» github_actions_public_key

Use this data source to retrieve information about a GitHub Actions public key. This data source is required to be used with other GitHub secrets interactions. Note that the provider `token` must have admin rights to a repository to retrieve it's action public key.

» Example Usage

```
data "github_secrets_public_key" "example" {
  owner      = "example_owner"
  repository = "example_repo"
}
```

» Argument Reference

- `owner` - (Required) Owner of the repository.
- `repository` - (Required) Name of the repository to get public key from.

» Attributes Reference

- `key_id` - ID of the key that has been retrieved.
- `key` - Actual key retrieved.

» github_collaborators

Use this data source to retrieve the collaborators for a given repository.

» Example Usage

```
data "github_collaborators" "test" {
  owner      = "example_owner"
  repository = "example_repository"
}
```

» Arguments Reference

- `owner` - (Required) The organization that owns the repository.
- `repository` - (Required) The name of the repository.

- `affiliation` - (Optional) Filter collaborators returned by their affiliation. Can be one of: `outside`, `direct`, `all`. Defaults to `all`.

» Attributes Reference

- `collaborator` - An Array of GitHub collaborators. Each `collaborator` block consists of the fields documented below.

The `collaborator` block consists of:

- `login` - The collaborator's login.
- `id` - The ID of the collaborator.
- `url` - The GitHub API URL for the collaborator.
- `html_url` - The GitHub HTML URL for the collaborator.
- `followers_url` - The GitHub API URL for the collaborator's followers.
- `following_url` - The GitHub API URL for those following the collaborator.
- `gists_url` - The GitHub API URL for the collaborator's gists.
- `starred_url` - The GitHub API URL for the collaborator's starred repositories.
- `subscriptions_url` - The GitHub API URL for the collaborator's subscribed repositories.
- `organizations_url` - The GitHub API URL for the collaborator's organizations.
- `repos_url` - The GitHub API URL for the collaborator's repositories.
- `events_url` - The GitHub API URL for the collaborator's events.
- `received_events_url` - The GitHub API URL for the collaborator's received events.
- `type` - The type of the collaborator (ex. `user`).
- `site_admin` - Whether the user is a GitHub admin.
- `permission` - The permission of the collaborator.

» github_release

Use this data source to retrieve information about a GitHub release in a specific repository.

» Example Usage

To retrieve the latest release that is present in a repository:

```
data "github_release" "example" {
    repository  = "example-repository"
    owner       = "example-owner"
    retrieve_by = "latest"
}
```

To retrieve a specific release from a repository based on it's ID:

```
data "github_release" "example" {
    repository  = "example-repository"
    owner       = "example-owner"
    retrieve_by = "id"
    id          = 12345
}
```

Finally, to retrieve a release based on it's tag:

```
data "github_release" "example" {
    repository  = "example-repository"
    owner       = "example-owner"
    retrieve_by = "tag"
    release_tag = "v1.0.0"
}
```

» Argument Reference

- `repository` - (Required) Name of the repository to retrieve the release from.
- `owner` - (Required) Owner of the repository.
- `retrieve_by` - (Required) Describes how to fetch the release. Valid values are `id`, `tag`, `latest`.
- `release_id` - (Optional) ID of the release to retrieve. Must be specified when `retrieve_by` = `id`.
- `release_tag` - (Optional) Tag of the release to retrieve. Must be specified when `retrieve_by` = `tag`.

» Attributes Reference

- `release_tag` - Tag of release
- `release_id` - ID of release
- `target_commitish` - Commitish value that determines where the Git release is created from
- `name` - Name of release
- `body` - Contents of the description (body) of a release
- `draft` - (`Boolean`) indicates whether the release is a draft
- `prerelease` - (`Boolean`) indicates whether the release is a prerelease
- `created_at` - Date of release creation
- `published_at` - Date of release publishing
- `url` - Base URL of the release
- `html_url` - URL directing to detailed information on the release
- `asserts_url` - URL of any associated assets with the release
- `upload_url` - URL that can be used to upload Assets to the release
- `zipball_url` - Download URL of a specific release in `zip` format
- `tarball_url` - Download URL of a specific release in `tar.gz` format

» github_repositories

Note: The data source will return a maximum of `1000` repositories as documented in official API docs.

Use this data source to retrieve a list of GitHub repositories using a search query.

