» launchdarkly_team_member

Provides a LaunchDarkly team member data source.

This resource allows you to retrieve team member information from your LaunchDarkly organization.

» Example Usage

```
data "launchdarkly_team_member" "example" {
  email = "example@example.com"
}
```

» Argument Reference

- `email` - (Required) The unique email address associated with the team member.

» Attributes Reference

In addition to the arguments above, the resource exports the following attributes:

- `id` - The 24 character alphanumeric ID of the team member.
- `first_name` - The team member's given name.
- `last_name` - The team member's family name.
- `role` - The role associated with team member. Possible roles are `owner`, `reader`, `writer`, or `admin`.
- `custom_role` - (Optional) The list of custom roles keys associated with the team member. Custom roles are only available to customers on enterprise plans. To learn more about enterprise plans, contact sales@launchdarkly.com.

» launchdarkly_project

Provides a LaunchDarkly project resource.

This resource allows you to create and manage projects within your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_project" "example" {
  key  = "example-project"
  name = "Example project"

  tags = [
    "terraform",
  ]
}
```

» Argument Reference

- `key` - (Required) The project's unique key.
- `name` - (Required) The project's name.
- `tags` - (Optional) The project's set of tags.

» Import

LaunchDarkly projects can be imported using the project's key, e.g.

```
$ terraform import launchdarkly_project.example example-project
```

» launchdarkly_environment

Provides a LaunchDarkly environment resource.

This resource allows you to create and manage environments in your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_environment" "staging" {
  name  = "Staging"
  key   = "staging"
  color = "ff00ff"
  tags  = ["terraform", "staging"]

  project_key = launchdarkly_project.example.key
}
```

» Argument Reference

- `project_key` - (Required) - The environment's project key.
- `name` - (Required) The name of the environment.
- `key` - (Required) The project-unique key for the environment.
- `color` - (Required) The color swatch as an RGB hex value with no leading `#`. For example: `000000`.
- `tags` - (Optional) Set of tags associated with the environment.
- `secure_mode` - (Optional) Set to `true` to ensure a user of the client-side SDK cannot impersonate another user.
- `default_track_events` - (Optional) Set to `true` to enable data export for every flag created in this environment after you configure this argument. To learn more, read Data Export.
- `default_ttl` - (Optional) The TTL for the environment. This must be between 0 and 60 minutes. The TTL setting only applies to environments using the PHP SDK. To learn more, read TTL settings.
- `require_comments` - (Optional) Set to `true` if this environment requires comments for flag and segment changes.
- `confirm_changes` - (Optional) Set to `true` if this environment requires confirmation for flag and segment changes.

» Attribute Reference

In addition to the arguments above, the resource exports the following attributes:

- `id` - The unique environment ID in the format `project_key/environment_key`.
- `api_key` - The environment's SDK key.
- `mobile_key` - The environment's mobile key.
- `client_side_id` - The environment's client-side ID.

» Import

You can import a LaunchDarkly environment using this format: `project_key/environment_key`.

For example:

```
$ terraform import launchdarkly_environment.staging example-project/staging
```

» launchdarkly_feature_flag

Provides a LaunchDarkly feature flag resource.

This resource allows you to create and manage feature flags within your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_feature_flag" "building_materials" {
  project_key = launchdarkly_project.example.key
  key         = "building-materials"
  name        = "Building materials"
  description = "this is a multivariate flag with string variations."

  variation_type = "string"
  variations {
    value       = "straw"
    name        = "Straw"
    description = "Watch out for wind."
  }
  variations {
    value       = "sticks"
    name        = "Sticks"
    description = "Sturdier than straw"
  }
  variations {
    value       = "bricks"
    name        = "Bricks"
    description = "The strongest variation"
  }

