External Provider

`external` is a special provider that exists to provide an interface between Terraform and external programs.

Using this provider it is possible to write separate programs that can participate in the Terraform workflow by implementing a specific protocol.

This provider is intended to be used for simple situations where you wish to integrate Terraform with a system for which a first-class provider doesn't exist. It is not as powerful as a first-class Terraform provider, so users of this interface should carefully consider the implications described on each of the child documentation pages (available from the navigation bar) for each type of object this provider supports.

Warning Terraform Enterprise does not guarantee availability of any particular language runtimes or external programs beyond standard shell utilities, so it is not recommended to use this provider within configurations that are applied within Terraform Enterprise.

External Data Source

The `external` data source allows an external program implementing a specific protocol (defined below) to act as a data source, exposing arbitrary data for use elsewhere in the Terraform configuration.

> **Warning** This mechanism is provided as an "escape hatch" for exceptional situations where a first-class Terraform provider is not more appropriate. Its capabilities are limited in comparison to a true data source, and implementing a data source via an external program is likely to hurt the portability of your Terraform configuration by creating dependencies on external programs and libraries that may not be available (or may need to be used differently) on different operating systems.

> **Warning** Terraform Enterprise does not guarantee availability of any particular language runtimes or external programs beyond standard shell utilities, so it is not recommended to use this data source within configurations that are applied within Terraform Enterprise.

Example Usage

```
data "external" "example" {
  program = ["python", "${path.module}/example-data-source.py"]

  query = {
    # arbitrary map from strings to strings, passed
    # to the external program as the data query.
    id = "abc123"
  }
}
```

External Program Protocol

The external program described by the `program` attribute must implement a specific protocol for interacting with Terraform, as follows.

The program must read all of the data passed to it on `stdin`, and parse it as a JSON object. The JSON object contains the contents of the `query` argument and its values will always be strings.

The program must then produce a valid JSON object on `stdout`, which will be used to populate the `result` attribute exported to the rest of the Terraform configuration. This JSON object must again have all of its values as strings. On successful completion it must exit with status zero.

If the program encounters an error and is unable to produce a result, it must print a human-readable error message (ideally a single line) to `stderr` and exit with a non-zero status. Any data on `stdout` is ignored if the program returns a non-zero status.

All environment variables visible to the Terraform process are passed through to the child program.

Terraform expects a data source to have *no observable side-effects*, and will re-run the program each time the state is refreshed.

Argument Reference

The following arguments are supported:

- `program` - (Required) A list of strings, whose first element is the program to run and whose subsequent elements are optional command line arguments to the program. Terraform does not execute the program through a shell, so it is not necessary to escape shell metacharacters nor add quotes around arguments containing spaces.
- `query` - (Optional) A map of string values to pass to the external program as the query arguments. If not supplied, the program will receive an empty object as its input.

Attributes Reference

The following attributes are exported:

- `result` - A map of string values returned from the external program.

Processing JSON in shell scripts

Since the external data source protocol uses JSON, it is recommended to use the utility `jq` (https://stedolan.github.io/jq/) to translate to and from JSON in a robust way when implementing a data source in a shell scripting language.

The following example shows some input/output boilerplate code for a data source implemented in bash:

```
#!/bin/bash

# Exit if any of the intermediate steps fail
set -e

# Extract "foo" and "baz" arguments from the input into
# FOO and BAZ shell variables.
# jq will ensure that the values are properly quoted
# and escaped for consumption by the shell.
eval "$(jq -r '@sh "FOO=\(.foo) BAZ=\(.baz)"')"

# Placeholder for whatever data-fetching logic your script implements
FOOBAZ="$FOO BAZ"

# Safely produce a JSON object containing the result value.
# jq will ensure that the value is properly quoted
# and escaped to produce a valid JSON string.
jq -n --arg foobaz "$FOOBAZ" '{"foobaz":$foobaz}'
```