» Example Usage

```
data "github_repositories" "example" {
  query = "org:hashicorp language:Go"
}
```

» Argument Reference

The following arguments are supported:

- `query` - (Required) Search query. See documentation for the search syntax.

- `sort` - (Optional) Sorts the repositories returned by the specified attribute. Valid values include `stars`, `fork`, and `updated`. Defaults to `updated`.

» Attributes Reference

- `full_names` - A list of full names of found repositories (e.g. `hashicorp/terraform`)
- `names` - A list of found repository names (e.g. `terraform`)

» github_repository

Use this data source to retrieve information about a GitHub repository.

» Example Usage

```
data "github_repository" "example" {
  full_name = "hashicorp/terraform"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Optional) The name of the repository.
- `full_name` - (Optional) Full name of the repository (in `org/name` format).

» Attributes Reference

- `description` - A description of the repository.
- `homepage_url` - URL of a page describing the project.
- `private` - Whether the repository is private.
- `has_issues` - Whether the repository has GitHub Issues enabled.
- `has_projects` - Whether the repository has the GitHub Projects enabled.
- `has_wiki` - Whether the repository has the GitHub Wiki enabled.
- `allow_merge_commit` - Whether the repository allows merge commits.
- `allow_squash_merge` - Whether the repository allows squash merges.
- `allow_rebase_merge` - Whether the repository allows rebase merges.
- `has_downloads` - Whether the repository has Downloads feature enabled.
- `default_branch` - The name of the default branch of the repository.
- `archived` - Whether the repository is archived.

- `topics` - The list of topics of the repository.
- `html_url` - URL to the repository on the web.
- `ssh_clone_url` - URL that can be provided to `git clone` to clone the repository via SSH.
- `http_clone_url` - URL that can be provided to `git clone` to clone the repository via HTTPS.
- `git_clone_url` - URL that can be provided to `git clone` to clone the repository anonymously via the git protocol.
- `svn_url` - URL that can be provided to `svn checkout` to check out the repository via GitHub's Subversion protocol emulation.

» github_team

Use this data source to retrieve information about a GitHub team.

» Example Usage

```
data "github_team" "example" {
  slug = "example"
}
```

» Argument Reference

- `slug` - (Required) The team slug.

» Attributes Reference

- `id` - the ID of the team.
- `name` - the team's full name.
- `description` - the team's description.
- `privacy` - the team's privacy type.
- `permission` - the team's permission level.
- `members` - List of team members

» github_user

Use this data source to retrieve information about a GitHub user.

» Example Usage

```
data "github_user" "example" {
  username = "example"
}
```

» Argument Reference

- `username` - (Required) The username.

» Attributes Reference

- `login` - the user's login.
- `avatar_url` - the user's avatar URL.
- `gravatar_id` - the user's gravatar ID.
- `site_admin` - whether the user is a GitHub admin.
- `name` - the user's full name.
- `company` - the user's company name.
- `blog` - the user's blog location.
- `location` - the user's location.
- `email` - the user's email.
- `gpg_keys` - list of user's GPG keys.
- `ssh_keys` - list of user's SSH keys.
- `bio` - the user's bio.
- `public_repos` - the number of public repositories.
- `public_gists` - the number of public gists.
- `followers` - the number of followers.
- `following` - the number of following users.
- `created_at` - the creation date.
- `updated_at` - the update date.

» github_actions_secret

This resource allows you to create and manage GitHub Actions secrets within your GitHub repositories. You must have write access to a repository to use this resource.

Secret values are encrypted using the Go '/crypto/box' module which is interoperable with libsodium. Libsodium is used by Github to decrypt secret values.

For the purposes of security, the contents of the `plaintext_value` field have been marked as `sensitive` to Terraform, but it is important to note that **this does not hide it from state files**. You should treat state as sensitive always.

It is also advised that you do not store plaintext values in your code but rather populate the plaintext_value using fields from a resource, data source or variable as, while encrypted in state, these will be easily accessible in your code. See below for an example of this abstraction.

» Example Usage

