GitLab Provider

The GitLab provider is used to interact with GitLab group or user resources.

It needs to be configured with the proper credentials before it can be used.

Use the navigation to the left to read about the available resources.

Example Usage

```
# Configure the GitLab Provider
provider "gitlab" {
    token = "${var.gitlab_token}"
}

# Add a project owned by the user
resource "gitlab_project" "sample_project" {
    name = "example"
}

# Add a hook to the project
resource "gitlab_project_hook" "sample_project_hook" {
    project = "${gitlab_project.sample_project.id}"
    url = "https://example.com/project_hook"
}

# Add a variable to the project
resource "gitlab_project_variable" "sample_project_variable" {
    project = "${gitlab_project.sample_project.id}"
    key = "project_variable_key"
    value = "project_variable_value"
}

# Add a deploy key to the project
resource "gitlab_deploy_key" "sample_deploy_key" {
    project = "${gitlab_project.sample_project.id}"
    title = "terraform example"
    key = "ssh-rsa AAAA..."
}

# Add a group
resource "gitlab_group" "sample_group" {
    name = "example"
    path = "example"
    description = "An example group"
}

# Add a project to the group - example/example
resource "gitlab_project" "sample_group_project" {
    name = "example"
    namespace_id = "${gitlab_group.sample_group.id}"
}
```

Argument Reference

The following arguments are supported in the `provider` block:

- `token` - (Optional) This is the GitLab personal access token. It must be provided, but it can also be sourced from the `GITLAB_TOKEN` environment variable.

- `base_url` - (Optional) This is the target GitLab base API endpoint. Providing a value is a requirement when working with GitLab CE or GitLab Enterprise e.g. `https://my.gitlab.server/api/v4/` . It is optional to provide this value and it can also be sourced from the `GITLAB_BASE_URL` environment variable. The value must end with a slash.

- `cacert_file` - (Optional) This is a file containing the ca cert to verify the gitlab instance. This is available for use when working with GitLab CE or Gitlab Enterprise with a locally-issued or self-signed certificate chain.

- `insecure` - (Optional; boolean, defaults to false) When set to true this disables SSL verification of the connection to the GitLab instance.

gitlab_group

Provides details about a specific group in the gitlab provider.

Example Usage

By group's ID

```
data "gitlab_group" "foo" {
  group_id = 123
}
```

By group's full path

```
data "gitlab_group" "foo" {
  full_path = "foo/bar"
}
```

Argument Reference

The following arguments are supported:

- `group_id` - (Optional) The ID of the group.
- `full_path` - (Optional) The full path of the group.

Note: exactly one of group_id or full_path must be provided.

Attributes Reference

The resource exports the following attributes:

- `id` - The unique ID assigned to the group.
- `name` - The name of this group.
- `path` - The path of the group.
- `description` - The description of the group.
- `lfs_enabled` - Boolean, is LFS enabled for projects in this group.
- `request_access_enabled` - Boolean, is request for access enabled to the group.
- `visibility_level` - Visibility level of the group. Possible values are `private`, `internal`, `public`.
- `parent_id` - Integer, ID of the parent group.

- `full_path` - The full path of the group.
- `full_name` - The full name of the group.
- `web_url` - Web URL of the group.

gitlab_project

Provides details about a specific project in the gitlab provider. The results include the name of the project, path, description, default branch, etc.

Example Usage

```
data "gitlab_project" "example" {
    id = 30
}
```

Argument Reference

The following arguments are supported:

- `id` - (Required) The integer that uniquely identifies the project within the gitlab install.

