» Data Source: heroku_addon

Use this data source to get information about a Heroku Addon.

» Example Usage

```
data "heroku_addon" "from_another_app" {
  name = "addon-from-another-app"
}

output "heroku_addon_data_basic" {
  value = [
    "Addon from another app",
    "id: ${data.heroku_addon.from_another_app.id}",
    "name: ${data.heroku_addon.from_another_app.name}",
    "app: ${data.heroku_addon.from_another_app.app}",
    "plan: ${data.heroku_addon.from_another_app.plan}",
    "provider_id: ${data.heroku_addon.from_another_app.provider_id}",
    "config_vars: ${join(", ", data.heroku_addon.from_another_app.config_vars)}",
  ]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The add-on name

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the add-on
- `name` - The add-on name
- `plan` - The plan name
- `provider_id` - The ID of the plan provider
- `config_vars` - The Configuration variables of the add-on

» Data Source: heroku_app

Use this data source to get information about a Heroku App.

» Example Usage

```
# Create a new Heroku app
data "heroku_app" "default" {
  name   = "my-cool-app"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the application. In Heroku, this is also the unique ID, so it must be unique and have a minimum of 3 characters.

» Attributes Reference

The following attributes are exported:

- `name` - (Required) The name of the application. In Heroku, this is also the unique .
- `stack` - (Optional) The application stack is what platform to run the application in.
- `buildpacks` - (Optional) A list of buildpacks that this app uses.
- `space` - (Optional) The private space in which the app runs. Not present if this is a common runtime app.
- `region` - (Required) The region in which the app is deployed.
- `git_url` - (Required) The Git URL for the application. This is used for deploying new versions of the app.
- `web_url` - (Required) The web (HTTP) URL that the application can be accessed at by default.
- `heroku_hostname` - (Required) A hostname for the Heroku application, suitable for pointing DNS records.
- `config_vars` - (Optional) A map of all of the configuration variables for the app.
- `acm` - (Required) True if Heroku ACM is enabled for this app, false otherwise.
- `organization` - (Optional) The Heroku Team that owns this app. The fields for this block are documented below.

The `organization` block supports:

- `name` (string) - The name of the Heroku Team.
- `locked` (boolean)
- `personal` (boolean)

» Data Source: heroku_space

Use this data source to get information about a Heroku Private Space.

» Example Usage

```
# Look up a Heroku Private Space
data "heroku_space" "default" {
  name   = "my-secret-space"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the Heroku Private Space.

» Attributes Reference

The following attributes are exported:

- `name` - The name of the Heroku Private Space. In Heroku, this is also the unique .
- `id` - The unique ID of the Heroku Private Space.
- `region` - The region in which the Heroku Private Space is deployed.
- `state` - The state of the Heroku Private Space. Either `allocating` or `allocated`.
- `shield` - Whether or not the space has Shield turned on. One of `on` or `off`.
- `organization` - The Heroku Team that owns this space. The fields for this block are documented below.
- `cidr` - The RFC-1918 CIDR the Private Space will use. It must be a /16 in 10.0.0.0/8, 172.16.0.0/12 or 192.168.0.0/16
- `data_cidr` - The RFC-1918 CIDR that the Private Space will use for the Heroku-managed peering connection that's automatically created when using Heroku Data add-ons. It must be between a /16 and a /20
- `outbound_ips` - The space's stable outbound NAT IPs.

The `organization` block supports:

- `name` (string) - The name of the Heroku Team.

» Data Source: heroku_space_peering_info

Use this data source to get peering information about a Heroku Private Space.

» Example Usage

```
# Look up a Heroku Private Space's peering info.
data "heroku_space_peering_info" "default" {
  name   = "my-secret-space"
}

# Initiate a VPC peering connection request.
resource "aws_vpc_peering_connection" "foo" {
  peer_owner_id = "${data.heroku_space_peering_info.default.aws_account_id}"
  peer_vpc_id   = "${data.heroku_space_peering_info.default.vpc_id}"
  vpc_id        = "${aws_vpc.foo.id}"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the Heroku Private Space.

» Attributes Reference

The following attributes are exported:

- `aws_account_id` - The AWS account ID that the Heroku Private Space runs in.
- `aws_region` - The AWS region that the Heroku Private Space runs in.
- `vpc_id` - The VPC ID of the Heroku Private Space.
- `vpc_cidr` - The CIDR block of the VPC ID.
- `dyno_cidr_blocks` - The CIDR blocks that the Dynos run on.
- `unavailable_cidr_blocks` - A list of unavailable CIDR blocks.

» Data Source: heroku_team

Use this data source to get information about a Heroku Team or Heroku Enterprise team.

