations to the Operator Lifecycle Manager Subscription.

Add infrastructure nodes to the OpenShift Container Platform cluster

Follow the procedures that are described in Creating infrastructure machine sets in the OpenShift Container Platform documentation. Infrastructure nodes are configured with a Kubernetes `taint` and `label` to keep non-management workloads from running on them.

To be compatible with the infrastructure node enablement provided by identity configuration management for Kubernetes, ensure your infrastructure nodes have the following `taint` and `label` applied:

```
metadata:
  labels:
    node-role.kubernetes.io/infra: ""
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
```

Operator Lifecycle Manager Subscription additional configuration

Add the following additional configuration before applying the Operator Lifecycle Manager Subscription:

```
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      effect: NoSchedule
      operator: Exists
```

Install on disconnected networks

You might need to install the identity configuration management for Kubernetes operator on Red Hat OpenShift Container Platform clusters that are not connected to the Internet. The procedure to install on a disconnected environment requires some of the same steps as the connected installation.

You must download copies of the packages to access them during the installation, rather than accessing them directly from the network during the installation.

NOTE	Upgrades are not supported from identity configuration management for Kubernetes v0.1.0. If you have v0.1.0 installed, you will need to uninstall it before installing v0.2.0 or higher.

- Prerequisites
- Confirm your OpenShift Container Platform installation
- Preparing to install on infrastructure nodes

Prerequisites

You must meet the following requirements before you install the identity configuration management for Kubernetes operator:

- Red Hat OpenShift Container Platform version 4.8.12 or later must be deployed in your environment, and you must be logged in with the command line interface (CLI).
- You need access to the catalog.redhat.com.

  Note: For managing bare metal clusters, you must have OpenShift Container Platform version 4.8.12 or later.

  See the OpenShift Container Platform version 4.8 documentation.

- Your Red Hat OpenShift Container Platform CLI must be version 4.8 or later, and configured to run `oc` commands. See Getting started with the CLI for information about installing and configuring the Red Hat OpenShift CLI.
- Your Red Hat OpenShift Container Platform permissions must allow you to create a namespace.
- You must have a workstation with Internet connection to download the dependencies for the operator.

Confirm your OpenShift Container Platform installation

- You must have a supported OpenShift Container Platform version, including the registry and storage services, installed and working in your cluster. For information about OpenShift Container Platform version 4.8, see OpenShift Container Platform Documentation.
- When and if you are connected, you can ensure that the OpenShift Container Platform cluster is set up correctly. Access the OpenShift Container Platform web console.

  Run the `kubectl -n openshift-console get route` command to access the OpenShift Container Platform web console. See the following example output:

  ```
  openshift-console          console             console-openshift-console.apps.new-
  coral.purple-chesterfield.com                       console              https
  reencrypt/Redirect     None
  ```

  The console URL in this example is: `https:// console-openshift-console.apps.new-coral.purple-chesterfield.com`. Open the URL in your browser and check the result.

  If the console URL displays `console-openshift-console.router.default.svc.cluster.local`, set the value for `openshift_master_default_subdomain` when you install OpenShift Container Platform.

See Sizing your cluster to learn about setting up capacity for your operator.

Installing in a disconnected environment

Important: You need to download the required images to a mirroring registry to install the operators in a disconnected environment. Without the download, you might receive `ImagePullBackOff` errors during your deployment.

Follow these steps to install the identity configuration management for Kubernetes operator in a disconnected environment:

1. Create a mirror registry. If you do not already have a mirror registry, create one by completing the procedure in the Mirroring images for a disconnected installation topic of the Red Hat OpenShift Container Platform documentation.

   If you already have a mirror registry, you can configure and use your existing one.

2. Create a YAML file that contains the `ImageContentSourcePolicy` with the name `idp-mgmt-config-policy.yaml`. **Note:** If you modify this on a running cluster, it causes a rolling restart of all nodes.

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: ImageContentSourcePolicy
   metadata:
     name: identity-config
   spec:
     repositoryDigestMirrors:
     - mirrors:
       - mirror.registry.com:5000/identity-config
       source: registry.redhat.io/identity-config
   ```

3. Apply the ImageContentSourcePolicy file by entering the following command:

   ```
   oc apply -f idp-mgmt-config-policy.yaml
   ```

4. Enable the disconnected Operator Lifecycle Manager Red Hat Operators and Community Operators.

   the identity configuration management for Kubernetes operator is included in the Operator Lifecycle Manager Red Hat Operator catalog.

5. Configure the disconnected Operator Lifecycle Manager for the Red Hat Operator catalog. Follow the steps in the https://access.redhat.com/documentation/en-us/openshift_container_platform/4.8/html/operators/administrator-tasks# Operator Lifecycle Manager-restricted-networks[Using Operator Lifecycle Manager on restricted networks] topic of the Red Hat OpenShift Container Platform documentation.
6. Now that you have the image in the disconnected Operator Lifecycle Manager, continue to install the identity configuration management for Kubernetes operator for Kubernetes from the Operator Lifecycle Manager catalog.

See Installing while connected online for the required steps.

Getting started

Introduction

Identity configuration management for Kubernetes is intended to allow you to quickly and easily configure OAuth authentication across your fleet of OpenShift clusters. Through the creation of an AuthRealm custom resource, you are able to specify one or more identity providers (such as GitHub or LDAP) and a placement rule to indicate to which clusters the identity providers should apply. As more clusters are brought under management, their OAuth configuration will be automatically configured simply by matching the specified placement rule. The existing OAuth configuration will be replaced by the configurations defined in the different AuthRealms. The existing OAuth configuration is saved in the cluster namespace and will be restored once the cluster is no longer under management.

An identity configuration management operator configuration is composed of:

- A namespace in which the identity configuration will be defined
- A managedclusterset custom resource which holds the managed clusters which may be part of the identity configuration.
- A placement custom resource which defines the matching labels to select the clusters.
- A managedclustersetbinding custom resource to bind that placement to the managedclusterset.
- For each identity configuration provider, a secret custom resource containing the client ID and client secret to connect to the identity configuration provider.
- An authrealm custom resource which defines the identiry configuration providers.

We will first walk through setting up your managed cluster set and placement, then prepare the configuration for your preferred identity provider(s).

Preparing your managed cluster set and placement

In order to add managed clusters to a managed cluster set, the managed clusters must be under management of the hub cluster. Follow these instructions if managed clusters need to be brought under management by the hub cluster:

- Red Hat Advanced Cluster Management - Importing a target managed cluster to the hub cluster
- Red Hat multicluster engine operator - Importing a target managed cluster to the hub cluster

Once there are managed clusters, create a managed cluster set and managedclustersetbinding. Then select the managed clusters that will be a part of the set:

- Red Hat Advanced Cluster Management - creating and managing managedclustersets
- Red Hat multicluster engine operator - creating and managing managedclustersets

Now a placement can be created. This will be used to determine which managed clusters in the managedclusterset will have the identity configuration applied. See:

- Red Hat Advanced Cluster Management - using managedclustersets with placement
- Red Hat multicluster engine operator - creating and managing managedclustersets

IMPORTANT

The identity configuration management for Kubernetes Technology Preview does not yet utilize custom role definitions for controlling access to work with AuthRealms. Instead, it relies on the role-based access control implemented by multicluster engine and Advanced Cluster Management around ManagedClusterSets, ManagedClusterSetBindings, and Placements. AuthRealms must exist in a namespace containing a Placement, and the Placement leverages the ManagedClusterSets that are bound to that namespace. Access to namespaces with existing bound ManagedClusterSets and/or existing Placements should be managed accordingly.

IMPORTANT

If you are using v0.1.0, before detaching a cluster from the hub, you must make sure the managed cluster is no longer managed by identity configuration management. You can remove the cluster from identity configuration management either by removing the cluster from the ManagedClusterSet, or by removing labels from the managed cluster as appropriate to no longer match the Placement. Once your cluster no longer matches the Placement of any AuthRealm, your original OAuth configuration should be restored on the cluster. If the cluster is detached without first being unmanaged from identity provider management, you will lose access to the cluster.

If you are using v0.2.0 or higher, it is recommended that you follow the steps above to take the cluster out of identity configuration management prior to detaching the cluster from the hub. In this case, your original OAuth

configuration should be restored for you on the managed cluster, and you can then detach the cluster from the hub. As an alternative, prior to detaching the managed cluster from the hub, you can copy your original OAuth from the ConfigMap named `idp-oauth-original` in the managed cluster's namespace on the hub cluster. After the cluster is completely detached from the hub cluster, manually edit the OAuth on the cluster you've just detached and refer to the contents of the idp-oauth-original ConfigMap to restore your original OAuth configuration.

Creating your AuthRealm

The AuthRealm custom resoure specifies a combination of one or more identity providers and a placement that determines to which managed clusters the identity provider(s) should be applied.

An AuthRealm resource example is as follows:

```
apiVersion: identityconfig.identitatem.io/v1alpha1
kind: AuthRealm
metadata:
  name: identity-config-developers
  namespace: identity-config-authrealm
spec:
  identityProviders:
  - github:
      clientID: <Github OAuth client ID>
      clientSecret:
        name: identitatem-github-client-secret
      organizations:
      - identitatem
    mappingMethod: claim
    name: identitatem-developers
    type: GitHub
  placementRef:
    name: dev-clusters-placement
  routeSubDomain: identitatem-devs
  type: dex
```

The AuthRealm `name` becomes an identity provider name on each managed cluster to which the AuthRealm applies, and is therefore the label that will appear on the managed cluster OpenShift login button.

The `identityProviders` list values directly correspond to OpenShift Container Platform's identity provider configuration. Only GitHub and LDAP are supported in this Technology Preview. Take special note of the `mappingMethod` field, which controls how mappings are established between the provider's identities and User objects.

The `placementRef` contains the name of the Placement custom resource located in the same namespace as the AuthRealm that will determine to which clusters this AuthRealm will apply.

The `routeSubDomain` is used to form the Issuer URL of the intermediary OpenID connect server that is installed and configured automatically on the hub cluster by identity configuration management in order to achieve fleet-wide identity configuration. The managed clusters' cluster OAuth configurations connect to this OpenID connect server via the Issuer URL, rather than connecting to the desired identity provider directly.

IMPORTANT	`routeSubDomain` must be unique among all AuthRealms on the hub cluster. It must be a max of 54 characters. It must meet the requirements of a DNS

> subdomain and therefore must also contain only lowercase alphanumeric characters, '-' or '.', and start and end with an alphanumeric character. Once specified, the `routeSubDomain` must not be modified. If you need to change this value, delete the AuthRealm and start again. These rules will be enforced in a future version of the operator, and failure to comply will result in unexpected results with the technology preview.

The `type` field must be set to `dex` for the Technology Preview. The Technology Preview leverages Dex under the covers as described above to achieve fleet-wide identity configuration.

The sections below outline more details and specific steps necessary to create your AuthRealm for GitHub or LDAP.

GitHub authentication

To use GitHub-based authentication, an OAuth application must be defined in GitHub.

GitHub OAuth

1. The GitHub Oauth app can be created in your personal profile or in a Github organization that you are an administrator for.
    - To create a GitHub Oauth app in your personal profile: Open a web browser and navigate to https://github.com, go to *Settings* > *Developer Settings* > *OAuth Apps* (The shortcut is https://github.com/settings/developers)
    - To create a GitHub Oauth app in a GitHub organization that you are an administrator for: Open a web browser and navigate to https://github.com/{organization_name}, go to *Settings* > *Developer Settings* > *OAuth Apps* (The shortcut is https://github.com/{organization_name}/settings/developers)
2. Add a **New OAuth App**. In the new OAuth app, **Generate a new client secret**. Copy the Client ID and Client Secret values and save them. They will be needed later.
3. Fill in the **Application name** with something to help identify the owner and hub it will be used for. NOTE: If you have more than one hub, each one will need its own OAuth app.
4. Fill in **Homepage URL** and **Authorization callback URL** with the OpenShift console URL. (NOTE: A little bit later we will correct the **Authorization callback URL** value once we have the proper value.)
5. Click **Register Application NOTE:** Leave this web page open so you can quickly correct the **Authorization callback URL** value once you have the proper value.

AuthRealm custom resource for GitHub

With your GitHub OAuth app created and your client id and secret in hand, you are ready to create your AuthRealm custom resource.

An example of an AuthRealm custom resource configured for the Github identity provider is provided below. Make note of the `identityProviders` field and the configuration of the `github` identity provider under it:

```
apiVersion: identityconfig.identitatem.io/v1alpha1
kind: AuthRealm
metadata:
  name: identity-config-developers
  namespace: identity-config-authrealm
spec:
  identityProviders:
  - github:
      clientID: <Github OAuth client ID>
      clientSecret:
        name: identitatem-github-client-secret
```

```
      organizations:
      - identitatem
    mappingMethod: claim
    name: identitatem-developers
    type: GitHub
  placementRef:
    name: dev-clusters-placement
  routeSubDomain: identitatem-devs
  type: dex
```

The `identityProviders` list contains the configurations for one or more identity providers. The example above contains a single identity provider (GitHub). An entry under `identityProviders` has the following fields:

- `name` contains the unique name that is used to identify the identity provider.
- `type` specifies the identity provider type and it is set to GitHub.
- `mappingMethod` (add, claim or lookup) controls how mappings are established between this provider's identities and User objects.
- `github` contains the GitHub specific configurations:
  - `clientID` contains the client ID of a registered GitHub OAuth application.
  - `clientSecret` contains a reference to a secret object containing the client secret of a registered GitHub OAuth application. Example secret:

    ```
    oc create secret generic <github-oauth-clientID-name> --from
    -literal=clientSecret=<github-oauth-clientID-secret>
    ```

  - `organizations` contains a list of organizations to authenticate the user against (authentication against specific teams within GitHub organizations is currently not supported). This field can be left blank to skip authentication against specific GitHub organizations. If organizations are specified in the config then the user must be a member of at least one of the specified orgs.
    Note: If the GitHub OAuth application is not owned by an organization specified in `organizations`, an organization owner must grant third-party access to use this option. This can be done in two ways:
    - by the GitHub organization's administrator from the GitHub organization settings,
    - or, during the first GitHub login when the user will be presented with a UI to explicitly request access to the GitHub organization. The request will flow to the GitHub organization's administrator for approval and the user will only be able to login after the request for access is approved.

To create your AuthRealm, copy the sample yaml above and update as appropriate with your client id and secret, GitHub organization(s) (if desired), and preferred names, namespace, route subdomain, and mapping method. Then, run the following command on your hub cluster:

```
oc create -f <authrealm_file_name>.yaml
```

With your AuthRealm created, you are ready to update your GitHub OAuth app's authorization callback URL. Leverage the `routeSubDomain` you specified in your AuthRealm, and run the script below while logged in to your hub cluster to generate the proper value:

```
export ROUTE_SUBDOMAIN=<your_route_subdomain>
export APPS=$(oc get infrastructure cluster -ojsonpath='{.status.apiServerURL}' | cut
-d':' -f2 | sed 's/\/\/api/apps/g')
echo "Authorization callback URL: https://${ROUTE_SUBDOMAIN}.${APPS}/callback"
```

At this point, any managed cluster that matches your Placement should be updated with an OAuth configuration that leverages an OpenID identity provider to connect to the proxy server that identity configuration management has stood up under the covers. Allow several minutes for the new login button to appear on your managed cluster.

LDAP authentication

LDAP

The LDAP identity provider configuration allows email/password based authentication backed by a LDAP directory. 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.

Setting up LDAP:

There are multiple options available for setting up an LDAP directory. For example:

- **OpenLDAP**: an open-source implementation of LDAP. The following article has information on hosting and deploying an OpenLDAP server: https://medium.com/ibm-garage/how-to-host-and-deploy-an-openldap-sever-in-openshift-affab06a4365
- **Secure LDAP using Azure Active Directory**: The following tutorial describes the steps for setting up Secure LDAP with Azure Active Directory: https://docs.microsoft.com/en-us/azure/active-directory-domain-services/tutorial-configure-ldaps

AuthRealm custom resource for LDAP

An example of an AuthRealm custom resource configured for the LDAP identity provider is provided below. Make note of the `identityProviders` field and the configuration of the `ldap` identity provider under it:

```
apiVersion: identityconfig.identitatem.io/v1alpha1
kind: AuthRealm
metadata:
  name: authrealm-ldap
  namespace: authrealm-ldap-ns
spec:
  type: dex
  routeSubDomain: identitatem-devs
  placementRef:
    name: authrealm-ldap-placement
  ldapExtraConfigs:
    openldap:
      baseDN: "dc=example,dc=com"
      filter: "(objectClass=person)"
  identityProviders:
    - name: openldap
      type: LDAP
      mappingMethod: add
      ldap:
        url: <LDAP server URL>
        insecure: false
```

```
      bindDN: cn=Manager,dc=example,dc=com
      ca:
        name: authrealm-ldap-ca
        namespace: authrealm-ldap-ns
      bindPassword:
        name: authrealm-ldap-secret
        namespace: authrealm-ldap-ns
      attributes:
        id:
          - DN
        preferredUsername:
          - mail
        name:
          - cn
        email:
          - mail
```

The `identityProviders` list contains the configurations for one or more identity providers. The example above contains a single identity provider (LDAP). An entry under `identityProviders` has the following fields:

- `name` contains the unique name that is used to identify the identity provider.
- `type` specifies the identity provider type and it is set to LDAP.
- `mappingMethod` (add, claim or lookup) controls how mappings are established between this provider's identities and User objects.
- `ldapExtraConfigs` contains extra server configuration setting for LDAP, the key being the idp.name
  - `baseDN` contains information to start the LDAP user search from. For example "cn=users,dc=example,dc=com"
  - `filter` contains optional filter to apply when searching the directory. For example "(objectClass=person)"
- `ldap` contains the LDAP specific configurations:
  - `url` contains the LDAP host and search parameters to use. The url syntax is `ldap://host:port/basedn?attribute?scope?filter`
  - `insecure` must be `false`. ldaps:// URLs will attempt to connect using TLS, and ldap:// URLs will be upgraded to TLS. The identity configuration management operator does not support insecure=true.
  - `bindDN` contains an optional DN to bind with during the search phase.
  - `bindPassword` contains an optional reference to a secret by name containing a password to bind with during the search phase. **Note**: bindDN and bindPassword need to be provided if the LDAP server requires authentication for search.
  - `ca` contains reference to the secret containing a trusted Root CA file - file name and format: "ca.crt" **Note**: If the server uses self-signed certificates, include files with names "tls.crt" and "tls.key" (representing client certificate and key) in the same secret

- `attributes` maps LDAP attributes to identities
  - `id` is the attribute whose value should be used as the user ID. Required. Only the first attribute is used; all other values are ignored. If the first attribute does not have a value, authentication fails. LDAP standard identity attribute is "dn".
  - `preferredUsername` is currently ignored by the identity configuration management operator.
  - `name` is the attribute whose values should be used as the display name. Required. Only the first attribute is used; all other values are ignored. LDAP standard display name attribute is "cn".
  - `email` is the attribute whose values should be used as the email address. Required. Only the first attribute is used; all other values are ignored.

To configure group search for LDAP provider, please refer https://identitatem.github.io/idp-mgmt-docs/groups

```
To create your AuthRealm, copy the sample yaml above and update as appropriate with
your ldap configuration, preferred names, namespace, route subdomain, and mapping
method. Then, run the following command on your hub cluster:
```

```
oc create -f <authrealm_file_name>.yaml
```

At this point, any managed cluster that matches your Placement should be updated with an OAuth configuration that leverages an OpenID identity provider to connect to the proxy server that identity configuration management has stood up under the covers. Allow several minutes for the new login button to appear on your managed cluster. You will be able to login with the ldap user.

OpenID authentication

Version 0.3.1 of the technology preview introduces limited support for integrating with OpenID Connect identity providers. We have only tested the integration with dex, Keycloak, and Red Hat Single Sign-On.

NOTE: When integrating with Keycloak, ensure the Keycloak client has default or optional client scopes enabled for email and profile.

AuthRealm custom resource for OpenID

An example of an AuthRealm custom resource configured with an OpenID identity provider is provided below. Make note of the `identityProviders` field and the configuration of the `opeinid` identity provider under it:

```
apiVersion: identityconfig.identitatem.io/v1alpha1
kind: AuthRealm
metadata:
  name: authrealm-openid
  namespace: authrealm-openid-ns
spec:
  placementRef:
    name: dev-clusters-placement
  routeSubDomain: identitatem-devs
  type: dex
  identityProviders:
    - name: identitatem-developers
      mappingMethod: add
      type: OpenID
      openID:
        clientID: <OpenID client ID>
        clientSecret:
          name: identitatem-openid-client-secret
        claims:
          preferredUsername:
            - username
          name:
            - name
          email:
            - mail
        issuer: <openid auth URL>
```

The `identityProviders` list contains the configurations for one or more identity providers. The example above contains a single identity provider (OpenID). An entry under `identityProviders` has the following fields:

- `name` contains the unique name that is used to identify the identity provider.
- `type` specifies the identity provider type and it is set to OpenID.

- `mappingMethod` (add, claim or lookup) controls how mappings are established between this provider's identities and User objects.
- `openID` contains the OpenID specific configurations:
  - `clientID` contains the client ID of a client registered with the OpenID provider.
  - `clientSecret` contains a reference to a secret object containing the OpenID client secret. Example secret:

    ```
    oc create secret generic <openid-oauth-clientID-name> --from
    -literal=clientSecret=<openid-oauth-clientID-secret>
    ```

- `claims` maps attributes on the user entry.
  - `preferredUsername` is the list of attributes whose values should be used as the preferred username. Optional. If unspecified, the preferred username is determined from the value of the sub claim. OpenShift supports an array of preferredUsername, but identity configuration management for Kubernetes is only supporting the first value from the array.
  - `name` is the list of attributes whose values should be used as the display name. Optional. If unspecified, no display name is set for the identity. OpenShift supports an array of name, but identity configuration management for Kubernetes is only supporting the first value from the array.
  - `email` is the list of attributes whose values should be used as the email address. Optional. If unspecified, no email is set for the identity. OpenShift supports an array of email, but identity configuration management for Kubernetes is only supporting the first value from the array.
- `issuer` the URL of the OpenID authentication provider.

To create your AuthRealm, copy the sample yaml above and update as appropriate with your OpenID configuration, preferred names, namespace, route subdomain, and mapping method. Then, run the following command on your hub cluster:

```
oc create -f <authrealm_file_name>.yaml
```

At this point, any managed cluster that matches your Placement should be updated with an OAuth configuration that leverages an OpenID identity provider to connect to the proxy server that identity configuration management has stood up under the covers. Allow several minutes for the new login button to appear on your managed cluster. You will be able to login with the OpenID authentication.

Using a custom certificate

By default, identity configuration management stands up an OpenID Connect proxy service that leverages the hub cluster's default ingress certificate. You can instead utilize a custom certificate by adding the following two fields to the AuthRealm spec:

```
spec:
  certificatesSecretRef:
    name: acme-cert
  host: https://identitatem-devs.acme.com
```

Create a tls secret containing your custom certificate in the same namespace as your AuthRealm and provide the secret name in `certificatesSecretRef.name`. For `host`, specify the full URL of the domain you want created for the OpenID Connect proxy service, including the `https://` protocol.

When creating the tls secret, be sure to specify the `tls` type. For example:

```
kubectl create secret tls <acme-cert> --cert=<path/to/cert/file>
--key=<path/to/key/file>
```

Note: When you specify a `host` URL, the `routeSubDomain` is used only for creating an OpenShift project on your hub cluster to hold resources pertaining to the AuthRealm. It is not reflected in the OpenID Connect proxy service issuer URL.

Command line interface

AuthRealm configuration can be created through the cm-cli.