Attributes Reference

The following attributes are exported:

- `path` - The path of the repository.
- `namespace_id` - The namespace (group or user) of the project. Defaults to your user. See `gitlab_group` (/docs/providers/gitlab/r/group.html) for an example.
- `description` - A description of the project.
- `default_branch` - The default branch for the project.
- `issues_enabled` - Enable issue tracking for the project.
- `merge_requests_enabled` - Enable merge requests for the project.
- `wiki_enabled` - Enable wiki for the project.
- `snippets_enabled` - Enable snippets for the project.
- `visibility_level` - Repositories are created as private by default.
- `id` - Integer that uniquely identifies the project within the gitlab install.
- `ssh_url_to_repo` - URL that can be provided to `git clone` to clone the repository via SSH.
- `http_url_to_repo` - URL that can be provided to `git clone` to clone the repository via HTTP.
- `web_url` - URL that can be used to find the project in a browser.
- `runners_token` - Registration token to use during runner setup.

- `archived` - Whether the project is in read-only mode (archived).

gitlab_user

Provides details about a specific user in the gitlab provider. Especially the ability to lookup the id for linking to other resources.

Example Usage

```
data "gitlab_user" "example" {
  username = "myuser"
}
```

Argument Reference

The following arguments are supported:

- `email` - (Optional) The e-mail address of the user. (Requires administrator privileges)
- `username` - (Optional) The username of the user.
- `user_id` - (Optional) The ID of the user.

Note: only one of email, user_id or username must be provided.

Attributes Reference

- `id` - The unique id assigned to the user by the gitlab server.
- `username` - The username of the user.
- `email` - The e-mail address of the user.
- `name` - The name of the user.
- `is_admin` - Whether the user is an admin.
- `can_create_group` - Whether the user can create groups.
- `can_create_project` - Whether the user can create projects.
- `projects_limit` - Number of projects the user can create.
- `created_at` - Date the user was created at.
- `state` - Whether the user is active or blocked.
- `external` - Whether the user is external.
- `extern_uid` - The external UID of the user.
- `user_provider` - The UID provider of the user.

- `organization` - The organization of the user.
- `two_factor_enabled` - Whether user's two factor auth is enabled.
- `avatar_url` - The avatar URL of the user.
- `bio` - The bio of the user.
- `location` - The location of the user.
- `skype` - Skype username of the user.
- `linkedin` - Linkedin profile of the user.
- `twitter` - Twitter username of the user.
- `website_url` - User's website URL.
- `theme_id` - User's theme ID.
- `color_scheme_id` - User's color scheme ID.
- `last_sign_in_at` - Last user's sign-in date.
- `current_sign_in_at` - Current user's sign-in date.

Note: some attributes might not be returned depending on if you're an admin or not. Please refer to doc (https://docs.gitlab.com/ce/api/users.html#single-user) for more details.

gitlab_users

Provides details about a list of users in the gitlab provider. The results include id, username, email, name and more about the requested users. Users can also be sorted and filtered using several options.

NOTE: Some of the available options require administrator privileges. Please visit Gitlab API documentation (https://docs.gitlab.com/ce/api/users.html#for-admins) for more information.

Example Usage

```
data "gitlab_users" "example" {
  sort = "desc"
  order_by = "name"
  created_before = "2019-01-01"
}
```

Argument Reference

The following arguments are supported:

- `search` - (Optional) Search users by username, name or email.
- `active` - (Optional) Filter users that are active.
- `blocked` - (Optional) Filter users that are blocked.
- `order_by` - (Optional) Order the users' list by `id`, `name`, `username`, `created_at` or `updated_at`. (Requires administrator privileges)
- `sort` - (Optional) Sort users' list in asc or desc order. (Requires administrator privileges)
- `extern_uid` - (Optional) Lookup users by external UID. (Requires administrator privileges)
- `extern_provider` - (Optional) Lookup users by external provider. (Requires administrator privileges)
- `created_before` - (Optional) Search for users created before a specific date. (Requires administrator privileges)
- `created_after` - (Optional) Search for users created after a specific date. (Requires administrator privileges)

Attributes Reference

The following attributes are exported:

- `users` - The list of users.
  - `id` - The unique id assigned to the user by the gitlab server.
  - `username` - The username of the user.
  - `email` - The e-mail address of the user.

