» github_ip_ranges

Use this data source to retrieve information about a Github's IP addresses.

» Example Usage

```
data "github_ip_ranges" "test" {}
```

» Attributes Reference

- `hooks` - An Array of IP addresses in CIDR format specifying the addresses that incoming service hooks will originate from.
- `git` - An Array of IP addresses in CIDR format specifying the Git servers.
- `pages` - An Array of IP addresses in CIDR format specifying the A records for GitHub Pages.

» 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_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_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 and teams, 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" "foo_master" {
  repository = "foo"
  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 = ["admins", "engineers"]
  }

  restrictions {
    users = ["foo-user"]
    teams = ["engineers"]
  }
}
```

» 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.
- `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
- `require_code_owner_reviews`: (Optional) Require an approved review in pull requests including files with a designated code owner. Defaults to `false`.

» 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.

`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_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_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:

- `name` - (Required) The type of the webhook. See a list of available hooks.
- `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`.

» Attributes Reference

The following additional attributes are exported:

- `url` - URL of the webhook

» 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
}
```

» 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_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) Meaningful only during create; set to `true` to produce an initial commit in the repository.
- `gitignore_template` - (Optional) Meaningful only during create, will be ignored after repository creation. Use the name of the template without the extension. For example, "Haskell".
- `license_template` - (Optional) Meaningful only during create, will be ignored after repository creation. Use the name of the template without the extension. For example, "mit" or "mozilla".
- `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.

» 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`.

» 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_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:

- `name` - (Required) The type of the webhook. See a list of available hooks.
- `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`.
- `active` - (Optional) Indicate of the webhook should receive events. Defaults to `true`.

» 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
```

» 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.

» 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`, `push`, or `admin`. Defaults to `pull`.

» Import

Github Team Membership can be imported using an id made up of `teamid:repository`, e.g.

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

» 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.
- `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
```