Gorgo Tasting OpenTofu and Terraform

September 08, 2026

Appendix B

Built-ins

Everything in this booklet came from one of two places: a provider, or OpenTofu itself. This appendix lists what OpenTofu itself provides, with no provider and no download: one resource type and one data source, the block types, the meta-arguments every resource accepts, the named values you can refer to, the operators, and the built-in functions. The function list is the commonly used subset; the full list is longer and lives in the documentation [1]. Every example below was evaluated with `tofu console`, and the results are shown as plain values; the console itself wraps typed collections as `tolist([...])`, `toset([...])`, or `tomap({...})`.

The built-in provider

The built-in provider needs no `required_providers` entry and defines two things [2]:

Name	What it is	Chapter
`terraform_data` resource	A placeholder resource. Arguments: `input` (any value, echoed back as the `output` attribute) and `triggers_replace` (a list; when any element changes, the resource is replaced). Attributes: `id`, `output` [3].	1
`terraform_remote_state` data source	Reads another configuration's state. Arguments: `backend` and `config` (the other configuration's backend settings), optional `workspace`. Attribute: `outputs`, a map of that configuration's outputs [4].	5

Block types

The top-level blocks a configuration is made of, and the nested blocks that only appear inside them:

Block	Purpose	Chapter
`terraform { }`	Settings for OpenTofu: `required_version`, `required_providers`, `backend` [5]	1, 2
`provider "NAME" { }`	Configure a provider [6]	2
`resource "TYPE" "NAME" { }`	A thing that should exist [7]	1
`data "TYPE" "NAME" { }`	A read-only lookup [8]	2
`variable "NAME" { }`	An input: `type`, `default`, `description`, `sensitive`, `validation` [9]	1

Block	Purpose	Chapter
`output "NAME" { }`	A return value: `value`, `description`, `sensitive`, `precondition` [10]	1
`locals { }`	Named expressions [11]	2
`module "NAME" { }`	Call a child module: `source`, `version`, its variables [12]	3
`check "NAME" { }`	A test that runs after plan and apply, holding `data` blocks and `assert` blocks [13]	2
`import { }`	Bring an existing object into state: `to`, `id` [14]	A
`moved { }`	Record a rename: `from`, `to` [15]	A
`removed { }`	Forget a resource without destroying it [7]	A
`provisioner "TYPE" { }`	Inside a resource: a command at creation (`local-exec`, `remote-exec`, `file`) [16]	1
`lifecycle { }`	Inside a resource: the meta-arguments below [17]	3
`dynamic "BLOCK" { }`	Inside a resource: generate nested blocks from a collection	4

Meta-arguments

Arguments OpenTofu understands on every `resource`, `data`, and (where noted) `module` block, regardless of provider:

Meta-argument	Meaning
`depends_on = [ ... ]`	Create after these, destroy before them; also on modules [18]
`count = N`	Create N instances, addressed `NAME[0]` and so on, with `count.index` inside [19]
`for_each = set or map`	One instance per element, addressed `NAME["key"]`, with `each.key` and `each.value` inside; also on modules [20]
`provider = NAME.ALIAS`	Use a non-default provider configuration [21]
`lifecycle { create_before_destroy = true }`	On replacement, create the new one first
`lifecycle { prevent_destroy = true }`	Fail any plan that would destroy this
`lifecycle { ignore_changes = [ ... ] }`	Do not plan updates for these arguments (drift you accept)
`lifecycle { replace_triggered_by = [ ... ] }`	Replace this when those resources change
`lifecycle { precondition { } }`	An assumption checked before the resource is planned
`lifecycle { postcondition { } }`	A guarantee checked after the resource is applied, with `self` available

`timeouts { }` looks like a meta-argument but is defined per resource type by its provider, so not every resource has one.

Named values

What an expression can refer to [22]:

Reference	What it is
`var.NAME`	An input variable
`local.NAME`	A local value

Reference	What it is
`TYPE.NAME` and `TYPE.NAME.ATTR`	A resource and its attributes
`data.TYPE.NAME.ATTR`	A data source result
`module.NAME.OUTPUT`	A child module's output
`self.ATTR`	The current resource, inside provisioners and postconditions
`each.key`, `each.value`	The current element inside a `for_each` resource or dynamic block
`count.index`	The current index inside a `count` resource
`path.module`, `path.root`, `path.cwd`	The module's directory, the root module's directory, the shell's directory
`terraform.workspace`	The current workspace name (`default` unless you use workspaces)