- `name` - The name of the user.
- `is_admin` - Whether the user is an admin.
- `can_create_group` - Whether the user can create groups.
- `can_create_project` - Whether the user can create projects.
- `projects_limit` - Number of projects the user can create.
- `created_at` - Date the user was created at.
- `state` - Whether the user is active or blocked.
- `external` - Whether the user is external.
- `extern_uid` - The external UID of the user.
- `provider` - The UID provider of the user.
- `organization` - The organization of the user.
- `two_factor_enabled` - Whether user's two factor auth is enabled.
- `avatar_url` - The avatar URL of the user.
- `bio` - The bio of the user.
- `location` - The location of the user.
- `skype` - Skype username of the user.
- `linkedin` - Linkedin profile of the user.
- `twitter` - Twitter username of the user.
- `website_url` - User's website URL.
- `theme_id` - User's theme ID.
- `color_scheme_id` - User's color scheme ID.
- `last_sign_in_at` - Last user's sign-in date.
- `current_sign_in_at` - Current user's sign-in date.

gitlab_branch_protection

This resource allows you to protect a specific branch by an access level so that the user with less access level cannot Merge/Push to the branch. GitLab EE features to protect by group or user are not supported.

Example Usage

```
resource "gitlab_branch_protection" "BranchProtect" {
  project = "12345"
  branch = "BranchProtected"
  push_access_level = "developer"
  merge_access_level = "developer"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) The id of the project.
- `branch` - (Required) Name of the branch.
- `push_access_level` - (Required) One of five levels of access to the project.
- `merge_access_level` - (Required) One of five levels of access to the project.

gitlab_deploy_key_enable

This resource allows you to enable pre-existing deploy keys for your GitLab projects.

the GITLAB KEY_ID for the deploy key must be known

Example Usage

```
# A repo to host the deployment key
resource "gitlab_project" "parent" {
  name = "parent_project"
}

# A second repo to use the deployment key from the parent project
resource "gitlab_project" "foo" {
  name = "foo_project"
}

# Upload a deployment key for the parent repo
resource "gitlab_deploy_key" "parent" {
  project = "${gitlab_project.parent.id}"
  title = "Example deploy key"
  key = "ssh-rsa AAAA..."
}

# Enable the deployment key on the second repo
resource "gitlab_deploy_key_enable" "foo" {
  project = "${gitlab_project.foo.id}"
  key_id = "${gitlab_deploy_key.parent.id}"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the deploy key to.
- `key_id` - (Required, string) The Gitlab key id for the pre-existing deploy key

Import

GitLab enabled deploy keys can be imported using an id made up of `{project_id}:{deploy_key_id}`, e.g.

```
$ terraform import gitlab_deploy_key_enable.example 12345:67890
```

gitlab_deploy_key

This resource allows you to create and manage deploy keys for your GitLab projects.

Example Usage

```
resource "gitlab_deploy_key" "example" {
  project = "example/deploying"
  title   = "Example deploy key"
  key     = "ssh-rsa AAAA..."
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the deploy key to.
- `title` - (Required, string) A title to describe the deploy key with.
- `key` - (Required, string) The public ssh key body.
- `can_push` - (Optional, boolean) Allow this deploy key to be used to push changes to the project. Defaults to `false`. **NOTE::** this cannot currently be managed.

Import

GitLab deploy keys can be imported using an id made up of `{project_id}:{deploy_key_id}`, e.g.

```
$ terraform import gitlab_deploy_key.test 1:3
```

gitlab_group

This resource allows you to create and manage GitLab groups. Note your provider will need to be configured with admin-level access for this resource to work.

Example Usage

```
resource "gitlab_group" "example" {
  name        = "example"
  path        = "example"
  description = "An example group"
}

// Create a project in the example group
resource "gitlab_project" "example" {
  name         = "example"
  description  = "An example project"
  parent_id = "${gitlab_group.example.id}"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of this group.
- `path` - (Required) The path of the group.
- `description` - (Optional) The description of the group.
- `lfs_enabled` - (Optional) Boolean, defaults to true. Whether to enable LFS support for projects in this group.
- `request_access_enabled` - (Optional) Boolean, defaults to false. Whether to enable users to request access to the group.
- `visibility_level` - (Optional) Set to `public` to create a public group. Valid values are `private`, `internal`, `public`. Groups are created as private by default.
- `parent_id` - (Optional) Integer, id of the parent group (creates a nested group).