```
cm create authrealm --values <values.yaml>
```

1. Fill the template form

   The template can be retreived by running:

   ```
   cm create authrealm -h
   ```

   Fill the template and save it as for example my-authrealm.yaml

   ```
   authRealm:
     # The name of the authrealm, can be override using the --name parameter
     name:
     # The namespace where the authrealm must be created, can be override using the
   --namespace
     namespace:
     #The strategy type, only dex is supported, can be override using --type
     type: dex
     # The routeSubDomain to use, can be override using --route-sub-domain
     routeSubDomain:
     # The placement rule to use, if not present then a new one will be created
     # in the authrealm namespace and having for labelSelector the matchLabels below.
     # It can be overridden using --placement
     placement:
     # The matchLabels to use to build the placement if not provided
     # For example:
     # matchLabels:
     #  authdeployment: east
     matchLabels:
     # The managedClusterSet to link the placement to, can be override using --cluster
   -set
     managedClusterSet:
     # The managedClusterSetBinding, if not present then it will be created to bind
     # the provided placement with the managedClusterSet
     # It can be overridden using --cluster-set-binding
     managedClusterSetBinding:
     # The list of identity providers
     identityProviders:
     # Example for github, this section will be copied into the authrealm CR.
     # Reference:
   https://github.com/openshift/api/blob/master/config/v1/0000_10_config-
   operator_01_oauth.crd.yaml#L80
     # The identity provider name
   ```

```
  - name: my-github-idp
    # The mappingMethod could be add, claim or lookup
    mappingMethod: claim
    # The identity provider type, here GitHub
    type: GitHub
    # The github specifics
    github:
      # The client ID of the github app
      clientID:
      # The github app secret, the cm-cli will create a local secret with it
      clientSecret:
      # Lists of GitHub Organizations (optionals)
      organizations:
      - myorg
    #,,,,
  # Example for LDAP, this section will be copied into the authrealm CR.
  # Reference:
https://github.com/openshift/api/blob/master/config/v1/0000_10_config-
operator_01_oauth.crd.yaml#L215
  # The identity provider name
  - name: my-ldap-idp
    # The mappingMethod could be add, claim or lookup
    mappingMethod: claim
    # The identity provider type, here LDAP
    type: LDAP
    # The ldap specifics
    ldap:
      # The LDAP server url
      url:
      # The bind Domain name
      bindDN:
      # The bind password, the cm-cli will create a local secret with it
      bindPassword:
    #....
  # Extra supported ldap configuration for the dex server
  #
  ldapExtraConfigs:
    # The name of the ldap identity provider
    my-ldap-idp:
      # The base Domain name
      baseDN:
      filter:
```

2. Create the authrealm

   a. Create directly

   ```
   cm create authrealm --values my-authrealm.yaml
   ```

   b. Create a file and then apply

Options --dry-run with --output-file can be used to get the rendered file

```
cm create authrealm --values my-authrealm.yaml --dry-run --output-file my-
authrealm-yamls.yaml
```

Then the my-authrealm-yamls.yaml can be applied using

```
oc apply -f my-authrealm-yamls.yaml
```

= Uninstalling

When you uninstall identity configuration management for Kubernetes, it might take up to 5 minutes to complete the uninstall process. The uninstall removes most operator components, excluding components such as custom resource definitions. After this uninstall, you must reinstall the operator before reinstalling the custom resource.

Prerequisite: Delete the IDPConfig and AuthRealm custom resources

Before you uninstall the identity configuration management for Kubernetes:

1. You must delete all of the AuthRealm custom resources that are managed by the operator.
2. You must delete the IDPConfig CR, this will delete several operators except the installer operator. The installer operator will be deleted when you delete the identity configuration management from the hub.

Note: When a cluster's authentication is no longer managed by identity configuration management, either because you've deleted all AuthRealms that apply to the cluster or because the cluster no longer matches any Placement rules in your AuthRealms, the original oauth configuration that was in place before identity configuration management began managing the cluster's oauth will be restored. This is only true if the managed cluster is still attached to the hub cluster at the point that the managed cluster becomes no longer manged by identity configuration management. Should you need to manually restore the original OAuth, you can find the backup in the ConfigMap named `idp-oauth-original` in the managed cluster's namespace on the hub cluster.

Removing resources by using commands

1. If you have not already, ensure that your OpenShift Container Platform CLI is configured to run `oc` commands. See Getting started with the OpenShift CLI in the OpenShift Container Platform documentation for more information about how to configure the `oc` commands.
2. Enter the following commands to locate and delete the identity configuration management `ClusterServiceVersion` in the namespace it is installed in:

```
 oc get csv -A
NAMESPACE            NAME                          DISPLAY
VERSION   REPLACES   PHASE
idp-mgmt-config      idp-mgmt-operator.v1.0.0      identity configuration management
for Kubernetes for Kubernetes   1.0.0                Succeeded

 oc delete clusterserviceversion idp-mgmt-operator.v1.0.0 -n idp-mgmt-config
 oc delete sub idp-mgmt-operator -n idp-mgmt-config
```

Note: The CSV version and operator namespace shown here may be different.

Deleting the components by using the console

When you use the RedHat OpenShift Container Platform console to uninstall, you remove the operator. Complete the following steps to uninstall by using the console:

1. In the OpenShift Container Platform console navigation, select **Operators** > **Installed Operators** > **identity configuration management for Kubernetes**.
2. Remove the *identity configuration management for Kubernetes* operator by selecting the *Options* menu and selecting **Uninstall operator**.

Manual cleanup steps

Once the identity configuration management operator has been removed, some manual cleanup is required to completely remove all artifacts of the Technology Preview.

1. Delete a remaining clusterrole

   Run the following command on the hub cluster:

   ```
   oc delete clusterrole dex-operator-dexsso
   ```

Troubleshooting Uninstall

If the identity configuration management custom resource is not being removed, remove any potential remaining artifacts by running the clean-up script.

1. Copy the following script into a file:

```
#!/bin/bash
oc delete crd authrealms.identityconfig.identitatem.io
oc delete crd clusteroauths.identityconfig.identitatem.io
oc delete crd strategies.identityconfig.identitatem.io
oc delete crd dexclients.auth.identitatem.io
oc delete crd dexservers.auth.identitatem.io
```

Troubleshooting

The installer pod CrashLoopBackOff

You launched the IDP installation from the OperatorHub and the installation failed. A reason could be that Red Hat Advanced Cluster Management or Multicluster Engine is not installed.

- Check the installer log

  ```
  oc logs -l control-plane=controller-manager -n <idp-installation-namespace>
  ```

  A message similar to the following means the prerequisite are not met.

  ```
  2022-01-10T14:17:21.962Z    ERROR   setup   unable to create controller
  {"controller": "Installer", "error": "IDP prerequisites are not met: neither Red
  Hat Advanced Cluster Management or Multicluster Engine installation has been
  detected, the product Red Hat Advanced Cluster Management is not installed on this
  cluster\nthe product Multicluster Engine is not installed on this cluster",
  "errorVerbose": "neither Red Hat Advanced Cluster Management or Multicluster Engine
  installation has been detected, the product Red Hat Advanced Cluster Management is
  not installed on this cluster\nthe product Multicluster Engine is not installed on
  this cluster\nIDP prerequisites are not met"}
  ```

Error appears on operator page after installing through OperatorHub

After instaling "identity configuration management", the "operator page" shows a 404 error.

- Check the installer log

```
oc logs -l control-plane=controller-manager -n <idp-installation-namespace>
```

A message similar to the following means a preview installation wasn't correctly cleaned.