```
data "github_actions_public_key" "example_public_key" {
  owner      = "example_owner"
  repository = "example_repository"
}

resource "github_actions_secret" "example_secret" {
  repository       = "example_repository"
  secret_name      = "example_secret_name"
  plaintext_value  = var.some_secret_string
  key_id           = github_actions_public_key.example_public_key.key_id
  public_key       = github_actions_public_key.example_public_key.key
}
```

» Argument Reference

The following arguments are supported:

- repository - (Required) Name of the repository
- secret_name - (Required) Name of the secret
- plaintext_value - (Required) Plaintext value of the secret to be encrypted
- key_id - (Required) ID if the key used for encryption
- public_key - (Required) Public key of the repository to be used in encryption of the plaintext_value

» github_branch_protection

Protects a GitHub branch.

This resource allows you to configure branch protection for repositories in your organization. When applied, the branch will be protected from forced pushes and deletion. Additional constraints, such as required status checks or restrictions on users, teams, and apps, can also be configured.

» Example Usage

```
# Protect the master branch of the foo repository. Additionally, require that
# the "ci/travis" context to be passing and only allow the engineers team merge
# to the branch.
resource "github_branch_protection" "example" {
  repository     = "${github_repository.example.name}"
  branch         = "master"
  enforce_admins = true

  required_status_checks {
    strict   = false
    contexts = ["ci/travis"]
  }

  required_pull_request_reviews {
    dismiss_stale_reviews = true
    dismissal_users       = ["foo-user"]
    dismissal_teams       = ["${github_team.example.slug}", "${github_team.second.slug}"]
  }

  restrictions {
    users = ["foo-user"]
    teams = ["${github_team.example.slug}"]
    apps  = ["foo-app"]
  }
}

resource "github_team" "example" {
  name = "Example Name"
}

resource "github_team_repository" "example" {
  team_id    = "${github_team.example.id}"
  repository = "${github_repository.example.name}"
  permission = "pull"
}
```

» Argument Reference

The following arguments are supported:

- `repository` - (Required) The GitHub repository name.
- `branch` - (Required) The Git branch to protect.

- `enforce_admins` - (Optional) Boolean, setting this to `true` enforces status checks for repository administrators.
- `require_signed_commits` - (Optional) Boolean, setting this to `true` requires all commits to be signed with GPG.
- `required_status_checks` - (Optional) Enforce restrictions for required status checks. See Required Status Checks below for details.
- `required_pull_request_reviews` - (Optional) Enforce restrictions for pull request reviews. See Required Pull Request Reviews below for details.
- `restrictions` - (Optional) Enforce restrictions for the users and teams that may push to the branch. See Restrictions below for details.

» Required Status Checks

`required_status_checks` supports the following arguments:

- `strict`: (Optional) Require branches to be up to date before merging. Defaults to `false`.
- `contexts`: (Optional) The list of status checks to require in order to merge into this branch. No status checks are required by default.

» Required Pull Request Reviews

`required_pull_request_reviews` supports the following arguments:

- `dismiss_stale_reviews`: (Optional) Dismiss approved reviews automatically when a new commit is pushed. Defaults to `false`.
- `dismissal_users`: (Optional) The list of user logins with dismissal access
- `dismissal_teams`: (Optional) The list of team slugs with dismissal access. Always use `slug` of the team, **not** its name. Each team already **has** to have access to the repository.
- `require_code_owner_reviews`: (Optional) Require an approved review in pull requests including files with a designated code owner. Defaults to `false`.
- `required_approving_review_count`: (Optional) Require x number of approvals to satisfy branch protection requirements. If this is specified it must be a number between 1-6. This requirement matches Github's API, see the upstream documentation for more information.

» Restrictions

`restrictions` supports the following arguments:

- `users`: (Optional) The list of user logins with push access.
- `teams`: (Optional) The list of team slugs with push access.

- `apps`: (Optional) The list of app slugs with push access. Always use `slug` of the team, **not** its name. Each team already **has** to have access to the repository.

`restrictions` is only available for organization-owned repositories.

» Import

GitHub Branch Protection can be imported using an ID made up of `repository:branch`, e.g.

```
$ terraform import github_branch_protection.terraform terraform:master
```

» github_issue_label

Provides a GitHub issue label resource.

This resource allows you to create and manage issue labels within your GitHub organization.

Issue labels are keyed off of their "name", so pre-existing issue labels result in a 422 HTTP error if they exist outside of Terraform. Normally this would not be an issue, except new repositories are created with a "default" set of labels, and those labels easily conflict with custom ones.

This resource will first check if the label exists, and then issue an update, otherwise it will create.

» Example Usage

```
# Create a new, red colored label
resource "github_issue_label" "test_repo" {
  repository = "test-repo"
  name       = "Urgent"
  color      = "FF0000"
}
```

» Argument Reference

The following arguments are supported:

- `repository` - (Required) The GitHub repository
- `name` - (Required) The name of the label.

- `color` - (Required) A 6 character hex code, **without the leading #**, identifying the color of the label.
- `description` - (Optional) A short description of the label.
- `url` - (Computed) The URL to the issue label

» Import

GitHub Issue Labels can be imported using an ID made up of `repository:name`, e.g.