» Example Usage

```
data "heroku_team" "my_heroku_team" {
  name = "name_of_my_heroku_team"
}

output "heroku_team_data_basic" {
  value = [
    "Heroku team",
    "id: ${data.heroku_team.my_heroku_team.id}",
    "default: ${data.heroku_team.my_heroku_team.default}",
    "membership_limit: ${data.heroku_team.my_heroku_team.membership_limit}",
    "provisioned_licenses: ${data.heroku_team.my_heroku_team.provisioned_licenses}",
    "type: ${data.heroku_team.my_heroku_team.type}",
  ]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The team name

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the team
- `default` - Whether to use this team when none is specified
- `credit_card_collections` - Whether charges incurred by the team are paid by credit card
- `membership_limit` - Upper limit of members allowed in a team
- `provisioned_licenses` - Whether the team is provisioned licenses by Salesforce
- `type` - type of team Will likely be either ”enterprise” or ”team”

» heroku_account_feature

This resource is used to create and manage User Features on Heroku.

NOTE: If this resource's HCL is removed from a `.tf` file, the behavior is to disable account feature and remove resource from state.

» Available Features

For a list of available features, use the Heroku CLI to fetch them for the current user:

```
heroku labs
```

The output will contain **User Features** that may be managed with this resource.

» Example Usage

```
# Create a new Heroku app
resource "heroku_account_feature" "example_metrics" {
  name = "metrics-request-volume"
  enabled = true
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) Name of the account feature
- `enabled` - (Required) Enable or disable the account feature

» Attributes Reference

The following attributes are exported:

- `id` - Comprised of acount email & feature name
- `description` - Description of account feature
- `state` - State of account feature

» Import

Existing account features can be imported using a combination of the account email (the email address tied to the Heroku API key) and the feature name.

For example: `$ terraform import heroku_account_feature.example_metrics name@example.com:metrics-request-volume`

» heroku_addon

Provides a Heroku Add-On resource. These can be attach services to a Heroku app.

» Example Usage

```
# Create a new Heroku app
resource "heroku_app" "default" {
  name = "test-app"
}

# Create a database, and configure the app to use it
resource "heroku_addon" "database" {
  app  = "${heroku_app.default.name}"
  plan = "heroku-postgresql:hobby-basic"
}

# Add a web-hook addon for the app
resource "heroku_addon" "webhook" {
  app  = "${heroku_app.default.name}"
  plan = "deployhooks:http"

  config = {
    url = "http://google.com"
  }
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The Heroku app to add to.
- `plan` - (Required) The addon to add.
- `config` - (Optional) Optional plan configuration.
- `name` - (Optional) Globally unique name of the add-on.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the add-on
- `name` - The add-on name
- `plan` - The plan name

- `provider_id` - The ID of the plan provider
- `config_vars` - The Configuration variables of the add-on

» Import

Addons can be imported using the Addon `id`, e.g.

```
$ terraform import heroku_addon.foobar 12345678
```

» heroku_addon_attachment

Attaches a Heroku Addon Resource to an additional Heroku App.

» Example Usage

```
resource "heroku_addon_attachment" "database" {
  app_id  = "${heroku_app.default.id}"
  addon_id = "${heroku_addon.database.id}"
}
```

» Argument Reference

The following arguments are supported:

- `app_id` - (Required) The ID of the Heroku App to attach to.
- `addon_id` - (Required) The ID of the existing Heroku Addon to attach.
- `name` - (Optional) A friendly name for the Heroku Addon Attachment.

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the add-on attachment

» Import

Addons can be imported using the unique Addon Attachment `id`, e.g.

```
$ terraform import heroku_addon_attachment.foobar 01234567-89ab-cdef-0123-456789abcdef
```

» heroku_app

Provides a Heroku App resource. This can be used to create and manage applications on Heroku.

» Example Usage

```
resource "heroku_app" "default" {
  name   = "my-cool-app"
  region = "us"

  config_vars = {
    FOOBAR = "baz"
  }

  buildpacks = [
    "heroku/go"
  ]
}
```

» Example Usage for a Team

A Heroku "team" was originally called an "organization", and that is still the identifier used in this resource.

```
resource "heroku_app" "default" {
  name   = "my-cool-app"
  region = "us"

  organization {
    name = "my-cool-team"
  }
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the application. In Heroku, this is also the unique ID, so it must be unique and have a minimum of 3 characters.
- `region` - (Required) The region that the app should be deployed in.
- `stack` - (Optional) The application stack is what platform to run the application in.

- `buildpacks` - (Optional) Buildpack names or URLs for the application. Buildpacks configured externally won't be altered if this is not present.
- `config_vars`[1] - (Optional) Configuration variables for the application. The config variables in this map are not the final set of configuration variables, but rather variables you want present. That is, other configuration variables set externally won't be removed by Terraform if they aren't present in this list.
- `sensitive_config_vars`[1] - (Optional) This argument is the same as `config_vars`. The main difference between the two is when `sensitive_config_vars` outputs are displayed on-screen following a terraform apply or terraform refresh, they are redacted, with displayed in place of their value. It is recommended to put private keys, passwords, etc in this argument.
- `space` - (Optional) The name of a private space to create the app in.
- `internal_routing` - (Optional) If true, the application will be routable only internally in a private space. This option is only available for apps that also specify `space`.
- `organization` - (Optional) A block that can be specified once to define Heroku Team settings for this app. The fields for this block are documented below.
- `acm` - (Optional) The flag representing Automated Certificate Management for the app.

The `organization` block supports:

- `name` (string) - The name of the Heroku Team
- `locked` (boolean)
- `personal` (boolean)

» Deleting vars

Deleting an entire `config_vars` or `sensitive_config_vars` map from a `heroku_app` configuration will not actually remove the vars on the remote resource. To remove an existing variable, leave these attribute maps in-place and delete only its entries from the map. Once these attributes are empty, the map itself may be deleted from the configuration. Otherwise if one deletes the map with existing entries, the config vars will not be deleted from the remote resource.