  tags = [
    "example",
    "terraform",
    "multivariate",
    "building-materials",
  ]
}
```

» Argument Reference

- `project_key` - (Required) The feature flag's project key.

- `key` - (Required) The unique feature flag key that references the flag in your application code.
- `name` - (Required) The human-readable name of the feature flag.
- `variation_type` - (Required) The feature flag's variation type: `boolean`, `string`, `number` or `json`.
- `variations` - (Required) List of nested blocks describing the variations associated with the feature flag. You must specify at least two variations. To learn more, read Nested Variations Blocks.
- `description` - (Optional) The feature flag's description.
- `tags` - (Optional) Set of feature flag tags.
- `maintainer_id` - (Optional) The feature flag maintainer's 24 character alphanumeric team member ID.
- `temporary` - (Optional) Specifies whether the flag is a temporary flag.
- `include_in_snippet` - (Optional) Specifies whether this flag should be made available to the client-side JavaScript SDK.
- `custom_properties` - (Optional) List of nested blocks describing the feature flag's custom properties. To learn more, read Nested Custom Properties.

» Nested Variations Blocks

Nested `variations` blocks have the following structure:

- `value` - (Required) The variation value. The value's type must correspond to the `variation_type` argument. For example: `variation_type = "boolean"` accepts only `true` or `false`.
- `name` - (Optional) The name of the variation.
- `description` - (Optional) The variation's description.

» Nested Custom Properties

Nested `custom_properties` have the following structure:

- `key` - (Required) The unique custom property key.
- `name` - (Required) The name of the custom property.
- `value` - (Required) The list of custom property value strings.

» Attributes Reference

In addition to the arguments above, the resource exports the following attribute:

- `id` - The unique feature flag ID in the format `project_key/flag_key`.

» Import

You can import a feature flag using the feature flag's ID in the format `project_key/flag_key`.

For example:

```
$ terraform import launchdarkly_feature_flag.building_materials example-project/building-mat
```

» launchdarkly_feature_flag_environment

Provides a LaunchDarkly environment-specific feature flag resource.

This resource allows you to create and manage environment-specific feature flags attributes within your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_feature_flag_environment" "number_env" {
  flag_id = launchdarkly_feature_flag.number.id
  env_key = launchdarkly_environment.staging.key

  targeting_enabled = true

  prerequisites {
    flag_key  = launchdarkly_feature_flag.basic.key
    variation = 0
  }

  user_targets {
    values = ["user0"]
  }
  user_targets {
    values = ["user1", "user2"]
  }
  user_targets {
    values = []
  }
```

```
  rules {
    clauses {
      attribute = "country"
      op        = "startsWith"
      values    = ["aus", "de", "united"]
      negate    = false
    }
    clauses {
      attribute = "segmentMatch"
      op        = "segmentMatch"
      values    = [launchdarkly_segment.example.key]
      negate    = false
    }
    variation = 0
  }

  flag_fallthrough {
    rollout_weights = [60000, 40000, 0]
  }
}
```

» Argument Reference

- `flag_id` - (Required) The feature flag's unique `id` in the format `project_key/flag_key`.
- `env_key` - (Required) The environment key.
- `targeting_enabled` - (Optional) Whether targeting is enabled.
- `track_events` - (Optional) Whether to send event data back to LaunchDarkly.
- `off_variation` - (Optional) The index of the variation to serve if targeting is disabled.
- `prerequisites` - (Optional) List of nested blocks describing prerequisite feature flags rules. To learn more, read Nested Prequisites Blocks.
- `user_targets` - (Optional) List of nested blocks describing the individual user targets for each variation. The order of the `user_targets` blocks determines the index of the variation to serve if a `user_target` is matched. To learn more, read Nested User Target Blocks.
- `rules` - (Optional) List of logical targeting rules. To learn more, read Nested Rules Blocks.

- `flag_fallthrough` - (Optional) Nested block describing the default variation to serve if no `prerequisites`, `user_target`, or `rules` apply. To learn more, read Nested Flag Fallthrough Block.

» Nested Prerequisites Blocks

Nested `prerequisites` blocks have the following structure:

- `flag_key` - (Required) The prerequisite feature flag's `key`.
- `variation` - (Required) The index of the prerequisite feature flag's variation to target.

» Nested User Targets Blocks

Nested `user_targets` blocks have the following structure:

- `values` - (Optional) List of `user` strings to target.

» Nested Flag Fallthrough Block

The nested `flag_fallthrough` block has the following structure:

- `variation` - (Optional) The default integer variation index to serve if no `prerequisites`, `user_target`, or `rules` apply. You must specify either `variation` or `rollout_weights`.
- `rollout_weights` - (Optional) List of integer percentage rollout weights to apply to each variation if no `prerequisites`, `user_target`, or `rules` apply. The sum of the `rollout_weights` must equal 1000000. You must specify either `variation` or `rollout_weights`.
- `bucket_by` - (Optional) Group percentage rollout by a custom attribute. This argument is only valid if `rollout_weights` is also specified.