```
2022-01-23T23:32:57.111Z    ERROR   controller.idpconfig    Reconciler error
{"reconciler group": "identityconfig.identitatem.io", "reconciler kind":
"IDPConfig", "name": "idp-config", "namespace": "idp-mgmt-config", "error": "\"idp-
mgmt-operator/leader_election_role_binding.yaml\" (string):
RoleBinding.rbac.authorization.k8s.io \"idp-mgmt-operator-leader-election-
rolebinding\" is invalid: roleRef: Invalid value:
rbac.RoleRef{APIGroup:\"rbac.authorization.k8s.io\", Kind:\"Role\", Name:\"leader-
election-operator-role\"}: cannot change roleRef", "errorVerbose": "\"idp-mgmt-
operator/leader_election_role_binding.yaml\" (string):
RoleBinding.rbac.authorization.k8s.io \"idp-mgmt-operator-leader-election-
rolebinding\" is invalid: roleRef: Invalid value:
rbac.RoleRef{APIGroup:\"rbac.authorization.k8s.io\", Kind:\"Role\", Name:\"leader-
election-operator-role\"}: cannot change roleRef\ngithub.com/identitatem/idp-mgmt-
operator/controllers/installer.(*IDPConfigReconciler).processIDPConfigCreation\n\t/
remote-
source/app/controllers/installer/installer_controller.go:168\ngithub.com/identitate
m/idp-mgmt-
operator/controllers/installer.(*IDPConfigReconciler).Reconcile\n\t/remote-
source/app/controllers/installer/installer_controller.go:126\nsigs.k8s.io/controlle
r-runtime/pkg/internal/controller.(*Controller).reconcileHandler\n\t/remote-
source/deps/gomod/pkg/mod/sigs.k8s.io/controller-
runtime@v0.9.7/pkg/internal/controller/controller.go:298\nsigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).processNextWorkItem\n\t/remote-
source/deps/gomod/pkg/mod/sigs.k8s.io/controller-
runtime@v0.9.7/pkg/internal/controller/controller.go:253\nsigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).Start.func2.2\n\t/remote-
source/deps/gomod/pkg/mod/sigs.k8s.io/controller-
runtime@v0.9.7/pkg/internal/controller/controller.go:214\nruntime.goexit\n\t/usr/li
b/golang/src/runtime/asm_amd64.s:1581"}
```

- Clean the RoleBinding

```
oc delete rolebinding idp-mgmt-operator-leader-election-rolebinding -n <idp-
installation-namespace>
```

Once the deletion done, reinstall the "identity configuration management" again.

General AuthRealm troubleshooting

The AuthRealm resource contains status information not only for itself, but for child resources used in configuring OAuth for your managed clusters. These child resources can include Strategy, Placement, ClusterOAuth and ManifestWork. The AuthRealm status should be the first place you check if the AuthRealm does not function properly. The AuthRealm status and conditions will allow you to quickly identify any resources that are not working properly and speed up the ability to apply corrective actions to fix the issue.

In addition, each Placement can have one or more PlacementDecision resources associated with it. The PlacementDecision will list the managed clusters the AuthRealm will be applied to, via ManifestWork. The name of the PlacementDecision resource will be based on the Placement name and have the format <Placement name>-decision-<number>, where <number> starts at one.

Note that AuthRealm child resources for Strategy, Placement and PlacementDecision are all in the same namespace as the AuthRealm. However, ClusterOAuth and ManifestWork are in the managed cluster namespace.

New login option specified in AuthRealm is not appearing on the managed cluster

You've created your AuthRealm and updated your managed cluster as appropriate such that it should match the Placement, but the new login option is not appearing on your managed cluster. Follow the steps below to determine the issue.

1. Wait several minutes. It takes time for the OpenShift authentication pods to restart and reflect the new configuration.
2. If the button does not appear after several minutes, run the following on the managed cluster to confirm the cluster OAuth configuration was propagated correctly from the hub: `oc get oauth cluster -oyaml` You should see an identity provider with a name matching your AuthRealm name and an issuer URL that reflects your hub cluster URL and routeSubDomain. If you do, continue to the next step. If not, check the following on the hub cluster:
   - Check the AuthRealm

     ```
     oc get authrealm <AuthRealm name> -n <AuthRealm namespace> -o yaml
     ```

     And verify the status.condition.type:Applied has a status of **"True"**. If the status is false, check the reason and message. Note: Applied **"True"** just mean the AuthRealm was successfully processed by the controller and subsequent resources are created; it doesn't mean the managed cluster authentication system is successfully updated.

   - Check for a ManifestWork named **idp-oauth** in the managed cluster namespace.

     ```
     oc get manifestworks idp-oauth -n <managed cluster namespace> -o yaml
     ```

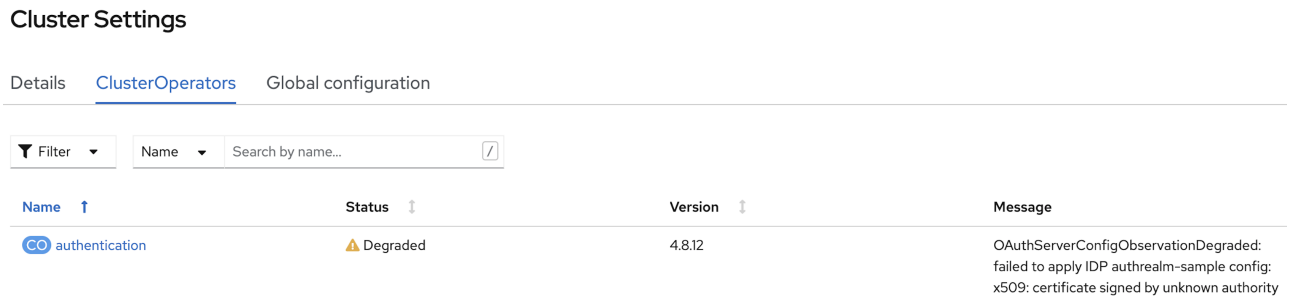
     And verify the status.condition.type:Applied has a status of **"True"**. If the status is false, wait a minute or two and recheck. If the ManifestWork does not exist, check the following on the hub cluster:

   - Check the PlacementDecision in the AuthRealm namespace to be sure the managed cluster was properly selected as part of the Placement and ManagedClusterSet.

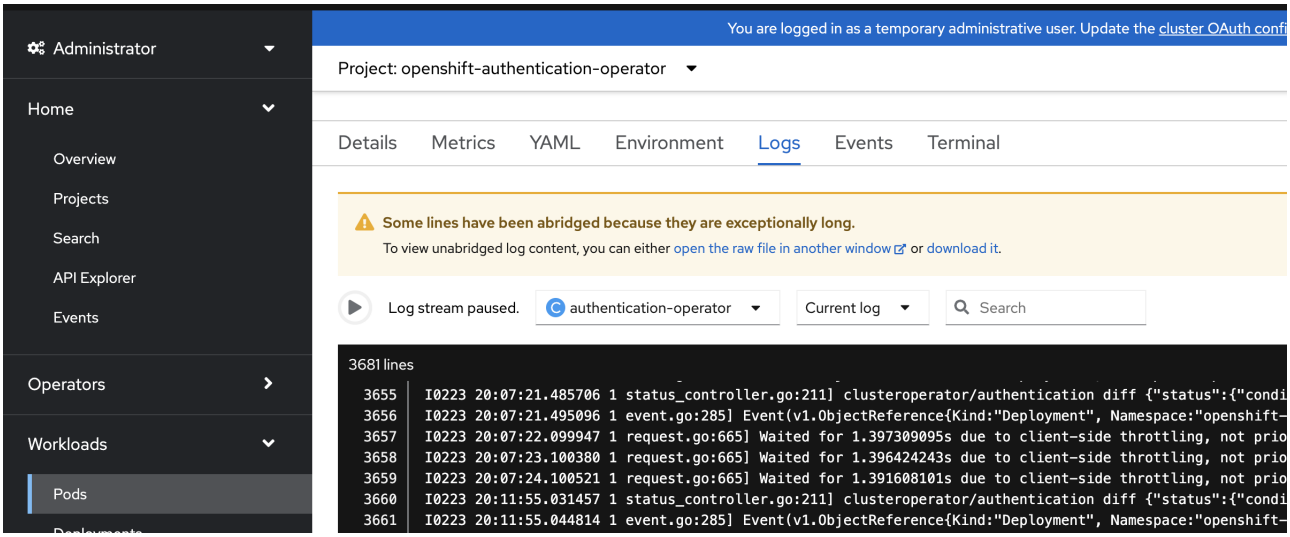
     ```
     oc get placementdecision -n <AuthRealm namespace> -o yaml
     ```

     Check the status.decision list contains an entry for the managed cluster. If not, verify the Placement, ManagedClusterSet and ManagedClusterSetBindings are properly configured to include the managed cluster.

3. From the OpenShift Container Platform console UI on the managed cluster, navigate to Administration→Cluster Settings→ClusterOperators. Check the Status of the `authentication` operator, and if it is not showing green Available, check the Message for an indication of the error.



4. From the OpenShift Container Platform console UI on the managed cluster, navigate to pods and search "authentication-operator". Check the logs for errors.



OpenShift authentication is failing with "An authentication error has occurred"

The new login option is showing up on your managed cluster, but when you try to complete the authentication flow, it ends with an error from OpenShift that says "An authentication error has occurred". Follow the steps below to determine the issue.

1. Check the logs of the authentication operator by running the following on the managed cluster:

   ```
   oc get pods -n openshift-authentication
   oc logs <pod_name>
   ```

2. If no errors are found, you may need to enable debug mode in the authentication operator:

   ```
   $ oc edit authentication.operator.openshift.io
     ...
     spec:
       logLevel: Debug  <-- change from Normal to Debug
     managementState: Managed
   ```

3. Wait for the openshift-authentication pods to restart and then check the logs again:

   ```
   watch oc get pods -n openshift-authentication
   oc logs <pod_name>
   ```

   Collisions with existing users and identities on the managed cluster can cause the "An authentication error has occurred" message. Consider whether your selected mapping method in your AuthRealm is right for your situation. Also remember that when you delete a user from a managed cluster, you will also need to delete the identity for that user in order for that user to be created correctly again on future login attempts with a new identity provider:

   ```
   oc get identity
   oc delete <identity_name>
   ```

OpenShift authentication is failing with "Application is not available"

Reaching the "Application is not available" 503 error page after clicking on the OpenShift login button can be an indication of a number of problems. Following are troubleshooting steps you can try.

Check your GitHub OAuth app callback URL

If you are using GitHub as your identity provider, and are able to successfully authenticate to GitHub, but your OpenShift login then ends with an "Application is not available" message, it is likely that you have an incorrect callback URL specified in your GitHub OAuth application. Review the callback URL in the browser address bar and confirm that it is your route subdomain specified in your AuthRealm (or the `host` if you are using a custom certificate).

Check if the OpenID Connect proxy server is running correctly

The OpenID Connect proxy server runs in the namespace `idp-mgmt-<AuthRealm.routeSubDomain>`. A DexServer custom resource is used by identity configuration management to configure and track this proxy server. From the `idp-mgmt-<AuthRealm.routeSubDomain>` OpenShift project on your hub cluster, run the following:

```
$ oc get dexserver dex-server -ojsonpath='{.status.conditions}'
[
  {
    "lastTransitionTime": "2021-11-04T20:30:35Z",
    "message": "DexServer is applied",
    "reason": "Applied",
    "status": "True",
    "type": "Applied"
  },
  {
    "lastTransitionTime": "2021-11-05T12:03:19Z",
    "message": "DexServer deployment is available",
    "reason": "Available",
    "status": "True",
    "type": "Available"
  }
]
```

Confirm no issues are reported. The type "Applied" indicates that the DexServer custom resource was successfully processed by the controller, and all related resources have been synchronized. The type "Available" reflects the status of the Dex server deploment pods.

Assuming your DexServer deployment is healthy, next check the ingress. Again from the `idp-mgmt-<AuthRealm.routeSubDomain>` OpenShift project on your hub cluster, run the following:

```
$ oc get ingress dex-server
NAME         CLASS    HOSTS                                        ADDRESS
PORTS     AGE
dex-server   <none>   yourRouteSubDomain.apps.clusterurl.com    router-
default.apps.clusterurl.com   80, 443   17h
```

An empty ADDRESS column is an indication that your ingress has a problem. One possible issue if you are using a custom certificate is that your certificate secret was not set up correctly in the `idp-mgmt-<AuthRealm.routeSubDomain` namespace. Run `oc get secrets` and confirm you see an entry with the name you specified in `AuthRealm.certificatesSecretRef.name` and that it is of type `kubernetes.io/tls`.

If the ingress is not found, an error message should be present in the DexServer custom resource status.

OpenShift authentication fails with "Bad Request" "Invalid client_id"

If after you press the OpenShift login button, you see a page displaying "Bad Request" and "Invalid client_id", there is an issue with the configuration for this managed cluster on the hub cluster.

On the hub cluster, look for the dex client that should have been created under the covers for this managed cluster. Run the following on the hub cluster, substituting your value for routeSubDomain:

```
$ oc get dexclients -n idp-mgmt-<routeSubDomain>
NAME                                  AGE
<managed_cluster>-<authrealm>         7m18s
```

You should see an entry for your managed cluster. If so, check the status to confirm the DexClient custom resource was successfully applied and an oauth2client was created on the Dex server.

```
$ oc get dexclient  <managed_cluster>-<authrealm> -n idp-mgmt-<routeSubDomain> -ojson
| jq -r '.status.conditions'
[
  {
    "lastTransitionTime": "2021-11-11T14:54:48Z",
    "message": "Dex client is created",
    "reason": "Created",
    "status": "True",
    "type": "Applied"
  },
  {
    "lastTransitionTime": "2021-11-11T14:54:48Z",
    "message": "oauth2client is created",
    "reason": "Created",
    "status": "True",
    "type": "OAuth2ClientCreated"
  }
]
```

You should see a status of "True" for both conditions. If not, review the errors. The type "Applied" indicates whether the controller has successfully processed the DexClient custom resource. The type "OAuth2ClientCreated" indicates whether an oauth2client was created on the Dex server for the managed cluster.

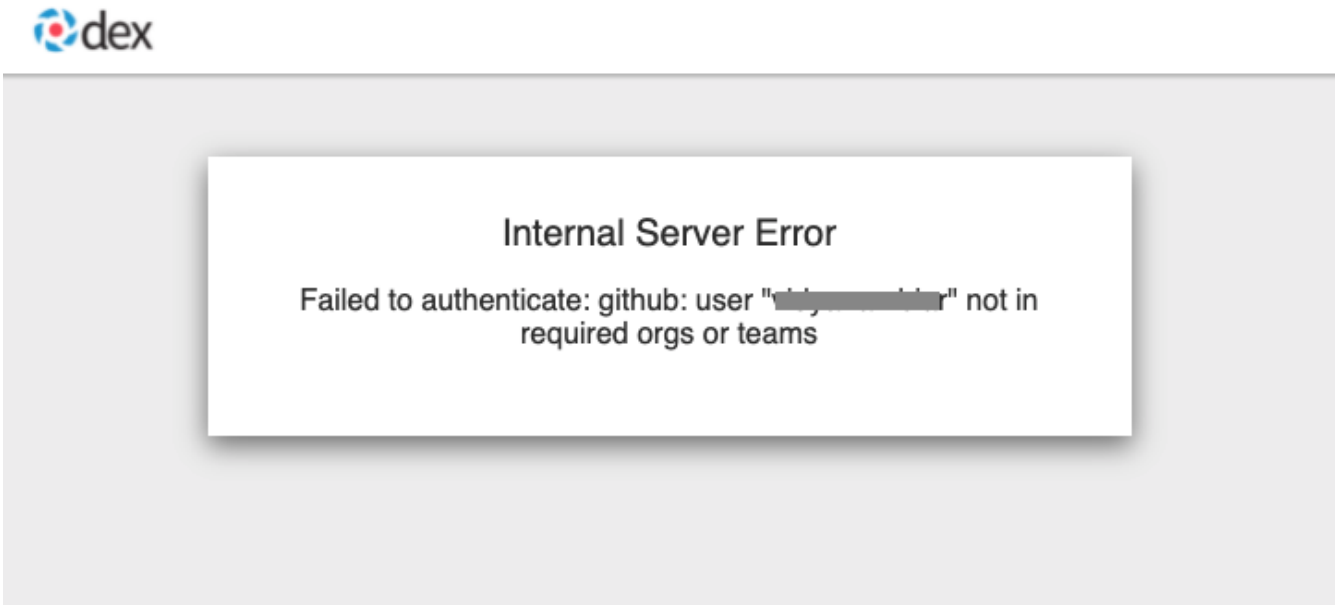
If you do not see a dex client for your managed cluster, check the operator logs for errors by running the following on the hub cluster:

```
$ oc logs -l control-plane=idp-mgmt-operator-manager -n idp-mgmt-config
```

Github login failure

User is not a member of the GitHub organization specified in the AuthRealm

If the AuthRealm custom resource specifies one or more values in the `github.organizations` field, the user attempting login to the cluster via GitHub will be authenticated against the specified organization(s). If the user is not a member of at least one of the organizations specified in the AuthRealm, they will not be able to login. The error will appear as follows:



- Check that the user attempting login is a member of at least one of the GitHub organizations specified in the AuthRealm.
- If the GitHub OAuth application is not owned by an organization specified in `github.organizations`, an organization owner must grant third-party access to use this option.

Internal Server Error: missing access_token

If after you press the OpenShift login button, you see a page displaying "Internal Server Error" and "Failed to authenticate: github: failed to get token: oauth2: server response missing access_token", you likely have an invalid GitHub OAuth client id and secret in your AuthRealm. Review your AuthRealm CR `spec.identityProviders.github.clientSecret` and `spec.identityProviders.github.clientSecret` fields, and ensure the base64-encoded client secret referenced in `spec.identityProviders.github.clientSecret` is correct by running the following:

```
$ oc get secret <your_secret_name> -n <your_authrealm_namespace> -ojson | jq -r
'.data.clientSecret' | base64 -d
```

You can match the trailing 8 characters of the base64-decoded secret with the Client secrets listed on your GitHub OAuth page. GitHub also indicates when the secret was last used, which can be a

clue as to whether your client secret has been applied successfully.

LDAP related issues

Issues with setting up the OpenID Connect proxy server correctly with the LDAP connector

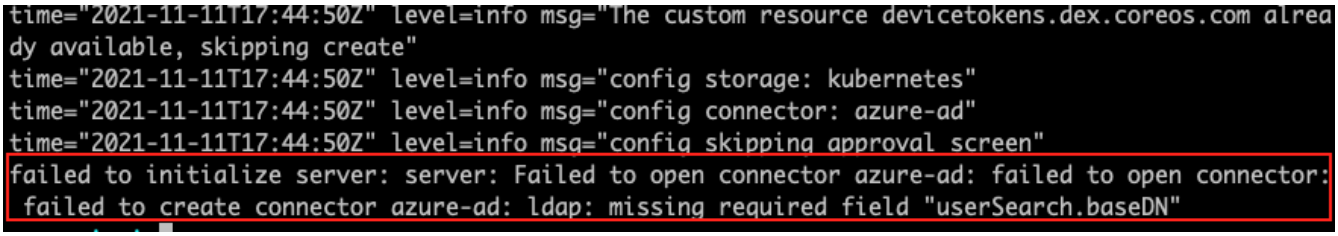