This is especially important if you are migrating all `config_vars` to `sensitive_config_vars` or migrating config vars to `heroku_app_config_association` resource.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the app. This is also the name of the application.
- `name` - The name of the application. In Heroku, this is also the unique ID.
- `stack` - The application stack is what platform to run the application in.
- `space` - The private space the app should run in.
- `internal_routing` - Whether internal routing is enabled the private space app.
- `region` - The region that the app should be deployed in.
- `git_url` - The Git URL for the application. This is used for deploying new versions of the app.
- `web_url` - The web (HTTP) URL that the application can be accessed at by default.
- `heroku_hostname` - A hostname for the Heroku application, suitable for pointing DNS records.
- `all_config_vars` - A map of all of the configuration variables that exist for the app, containing both those set by Terraform and those set externally. (These are treated as "sensitive" so that their values are redacted in console output.)
- `uuid` - The unique UUID of the Heroku app. **NOTE:** Use this for `null_resource` triggers.

» Import

Apps can be imported using the App `id`, e.g.

```
$ terraform import heroku_app.foobar MyApp
```

» heroku_app_config_association

Provides a Heroku App Config Association resource, making it possible to set/update/remove heroku app config vars independently from the `heroku_app` resource. An example usage scenario could be:

- User has separate git repositories for various micro-services. Multiple micro-services use Kafka.
- User has a separate repository for kafka terraform files with blue/green support.
- User builds out new clusters.
- Prior to this resource's introduction, user would need one `terraform apply` to update state and X number of `terraform apply` for each micro-service to pick up the new kafka clusters. However with this resource, user can do one `terraform apply` and let Heroku handle the rolling restarts to pick up the new config vars.

» "Sensitive" is not secret

Heroku does not have a 'sensitivity' distinction for its config variables. This distinction is only made during terraform `plan` and `apply` to avoid leaking sensitive data in the console output.

» Beware of conflicting vars

Be careful when having config variables defined in both `heroku_app` and `heroku_app_config_association` resources. As the latter resource has a dependency on the former, any overlapping config variables in `heroku_app` will be overwritten in `heroku_app_config_association` during a `terraform apply`. Furthermore, this overlap will cause an infinite dirty terraform plan if config variables have different values on both resources at the same time. It is recommended to use one or the other resource, not both, to manage your app(s) config vars.

» Example HCL

```
resource "heroku_config" "common" {
    name = "common-vars"

    vars = {
        LOG_LEVEL = "info"
    }

    sensitive_vars = {
        PRIVATE_KEY = "some_private_key"
    }
}

resource "heroku_app" "foobar" {
  name   = "my-cool-app"
  region = "us"
}

resource "heroku_app" "foobar2" {
  name   = "my-cool-app2"
  region = "us"
}

resource "heroku_app_config_association" "foobar" {
  app_id = "${heroku_app.foobar.id}"
```

```
  vars = "${heroku_config.common.vars}"
  sensitive_vars = "${heroku_config.common.sensitive_vars}"
}

resource "heroku_app_config_association" "foobar2" {
  app_id = "${heroku_app.foobar2.id}"

  vars = "${heroku_config.common.vars}"
  sensitive_vars = {
    DATABASE_URL = "some_db_url_that_has_auth_info"
  }
}
```

» Argument Reference

- `app_id` - (Required) A Heroku app's `UUID`. Can also be the name of the Heroku app but `UUID` is preferred as it is idempotent.
- `vars` - Map of config vars that are output in plaintext.
- `sensitive_vars` - This is the same as `vars`. The main difference between the two attributes is `sensitive_vars` outputs are redacted on-screen and replaced by a placeholder, following a terraform plan or apply. It is recommended to put private keys, passwords, etc in this argument.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the app config association.

» Import

The `heroku_app_config_association` resource's primary attributes are managed only within Terraform state. It does not exist as a native Heroku resource. Therefore, it is not possible to import an existing `heroku_app_config_association` resource.

» heroku_app_feature

This resource is used to create and manage App Features on Heroku.

» Available Features

For a list of available features, use the Heroku CLI to fetch them for one of your existing apps:

`heroku labs --app foobar`

The output will contain **User Features** and **App Features**. This resource manages App Features. If you need to manage User Features, use the `heroku_account_feature` resource.

» Example Usage

```
resource "heroku_app_feature" "log_runtime_metrics" {
  app = "test-app"
  name = "log-runtime-metrics"
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The Heroku app to link to.
- `name` - (Required) The name of the App Feature to manage.
- `enabled` - (Optional) Whether to enable or disable the App Feature. The default value is true.

» Import

App features can be imported using the combination of the application name, a colon, and the feature's name.

For example: `$ terraform import heroku_app_feature.log-runtime-metrics foobar:log-runtime-metrics`

» heroku_app_release

Provides a Heroku App Release resource.

An app release represents a combination of code, config vars and add-ons for an app on Heroku.

NOTE: This resource requires the slug be uploaded to Heroku using `heroku_slug` or with external tooling prior to running terraform.

» Example Usage

```
resource "heroku_app" "foobar" {
    name = "foobar"
    region = "us"
}

# Upload your slug

resource "heroku_app_release" "foobar-release" {
    app = "${heroku_app.foobar.name}"
    slug_id = "01234567-89ab-cdef-0123-456789abcdef"
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The name of the application
- `slug_id` - unique identifier of slug
- `description` - description of changes in this release

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the app release

» Import

Existing app releases can be imported using the combination of the application name, a colon, and the formation's type.

