Configure GitLab OAuth authentication

- Before you begin
  - Create a GitLab OAuth Application
- Configure GitLab authentication client using the Grafana UI
- Configure GitLab authentication client using the Terraform provider
- Configure GitLab authentication client using the Grafana configuration file
  - Steps
  - Configure a refresh token
  - Configure JWT ID token validation
  - Configure allowed groups
  - Configure role mapping
  - Role mapping examples
    - Org roles mapping example
    - Map roles using user information from OAuth token
    - Map roles using groups
    - Map server administrator role
    - Map one role to all users
  - Example of GitLab configuration in Grafana
- Configure team synchronization
- Configuration options

{{< docs/shared lookup="auth/intro.md" source="grafana" version="" >}}

This topic describes how to configure GitLab OAuth authentication.

{{< admonition type="note" >}} If Users use the same email address in GitLab that they use with other authentication providers (such as Grafana.com), you need to do additional configuration to ensure that the users are matched correctly. Please refer to the Using the same email address to

login with different identity providers documentation for more information. {{< /admonition >}}

Before you begin

Ensure you know how to create a GitLab OAuth application. Consult GitLab's documentation on creating a GitLab OAuth application for more information.

Create a GitLab OAuth Application

1. Log in to your GitLab account and go to **Profile > Preferences > Applications**.
2. Click **Add new application**.
3. Fill out the fields.
   - In the **Redirect URI** field, enter the following: `https://<YOUR-GRAFANA-URL>/login/gitlab` and check `openid`, `email`, `profile` in the **Scopes** list.
   - Leave the **Confidential** checkbox checked.
4. Click **Save application**.
5. Note your **Application ID** (this is the `Client Id`) and **Secret** (this is the `Client Secret`).

Configure GitLab authentication client using the Grafana UI

As a Grafana Admin, you can configure GitLab OAuth client from within Grafana using the GitLab UI. To do this, navigate to **Administration > Authentication > GitLab** page and fill in the form. If you have a current configuration in the Grafana configuration file then the form will be pre-populated with those values otherwise the form will contain default values.

After you have filled in the form, click **Save** to save the configuration. If the save was successful, Grafana will apply the new configurations.

If you need to reset changes you made in the UI back to the default values, click **Reset**. After you have reset the changes, Grafana will apply the configuration from the Grafana configuration file (if there is any configuration) or the default values.

{{< admonition type="note" >}} If you run Grafana in high availability mode, configuration changes may not get applied to all Grafana instances immediately. You may need to wait a few minutes for the configuration to propagate to all Grafana instances. {{< /admonition >}}

Refer to configuration options for more information.

Configure GitLab authentication client using the Terraform provider

```
resource "grafana_sso_settings" "gitlab_sso_settings" {
  provider_name = "gitlab"
  oauth2_settings {
    name                   = "Gitlab"
    client_id              = "YOUR_GITLAB_APPLICATION_ID"
    client_secret          = "YOUR_GITLAB_APPLICATION_SECRET"
    allow_sign_up          = true
    auto_login             = false
    scopes                 = "openid email profile"
    allowed_domains        = "mycompany.com mycompany.org"
    role_attribute_path    = "contains(groups[*], 'example-group') && 'Editor' || 'Viewer'"
    role_attribute_strict = false
    allowed_groups         = "[\"admins\", \"software engineers\", \"developers/frontend\"]"
    use_pkce               = true
    use_refresh_token      = true
  }
}
```

Go to Terraform Registry for a complete reference on using the `grafana_sso_settings` resource.

Configure GitLab authentication client using the Grafana configuration file

Ensure that you have access to the Grafana configuration file.

Steps

To configure GitLab authentication with Grafana, follow these steps:

1. Create an OAuth application in GitLab.
   1. Set the redirect URI to `http://<my_grafana_server_name_or_ip>:<grafana_server_port>/login/gitlab`.
      Ensure that the Redirect URI is the complete HTTP address that you use to access Grafana via your browser, but with the appended path of `/login/gitlab`.
      For the Redirect URI to be correct, it might be necessary to set the `root_url` option in the `[server]` section of the Grafana configuration file. For example, if you are serving Grafana behind a proxy.
   2. Set the OAuth2 scopes to `openid`, `email` and `profile`.
2. Refer to the following table to update field values located in the `[auth.gitlab]` section of the Grafana configuration file:

Field	Description
`client_id`, `client_secret`	These values must match the `Application ID` and `Secret` from your GitLab OAuth application.
`enabled`	Enables GitLab authentication. Set this value to `true`.

