HashiCorp
(https://www.hashicorp.com)

Command: validate

JUMP TO SECTION

The `terraform validate` command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state. It is thus primarily useful for general verification of reusable modules, including correctness of attribute names and value types.

It is safe to run this command automatically, for example as a post-save check in a text editor or as a test step for a re-usable module in a CI system.

Validation requires an initialized working directory with any referenced plugins and modules installed. To initialize a working directory for validation without accessing any configured remote backend, use:

```
$ terraform init -backend=false
```

To verify configuration in the context of a particular run (a particular target workspace, input variable values, etc), use the `terraform plan` command instead, which includes an implied validation check.

Usage

Usage: `terraform validate [options]`

This command accepts the following options:

- `-json` - Produce output in a machine-readable JSON format, suitable for use in text editor integrations and other automated systems. Always disables color.

- `-no-color` - If specified, output won't contain any color.

JSON Output Format

When you use the `-json` option, Terraform will produce validation results in JSON format to allow using the validation result for tool integrations, such as highlighting errors in a text editor.

As with all JSON output options, it's possible that Terraform will encounter an error prior to beginning the validation task that will thus not be subject to the JSON output setting. For that reason, external software consuming Terraform's output should be prepared to find data on stdout that *isn't* valid JSON, which it should then treat as a generic error case.

Note: The output includes a `format_version` key, which currently has major version zero to indicate that the format is experimental and subject to change. A future version will assign a non-zero major version and make stronger promises about compatibility. We do not anticipate any significant breaking changes to the format before its first major version, however.

In the normal case, Terraform will print a JSON object to the standard output stream. The top-level JSON object will have the following properties:

- `valid` (boolean): Summarizes the overall validation result, by indicating `true` if Terraform considers the current configuration to be valid or `false` if it detected any errors.

- `error_count` (number): A zero or positive whole number giving the count of errors Terraform detected. If `valid` is `false` then `error_count` will always be zero, because it is the presence of errors that indicates that a configuration is invalid.

- `warning_count` (number): A zero or positive whole number giving the count of warnings Terraform detected. Warnings do not cause Terraform to consider a configuration to be invalid, but they do indicate potential caveats that a user should consider and possibly resolve.

- `diagnostics` (array of objects): A JSON array of nested objects that each describe an error or warning from Terraform.

The nested objects in `diagnostics` have the following properties:

- `severity` (string): A string keyword, currently either `"error"` or `"warning"`, indicating the diagnostic severity.

  The presence of errors causes Terraform to consider a configuration to be invalid, while warnings are just advice or caveats to the user which do not block working with the configuration. Later versions of Terraform may introduce new severity keywords, so consumers should be prepared to accept and ignore severity values they don't understand.

- `summary` (string): A short description of the nature of the problem that the diagnostic is reporting.

  In Terraform's usual human-oriented diagnostic messages, the summary serves as a sort of "heading" for the diagnostic, printed after the "Error:" or "Warning:" indicator.

  Summaries are typically short, single sentences, but can sometimes be longer as a result of returning errors from subsystems that are not designed to return full diagnostics, where the entire error message therefore becomes the summary. In those cases, the summary might include newline characters which a renderer should honor when presenting the message visually to a user.

- `detail` (string): An optional additional message giving more detail about the problem.

  In Terraform's usual human-oriented diagnostic messages, the detail provides the paragraphs of text that appear after the heading and the source location reference.

  Detail messages are often multiple paragraphs and possibly interspersed with non-paragraph lines, so tools which aim to present detail messages to the user should distinguish between lines without leading spaces, treating them as paragraphs, and lines with leading spaces, treating them as preformatted text. Renderers should then soft-wrap the paragraphs to fit the width of the rendering container, but leave the preformatted lines unwrapped.

Some Terraform detail messages currently contain an approximation of bullet lists using ASCII characters to mark the bullets. This is not currently a contractural formatting convention and so renderers should avoid depending on it and should instead treat those lines as either paragraphs or preformatted text per the rules above. A future version of this format may define some additional rules for processing other text conventions, but will do so within the bounds of the rules above to achieve backward-compatibility.

- `range` (object): An optional object referencing a portion of the configuration source code that the diagnostic message relates to. For errors, this will typically indicate the bounds of the specific block header, attribute, or expression which was detected as invalid.

  A source range is an object with a property `filename` which gives the filename as a relative path from the current working directory, and then two properties `start` and `end` which are both themselves objects describing source positions, as described below.

  Not all diagnostic messages are connected with specific portions of the configuration, so `range` will be omitted or `null` for diagnostic messages where it isn't relevant.

- `snippet` (object): An optional object including an excerpt of the configuration source code that the diagnostic message relates to.

  The snippet information includes:

  - `context` (string): An optional summary of the root context of the diagnostic. For example, this might be the resource block containing the expression which triggered the diagnostic. For some diagnostics this information is not available, and then this property will be `null`.

  - `code` (string): A snippet of Terraform configuration including the source of the diagnostic. This can be multiple lines and may include additional configuration source code around the expression which triggered the diagnostic.

  - `start_line` (number): A one-based line count representing the position in the source file at which the `code` excerpt begins. This is not necessarily the same

value as `range.start.line`, as it is possible for `code` to include one or more lines of context before the source of the diagnostic.

  - `highlight_start_offset` (number): A zero-based character offset into the `code` string, pointing at the start of the expression which triggered the diagnostic.

  - `highlight_end_offset` (number): A zero-based character offset into the `code` string, pointing at the end of the expression which triggered the diagnostic.

  - `values` (array of objects): Contains zero or more expression values which may be useful in understanding the source of a diagnostic in a complex expression. These expression value objects are described below.

Source Position

A source position object, as used in the `range` property of a diagnostic object, has the following properties:

- `byte` (number): A zero-based byte offset into the indicated file.
- `line` (number): A one-based line count for the line containing the relevant position in the indicated file.
- `column` (number): A one-based count of *Unicode characters* from the start of the line indicated in `line`.

A `start` position is inclusive while an `end` position is exclusive. The exact positions used for particular error messages are intended for human interpretation only and subject to change in future versions of Terraform due either to improvements to the error reporting or changes in implementation details of the language parser/evaluator.

Expression Value

An expression value object gives additional information about a value which is part of the expression which triggered the diagnostic. This is especially useful when using `for_each` or similar constructs, in order to identify exactly which values are responsible for an error.

The object has two properties:

- `traversal` (string): An HCL-like traversal string, such as `var.instance_count`. Complex index key values may be elided, so this will not always be valid, parseable HCL. The contents of this string are intended to be human-readable and are subject to change in future versions of Terraform.
- `statement` (string): A short English-language fragment describing the value of the expression when the diagnostic was triggered. The contents of this string are intended to be human-readable and are subject to change in future versions of Terraform.