Operators and expressions

Syntax	Meaning
`+ - * / %`	Arithmetic; `/` is floating point, `%` is remainder [23]
`== != < <= > >=`	Comparison
`&& || !`	Logical and, or, not
`cond ? a : b`	Conditional [24]
`[for x in list : expr]`	For expression producing a list [25]
`{for k, v in map : k => expr}`	For expression producing a map
`[for x in list : expr if cond]`	For expression with a filter
`list[*].attr`	Splat: the attribute of every element [26]
`"a ${expr} b"`	String interpolation [27]
`"%{ if cond }yes%{ endif }"`	String directive; also `%{ for }`
`<<-EOT ... EOT`	Indented heredoc
`f(a, b)` and `f(list...)`	Function call; `...` spreads a list into arguments [28]
`list[0]`, `map["key"]`, `map.key`	Indexing and attribute access

Functions

Strings

Example	Result
`format("%s-%03d", "web", 7)`	`"web-007"`
`join("-", ["a", "b", "c"])`	`"a-b-c"`
`split(",", "a,b,c")`	`["a", "b", "c"]`
`replace("hello world", "world", "gorgo")`	`"hello gorgo"`
`trimspace("  hi  ")`	`"hi"`
`trim("xxhixx", "x")`	`"hi"`
`trimprefix("motd:ch2", "motd:")`	`"ch2"`
`trimsuffix("main.go", ".go")`	`"main"`
`lower("Hello")`, `upper("hello")`	`"hello"`, `"HELLO"`
`title("hello world")`	`"Hello World"`
`substr("abcdef", 2, 3)`	`"cde"`
`startswith("hello", "he")`	`true`
`endswith("main.go", ".go")`	`true`

Example	Result
`strcontains("hello", "ell")`	`true`
`regex("[0-9]+", "port 8080")`	`"8080"` (an error when nothing matches)
`regexall("[a-z]+", "a1b2")`	`["a", "b"]`
`indent(2, "a\nb")`	`"a\n  b"` (every line but the first)
`chomp("line\n")`	`"line"`
`length("hello")`	`5`

Collections

Example	Result
`length(["a", "b", "c"])`	`3`
`element(["a", "b", "c"], 4)`	`"b"` (the index wraps around)
`index(["a", "b", "c"], "b")`	`1`
`lookup({a = 1}, "b", 0)`	`0` (the default)
`keys({a = 1, b = 2})`	`["a", "b"]`
`values({a = 1, b = 2})`	`[1, 2]`
`merge({a = 1}, {b = 2})`	`{a = 1, b = 2}`
`concat([1, 2], [3])`	`[1, 2, 3]`
`flatten([[1, 2], [3]])`	`[1, 2, 3]`
`distinct([1, 1, 2])`	`[1, 2]`
`sort(["b", "a"])`	`["a", "b"]`
`reverse([1, 2, 3])`	`[3, 2, 1]`
`slice([1, 2, 3, 4], 1, 3)`	`[2, 3]`
`contains(["a", "b"], "a")`	`true`
`range(3)`	`[0, 1, 2]`
`zipmap(["a", "b"], [1, 2])`	`{a = 1, b = 2}`
`setproduct(["a", "b"], [1, 2])`	`[["a", 1], ["a", 2], ["b", 1], ["b", 2]]`
`setunion([1], [2])`	`[1, 2]` (as a set)
`one([42])`	`42` (an error for more than one element)
`coalesce("", "x")`	`"x"` (the first non-empty)
`coalescelist([], [1])`	`[1]`
`compact(["a", "", "b"])`	`["a", "b"]`
`alltrue([true, false])`	`false`
`anytrue([true, false])`	`true`
`sum([1, 2, 3])`	`6`

Numbers

Example	Result
`min(3, 1, 2), max(3, 1, 2)`	`1, 3`
`abs(-4)`	`4`
`ceil(1.2), floor(1.8)`	`2, 1`
`pow(2, 10)`	`1024`
`parseint("ff", 16)`	`255`

Types and errors

Example	Result
`tostring(42)`	`"42"`
`tonumber("42")`	`42`
`tobool("true")`	`true`
`tolist(toset(["b", "a", "b"]))`	`["a", "b"]`
`toset(["b", "a", "b"])`	`["a", "b"]` (as a set)
`tomap({a = 1})`	`{a = 1}`
`type(["a"])`	`tuple([string])`
`can(regex("[0-9]+", "abc"))`	`false` (the error became false)
`try(regex("[0-9]+", "abc"), "none")`	`"none"` (the first argument that does not error)
`sensitive("secret")`	`(sensitive value)`
`nonsensitive(sensitive("secret"))`	`"secret"`

Encoding

Example	Result
`jsonencode({a = 1})`	`"{\"a\":1}"`
`jsondecode("{\"a\": 1}")`	`{a = 1}`
`yamlencode({a = [1, 2]})`	`"\"a\":\n- 1\n- 2\n"`
`yamldecode("a: 1")`	`{a = 1}`
`base64encode("hi")`	`"aGk="`
`base64decode("aGk=")`	`"hi"`
`urlencode("a b&c")`	`"a+b%26c"`