For example: `$ terraform import heroku_app_release.foobar-release foobar`

» heroku_build

Provides a Heroku Build resource, to deploy source code to a Heroku app.

Either a URL or local path, pointing to a tarball of the source code, may be deployed. If a local path is used, it may instead point to a directory of source code, which will be tarballed automatically and then deployed.

This resource waits until the build & release completes.

If the build fails, the error will contain a URL to view the build log. `curl "https://the-long-log-url-in-the-error"`.

To start the app from a successful build, use a Formation resource to specify the process, dyno size, and dyno quantity.

» Source code layout

The code contained in the source directory or tarball must follow the layout required by the buildpack or the `Dockerfile` for container builds.

» Building with Buildpacks

This is the default build process.

For apps that do not have a buildpack set, the official Heroku buildpacks will be searched until a match is detected and used to compile the app.

A `Procfile` may be required to successfully launch the app. Some buildpacks provide a default web process, such as `npm start` for Node.js. Other buildpacks may require a `Procfile`, like for a pure Ruby app.

» Building with Docker

To use container builds, set the parent `heroku_app` resource's `stack = "container"`

A `heroku.yml` manifest file is required to declare which `Dockerfile` to build for each process. Be careful not to create conflicting configuration between `heroku.yml` and Terraform, such as addons or config vars.

» Source URLs

A `source.url` may point to any `https://` URL that responds to a `GET` with a tarball source code. When running `terraform apply`, the source code will only be fetched once for a successful build. Change the URL to force a new resource.

Useful for building public, open-source source code, such as projects that publish releases on GitHub.

» GitHub URLs

GitHub provides release tarballs through URLs. Create a release and then use the tag as a `source.url`, such as:

`https://github.com/username/example/archive/v1.0.0.tar.gz`

Using a branch or master `source.url` is possible, but be aware that tracking down exactly what commit was deployed for a given `terraform apply` may be difficult. On the other hand, using stable release tags ensures repeatability of the Terraform configuration.

» Example Usage with Source URL

```
resource "heroku_app" "foobar" {
    name   = "foobar"
    region = "us"
}

resource "heroku_build" "foobar" {
  app        = "${heroku_app.foobar.id}"
  buildpacks = ["https://github.com/mars/create-react-app-buildpack"]

  source = {
    # This app uses a community buildpack, set it in `buildpacks` above.
    url     = "https://github.com/mars/cra-example-app/archive/v2.1.1.tar.gz"
    version = "v2.1.1"
  }
}

resource "heroku_formation" "foobar" {
  app        = "${heroku_app.foobar.id}"
  type       = "web"
  quantity   = 1
  size       = "Standard-1x"
  depends_on = ["heroku_build.foobar"]
}
```

» Local source

A `source.path` may point to either:

- a tarball of source code
- a directory of source code
  - use `src/appname` relative path to reference `src/` sub-directories, monorepo-style

  - use `../appname` relative or `/opt/src/appname` absolute paths to external directories
  - *something else will need to manage the state of that external source code, before Heroku Build*

When running `terraform apply`, if the contents (SHA256) of the source path changed since the last `apply`, then a new build will start.

» Example Usage with Local Source Directory

```
resource "heroku_app" "foobar" {
    name   = "foobar"
    region = "us"
}

resource "heroku_build" "foobar" {
  app = "${heroku_app.foobar.id}"

  source = {
    # A local directory, changing its contents will
    # force a new build during `terraform apply`
    path = "../example-app"
  }
}

resource "heroku_formation" "foobar" {
  app        = "${heroku_app.foobar.id}"
  type       = "web"
  quantity   = 1
  size       = "Standard-1x"
  depends_on = ["heroku_build.foobar"]
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The ID of the Heroku app
- `buildpacks` - List of buildpack GitHub URLs
- `source` - (Required) A block that specifies the source code to build & release:
  - `checksum` - Hash of the source archive for verifying its integrity, auto-generated when `source.path` is set, `SHA256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b93`
  - `path` - (Required unless `source.url` is set) Local path to the source directory or tarball archive for the app

  - `url` - (Required unless `source.path` is set) `https` location of the source archive for the app
  - `version` - Use to track what version of your source originated this build. If you are creating builds from git-versioned source code, for example, the commit hash, or release tag would be a good value to use for the version parameter.

» Attributes Reference

The following attributes are exported:

- `uuid` - The ID of the build
- `output_stream_url` - URL that streams the log output from the build
- `release_id` - The Heroku app release created with a build's slug
- `slug_id` - The Heroku slug created by a build
- `stack` - Name or ID of the Heroku stack
- `status` - The status of a build. Possible values are `pending`, `successful` and `failed`
- `user` - Heroku account that created a build
  - `email`
  - `id`

» Import

Existing builds can be imported using the combination of the application name, a colon, and the build ID.

For example: `$ terraform import heroku_build.foobar bazbux:4f1db8ef-ed5c-4c42-a3d6-3c28262d5`

- `foobar` is the **heroku_build** resource's name
- `bazbux` is the Heroku app name (or ID) that the build belongs to
- `:` separates the app identifier & the build identifier
- `4f1db8ef…` is the build ID

» heroku_cert

Provides a Heroku SSL certificate resource. It allows to set a given certificate for a Heroku app.

» Example Usage