» Nested Rules Blocks

Nested `rules` blocks have the following structure:

- `clauses` - (Required) List of nested blocks specifying the logical clauses to evaluate. To learn more, read Nested Clauses Blocks.
- `variation` - (Optional) The integer variation index to serve if the rule clauses evaluate to `true`. You must specify either `variation` or `rollout_weights`.

- `rollout_weights` - (Optional) List of integer percentage rollout weights to apply to each variation if the rule clauses evaluates to `true`. The sum of the `rollout_weights` must equal 1000000. You must specify either `variation` or `rollout_weights`.
- `bucket_by` - (Optional) Group percentage rollout by a custom attribute. This argument is only valid if `rollout_weights` is also specified.

» Nested Clauses Blocks

Nested `clauses` blocks have the following structure:

- `attribute` - (Required) The user attribute to operate on.
- `op` - (Required) The operator associated with the rule clause. Available options are `in`, `endsWith`, `startsWith`, `matches`, `contains`, `lessThan`, `lessThanOrEqual`, `greaterThanOrEqual`, `before`, `after`, `segmentMatch`, `semVerEqual`, `semVerLessThan`, and `semVerGreaterThan`.
- `values` - (Required) The list of values associated with the rule clause.
- `negate` - (Required) Whether to negate the rule clause.

Nested `flag_fallthrough` blocks have the following structure:

- `variation` - (Optional) The integer variation index to serve if the rule clauses evaluate to `true`. You must specify either `variation` or `rollout_weights`.
- `rollout_weights` - (Optional) List of integer percentage rollout weights to apply to each variation if the rule clauses evaluates to `true`. The sum of the `rollout_weights` must equal 1000000. You must specify either `variation` or `rollout_weights`.

» Attributes Reference

In addition to the arguments above, the resource exports the following attribute:

- `id` - The unique feature flag environment ID in the format `project_key/env_key/flag_key`.

» Import

LaunchDarkly feature flag environments can be imported using the segment's ID in the form `project_key/env_key/flag_key`, e.g.

```
$ terraform import launchdarkly_segment.example example-project/example-environment/example-
```

» launchdarkly_segment

Provides a LaunchDarkly segment resource.

This resource allows you to create and manage segments within your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_segment" "example" {
  key         = "example-segment-key"
  project_key = launchdarkly_project.example.key
  env_key     = launchdarkly_environment.example.key
  name        = "example segment"
  description = "This segment is managed by Terraform"
  tags        = ["segment-tag-1", "segment-tag-2"]
  included    = ["user1", "user2"]
  excluded    = ["user3", "user4"]

  rules {
    clauses {
      attribute = "country"
      op        = "startsWith"
      values    = ["en", "de", "un"]
      negate    = false
    }
  }
}
```

» Argument Reference

- `key` - (Required) The unique key that references the segment.
- `project_key` - (Required) The segment's project key.
- `env_key` - (Required) The segment's environment key.
- `name` - (Required) The human-friendly name for the segment.
- `description` - (Optional) The description of the segment's purpose.
- `tags` - (Optional) Set of tags for the segment.
- `included` - (Optional) List of users included in the segment.
- `excluded` - (Optional) List of user excluded from the segment.

- `rules` - (Optional) List of nested custom rule blocks to apply to the segment. To learn more, read Nested Rules Blocks.

» Nested Rules Blocks

Nested `rules` blocks have the following structure:

- `weight` - (Optional) The integer weight of the rule (between 1 and 100000).
- `bucket_by` - (Optional) The operator used to group users together. Available options are `in`, `endsWith`, `startsWith`, `matches`, `contains`, `lessThan`, `lessThanOrEqual`, `greaterThanOrEqual`, `before`, `after`, `segmentMatch`, `semVerEqual`, `semVerLessThan`, and `semVerGreaterThan`.
- `clauses` - (Optional) List of nested custom rule clause blocks. To learn more, read Nested Clauses Blocks.

» Nested Clauses Blocks

Nested `clauses` blocks have the following structure:

- `attribute` - (Required) The user attribute to operate on.
- `op` - (Required) The operator associated with the rule clause. Available options are `in`, `endsWith`, `startsWith`, `matches`, `contains`, `lessThan`, `lessThanOrEqual`, `greaterThanOrEqual`, `before`, `after`, `segmentMatch`, `semVerEqual`, `semVerLessThan`, and `semVerGreaterThan`.
- `values` - (Required) The list of values associated with the rule clause.
- `negate` - (Required) Whether to negate the rule clause.

» Attributes Reference

In addition to the arguments above, the resource exports the following attribute:

- `id` - The unique environment ID in the format `project_key/env_key/segment_key`.

» Import

LaunchDarkly segments can be imported using the segment's ID in the form `project_key/env_key/segment_key`, e.g.

