Terraform Provider

The terraform provider provides access to outputs from the Terraform state of shared infrastructure.

Use the navigation to the left to read about the available data sources.

Example Usage

```
# Shared infrastructure state stored in Atlas
data "terraform_remote_state" "vpc" {
  backend = "atlas"

  config {
    name = "hashicorp/vpc-prod"
  }
}

resource "aws_instance" "foo" {
  # ...
  subnet_id = "${data.terraform_remote_state.vpc.subnet_id}"
}
```

Atlas Provider

This provider is deprecated, and the service it interacts with has been discontinued.

The Atlas provider was used to interact with objects in Atlas's artifact registry, which were usually machine images built with Packer. In 2018, the Atlas suite was discontinued. It is replaced by Terraform Enterprise (https://www.terraform.io/docs/enterprise/index.html), which focuses on the core Terraform workflow and does not include an artifact registry. For more information, see Differences Between Current and Legacy Terraform Enterprise (https://www.terraform.io/docs/enterprise/upgrade/differences.html).

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

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

Example Usage

```
# Configure the Atlas provider
provider "atlas" {
  token = "${var.atlas_token}"
}

# Fetch an artifact configuration
data "atlas_artifact" "web" {
  # ...
}
```

Argument Reference

The following arguments are supported:

- `address` - (Optional) Atlas server endpoint. Defaults to public Atlas. This is only required when using an on-premise deployment of Atlas. This can also be specified with the `ATLAS_ADDRESS` shell environment variable.
- `token` - (Required) API token. This can also be specified with the `ATLAS_TOKEN` shell environment variable.

atlas_artifact

Provides a Data Source (/docs/configuration/data-sources.html) to access to deployment artifacts managed by Atlas. This can be used to dynamically configure instantiation and provisioning of resources.

This provider is deprecated, and the service it interacts with has been discontinued.

Example Usage

An artifact can be created that has metadata representing an AMI in AWS. This AMI can be used to configure an instance. Any changes to this artifact will trigger a change to that instance.

```
# Read the AMI
data "atlas_artifact" "web" {
  name  = "hashicorp/web"
  type  = "amazon.image"
  build = "latest"

  metadata {
    arch = "386"
  }
}

# Start our instance with the dynamic ami value
# Remember to include the AWS region as it is part of the full ID
resource "aws_instance" "app" {
  ami = "${data.atlas_artifact.web.metadata_full.region-us-east-1}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) Name of the artifact in Atlas. This is given in slug format like "organization/artifact".
- `type` - (Required) The type of artifact to query for.
- `build` - (Optional) The build number responsible for creating the version of the artifact to filter on. This can be "latest", to find a matching artifact in the latest build, "any" to find a matching artifact in any build, or a specific number to pin to that build. If `build` and `version` are unspecified, `version` will default to "latest". Cannot be specified with `version`. Note: `build` is only present if Atlas builds the image.
- `version` - (Optional) The version of the artifact to filter on. This can be "latest", to match against the latest version, "any" to find a matching artifact in any version, or a specific number to pin to that version. Defaults to "latest" if neither `build` or `version` is specified. Cannot be specified with `build`.
- `metadata_keys` - (Optional) If given, only an artifact containing the given keys will be returned. This is used to disambiguate when multiple potential artifacts match. An example is "aws" to filter on an AMI.

- `metadata` - (Optional) If given, only an artifact matching the metadata filters will be returned. This is used to disambiguate when multiple potential artifacts match. An example is "arch" = "386" to filter on architecture.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the artifact. This could be an AMI ID, GCE Image ID, etc.
- `file_url` - For artifacts that are binaries, this is a download path.
- `metadata_full` - Contains the full metadata of the artifact. The keys are sanitized to replace any characters that are invalid in a resource name with a hyphen. For example, the "region.us-east-1" key will become "region-us-east-1".
- `version_real` - The matching version of the artifact
- `slug` - The artifact slug in Atlas

atlas_artifact

Provides access to deployment artifacts managed by Atlas. This can be used to dynamically configure instantiation and provisioning of resources.

This provider is deprecated, and the service it interacts with has been discontinued.

This resource is deprecated! Please use the Artifact Data Source (/docs/providers/terraform-enterprise/d/artifact.html)

Example Usage

An artifact can be created that has metadata representing an AMI in AWS. This AMI can be used to configure an instance. Any changes to this artifact will trigger a change to that instance.