```
$ terraform import github_issue_label.panic_label terraform:panic
```

» github_membership

Provides a GitHub membership resource.

This resource allows you to add/remove users from your organization. When applied, an invitation will be sent to the user to become part of the organization. When destroyed, either the invitation will be cancelled or the user will be removed.

» Example Usage

```
# Add a user to the organization
resource "github_membership" "membership_for_some_user" {
  username = "SomeUser"
  role     = "member"
}
```

» Argument Reference

The following arguments are supported:

- `username` - (Required) The user to add to the organization.
- `role` - (Optional) The role of the user within the organization. Must be one of `member` or `admin`. Defaults to `member`.

» Import

GitHub Membership can be imported using an ID made up of `organization:username`, e.g.

```
$ terraform import github_membership.member hashicorp:someuser
```

» github_organization_block

This resource allows you to create and manage blocks for GitHub organizations.

» Example Usage

```
resource "github_organization_block" "example" {
  username = "paultyng"
}
```

» Argument Reference

The following arguments are supported:

- `username` - (Required) The name of the user to block.

» github_organization_project

This resource allows you to create and manage projects for GitHub organization.

» Example Usage

```
resource "github_organization_project" "project" {
  name = "A Organization Project"
  body = "This is a organization project."
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the project.
- `body` - (Optional) The body of the project.

» Attributes Reference

The following additional attributes are exported:

- `url` - URL of the project

» github_organization_webhook

This resource allows you to create and manage webhooks for GitHub organization.

» Example Usage

```
resource "github_organization_webhook" "foo" {
  name = "web"

  configuration {
    url          = "https://google.de/"
    content_type = "form"
    insecure_ssl = false
  }

  active = false

  events = ["issues"]
}
```

» Argument Reference

The following arguments are supported:

- `events` - (Required) A list of events which should trigger the webhook. See a list of available events
- `configuration` - (Required) key/value pair of configuration for this webhook. Available keys are `url`, `content_type`, `secret` and `insecure_ssl`.
- `active` - (Optional) Indicate of the webhook should receive events. Defaults to `true`.
- `name` - (Optional) The type of the webhook. `web` is the default and the only option.

» Attributes Reference

The following additional attributes are exported:

- `url` - URL of the webhook

» github_project_column

This resource allows you to create and manage columns for GitHub projects.

» Example Usage

```
resource "github_organization_project" "project" {
  name = "A Organization Project"
  body = "This is an organization project."
}

resource "github_project_column" "column" {
  project_id = "${github_organization_project.project.id}"
  name       = "a column"
}
```

» Argument Reference

The following arguments are supported:

- `project_id` - (Required) The ID of an existing project that the column will be created in.
- `name` - (Required) The name of the column.

» github_repository

This resource allows you to create and manage repositories within your GitHub organization.

This resource cannot currently be used to manage *personal* repositories, outside of organizations.

» Example Usage

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

  private = true

  template {
    owner = "github"
    repository = "terraform-module-template"
  }
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the repository.
- `description` - (Optional) A description of the repository.
- `homepage_url` - (Optional) URL of a page describing the project.
- `private` - (Optional) Set to `true` to create a private repository. Repositories are created as public (e.g. open source) by default.
- `has_issues` - (Optional) Set to `true` to enable the GitHub Issues features on the repository.
- `has_projects` - (Optional) Set to `true` to enable the GitHub Projects features on the repository. Per the GitHub documentation when in an organization that has disabled repository projects it will default to `false` and will otherwise default to `true`. If you specify `true` when it has been disabled it will return an error.
- `has_wiki` - (Optional) Set to `true` to enable the GitHub Wiki features on the repository.
- `allow_merge_commit` - (Optional) Set to `false` to disable merge commits on the repository.
- `allow_squash_merge` - (Optional) Set to `false` to disable squash merges on the repository.
- `allow_rebase_merge` - (Optional) Set to `false` to disable rebase merges on the repository.
- `has_downloads` - (Optional) Set to `true` to enable the (deprecated) downloads features on the repository.