```
$ terraform import launchdarkly_segment.example example-project/example-environment/example-
```

» launchdarkly_webhook

Provides a LaunchDarkly webhook resource.

This resource allows you to create and manage webhooks within your LaunchDarkly organization.

» Example Usage

```
resource "launchdarkly_webhook" "example" {
  url     = "http://webhooks.com/webhook"
  name    = "Example Webhook"
  tags    = ["terraform"]
  enabled = true
}
```

» Argument Reference

- `url` - (Required) - The URL of the remote webhook.
- `enabled` - (Required) - Specifies whether the webhook is enabled.
- `name` - (Optional) - The webhook's human-readable name.
- `secret` - (Optional) - The secret used to sign the webhook.
- `tags` - (Optional) - Set of tags associated with the webhook.

» Attributes Reference

In addition to the arguments above, the resource exports following attribute:

- `id` - The unique webhook ID.

» Import

LaunchDarkly webhooks can be imported using the webhook's 24 character ID, e.g.

```
$ terraform import launchdarkly_webhook.example 57c0af609969090743529967
```

» launchdarkly_custom_role

Provides a LaunchDarkly custom role resource.

This resource allows you to create and manage custom roles within your LaunchDarkly organization.

Note: Custom roles are only available to customers on enterprise plans. To learn more about enterprise plans, contact sales@launchdarkly.com.

» Example Usage

```
resource "launchdarkly_custom_role" "example" {
  key         = "example-role-key-1"
  name        = "example role"
  description = "This is an example role"

  policy {
    effect    = "allow"
    resources = ["proj/*:env/production"]
    actions   = ["*"]
  }
}
```

» Argument Reference

- `key` - (Required) The unique key that references the custom role.
- `name` - (Required) The human-readable name for the custom role.
- `description` - (Optional) The description of the custom role.
- `policy` - (Required) The custom role policy block. To learn more, read Policies in custom roles.

Custom role `policy` blocks are composed of the following arguments:

- `effect` - (Required) - Either `allow` or `deny`. This argument defines whether the statement allows or denies access to the named resources and actions.
- `resources` - (Required) - The list of resource specifiers defining the resources to which the statement applies or does not apply.
- `actions` - (Required) The list of action specifiers defining the actions to which the statement applies.

» Import

You can import LaunchDarkly custom roles by using an existing custom role `key`.

For example:

```
$ terraform import launchdarkly_custom_role.example example-role-key-1
```

» launchdarkly_team_member

Provides a LaunchDarkly team member resource.

This resource allows you to create and manage team members within your LaunchDarkly organization.

Note: You can only manage team members with "admin" level personal access tokens. To learn more, read Managing Teams.

» Example Usage

```
resource "launchdarkly_team_member" "example" {
  email        = "example.user@example.com"
  first_name   = "John"
  last_name    = "Smith"
  role         = "writer"
}
```

» Argument Reference

- `email` - (Required) The unique email address associated with the team member.
- `first_name` - (Optional) The team member's given name.
- `last_name` - (Optional) The team member's family name.
- `role` - (Optional) The role associated with team member. Supported roles are `reader`, `writer`, or `admin`. If you don't specify a role, `reader` is assigned by default.
- `custom_roles` - (Optional) The list of custom roles keys associated with the team member. Custom roles are only available to customers on enterprise plans. To learn more about enterprise plans, contact sales@ launchdarkly.com.

Note: each `launchdarkly_team_member` must have either a `role` or `custom_roles` argument.

» Attributes Reference

In addition to the arguments above, the resource exports the following attribute:

- `id` - The 24 character alphanumeric ID of the team member.

» Import

LaunchDarkly team members can be imported using the team member's 24 character ID, e.g.

```
$ terraform import launchdarkly_team_member.example 5f05565b48be0b441fb63020
```