```
# Create a new Heroku app
resource "heroku_app" "default" {
```

```
  name = "test-app"
}

# Add-on SSL to application
resource "heroku_addon" "ssl" {
  app  = "${heroku_app.default.name}"
  plan = "ssl"
}

# Establish certificate for a given application
resource "heroku_cert" "ssl_certificate" {
  app               = "${heroku_app.default.name}"
  certificate_chain = "${file("server.crt")}"
  private_key       = "${file("server.key")}"
  depends_on        = ["heroku_addon.ssl"]
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The Heroku app to add to.
- `certificate_chain` - (Required) The certificate chain to add
- `private_key` - (Required) The private key for a given certificate chain

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the add-on
- `cname` - The CNAME for the SSL endpoint
- `name` - The name of the SSL certificate

» Importing

When importing a Heroku cert resource, the ID must be built using the app name colon the unique ID from the Heroku API. For an app named `production-api` with a certificate ID of `b85d9224-310b-409b-891e-c903f5a40568`, you would import it as:

```
$ terraform import heroku_cert.production_api production-api:b85d9224-310b-409b-891e-c903f5a
```

» heroku_config

Provides a Heroku Config resource, making it possible to define variables to be used throughout your Heroku terraform configurations. Combined with `heroku_app_config_association`, these two resources enable users to decouple setting config var(s) from the `heroku_app` resource.

NOTE: Unlike most Terraform resources, this resource **DOES NOT** by itself create, update or delete anything in Heroku. A `heroku_app_config_association`, `heroku_app.config_vars`, or `heroku_app.sensitive_config_vars` is required to actually set these values on Heroku apps.

» Example HCL

```
resource "heroku_config" "endpoints" {
    vars = {
        x = "https://..."
        y = "https://..."
        z = "https://..."
    }

    sensitive_vars = {
        PRIVATE_KEY = "some_private_key"
    }
}
```

» Argument Reference

- `vars` - Map of vars that are can be outputted in plaintext.
- `sensitive_vars` - This is the same as `vars`. The main difference between the two attributes is `sensitive_vars` outputs are redacted on-screen and replaced by a placeholder, following a terraform `plan` or `apply`. It is recommended to put private keys, passwords, etc in this argument.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the config.

» Import

The `heroku_config` resource is a meta-resource, managed only within Terraform state. It does not exist as a native Heroku resource. Therefore, it is not possible to import an existing `heroku_config` configuration.

» heroku_domain

Provides a Heroku Domain resource. This can be used to create and manage custom domains on Heroku.

» Example Usage

```
# Create a new Heroku app
resource "heroku_app" "default" {
  name = "test-app"
}

# Associate a custom domain
resource "heroku_domain" "default" {
  app      = "${heroku_app.default.name}"
  hostname = "terraform.example.com"
}
```

» Argument Reference

The following arguments are supported:

- `hostname` - (Required) The hostname to serve requests from.
- `app` - (Required) The Heroku app to link to.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the of the domain record.
- `hostname` - The hostname traffic will be served as.
- `cname` - The CNAME traffic should route to.

» Importing

When importing a Heroku domain resource, the ID must be built using the app name colon the unique ID from the Heroku API. For an app named `production-api` with a domain ID of `b85d9224-310b-409b-891e-c903f5a40568`, you would import it as:

```
$ terraform import heroku_domain.production_api production-api:b85d9224-310b-409b-891e-c903f
```

» heroku_drain

Provides a Heroku Drain resource. This can be used to create and manage Log Drains on Heroku.

» Example Usage

```
resource "heroku_drain" "default" {
  app = "test-app"
  url = "syslog://terraform.example.com:1234"
}
```

» Argument Reference

The following arguments are supported:

- `url` - (Required) The URL for Heroku to drain your logs to.
- `app` - (Required) The Heroku app to link to.

» Attributes Reference

The following attributes are exported:

- `token` - The unique token for your created drain.

» Importing

When importing a Heroku drain resource, the ID must be built using the app name colon the unique ID from the Heroku API. For an app named `production-api` with a drain ID of `b85d9224-310b-409b-891e-c903f5a40568`, you would import it as:

```
$ terraform import heroku_drain.production_api production-api:b85d9224-310b-409b-891e-c903f5
```

» heroku_formation

Provides a Heroku Formation resource.

A formation represents the formation of processes that should be set for an application.

NOTE: - The application must have a dyno in order to update its formation. - If the heroku formation resource is removed and deleted, this will be a no-op action in Heroku. The Heroku Platform does not have a `DELETE` endpoint for `formation`. - This resource works well with the `heroku_app_release` resource, which allows you to deploy a slug/release to an application before the formation can be updated.

» Example Usage

```
# Creates a new application called foobar
resource "heroku_app" "foobar" {
    name = "foobar"
    region = "us"
}

# Creates a new release for application foobar using a slug id
resource "heroku_app_release" "foobar-release" {
    app = "${heroku_app.foobar.name}"
    slug_id = "01234567-89ab-cdef-0123-456789abcdef"
}

# Update the web formation for the foobar application's web
resource "heroku_formation" "foobar-web" {
    app = "${heroku_app.foobar.name}"
    type = "web"
    quantity = 2
    size = "standard-2x"

    # Tells Terraform that this formation must be created/updated only after the app release
    depends_on = ["heroku_app_release.foobar-release"]
}
```

» Argument Reference

- `app` - (Required) The name of the application
- `type` - (Required) type of process such as ”web”
- `quantity` - (Required) number of processes to maintain

- `size` - (Required) dyno size (Example: "standard-1X"). Capitalization does not matter.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the formation

» Import

Existing formations can be imported using the combination of the application name, a colon, and the formation's type.