Attributes Reference

The resource exports the following attributes:

- `id` - The unique id assigned to the group by the GitLab server. Serves as a namespace id where one is needed.
- `full_path` - The full path of the group.
- `full_name` - The full name of the group.

- `web_url` - Web URL of the group.

Importing groups

You can import a group state using `terraform import <resource> <id>`. The `id` can be whatever the details of a group (https://docs.gitlab.com/ee/api/groups.html#details-of-a-group) api takes for its `:id` value, so for example:

```
terraform import gitlab_group.example example
```

gitlab_group_membership

This resource allows you to add a user to an existing group.

Example Usage

```
resource "gitlab_group_membership" "test" {
  group_id     = "12345"
  user_id      = 1337
  access_level = "guest"
  expires_at   = "2020-12-31"
}
```

Argument Reference

The following arguments are supported:

- `group_id` - (Required) The id of the group.
- `user_id` - (Required) The id of the user.
- `access_level` - (Required) Acceptable values are: guest, reporter, developer, master, owner.
- `expires_at` - (Optional) Expiration date for the group membership. Format: `YYYY-MM-DD`

Import

GitLab group membership can be imported using an id made up of `group_id:user_id`, e.g.

```
$ terraform import gitlab_group_membership.test "12345:1337"
```

gitlab_group_variable

This resource allows you to create and manage CI/CD variables for your GitLab groups. For further information on variables, consult the gitlab documentation (https://docs.gitlab.com/ce/ci/variables/README.html#variables).

Example Usage

```
resource "gitlab_group_variable" "example" {
   group     = "12345"
   key       = "group_variable_key"
   value     = "group_variable_value"
   protected = false
}
```

Argument Reference

The following arguments are supported:

- `group` - (Required, string) The name or id of the group to add the hook to.
- `key` - (Required, string) The name of the variable.
- `value` - (Required, string) The value of the variable.
- `variable_type` - (Optional, string) The type of a variable. Available types are: env_var (default) and file.
- `protected` - (Optional, boolean) If set to `true`, the variable will be passed only to pipelines running on protected branches and tags. Defaults to `false`.

Import

GitLab group variables can be imported using an id made up of `groupid:variablename`, e.g.

```
$ terraform import gitlab_group_variable.example 12345:group_variable_key
```

gitlab_label

This resource allows you to create and manage labels for your GitLab projects. For further information on labels, consult the gitlab documentation (https://docs.gitlab.com/ee/user/project/labels.htm).

Example Usage

```
resource "gitlab_label" "fixme" {
  project     = "example"
  name        = "fixme"
  description = "issue with failing tests"
  color       = "#ffcc00"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) The name or id of the project to add the label to.
- `name` - (Required) The name of the label.
- `color` - (Required) The color of the label given in 6-digit hex notation with leading '#' sign (e.g. #FFAABB) or one of the CSS color names (https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#Color_keywords).
- `description` - (Optional) The description of the label.

Attributes Reference

The resource exports the following attributes:

- `id` - The unique id assigned to the label by the GitLab server (the name of the label).

gitlab_pipeline_schedule

This resource allows you to create and manage pipeline schedules. For further information on clusters, consult the gitlab documentation (https://docs.gitlab.com/ce/user/project/pipelines/schedules.html).

Example Usage

```
resource "gitlab_pipeline_schedule" "example" {
   project     = "12345"
   description = "Used to schedule builds"
   ref         = "master"
   cron        = "0 1 * * *"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the schedule to.
- `description` - (Required, string) The description of the pipeline schedule.
- `ref` - (Required, string) The branch/tag name to be triggered.
- `cron` - (Required, string) The cron (e.g. `0 1 * * *`).
- `cron_timezone` - (Optional, string) The timezone.
- `active` - (Optional, bool) The activation of pipeline schedule. If false is set, the pipeline schedule will deactivated initially.