```
# Read the AMI
resource "atlas_artifact" "web" {
  name  = "hashicorp/web"
  type  = "amazon.image"
  build = "latest"

  metadata {
    arch = "386"
  }
}

# Start our instance with the dynamic ami value
# Remember to include the AWS region as it is part of the full ID
resource "aws_instance" "app" {
  ami = "${atlas_artifact.web.metadata_full.region-us-east-1}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) Name of the artifact in Atlas. This is given in slug format like "organization/artifact".
- `type` - (Required) The type of artifact to query for.
- `build` - (Optional) The build number responsible for creating the version of the artifact to filter on. This can be "latest", to find a matching artifact in the latest build, "any" to find a matching artifact in any build, or a specific number to pin to that build. If `build` and `version` are unspecified, `version` will default to "latest". Cannot be specified with `version`. Note: `build` is only present if Atlas builds the image.
- `version` - (Optional) The version of the artifact to filter on. This can be "latest", to match against the latest version, "any" to find a matching artifact in any version, or a specific number to pin to that version. Defaults to "latest" if neither `build` or `version` is specified. Cannot be specified with `build`.

- `metadata_keys` - (Optional) If given, only an artifact containing the given keys will be returned. This is used to disambiguate when multiple potential artifacts match. An example is "aws" to filter on an AMI.
- `metadata` - (Optional) If given, only an artifact matching the metadata filters will be returned. This is used to disambiguate when multiple potential artifacts match. An example is "arch" = "386" to filter on architecture.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the artifact. This could be an AMI ID, GCE Image ID, etc.
- `file_url` - For artifacts that are binaries, this is a download path.
- `metadata_full` - Contains the full metadata of the artifact. The keys are sanitized to replace any characters that are invalid in a resource name with a hyphen. For example, the "region.us-east-1" key will become "region-us-east-1".
- `version_real` - The matching version of the artifact
- `slug` - The artifact slug in Atlas

remote_state

Retrieves state data from a Terraform backend (/docs/backends/index.html). This allows you to use the root-level outputs of one or more Terraform configurations as input data for another configuration.

Although this data source uses Terraform's backends (/docs/backends/index.html), it doesn't have the same limitations as the main backend configuration. You can use any number of `remote_state` data sources with differently configured backends, and you can use interpolations when configuring them.

Example Usage

```
data "terraform_remote_state" "vpc" {
  backend = "atlas"
  config {
    name = "hashicorp/vpc-prod"
  }
}

resource "aws_instance" "foo" {
  # ...
  subnet_id = "${data.terraform_remote_state.vpc.subnet_id}"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Required) The remote backend to use.
- `workspace` - (Optional) The Terraform workspace to use, if the backend supports workspaces.
- `config` - (Optional; block) The configuration of the remote backend. The `config` block can use any arguments that would be valid in the equivalent `terraform { backend "<TYPE>" { ... } }`block. See the documentation of your chosen backend (/docs/backends/types/index.html) for details.
- `defaults` - (Optional; block) Default values for outputs, in case the state file is empty or lacks a required output.

Attributes Reference

The following attributes are exported:

- `backend` - See Argument Reference above.
- `config` - See Argument Reference above.
- `<OUTPUT NAME>` - Each root-level output (/docs/configuration/outputs.html) in the remote state appears as a top level attribute on the data source.

Root Outputs Only

Only the root-level outputs from the remote state are accessible. Outputs from modules within the state cannot be accessed. If you want a module output or a resource attribute to be accessible via a remote state, you must thread the output through to a root output.

For example:

```
module "app" {
  source = "..."
}

output "app_value" {
  value = "${module.app.value}"
}
```

In this example, the output `value` from the "app" module is available as `app_value`. If this root level output hadn't been created, then a remote state resource wouldn't be able to access the `value` output on the module.