   Review the list of other GitLab configuration options and complete them, as necessary.
3. Optional: Configure a refresh token:
   a. Set `use_refresh_token` to `true` in `[auth.gitlab]` section in Grafana configuration file.
4. Configure role mapping.
5. Optional: Configure team synchronization.
6. Restart Grafana.
   You should now see a GitLab login button on the login page and be able to log in or sign up with your GitLab accounts.

Configure a refresh token

When a user logs in using an OAuth provider, Grafana verifies that the access token has not expired. When an access token expires, Grafana uses the provided refresh token (if any exists) to obtain a new access token.

Grafana uses a refresh token to obtain a new access token without requiring the user to log in again. If a refresh token doesn't exist, Grafana logs the user out of the system after the access token has expired.

By default, GitLab provides a refresh token.

Refresh token fetching and access token expiration check is enabled by default for the GitLab provider since Grafana v10.1.0. If you would like to disable access token expiration check then set the `use_refresh_token` configuration value to `false`.

{{< admonition type="note" >}} The `accessTokenExpirationCheck` feature toggle has been removed in Grafana v10.3.0. Use the `use_refresh_token` configuration value instead for configuring refresh token fetching and access token expiration check. {{< /admonition >}}

Configure JWT ID token validation

By default, Grafana extracts user information from ID tokens without validating their cryptographic signatures. To enhance security, you can enable JWT signature validation to ensure that ID tokens are authentic and have not been tampered with.

To enable JWT ID token validation:

1. Set `validate_id_token` to `true` in the `[auth.gitlab]` section of the Grafana configuration file.
2. Configure `jwk_set_url` with the URL of your GitLab instance's JSON Web Key Set (JWKS) endpoint. This endpoint provides the public keys used to verify JWT signatures.
   For GitLab.com, the JWKS endpoint is: `https://gitlab.com/oauth/discovery/keys`
   For self-hosted GitLab instances, the JWKS endpoint is typically: `https://<your-gitlab-domain>/oauth/discovery/keys`

Example configuration:

```
[auth.gitlab]
enabled = true
validate_id_token = true
jwk_set_url = https://gitlab.com/oauth/discovery/keys
client_id = <client id>
client_secret = <client secret>
scopes = openid email profile
```

{{< admonition type="note" >}} When JWT validation is enabled, Grafana caches the JWKS keys to improve performance. The cache respects the `Cache-Control` header from the JWKS endpoint response. If no cache expiration is specified, keys are cached for 5 minutes by default. {{< /admonition >}}

{{< admonition type="caution" >}} If `validate_id_token` is set to `true`, you must configure `jwk_set_url`. Authentication will fail if the JWK Set URL is not provided or if the ID token signature cannot be verified. {{< /admonition >}}

Configure allowed groups

To limit access to authenticated users that are members of one or more GitLab groups, set `allowed_groups` to a comma or space-separated list of groups.

GitLab's groups are referenced by the group name. For example, `developers`. To reference a subgroup `frontend`, use `developers/frontend`. Note that in GitLab, the group or subgroup name does not always match its display name, especially if the display name contains spaces or special characters. Make sure you always use the group or subgroup name as it appears in the URL of the group or subgroup.

Configure role mapping

Unless `skip_org_role_sync` option is enabled, the user's role will be set to the role retrieved from GitLab upon user login.

The user's role is retrieved using a JMESPath expression from the `role_attribute_path` configuration option. To map the server administrator role, use the `allow_assign_grafana_admin` configuration option. Refer to configuration options for more information.

You can use the `org_mapping` configuration option to assign the user to multiple organizations and specify their role based on their GitLab group membership. For more information, refer to Org roles mapping example. If the org role mapping (`org_mapping`) is specified and Entra ID returns a valid role, then the user will get the highest of the two roles.

If no valid role is found, the user is assigned the role specified by the `auto_assign_org_role` option. You can disable this default role assignment by setting `role_attribute_strict = true`. This setting denies user access if no role or an invalid role is returned after evaluating the `role_attribute_path` and the `org_mapping` expressions.

To ease configuration of a proper JMESPath expression, go to JMESPath to test and evaluate expressions with custom payloads.

Role mapping examples

This section includes examples of JMESPath expressions used for role mapping.

Org roles mapping example

The GitLab integration uses the external users' groups in the `org_mapping` configuration to map organizations and roles based on their GitLab group membership.

In this example, the user has been granted the role of a `Viewer` in the `org_foo` organization, and the role of an `Editor` in the `org_bar` and `org_baz` orgs.