gitlab_pipeline_trigger

This resource allows you to create and manage pipeline triggers

Example Usage

```
resource "gitlab_pipeline_trigger" "example" {
   project   = "12345"
   description = "Used to trigger builds"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the trigger to.
- `description` - (Required, string) The description of the pipeline trigger.

gitlab_project_cluster

This resource allows you to create and manage project clusters for your GitLab projects. For further information on clusters, consult the gitlab documentation (https://docs.gitlab.com/ce/user/project/clusters/index.html).

Example Usage

```
resource "gitlab_project" "foo" {
  name = "foo-project"
}

resource gitlab_project_cluster "bar" {
  project                      = "${gitlab_project.foo.id}"
  name                         = "bar-cluster"
  domain                       = "example.com"
  enabled                      = true
  kubernetes_api_url           = "https://124.124.124"
  kubernetes_token             = "some-token"
  kubernetes_ca_cert           = "some-cert"
  kubernetes_namespace         = "namespace"
  kubernetes_authorization_type = "rbac"
  environment_scope            = "*"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The id of the project to add the cluster to.
- `name` - (Required, string) The name of cluster.
- `domain` - (Optional, string) The base domain of the cluster.
- `enabled` - (Optional, boolean) Determines if cluster is active or not. Defaults to `true`. This attribute cannot be read.
- `managed` - (Optional, boolean) Determines if cluster is managed by gitlab or not. Defaults to `true`. This attribute cannot be read.
- `kubernetes_api_url` - (Required, string) The URL to access the Kubernetes API.
- `kubernetes_token` - (Required, string) The token to authenticate against Kubernetes.
- `kubernetes_ca_cert` - (Optional, string) TLS certificate (needed if API is using a self-signed TLS certificate).
- `kubernetes_namespace` - (Optional, string) The unique namespace related to the project.
- `kubernetes_authorization_type` - (Optional, string) The cluster authorization type. Valid values are `rbac`, `abac`, `unknown_authorization`. Defaults to `rbac`.
- `environment_scope` - (Optional, string) The associated environment to the cluster. Defaults to `*`.

Import

GitLab project clusters can be imported using an id made up of `projectid:clusterid`, e.g.

```
$ terraform import gitlab_project_cluster.bar 123:321
```

gitlab_project_hook

This resource allows you to create and manage hooks for your GitLab projects. For further information on hooks, consult the gitlab documentation (https://docs.gitlab.com/ce/user/project/integrations/webhooks.html).

Example Usage

```
resource "gitlab_project_hook" "example" {
  project               = "example/hooked"
  url                   = "https://example.com/hook/example"
  merge_requests_events = true
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) The name or id of the project to add the hook to.
- `url` - (Required) The url of the hook to invoke.
- `token` - (Optional) A token to present when invoking the hook.
- `enable_ssl_verification` - (Optional) Enable ssl verification when invoking the hook.
- `push_events` - (Optional) Invoke the hook for push events.
- `issues_events` - (Optional) Invoke the hook for issues events.
- `merge_requests_events` - (Optional) Invoke the hook for merge requests.
- `tag_push_events` - (Optional) Invoke the hook for tag push events.
- `note_events` - (Optional) Invoke the hook for notes events.
- `job_events` - (Optional) Invoke the hook for job events.
- `pipeline_events` - (Optional) Invoke the hook for pipeline events.
- `wiki_page_events` - (Optional) Invoke the hook for wiki page events.

Attributes Reference

The resource exports the following attributes:

- `id` - The unique id assigned to the hook by the GitLab server.

gitlab_project

This resource allows you to create and manage projects within your GitLab group or within your user.

Example Usage