- `auto_init` - (Optional) Set to `true` to produce an initial commit in the repository.
- `gitignore_template` - (Optional) Use the name of the template without the extension. For example, ”Haskell”.
- `license_template` - (Optional) Use the name of the template without the extension. For example, ”mit” or ”mpl-2.0”.
- `default_branch` - (Optional) The name of the default branch of the repository. **NOTE:** This can only be set after a repository has already been created, and after a correct reference has been created for the target branch inside the repository. This means a user will have to omit this parameter from the initial repository creation and create the target branch inside of the repository prior to setting this attribute.
- `archived` - (Optional) Specifies if the repository should be archived. Defaults to `false`. **NOTE** Currently, the API does not support unarchiving.
- `topics` - (Optional) The list of topics of the repository.
- `template` - (Optional) Use a template repository to create this resource. See Template Repositories below for details.

» Template Repositories

`template` supports the following arguments:

- `owner`: The GitHub organization or user the template repository is owned by.
- `repository`: The name of the template repository.

» Attributes Reference

The following additional attributes are exported:

- `full_name` - A string of the form ”orgname/reponame”.
- `html_url` - URL to the repository on the web.
- `ssh_clone_url` - URL that can be provided to `git clone` to clone the repository via SSH.
- `http_clone_url` - URL that can be provided to `git clone` to clone the repository via HTTPS.
- `git_clone_url` - URL that can be provided to `git clone` to clone the repository anonymously via the git protocol.
- `svn_url` - URL that can be provided to `svn checkout` to check out the repository via GitHub’s Subversion protocol emulation.

» Import

Repositories can be imported using the `name`, e.g.

```
$ terraform import github_repository.terraform terraform
```

» github_repository_collaborator

Provides a GitHub repository collaborator resource.

This resource allows you to add/remove collaborators from repositories in your organization. Collaborators can have explicit (and differing levels of) read, write, or administrator access to specific repositories in your organization, without giving the user full organization membership.

When applied, an invitation will be sent to the user to become a collaborator on a repository. When destroyed, either the invitation will be cancelled or the collaborator will be removed from the repository.

Further documentation on GitHub collaborators:

- Adding outside collaborators to repositories in your organization
- Converting an organization member to an outside collaborator

» Example Usage

```
# Add a collaborator to a repository
resource "github_repository_collaborator" "a_repo_collaborator" {
  repository = "our-cool-repo"
  username   = "SomeUser"
  permission = "admin"
}
```

» Argument Reference

The following arguments are supported:

- `repository` - (Required) The GitHub repository
- `username` - (Required) The user to add to the repository as a collaborator.
- `permission` - (Optional) The permission of the outside collaborator for the repository. Must be one of `pull`, `push`, or `admin`. Defaults to `push`.

» Attribute Reference

In addition to the above arguments, the following attributes are exported:

- `invitation_id` - ID of the invitation to be used in `github_user_invitation_accepter`

» Import

GitHub Repository Collaborators can be imported using an ID made up of `repository:username`, e.g.

```
$ terraform import github_repository_collaborator.collaborator terraform:someuser
```

» github_repository_deploy_key

Provides a GitHub repository deploy key resource.

A deploy key is an SSH key that is stored on your server and grants access to a single GitHub repository. This key is attached directly to the repository instead of to a personal user account.

This resource allows you to add/remove repository deploy keys.

Further documentation on GitHub repository deploy keys: - About deploy keys

» Example Usage

```
# Add a deploy key
resource "github_repository_deploy_key" "example_repository_deploy_key" {
  title      = "Repository test key"
  repository = "test-repo"
  key        = "ssh-rsa AAA..."
  read_only  = "false"
}
```

» Argument Reference

The following arguments are supported:

- `key` - (Required) A SSH key.
- `read_only` - (Required) A boolean qualifying the key to be either read only or read/write.
- `repository` - (Required) Name of the GitHub repository.
- `title` - (Required) A title.

Changing any of the fields forces re-creating the resource.

» Import

Repository deploy keys can be imported using a colon-separated pair of repository name and GitHub's key id. The latter can be obtained by GitHub's SDKs and API.

```
$ terraform import github_repository_deploy_key.foo test-repo:23824728
```

» github_repository_file

This resource allows you to create and manage files within a GitHub repository.