If you do not see the new login option even after several minutes, it is possible that the OpenID Connect proxy server is not deployed correctly as it is unable to set up the LDAP connector based on the information provided in the AuthRealm CR. For instance, not specifying `ldapExtraConfigs.<idp-name>.baseDN` in the AuthRealm CR or missing to create the secret containing the `bindPassword` can prevent the OpenID Connect proxy server from being deployed correctly.

1. Refer to the step above to view the status information in the DexServer custom resource.
2. Check the logs of the dex server pod by running the following on the hub cluster, substituting your value for `routeSubDomain`:

Fetch logs from the pod for more information on the error:

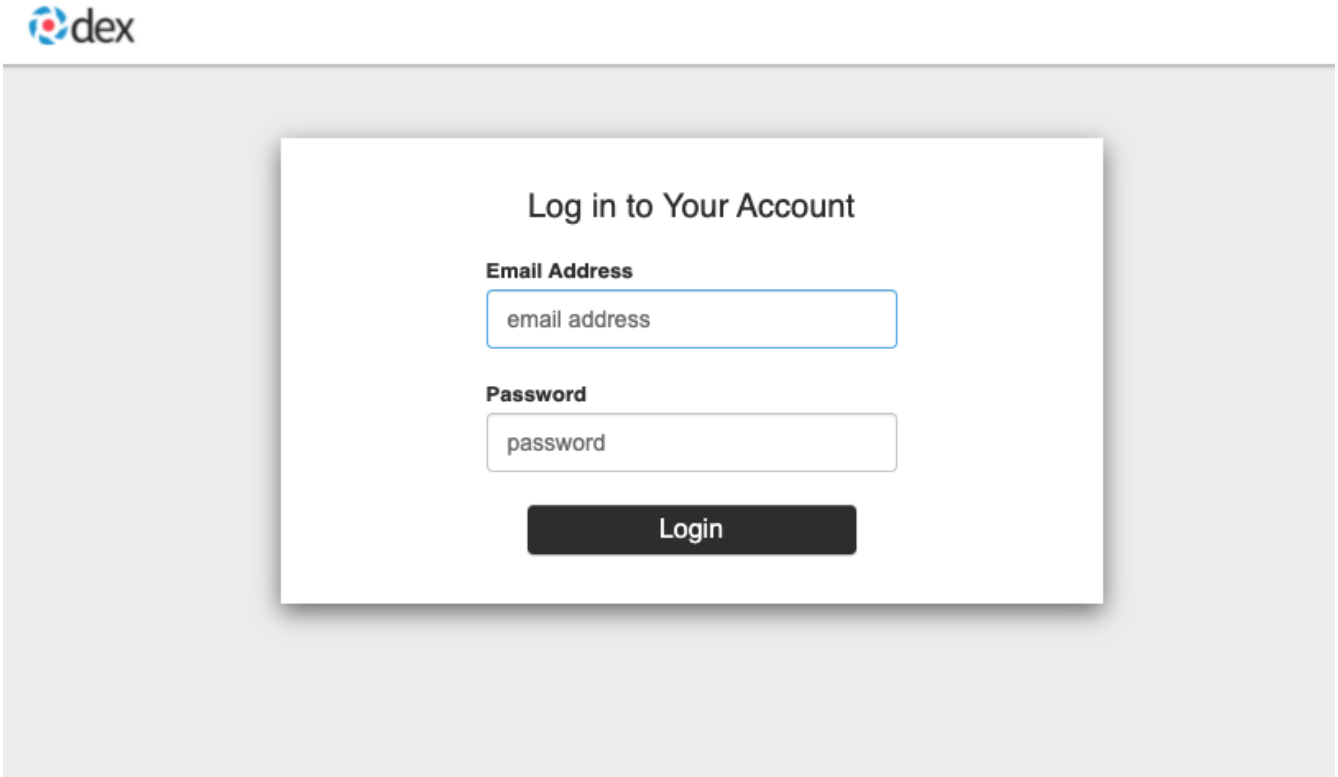
```
$ oc logs -l control-plane=dex-server -n idp-mgmt-<routeSubDomain>
```

Example:



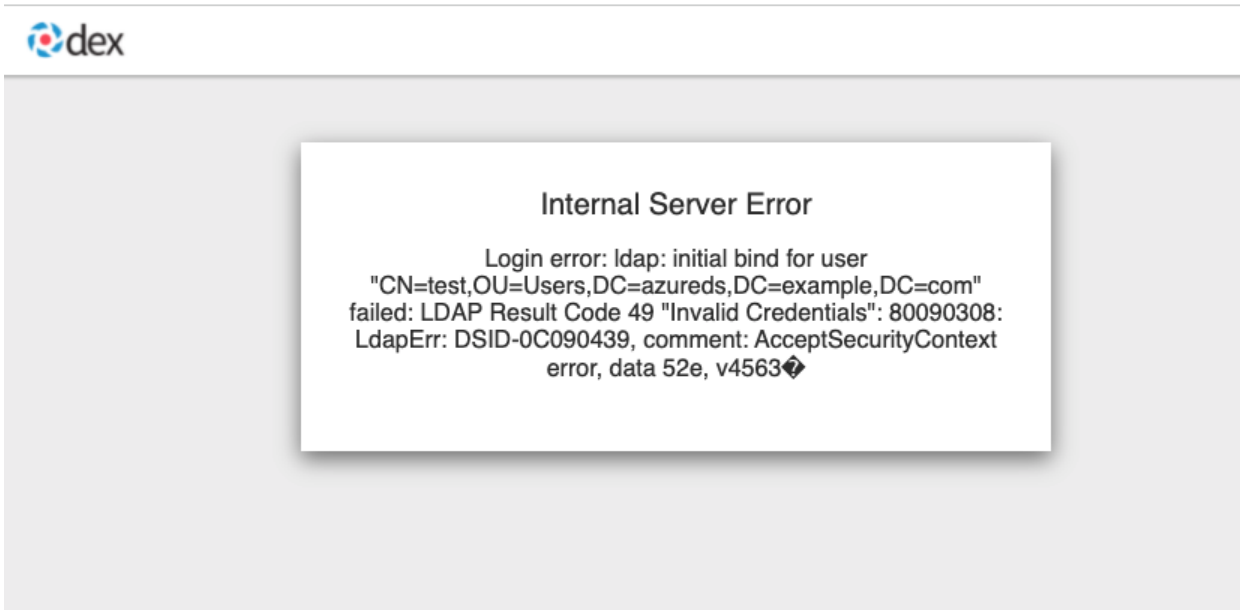
LDAP login failures

The new login option is showing up on your managed cluster, but when you try to complete the authentication flow via the LDAP login UI, it ends with an error.

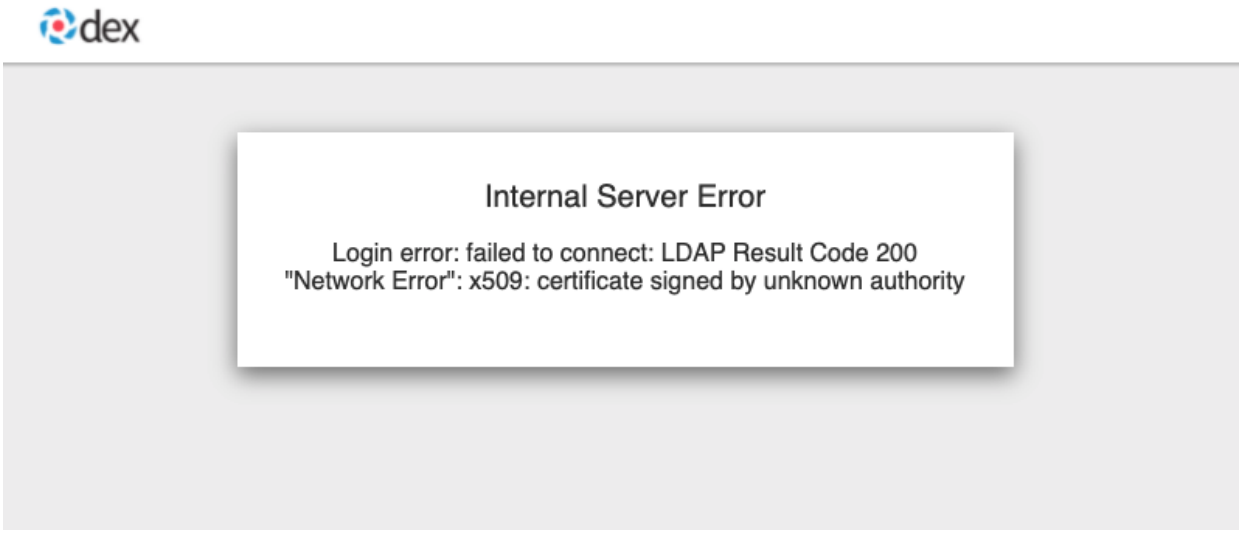


Examples of errors that could occur during this flow:

- Incorrect bind credentials used to authenticate with the LDAP server
  - Check that the correct bindDN and bindPassword was provided for the LDAP identity provider in the AuthRealm CR.



- Incorrect certificate for the LDAP server
  - Check that the correct certificate was provided in the secret that is referenced in the `ca` field for the ldap identity provider in the AuthRealm CR.



troubleshoot import groups

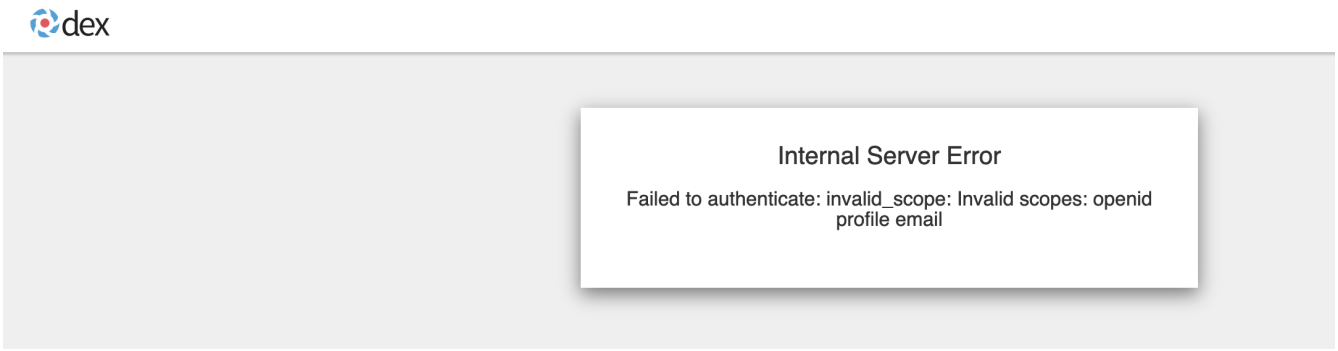
Groups are not created.

- Check the dex-server logs if the groups are displaied, if not you probably have an issue with the relationship between the user and the group in your IDP configuration. For example for LDAP, the user is not a member of the group.
- Check the authentication-operator log for errors: Authentication Operator Log

OpenID related issues

Invalid scopes: openid profile email

If you receive the following error:



The OpenID client needs to add default or optional client scopes for profile and email.