```
resource "gitlab_project" "example" {
  name        = "example"
  description = "My awesome codebase"

  visibility_level = "public"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the project.
- `path` - (Optional) The path of the repository.
- `namespace_id` - (Optional) The namespace (group or user) of the project. Defaults to your user. See `gitlab_group` (/docs/providers/gitlab/r/group.html) for an example.
- `description` - (Optional) A description of the project.
- `tags` - (Optional) Tags (topics) of the project.
- `default_branch` - (Optional) The default branch for the project.
- `issues_enabled` - (Optional) Enable issue tracking for the project.
- `merge_requests_enabled` - (Optional) Enable merge requests for the project.
- `approvals_before_merge` - (Optional) Number of merge request approvals required for merging. Default is 0.
- `wiki_enabled` - (Optional) Enable wiki for the project.
- `snippets_enabled` - (Optional) Enable snippets for the project.
- `container_registry_enabled` - (Optional) Enable container registry for the project.
- `visibility_level` - (Optional) Set to `public` to create a public project. Valid values are `private`, `internal`, `public`. Repositories are created as private by default.
- `merge_method` - (Optional) Set to `ff` to create fast-forward merges Valid values are `merge`, `rebase_merge`, `ff` Repositories are created with `merge` by default
- `only_allow_merge_if_pipeline_succeeds` - (Optional) Set to true if you want allow merges only if a pipeline succeeds.
- `only_allow_merge_if_all_discussions_are_resolved` - (Optional) Set to true if you want allow merges only if all

discussions are resolved.

- `shared_runners_enabled` - (Optional) Enable shared runners for this project.
- `shared_with_groups` - (Optional) Enable sharing the project with a list of groups (maps).
    - `group_id` - (Required) Group id of the group you want to share the project with.
    - `group_access_level` - (Required) Group's sharing permissions. See group members permission (https://docs.gitlab.com/ce/user/permissions.html#group-members-permissions) for more info. Valid values are `guest`, `reporter`, `developer`, `master`.
- `archived` - (Optional) Whether the project is in read-only mode (archived). Repositories can be archived/unarchived by toggling this parameter.
- `initialize_with_readme` - (Optional) Create master branch with first commit containing a README.md file.

Attributes Reference

The following additional attributes are exported:

- `id` - Integer that uniquely identifies the project within the gitlab install.
- `ssh_url_to_repo` - URL that can be provided to `git clone` to clone the repository via SSH.
- `http_url_to_repo` - URL that can be provided to `git clone` to clone the repository via HTTP.
- `web_url` - URL that can be used to find the project in a browser.
- `runners_token` - Registration token to use during runner setup.
- `shared_with_groups` - List of the groups the project is shared with.
    - `group_name` - Group's name.

Importing projects

You can import a project state using `terraform import <resource> <id>`. The `id` can be whatever the get single project api (https://docs.gitlab.com/ee/api/projects.html#get-single-project) takes for its `:id` value, so for example:

```
terraform import gitlab_project.example richardc/example
```

gitlab_project_membership

This resource allows you to add a current user to an existing project with a set access level.

Example Usage

```
resource "gitlab_project_membership" "test" {
  project_id   = "12345"
  user_id      = 1337
  access_level = "guest"
}
```

Argument Reference

The following arguments are supported:

- `project_id` - (Required) The id of the project.
- `user_id` - (Required) The id of the user.
- `access_level` - (Required) One of five levels of access to the project.

Import

GitLab group membership can be imported using an id made up of `group_id:user_id`, e.g.

```
$ terraform import gitlab_project_membership.test "12345:1337"
```

gitlab_project_push_rules

This resource allows you to create and manage push rules for your GitLab projects. For further information on push rules, consult the gitlab documentation (https://docs.gitlab.com/ce/push_rules/push_rules.html#push-rules).