» Example Usage

```
resource "github_repository_file" "gitignore" {
  repository = "example"
  file       = ".gitignore"
  content    = "**/*.tfstate"
}
```

» Argument Reference

The following arguments are supported:

- `repo` - (Required) The repository to create the file in.
- `file` - (Required) The path of the file to manage.
- `content` - (Required) The file content.
- `branch` - (Optional) Git branch (defaults to `master`). The branch must already exist, it will not be created if it does not already exist.
- `commit_author` - (Optional) Committer author name to use.
- `commit_email` - (Optional) Committer email address to use.
- `commit_message` - (Optional) Commit message when adding or updating the managed file.

» Attributes Reference

The following additional attributes are exported:

- `sha` - The SHA blob of the file.

» Import

Repository files can be imported using a combination of the `repo` and `file`, e.g.

```
$ terraform import github_repository.gitignore example/.gitignore
```

To import a file from a branch other than master, append `:` and the branch name, e.g.

```
$ terraform import github_repository.gitignore example/.gitignore:dev
```

» github_repository_project

This resource allows you to create and manage projects for GitHub repository.

» Example Usage

```
resource "github_repository" "example" {
  name         = "example"
  description  = "My awesome codebase"
  has_projects = true
}

resource "github_repository_project" "project" {
  name       = "A Repository Project"
  repository = "${github_repository.example.name}"
  body       = "This is a repository project."
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the project.
- `repository` - (Required) The repository of the project.
- `body` - (Optional) The body of the project.

» Attributes Reference

The following additional attributes are exported:

- `url` - URL of the project

» github_repository_webhook

This resource allows you to create and manage webhooks for repositories within your GitHub organization.

This resource cannot currently be used to manage webhooks for *personal* repositories, outside of organizations.

» Example Usage

```
resource "github_repository" "repo" {
  name         = "foo"
  description  = "Terraform acceptance tests"
  homepage_url = "http://example.com/"

  private = false
}

resource "github_repository_webhook" "foo" {
  repository = "${github_repository.repo.name}"

  name = "web"

  configuration {
    url          = "https://google.de/"
    content_type = "form"
    insecure_ssl = false
  }

  active = false

  events = ["issues"]
}
```

» Argument Reference

The following arguments are supported:

- `repository` - (Required) The repository of the webhook.
- `events` - (Required) A list of events which should trigger the webhook. See a list of available events.
- `configuration` - (Required) key/value pair of configuration for this webhook. Available keys are `url`, `content_type`, `secret` and `insecure_ssl`. `secret` is the shared secret, see API documentation.
- `active` - (Optional) Indicate of the webhook should receive events. Defaults to `true`.
- `name` - (Optional) The type of the webhook. `web` is the default and the only option.

» Attributes Reference

The following additional attributes are exported:

- `url` - URL of the webhook

» Import

Repository webhooks can be imported using the `name` of the repository, combined with the `id` of the webhook, separated by a `/` character. The `id` of the webhook can be found in the URL of the webhook. For example: `"https://github.com/foo-org/foo-repo/settings/hooks/14711452"`.

Importing uses the name of the repository, as well as the ID of the webhook, e.g.

```
$ terraform import github_repository_webhook.terraform terraform/11235813
```

If secret is populated in the webhook's configuration, the value will be imported as "********".

» github_team

Provides a GitHub team resource.

This resource allows you to add/remove teams from your organization. When applied, a new team will be created. When destroyed, that team will be removed.

» Example Usage

```
# Add a team to the organization
resource "github_team" "some_team" {
  name        = "some-team"
  description = "Some cool team"
  privacy     = "closed"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the team.
- `description` - (Optional) A description of the team.
- `privacy` - (Optional) The level of privacy for the team. Must be one of `secret` or `closed`. Defaults to `secret`.
- `parent_team_id` - (Optional) The ID of the parent team, if this is a nested team.
- `ldap_dn` - (Optional) The LDAP Distinguished Name of the group where membership will be synchronized. Only available in GitHub Enterprise.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the created team.
- `slug` - The slug of the created team, which may or may not differ from `name`, depending on whether `name` contains "URL-unsafe" characters. Useful when referencing the team in `github_branch_protection`.

» Import

GitHub Teams can be imported using the GitHub team ID e.g.

```
$ terraform import github_team.core 1234567
```

» github_team_membership

Provides a GitHub team membership resource.