For example:

```
$ terraform import heroku_formation.foobar-web foobar:web
```

» heroku_pipeline

Provides a Heroku Pipeline resource.

A pipeline is a group of Heroku apps that share the same codebase. Once a pipeline is created, and apps are added to different stages using `heroku_pipeline_coupling`, you can promote app slugs to the next stage.

» Example Usage

```
# Create Heroku apps for staging and production
resource "heroku_app" "staging" {
  name = "test-app-staging"
}

resource "heroku_app" "production" {
  name = "test-app-production"
}

# Create a Heroku pipeline
resource "heroku_pipeline" "test-app" {
  name = "test-app"
}

# Couple apps to different pipeline stages
```

```
resource "heroku_pipeline_coupling" "staging" {
  app      = "${heroku_app.staging.name}"
  pipeline = "${heroku_pipeline.test-app.id}"
  stage    = "staging"
}

resource "heroku_pipeline_coupling" "production" {
  app      = "${heroku_app.production.name}"
  pipeline = "${heroku_pipeline.test-app.id}"
  stage    = "production"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the pipeline.

» Attributes Reference

The following attributes are exported:

- `id` - The UUID of the pipeline.
- `name` - The name of the pipeline.

» Import

Pipelines can be imported using the Pipeline `id`, e.g.

```
$ terraform import heroku_pipeline.foobar 12345678
```

» heroku_pipeline_coupling

Provides a Heroku Pipeline Coupling resource.

A pipeline is a group of Heroku apps that share the same codebase. Once a pipeline is created using `heroku_pipeline`, and apps are added to different stages using `heroku_pipeline_coupling`, you can promote app slugs to the downstream stages.

» Example Usage

```
# Create Heroku apps for staging and production
resource "heroku_app" "staging" {
  name = "test-app-staging"
}

resource "heroku_app" "production" {
  name = "test-app-production"
}

# Create a Heroku pipeline
resource "heroku_pipeline" "test-app" {
  name = "test-app"
}

# Couple apps to different pipeline stages
resource "heroku_pipeline_coupling" "staging" {
  app      = "${heroku_app.staging.name}"
  pipeline = "${heroku_pipeline.test-app.id}"
  stage    = "staging"
}

resource "heroku_pipeline_coupling" "production" {
  app      = "${heroku_app.production.name}"
  pipeline = "${heroku_pipeline.test-app.id}"
  stage    = "production"
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The name of the app for this coupling.
- `pipeline` - (Required) The ID of the pipeline to add this app to.
- `stage` - (Required) The stage to couple this app to. Must be one of `review`, `development`, `staging`, or `production`.

» Attributes Reference

The following attributes are exported:

- `id` - The UUID of this pipeline coupling.
- `app` - The name of the application.
- `app_id` - The ID of the application.

- `pipeline` - The UUID of the pipeline.
- `stage` - The stage for this coupling.

» Import

Pipeline couplings can be imported using the Pipeline coupling `id`, e.g.

```
$ terraform import heroku_pipeline_coupling.foobar 12345678
```

» heroku_slug

Provides a Heroku Slug resource.

This resource supports uploading a pre-generated archive file of executable code, making it possible to launch apps directly from a Terraform config. This resource does not itself generate the slug archive. A guide to creating slug archives is available in the Heroku Dev Center.

» Minimal Example

Create a ready-to-release slug:

- `file_url` or `file_path` must reference a file containing a slug archive of executable code and must follow the prescribed layout from Create slug archive in the Heroku Dev Center (nested within an `./app` directory)
- The archive may be created by an external build system, downloaded from another Heroku app, or otherwise provided outside of the context of this Terraform resource
- If the content (SHA256) of `file_path` changes, then a new resource will be forced on the next plan/apply; if the file does not exist, the difference is ignored.
- The `file_url` is only fetched during resource creation. To trigger another fetch the `file_url` should be changed, then a new resource will be forced on the next plan/apply.

```
resource "heroku_slug" "foobar" {
  app      = "${heroku_app.foobar.id}"
  file_url = "https://github.com/terraform-providers/terraform-provider-heroku/raw/master/he

  process_types = {
    web = "ruby server.rb"
  }
}
```

» Example Usage

Complete config to launch a Heroku app:

```
resource "heroku_app" "foobar" {
    name = "foobar"
    region = "us"
}

# Create a slug for the app with a local slug archive file
resource "heroku_slug" "foobar" {
  app                            = "${heroku_app.foobar.id}"
  buildpack_provided_description = "Ruby"
  // The slug archive file must already exist
  file_path                      = "slug.tgz"

  process_types = {
    web = "ruby server.rb"
  }
}

# Deploy a release to the app with the slug
resource "heroku_app_release" "foobar" {
  app     = "${heroku_app.foobar.id}"
  slug_id = "${heroku_slug.foobar.id}"
}

# Launch the app's web process by scaling-up
resource "heroku_formation" "foobar" {
  app        = "${heroku_app.foobar.id}"
  type       = "web"
  quantity   = 1
  size       = "Standard-1x"
  depends_on = ["heroku_app_release.foobar"]
}
```

» Argument Reference

The following arguments are supported:

- `app` - (Required) The ID of the Heroku app
- `buildpack_provided_description` - Description of language or app framework, `"Ruby/Rack"`; displayed as the app's language in the Heroku Dashboard
- `checksum` - Hash of the slug for verifying its integrity, auto-generated from contents of `file_path` or `file_url`, `SHA256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4`

- `commit` - Identification of the code with your version control system (eg: SHA of the git HEAD), `"60883d9e8947a57e04dc9124f25df004866a2051"`
- `commit_description` - Description of the provided commit
- `file_path` - (Required unless `file_url` is set) Local path to a slug archive, `"slugs/current.tgz"`
- `file_url` - (Required unless `file_path` is set) **https** URL to a slug archive, `"https://example.com/slugs/app-v1.tgz"`
- `process_types` - (Required) Map of processes to launch on Heroku Dynos
- `stack` - Name or ID of the Heroku stack

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the slug
- `app` - The ID or unique name of the Heroku app
- `blob` - Slug archive (compressed tar of executable code)
  - `method` - HTTP method to upload the archive
  - `url` - Pre-signed, expiring URL to upload the archive
- `buildpack_provided_description` - Description of language or app framework, `"Ruby/Rack"`
- `checksum` - Hash of the slug for verifying its integrity, auto-generated from contents of `file_path` or `file_url`
- `commit` - Identification of the code with your version control system (eg: SHA of the git HEAD), `"60883d9e8947a57e04dc9124f25df004866a2051"`
- `commit_description` - Description of the provided commit
- `process_types` - Map of processes to launch on Heroku Dynos
- `size` - Slug archive filesize in bytes
- `stack` - Heroku stack name
- `stack_id` - Heroku stack ID

» Import

Existing slugs can be imported using the combination of the application name, a colon, and the slug ID.

For example:

```
$ terraform import heroku_slug.foobar bazbux:4f1db8ef-ed5c-4c42-a3d6-3c28262d5abc
```

- `foobar` is the **heroku_slug** resource's name
- `bazbux` is the Heroku app name (or ID) that the slug belongs to
- `:` separates the app identifier & the slug identifier
- `4f1db8ef…` is the slug ID

» heroku_space

Provides a Heroku Private Space resource for running apps in isolated, highly available, secure app execution environments.

» Example Usage

A Heroku "team" was originally called an "organization", and that is still the identifier used in this resource.

```
// Create a new Heroku space
resource "heroku_space" "default" {
  name = "test-space"
  organization = "my-company"
  region = "virginia"
}

// Create a new Heroku app in test-space
resource "heroku_app" "default" {
  name = "test-app"
  space = "${heroku_space.default.name}"
  organization = {
    name = "my-company"
  }
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the Private Space.
- `organization` - (Required) The name of the Heroku Team which will own the Private Space.
- `cidr` - (Optional) The RFC-1918 CIDR the Private Space will use. It must be a /16 in 10.0.0.0/8, 172.16.0.0/12 or 192.168.0.0/16
- `data_cidr` - (Optional) The RFC-1918 CIDR that the Private Space will use for the Heroku-managed peering connection that's automatically created when using Heroku Data add-ons. It must be between a /16 and a /20
- `region` - (Optional) provision in a specific Private Spaces region.
- `shield` - (Optional) provision as a Shield Private Space.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the space.
- `name` - The space's name.
- `organization` - The space's Heroku Team.
- `region` - The space's region.
- `cidr` - The space's CIDR.
- `data_cidr` - The space's Data CIDR.
- `outbound_ips` - The space's stable outbound NAT IPs.

» Import

Spaces can be imported using the space `id`, e.g.

```
$ terraform import heroku_space.foobar MySpace
```

» heroku_space_app_access

Provides a resource for managing permissions for the entire Private Space. Members with the admin role will always have full permissions in the Private Space, so using this resource on an admin will have no effect. The provided email must already be a member of the Heroku Team. Currently the only supported permission is `create_apps`.

» Example Usage

```
// Create a new Heroku Private Space
resource "heroku_space" "default" {
  name = "test-space"
  organization = "my-company"
  region = "virginia"
}

// Give an existing team member create_apps permissions to the Private Space
resource "heroku_space_app_access" "member1" {
  space = "${heroku_space.default.name}"
  email = "member1@example.com"
  permissions = ["create_apps"]
}

// Remove all permissions from an existing team member
```

```
resource "heroku_space_app_access" "member2" {
  space = "${heroku_space.default.name}"
  email = "member2@example.com"
  permissions = []
}
```

» Argument Reference

The following arguments are supported:

- `space` - (Required) The name of the Private Space.
- `email` - (Required) The email of the existing Heroku Team member.
- `permissions` - (Required) The permissions to grant the team member for the Private Space. Currently `create_apps` is the only supported permission. If not provided the member will have no permissions to the space. Members with admin role will always have `create_apps` permissions, which cannot be removed.

» Importing

Existing permissions can be imported using the combination of the Private Space name, a colon, and the member email.

For example:

```
$ terraform import heroku_space_app_access.member1 my-space:member1@foobar.com
```

» heroku_space_inbound_ruleset

Provides a resource for managing inbound rulesets for Heroku Private Spaces.

» Example Usage

```
# Create a new Heroku space
resource "heroku_space" "default" {
  name         = "test-space"
  organization = "my-company"
  region       = "virginia"
}

# Allow all traffic EXCEPT 8.8.4.4 to access the HPS.
resource "heroku_space_inbound_ruleset" "default" {
  space = "${heroku_space.default.name}"
```

