» template_file

The `template_file` data source renders a template from a template string, which is usually loaded from an external file.

Note In Terraform 0.12 and later, the `templatefile` function offers a built-in mechanism for rendering a template from a file. Use that function instead, unless you are using Terraform 0.11 or earlier.

» Example Usage

```
data "template_file" "init" {
  template = "${file("${path.module}/init.tpl")}"
  vars = {
    consul_address = "${aws_instance.consul.private_ip}"
  }
}
```

Inside `init.tpl` you can include the value of `consul_address`. For example:

```
#!/bin/bash

echo "CONSUL_ADDRESS = ${consul_address}" > /tmp/iplist
```

Although in principle `template_file` can be used with an inline template string, we don't recommend this approach because it requires awkward escaping. Instead, just use template syntax directly in the configuration. For example:

```
  user_data = <<-EOT
    echo "CONSUL_ADDRESS = ${aws_instance.consul.private_ip}" > /tmp/iplist
  EOT
```

» Argument Reference

The following arguments are supported:

- `template` - (Required) The contents of the template, as a string using Terraform template syntax. Use the `file` function to load the template source from a separate file on disk.
- `vars` - (Optional) Variables for interpolation within the template. Note that variables must all be primitives. Direct references to lists or maps will cause a validation error.

Earlier versions of `template_file` accepted another argument `filename` as an alternative to `template`. This has now been removed. Use the `template` argument with the `file` function to get the same effect.

» Template Syntax

The `template` argument is processed as Terraform template syntax.

However, this provider has its own copy of the template engine embedded in it, separate from Terraform itself, and so which features are available are decided based on what Terraform version the provider was compiled against, and not on which Terraform version you are running.

For more consistent results, Terraform 0.12 has a built in function `templatefile` which serves the same purpose as this data source. Use that function instead if you are using Terraform 0.12 or later. Its template and expression capabilities will always match the version of Terraform you are using.

» Attributes Reference

The following attributes are exported:

- `template` - See Argument Reference above.
- `vars` - See Argument Reference above.
- `rendered` - The final rendered template.

» template_cloudinit_config

Renders a multipart MIME configuration for use with Cloud-init.

Cloud-init is a commonly-used startup configuration utility for cloud compute instances. It accepts configuration via provider-specific user data mechanisms, such as `user_data` for Amazon EC2 instances. Multipart MIME is one of the data formats it accepts. For more information, see User-Data Formats in the Cloud-init manual.

This is not a generalized utility for producing multipart MIME messages. Its featureset is specialized for the features of cloud-init.

» Example Usage

```
# Render a part using a `template_file`
data "template_file" "script" {
  template = "${file("${path.module}/init.tpl")}"

  vars {
    consul_address = "${aws_instance.consul.private_ip}"
  }
}
```

```
# Render a multi-part cloud-init config making use of the part
# above, and other source files
data "template_cloudinit_config" "config" {
  gzip          = true
  base64_encode = true

  # Main cloud-config configuration file.
  part {
    filename     = "init.cfg"
    content_type = "text/cloud-config"
    content      = "${data.template_file.script.rendered}"
  }

  part {
    content_type = "text/x-shellscript"
    content      = "baz"
  }

  part {
    content_type = "text/x-shellscript"
    content      = "ffbaz"
  }
}

# Start an AWS instance with the cloud-init config as user data
resource "aws_instance" "web" {
  ami              = "ami-d05e75b8"
  instance_type    = "t2.micro"
  user_data_base64 = "${data.template_cloudinit_config.config.rendered}"
}
```

» Argument Reference

The following arguments are supported:

- `gzip` - (Optional) Specify whether or not to gzip the rendered output. Defaults to `true`.
- `base64_encode` - (Optional) Base64 encoding of the rendered output. Defaults to `true`, and cannot be disabled if `gzip` is `true`.
- `part` - (Required) A nested block type which adds a file to the generated cloud-init configuration. Use multiple `part` blocks to specify multiple files, which will be included in order of declaration in the final MIME document.

Each `part` block expects the following arguments:

- `content` - (Required) Body content for the part.
- `filename` - (Optional) A filename to report in the header for the part.
- `content_type` - (Optional) A MIME-style content type to report in the header for the part.
- `merge_type` - (Optional) A value for the `X-Merge-Type` header of the part, to control cloud-init merging behavior.

» Attributes Reference

The following attributes are exported:

- `rendered` - The final rendered multi-part cloud-init config.

» template_dir

Renders a directory containing templates into a separate directory of corresponding rendered files.

`template_dir` is similar to `template_file` but it walks a given source directory and treats every file it encounters as a template, rendering it to a corresponding file in the destination directory.

Note When working with local files, Terraform will detect the resource as having been deleted each time a configuration is applied on a new machine where the destination dir is not present and will generate a diff to create it. This may cause "noise" in diffs in environments where configurations are routinely applied by many different users or within automation systems.

» Example Usage

The following example shows how one might use this resource to produce a directory of configuration files to upload to a compute instance, using Amazon EC2 as a placeholder.

```
resource "template_dir" "config" {
  source_dir      = "${path.module}/instance_config_templates"
  destination_dir = "${path.cwd}/instance_config"

  vars = {
    consul_addr = "${var.consul_addr}"
  }
}
```

```
resource "aws_instance" "server" {
  ami           = "${var.server_ami}"
  instance_type = "t2.micro"

  connection {
    # ...connection configuration...
  }

  provisioner "file" {
    # Referencing the template_dir resource ensures that it will be
    # created or updated before this aws_instance resource is provisioned.
    source      = "${template_dir.config.destination_dir}"
    destination = "/etc/myapp"
  }
}

variable "consul_addr" {}

variable "server_ami" {}
```

» Argument Reference

The following arguments are supported:

- `source_dir` - (Required) Path to the directory where the files to template reside.
- `destination_dir` - (Required) Path to the directory where the templated files will be written.
- `vars` - (Optional) Variables for interpolation within the template. Note that variables must all be primitives. Direct references to lists or maps will cause a validation error.

Any required parent directories of `destination_dir` will be created automatically, and any pre-existing file or directory at that location will be deleted before template rendering begins.

After rendering, this resource remembers the content of both the source and destination directories in the Terraform state, and will plan to recreate the output directory if any changes are detected during the plan phase.

Note that it is *not* safe to use the `file` interpolation function to read files created by this resource, since that function can be evaluated before the destination directory has been created or updated. It *is* safe to use the generated files with resources that directly take filenames as arguments, as long as the path is constructed using the `destination_dir` attribute to create a dependency relationship with the `template_dir` resource.

» Template Syntax

The `template` argument is processed as Terraform template syntax.

However, this provider has its own copy of the template engine embedded in it, separate from Terraform itself, and so which features are available are decided based on what Terraform version the provider was compiled against, and not on which Terraform version you are running.

To include values from your configuration in rendered templates, pass them via the `vars` argument as shown below:

```
resource "template_dir" "init" {
  # ...

  vars = {
    foo  = "${var.foo}"
    attr = "${aws_instance.foo.private_ip}"
  }
}
```

» Attributes

This resource exports the following attributes:

- `destination_dir` - The destination directory given in configuration. Interpolate this attribute into other resource configurations to create a dependency to ensure that the destination directory is populated before another resource attempts to read it.