This resource allows you to add/remove users from teams in your organization. When applied, the user will be added to the team. If the user hasn't accepted their invitation to the organization, they won't be part of the team until they do. When destroyed, the user will be removed from the team.

» Example Usage

```
# Add a user to the organization
resource "github_membership" "membership_for_some_user" {
  username = "SomeUser"
  role     = "member"
}

resource "github_team" "some_team" {
  name        = "SomeTeam"
  description = "Some cool team"
}

resource "github_team_membership" "some_team_membership" {
  team_id  = "${github_team.some_team.id}"
  username = "SomeUser"
  role     = "member"
}
```

» Argument Reference

The following arguments are supported:

- `team_id` - (Required) The GitHub team id
- `username` - (Required) The user to add to the team.
- `role` - (Optional) The role of the user within the team. Must be one of `member` or `maintainer`. Defaults to `member`.

» Import

GitHub Team Membership can be imported using an ID made up of `teamid:username`, e.g.

```
$ terraform import github_team_membership.member 1234567:someuser
```

» github_team_repository

This resource manages relationships between teams and repositories in your GitHub organization.

Creating this resource grants a particular team permissions on a particular repository.

The repository and the team must both belong to the same organization on GitHub. This resource does not actually *create* any repositories; to do that, see `github_repository`.

» Example Usage

```
# Add a repository to the team
resource "github_team" "some_team" {
  name        = "SomeTeam"
  description = "Some cool team"
}

resource "github_repository" "some_repo" {
  name = "some-repo"
}

resource "github_team_repository" "some_team_repo" {
  team_id    = "${github_team.some_team.id}"
  repository = "${github_repository.some_repo.name}"
  permission = "pull"
}
```

» Argument Reference

The following arguments are supported:

- `team_id` - (Required) The GitHub team id
- `repository` - (Required) The repository to add to the team.
- `permission` - (Optional) The permissions of team members regarding the repository. Must be one of `pull`, `triage`, `push`, `maintain`, or `admin`. Defaults to `pull`.

» Import

GitHub Team Repository can be imported using an ID made up of `teamid:repository`, e.g.

```
$ terraform import github_team_repository.terraform_repo 1234567:terraform
```

» github_user_gpg_key

Provides a GitHub user's GPG key resource.

This resource allows you to add/remove GPG keys from your user account.

» Example Usage

```
resource "github_user_gpg_key" "example" {
  armored_public_key = "-----BEGIN PGP PUBLIC KEY BLOCK-----\n...\n-----END PGP PUBLIC KEY B
}
```

» Argument Reference

The following arguments are supported:

- `armored_public_key` - (Required) Your public GPG key, generated in ASCII-armored format. See Generating a new GPG key for help on creating a GPG key.

» Attributes Reference

The following attributes are exported:

- `id` - The GitHub ID of the GPG key, e.g. `401586`
- `key_id` - The key ID of the GPG key, e.g. `3262EFF25BA0D270`

» Import

GPG keys are not importable due to the fact that API does not return previously uploaded GPG key.

» github_user_invitation_accepter

Provides a resource to manage GitHub repository collaborator invitations.

» Example Usage

```
resource "github_repository" "example" {
  name = "example-repo"
}
```

```
resource "github_repository_collaborator" "example" {
  repository = "${github_repository.example.name}"
  username   = "example-username"
  permission = "push"
}

provider "github" {
  alias = "invitee"
  token = "${var.invitee_token}"
}

resource "github_user_invitation_accepter" "example" {
  provider      = "github.invitee"
  invitation_id = "${github_repository_collaborator.example.invitation_id}"
}
```

» Argument Reference

The following arguments are supported:

- `invitation_id` - (Required) ID of the invitation to accept

» github_user_ssh_key

Provides a GitHub user's SSH key resource.

This resource allows you to add/remove SSH keys from your user account.

» Example Usage

```
resource "github_user_ssh_key" "example" {
  title = "example title"
  key   = "${file("~/.ssh/id_rsa.pub")}"
}
```

» Argument Reference

The following arguments are supported:

- `title` - (Required) A descriptive name for the new key. e.g. `Personal MacBook Air`
- `key` - (Required) The public SSH key to add to your GitHub account.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the SSH key
- `url` - The URL of the SSH key

» Import

SSH keys can be imported using their ID e.g.

```
$ terraform import github_user_ssh_key.example 1234567
```