```
  rule {
    action = "allow"
    source = "0.0.0.0/0"
  }

  rule {
    action = "deny"
    source = "8.8.4.4/32"
  }
}
```

» Argument Reference

The following arguments are supported:

- `space` - (Required) The name of the space.
- `rule` - (Required) At least one `rule` block. Rules are documented below.

A `rule` block supports the following arguments:

- `action` - (Required) The action to apply this rule to. Must be one of `allow` or `deny`.
- `source` - (Required) A CIDR block source for the rule.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the inbound ruleset.

» heroku_space_peering_connection_accepter

Provides a resource for accepting VPC peering requests to Heroku Private Spaces.

» Example Usage

```
# Fetch the peering information for the Heroku Private Space.
data "heroku_space_peering_info" "peer_space" {
  name = "my-fancy-space"
}
```

```
# Initiate the request.
resource "aws_vpc_peering_connection" "request" {
  peer_owner_id = "${data.heroku_space_peering_info.peer_space.aws_account_id}"
  peer_vpc_id   = "${data.heroku_space_peering_info.peer_space.vpc_id}"
  vpc_id        = "${aws_vpc.main.id}"
}

# Accept the request.
resource "heroku_space_peering_connection_accepter" "accept" {
  space                     = "${heroku_space.peer_space.name}"
  vpc_peering_connection_id = "${aws_vpc_peering_connection.request.id}"
}
```

» Argument Reference

The following arguments are supported:

- `space` - (Required) The name of the space.
- `vpc_peering_connection_id` - (Required) The peering connection request ID.

» Attributes Reference

The following attributes are exported:

- `status` - The status of the peering connection request.
- `type` - The type of the peering connection.

» heroku_space_vpn_connection

Provides a resource for creating a VPN connection between a network and a Heroku Private Space. For more information, see Private Spaces VPN Connection in the Heroku DevCenter.

» Example Usage

```
// Create a new Heroku space
resource "heroku_space" "default" {
  name         = "test-space"
  organization = "my-company"
  region       = "virginia"
}
```

```
// Connect the Heroku space to another network with a VPN
resource "heroku_space_vpn_connection" "office" {
  name           = "office"
  space          = "${heroku_space.default.name}"
  public_ip      = "203.0.113.1"
  routable_cidrs = ["192.168.1.0/24"]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the VPN connection.
- `space` - (Required) The name of the Heroku Private Space where the VPN connection will be established.
- `public_ip` - (Required) The public IP address of the VPN endpoint on the network where the VPN connection will be established.
- `routable_cidrs` - (Required) A list of IPv4 CIDR blocks used by the network where the VPN connection will be established.

» Attributes Reference

The following attributes are exported:

- `space_cidr_block` - The CIDR block for the Heroku Private Space. The network where the VPN will be established should be configured to route traffic destined for this CIDR block over the VPN link.
- `ike_version` - The IKE version used to setup the IPsec tunnel.
- `tunnels` - Details about each VPN tunnel endpoint.
  - `ip` - The public IP address of the tunnel.
  - `pre_shared_key` - The pre-shared IPSec secret for the tunnel.

» heroku_team_collaborator

A Heroku Team Collaborator receives access to a specific Team-owned app.

To create a Heroku Team, use the New Team feature of Heroku Dashboard. For Heroku Enterprise accounts, new Teams may be created within the account by users with the right permissions.

A Heroku "team" was originally called an "organization", and that is still the identifier used elsewhere in this provider. For `heroku_app` & `heroku_space` resources, set the Heroku Team name as the "organization".

NOTE: This resource only works for Team-owned apps

» Example Usage

```
# Create a new team collaborator for the foobar application that has view, operate, manage p
resource "heroku_team_collaborator" "foobar-collaborator" {
    app = "${heroku_app.foobar.name}"
    email = "collaborator@foobar.com"
    permissions = ["view", "operate", "manage"]
}
```

» Argument Reference

- `app` - (Required) The name of the team app that the team collaborator will be added to.
- `email` - (Required) Email address of the team collaborator
- `permissions` - (Required) List of permissions that will be granted to the team collaborator. The order in which individual permissions are set here does not matter. Please visit this link for more information on available permissions.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the team collaborator

» Import

Team Collaborators can be imported using the combination of the team application name, a colon, and the collaborator's email address

For example:

```
$ terraform import heroku_team_collaborator.foobar-collaborator foobar_app:collaborator@foob
```

» heroku_team_member

A Heroku Team Member receives access to everything owned by the Team.

To create a Heroku Team, use the New Team feature of Heroku Dashboard. For Heroku Enterprise accounts, new Teams may be created within the account by users with the right permissions.

A Heroku "team" was originally called an "organization", and that is still the identifier used elsewhere in this provider. For `heroku_app` & `heroku_space` resources, set the Heroku Team name as the "organization".

» Example Usage

```
# Adds a Heroku user to a Heroku team as a viewer.
resource "heroku_team_member" "foobar-member" {
  team  = "my-team"
  email = "some-user@example.com"
  role  = "member"
}
```

» Argument Reference

- `team` - (Required) The name of the Heroku Team.
- `email` - (Required) Email address of the member
- `role` - (Required) The role to assign the member. See the API docs for available options.

» Import

Team members can be imported using the combination of the team application name, a colon, and the member's email address.

```
$ terraform import heroku_team_member.foobar-member my-team-foobar:some-user@example.com
```