» template_file

Renders a template from a file.

» Example Usage

Option 1: From a file:

Reference the template path:

```
data "template_file" "init" {
  template = "${file("${path.module}/init.tpl")}"

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

Inside the file, reference the variable as such:

```
#!/bin/bash

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

Option 2: Inline:

```
data "template_file" "init" {
  template = "$${consul_address}:1234"

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

» Argument Reference

The following arguments are supported:

- `template` - (Required) The contents of the template. These can be loaded from a file on disk using the `file()` interpolation function.
- `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.

The following arguments are maintained for backwards compatibility and may be removed in a future version:

- `filename` - *Deprecated, please use* `template` *instead.* The filename for the template. Use path variables to make this path relative to different path roots.

» Attributes Reference

The following attributes are exported:

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

» Template Syntax

The syntax of the template files is the same as standard interpolation syntax, but you only have access to the variables defined in the `vars` section.

To access interpolations that are normally available to Terraform configuration (such as other variables, resource attributes, module outputs, etc.) you'll have to expose them via `vars` as shown below:

```
data "template_file" "init" {
  # ...

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

» Inline Templates

Inline templates allow you to specify the template string inline without loading a file. An example is shown below:

```
data "template_file" "init" {
  template = "$${consul_address}:1234"

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

Important: Template variables in an inline template (such as `consul_address` above) must be escaped with a double-`$`. Unescaped interpolations will be processed by Terraform normally prior to executing the template.

An example of mixing escaped and non-escaped interpolations in a template:

```
variable "port" { default = 80 }

data "template_file" "init" {
  template = "$${foo}:${var.port}"

  vars {
    foo = "${count.index}"
  }
}
```

In the above example, the template is processed by Terraform first to turn it into: `${foo}:80`. After that, the template is processed as a template to interpolate `foo`.

In general, you should use template variables in the `vars` block and try not to mix interpolations. This keeps it understandable and has the benefit that you don't have to change anything to switch your template to a file.

» template_cloudinit_config

Renders a multi-part cloud-init config from source files.

» 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 cloudinit config making use of the part
# above, and other source files
data "template_cloudinit_config" "config" {
  gzip          = true
  base64_encode = true

  # Setup hello world script to be called by the cloud-config
  part {
    filename     = "init.cfg"
    content_type = "text/part-handler"
```

```
    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 cloudinit config as user data
resource "aws_instance" "web" {
  ami           = "ami-d05e75b8"
  instance_type = "t2.micro"
  user_data     = "${data.template_cloudinit_config.config.rendered}"
}
```

» Argument Reference

The following arguments are supported:

- `gzip` - (Optional) Specify whether or not to gzip the rendered output. Default to `true`
- `base64_encode` - (Optional) Base64 encoding of the rendered output. Default to `true`
- `part` - (Required) One may specify this many times, this creates a fragment of the rendered cloud-init config file. The order of the parts is maintained in the configuration is maintained in the rendered template.

The `part` block supports:

- `filename` - (Optional) Filename to save part as.
- `content_type` - (Optional) Content type to send file as.
- `content` - (Required) Body for the part.
- `merge_type` - (Optional) Gives the ability to merge multiple blocks of cloud-config together.

» Attributes Reference

The following attributes are exported:

- `rendered` - The final rendered multi-part cloudinit 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 create 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 syntax of the template files is the same as standard interpolation syntax, but you only have access to the variables defined in the `vars` section.

To access interpolations that are normally available to Terraform configuration (such as other variables, resource attributes, module outputs, etc.) you can

expose them via `vars` 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.