Example Usage

```
resource "gitlab_project_push_rules" "example" {
  commit_message_regex = "^(feat|feature|fix|chore|docs|BREAKING_CHANGE):.*"
  prevent_secrets = true
  branch_name_regex = "^PROJ-\d+-.*"
  author_email_regex = "@my-company.com$"
  commit_committer_check = true
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the push rules to.
- `commit_message_regex` - (Optional, string) All commit messages must match this regex, e.g. "Fixed \d+..*"
- `deny_delete_tag` - (Optional, bool) Deny deleting a tag
- `member_check` - (Optional, bool) Restrict commits by author (email) to existing GitLab users
- `prevent_secrets` - (Optional, bool) GitLab will reject any files that are likely to contain secrets
- `branch_name_regex` - (Optional, string) All branch names must match this regex, e.g. "(feature|hotfix)\/*"
- `author_email_regex` - (Optional, string) All commit author emails must match this regex, e.g. "@my-company.com$"
- `file_name_regex` - (Optional, string) All commited filenames must not match this regex, e.g. "(jar|exe)$"
- `max_file_size` - (Optional, int) Maximum file size (MB)
- `commit_committer_check` - (Optional, bool) Users can only push commits to this repository that were committed with one of their own verified emails

Attributes Reference

The resource exports the following attributes:

- `id` - The unique id assigned to the push rules by the GitLab server.

gitlab_project_share_group

This resource allows you to share a project with a group

Example Usage

```
resource "gitlab_project_share_group" "test" {
project_id = "12345"
group_id = 1337
access_level = "guest"
}
```

Argument Reference

The following arguments are supported:

- `project_id` - (Required) The id of the project.
- `group_id` - (Required) The id of the group.
- `access_level` - (Required) One of five levels of access to the project.

Import

GitLab project group shares can be imported using an id made up of `projectid:groupid` , e.g.

```
$ terraform import gitlab_project_share_group.test 12345:1337
```

gitlab_project_variable

This resource allows you to create and manage CI/CD variables for your GitLab projects. For further information on variables, consult the gitlab documentation (https://docs.gitlab.com/ce/ci/variables/README.html#variables).

Example Usage

```
resource "gitlab_project_variable" "example" {
   project   = "12345"
   key       = "project_variable_key"
   value     = "project_variable_value"
   protected = false
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required, string) The name or id of the project to add the hook to.
- `key` - (Required, string) The name of the variable.
- `value` - (Required, string) The value of the variable.
- `variable_type` - (Optional, string) The type of a variable. Available types are: env_var (default) and file.
- `protected` - (Optional, boolean) If set to `true`, the variable will be passed only to pipelines running on protected branches and tags. Defaults to `false`.
- `masked` - (Optional, boolean) If set to `true`, the variable will be masked if it would have been written to the logs. Defaults to `false`.
- `environment_scope` - (Optional, string) The environment_scope of the variable

Import

GitLab project variables can be imported using an id made up of `projectid:variablename`, e.g.

```
$ terraform import gitlab_project_variable.example 12345:project_variable_key
```

gitlab_service_jira

This resource allows you to manage Jira integration.

Example Usage

```
resource "gitlab_project" "awesome_project" {
  name = "awesome_project"
  description = "My awesome project."
  visibility_level = "public"
}

resource "gitlab_service_jira" "jira" {
  project  = "${gitlab_project.awesome_project.id}"
  url      = "https://jira.example.com"
  username = "user"
  password = "mypass"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) ID of the project you want to activate integration on.
- `url` - (Required) The URL to the JIRA project which is being linked to this GitLab project. For example, https://jira.example.com (https://jira.example.com).
- `username` - (Required) The username of the user created to be used with GitLab/JIRA.
- `password` - (Required) The password of the user created to be used with GitLab/JIRA.
- `project_key` - (Required) The short identifier for your JIRA project, all uppercase, e.g., PROJ.
- `jira_issue_transition_id` - (Optional) The ID of a transition that moves issues to a closed state. You can find this number under the JIRA workflow administration (Administration > Issues > Workflows) by selecting View under Operations of the desired workflow of your project. By default, this ID is set to 2.

Importing Jira service

You can import a service_jira state using `terraform import <resource> <project_id>`:

```
$ terraform import gitlab_service_jira.jira 1
```

gitlab_service_slack

This resource allows you to manage Slack notifications integration.

Example Usage