The external user is part of the following GitLab groups: `groupd-1` and `group-2`.

Config:

```
org_mapping = group-1:org_foo:Viewer groupd-1:org_bar:Editor *:org_baz:Editor
```

Map roles using user information from OAuth token

In this example, the user with email `admin@company.com` has been granted the `Admin` role. All other users are granted the `Viewer` role.

```
role_attribute_path = email=='admin@company.com' && 'Admin' || 'Viewer'
```

Map roles using groups

In this example, the user from GitLab group 'example-group' have been granted the `Editor` role. All other users are granted the `Viewer` role.

```
role_attribute_path = contains(groups[*], 'example-group') && 'Editor' || 'Viewer'
```

Map server administrator role

In this example, the user with email `admin@company.com` has been granted the `Admin` organization role as well as the Grafana server admin role. All other users are granted the `Viewer` role.

```
role_attribute_path = email=='admin@company.com' && 'GrafanaAdmin' || 'Viewer'
```

Map one role to all users

In this example, all users will be assigned `Viewer` role regardless of the user information received from the identity provider.

```
role_attribute_path = "'Viewer'"
skip_org_role_sync = false
```

Example of GitLab configuration in Grafana

This section includes an example of GitLab configuration in the Grafana configuration file.

```
[auth.gitlab]
enabled = true
allow_sign_up = true
auto_login = false
client_id = YOUR_GITLAB_APPLICATION_ID
client_secret = YOUR_GITLAB_APPLICATION_SECRET
scopes = openid email profile
auth_url = https://gitlab.com/oauth/authorize
token_url = https://gitlab.com/oauth/token
api_url = https://gitlab.com/api/v4
role_attribute_path = contains(groups[*], 'example-group') && 'Editor' || 'Viewer'
role_attribute_strict = false
allow_assign_grafana_admin = false
allowed_groups = ["admins", "software engineers", "developers/frontend"]
allowed_domains = mycompany.com mycompany.org
tls_skip_verify_insecure = false
use_pkce = true
use_refresh_token = true
```

Configure team synchronization

{{< admonition type="note" >}} Available in Grafana Enterprise and Grafana Cloud {{< /admonition >}}

By using Team Sync, you can map GitLab groups to teams within Grafana. This will automatically assign users to the appropriate teams. Teams for each user are synchronized when the user logs in.

GitLab groups are referenced by the group name. For example, `developers`. To reference a subgroup `frontend`, use `developers/frontend`. Note that in GitLab, the group or subgroup name does not always match its display name, especially if the display name contains spaces or special characters. Make sure you always use the group or subgroup name as it appears in the URL of the group or subgroup.

To learn more about Team Sync, refer to Configure team sync.

Configuration options

The following table describes all GitLab OAuth configuration options. You can apply these options as environment variables, similar to any other configuration within Grafana. For more information, refer to Override configuration with environment variables.

{{< admonition type="note" >}} If the configuration option requires a JMESPath expression that includes a colon, enclose the entire expression in quotes to prevent parsing errors. For example `role_attribute_path: "role:view"` {{< /admonition >}}

Setting	Required	Supported on Cloud	Description	Default
`enabled`	Yes	Yes	Whether GitLab OAuth authentication is allowed.	`false`
`client_id`	Yes	Yes	Client ID provided by your GitLab OAuth app.	
`client_secret`	Yes	Yes	Client secret provided by your GitLab OAuth app.	
`auth_url`	Yes	Yes	Authorization endpoint of your GitLab OAuth provider. If you use your own instance of GitLab instead of gitlab.com, adjust `auth_url` by replacing the `gitlab.com` hostname with your own.	`https://gitlab.com/oauth/authorize`
`token_url`	Yes	Yes	Endpoint used to obtain GitLab OAuth access token. If you use your own instance of GitLab instead of gitlab.com, adjust `token_url` by replacing the `gitlab.com` hostname with your own.	`https://gitlab.com/oauth/token`

Setting	Required	Supported on Cloud	Description	Default
`token_exchange_timeout`	No	Yes	Timeout in seconds for the OAuth token exchange request. If set to `0` or left unset, defaults to `15` seconds.	`15`
`api_url`	No	Yes	Grafana uses `<api_url>/user` endpoint to obtain GitLab user information compatible with OpenID UserInfo.	`https://gitlab.com/api/v4`
`name`	No	Yes	Name used to refer to the GitLab authentication in the Grafana user interface.	`GitLab`
`icon`	No	Yes	Icon used for GitLab authentication in the Grafana user interface.	`gitlab`
`scopes`	No	Yes	List of comma or space-separated GitLab OAuth scopes.	`openid email profile`
`allow_sign_up`	No	Yes	Whether to allow new Grafana user creation through GitLab login. If set to `false`, then only existing Grafana users can log in with GitLab OAuth.	`true`
`auto_login`	No	Yes	Set to `true` to enable users to bypass the login screen and automatically log in. This setting is ignored if you configure multiple auth providers to use auto-login.	`false`
`login_prompt`	No	Yes	Indicates the type of user interaction when the user logs in with GitLab. Available values are `login`, `consent` and `select_account`.	