```
resource "gitlab_project" "awesome_project" {
  name = "awesome_project"
  description = "My awesome project."
  visibility_level = "public"
}

resource "gitlab_service_slack" "slack" {
  project                    = "${gitlab_project.awesome_project.id}"
  webhook                    = "https://webhook.com"
  username                   = "myuser"
  push_events                = true
  push_channel               = "push_chan"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) ID of the project you want to activate integration on.
- `webhook` - (Required) Webhook URL (ex.: https://hooks.slack.com/services/.. (https://hooks.slack.com/services/..).)
- `username` - (Optional) Username to use.
- `notify_only_broken_pipelines` - (Optional) Send notifications for broken pipelines.
- `notify_only_default_branch` - (Optional) Send notifications only for the default branch.
- `push_events` - (Optional) Enable notifications for push events.
- `push_channel` - (Optional) The name of the channel to receive push events notifications.
- `issues_events` - (Optional) Enable notifications for issues events.
- `issue_channel` - (Optional) The name of the channel to receive issue events notifications.
- `confidential_issues_events` - (Optional) Enable notifications for confidential issues events.
- `confidential_issue_channel` - (Optional) The name of the channel to receive confidential issue events notifications.
- `merge_requests_events` - (Optional) Enable notifications for merge requests events.
- `merge_request_channel` - (Optional) The name of the channel to receive merge request events notifications.
- `tag_push_events` - (Optional) Enable notifications for tag push events.

- `tag_push_channel` - (Optional) The name of the channel to receive tag push events notifications.
- `note_events` - (Optional) Enable notifications for note events.
- `note_channel` - (Optional) The name of the channel to receive note events notifications.
- `confidential_note_events` - (Optional) Enable notifications for confidential note events.
- `pipeline_events` - (Optional) Enable notifications for pipeline events.
- `pipeline_channel` - (Optional) The name of the channel to receive pipeline events notifications.
- `wiki_page_events` - (Optional) Enable notifications for wiki page events.
- `wiki_page_channel` - (Optional) The name of the channel to receive wiki page events notifications.

Importing Slack service

You can import a service_slack state using `terraform import <resource> <project_id>`:

```
$ terraform import gitlab_service_slack.slack 1
```

gitlab_tag_protection

This resource allows you to protect a specific tag or wildcard by an access level so that the user with less access level cannot Create the tags.

Example Usage

```
resource "gitlab_tag_protection" "TagProtect" {
  project = "12345"
  tag = "TagProtected"
  create_access_level = "developer"
}
```

Argument Reference

The following arguments are supported:

- `project` - (Required) The id of the project.
- `tag` - (Required) Name of the tag or wildcard.
- `create_access_level` - (Required) One of five levels of access to the project.

gitlab_user

This resource allows you to create and manage GitLab users. Note your provider will need to be configured with admin-level access for this resource to work.

Example Usage

```
resource "gitlab_user" "example" {
  name             = "Example Foo"
  username         = "example"
  password         = "superPassword"
  email            = "gitlab@user.create"
  is_admin         = true
  projects_limit   = 4
  can_create_group = false
  is_external      = true
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the user.
- `username` - (Required) The username of the user.
- `password` - (Required) The password of the user.
- `email` - (Required) The e-mail address of the user.
- `is_admin` - (Optional) Boolean, defaults to false. Whether to enable administrative priviledges for the user.
- `projects_limit` - (Optional) Integer, defaults to 0. Number of projects user can create.
- `can_create_group` - (Optional) Boolean, defaults to false. Whether to allow the user to create groups.
- `skip_confirmation` - (Optional) Boolean, defaults to true. Whether to skip confirmation.
- `is_external` - (Optional) Boolean, defaults to false. Whether a user has access only to some internal or private projects. External users can only access projects to which they are explicitly granted access.

Attributes Reference

The resource exports the following attributes:

- `id` - The unique id assigned to the user by the GitLab server.

Importing users

You can import a user to terraform state using `terraform import <resource> <id>`. The `id` must be an integer for the id of the user you want to import, for example:

```
terraform import gitlab_user.example 42
```