Setting	Required	Supported on Cloud	Description	Default
`role_attribute_path`	No	Yes	JMESPath expression to use for Grafana role lookup. Grafana will first evaluate the expression using the GitLab OAuth token. If no role is found, Grafana creates a JSON data with `groups` key that maps to groups obtained from GitLab's `/oauth/userinfo` endpoint, and evaluates the expression using this data. Finally, if a valid role is still not found, the expression is evaluated against the user information retrieved from `api_url/users` endpoint and groups retrieved from `api_url/groups` endpoint. The result of the evaluation should be a valid Grafana role (`None`, `Viewer`, `Editor`, `Admin` or `GrafanaAdmin`). For more information on user role mapping, refer to Configure role mapping.	
`role_attribute_strict`	No	Yes	Set to `true` to deny user login if the Grafana role cannot be extracted using `role_attribute_path`. For more information on user role mapping, refer to Configure role mapping.	`false`

Setting	Required	Supported on Cloud	Description	Default
`org_mapping`	No	No	List of comma- or space-separated `<ExternalGitlabGroupName>:<OrgIdOrName>:<Role>` mappings. Value can be `*` meaning "All users". Role is optional and can have the following values: `None`, `Viewer`, `Editor` or `Admin`. For more information on external organization to role mapping, refer to Org roles mapping example.	
`skip_org_role_sync`	No	Yes	Set to `true` to stop automatically syncing user roles.	`false`
`allow_assign_grafana_admin`	No	No	Set to `true` to enable automatic sync of the Grafana server administrator role. If this option is set to `true` and the result of evaluating `role_attribute_path` for a user is `GrafanaAdmin`, Grafana grants the user the server administrator privileges and organization administrator role. If this option is set to `false` and the result of evaluating `role_attribute_path` for a user is `GrafanaAdmin`, Grafana grants the user only organization administrator role. For more information on user role mapping, refer to Configure role mapping.	`false`

Setting	Required	Supported on Cloud	Description	Default
`allowed_domains`	No	Yes	List of comma or space-separated domains. User must belong to at least one domain to log in.	
`allowed_groups`	No	Yes	List of comma or space-separated groups. The user should be a member of at least one group to log in.	
`tls_skip_verify_insecure`	No	No	If set to `true`, the client accepts any certificate presented by the server and any host name in that certificate. *You should only use this for testing*, because this mode leaves SSL/TLS susceptible to man-in-the-middle attacks.	`false`
`tls_client_cert`	No	No	The path to the certificate.	
`tls_client_key`	No	No	The path to the key.	
`tls_client_ca`	No	No	The path to the trusted certificate authority list.	
`use_pkce`	No	Yes	Set to `true` to use Proof Key for Code Exchange (PKCE). Grafana uses the SHA256 based `S256` challenge method and a 128 bytes (base64url encoded) code verifier.	`true`
`use_refresh_token`	No	Yes	Set to `true` to use refresh token and check access token expiration. The `accessTokenExpirationCheck` feature toggle should also be enabled to use refresh token.	`true`

Setting	Required	Supported on Cloud	Description	Default
`validate_id_token`	No	Yes	If enabled, Grafana will validate the JWT signature of ID tokens using the JWKS endpoint. This enhances security by ensuring tokens are authentic and have not been tampered with.	`false`
`jwk_set_url`	No	Yes	URL of the JSON Web Key Set (JWKS) endpoint used to verify JWT ID token signatures. Required when ID token validation is enabled. For GitLab.com, use `https://gitlab.com/oauth/discovery/keys`. For self-hosted GitLab instances, use `https://<your-gitlab-domain>/oauth/discovery/keys`.	
`signout_redirect_url`	No	Yes	URL to redirect to after the user logs out.	

□□□ #2
□ □□□ □□□ 2026-10-31 16:12:33 UTC
□ □□□ □□□ 2026-10-31 16:12:37 UTC