Files and paths

Example	Result
`file("hello.txt")`	The file's contents as a string
`fileexists("hello.txt")`	`true`
`fileset("app", "*.go")`	`["main.go"]` (as a set)
`filesha1("hello.txt")`	`"f572d396..."`
`filesha256("hello.txt")`	`"5891b5b5..."`
`templatefile("greet.tftpl", { name = "Gorgo" })`	`"Hi Gorgo!\n"` for a template holding `Hi ${name}!`
`abspath("app")`	`"/home/you/webserver/app"`
`dirname("app/main.go")`	`"app"`
`basename("app/main.go")`	`"main.go"`
`pathexpand("~/.kube/config")`	`"/home/you/.kube/config"`

Hashes and identifiers

Example	Result
`sha1("hello")`	`"aaf4c61d..."`
`sha256("hello")`	`"2cf24dba..."`

Example	Result
`md5("hello")`	`"5d41402a..."`
`uuid()`	A new random universally unique identifier (UUID) on every evaluation

Dates

Example	Result
`timestamp()`	The current UTC time, RFC 3339, new on every evaluation
`formatdate("YYYY-MM-DD", "2026-09-05T10:30:00Z")`	`"2026-09-05"`
`timeadd("2026-09-05T10:30:00Z", "48h")`	`"2026-09-07T10:30:00Z"`

Networks

Example	Result
`cidrsubnet("10.0.0.0/16", 8, 1)`	`"10.0.1.0/24"`
`cidrhost("10.0.1.0/24", 5)`	`"10.0.1.5"`
`cidrnetmask("10.0.1.0/24")`	`"255.255.255.0"`
`cidrsubnets("10.0.0.0/16", 8, 8)`	`["10.0.0.0/24", "10.0.1.0/24"]`

Trap: `uuid()` and `timestamp()` give a new value on every plan, so a resource argument built from them changes on every run. Use them only where a changing value is the point, and reach for the `random` provider when you want a value generated once and kept.

[1] OpenTofu Project, "Built-in functions." 2026. Available: https://opentofu.org/docs/language/functions/

[2] OpenTofu Project, "Built-in provider." 2026. Available: https://opentofu.org/docs/language/providers/builtin/

[3] HashiCorp, "The terraform_data managed resource type." 2026. Available: https://developer.hashicorp.com/terraform/language/resources/terraform-data

[4] OpenTofu Project, "The terraform_remote_state data source." 2026. Available: https://opentofu.org/docs/language/state/remote-state-data/

[5] OpenTofu Project, "OpenTofu settings." 2026. Available: https://opentofu.org/docs/language/settings/

[6] OpenTofu Project, "Provider configuration." 2026. Available: https://opentofu.org/docs/language/providers/configuration/

[7] OpenTofu Project, "Resource blocks." 2026. Available: https://opentofu.org/docs/language/resources/syntax/

[8] OpenTofu Project, "Data sources." 2026. Available: https://opentofu.org/docs/language/data-sources/

[9] OpenTofu Project, "Input variables." 2026. Available: https://opentofu.org/docs/language/values/variables/

[10] OpenTofu Project, "Output values." 2026. Available: https://opentofu.org/docs/language/values/outputs/

[11] OpenTofu Project, "Local values." 2026. Available: https://opentofu.org/docs/language/values/locals/

[12] OpenTofu Project, "Module blocks." 2026. Available: https://opentofu.org/docs/language/modules/syntax/

[13] OpenTofu Project, "Checks with assertions." 2026. Available: https://opentofu.org/docs/language/checks/

[14] OpenTofu Project, "Import." 2026. Available: https://opentofu.org/docs/language/import/

[15] OpenTofu Project, "Refactoring." 2026. Available: https://opentofu.org/docs/language/modules/develop/refactoring/

[16] OpenTofu Project, "Provisioners." 2026. Available: https://opentofu.org/docs/language/resources/provisioners/syntax/

[17] OpenTofu Project, "The lifecycle meta-argument." 2026. Available: https://opentofu.org/docs/language/meta-arguments/lifecycle/

[18] OpenTofu Project, "The depends_on meta-argument." 2026. Available: https://opentofu.org/docs/language/meta-arguments/depends_on/

[19] OpenTofu Project, "The count meta-argument." 2026. Available: https://opentofu.org/docs/language/meta-arguments/count/

[20] OpenTofu Project, "The for_each meta-argument." 2026. Available: https://opentofu.org/docs/language/meta-arguments/for_each/

[21] OpenTofu Project, "The resource provider meta-argument." 2026. Available: https://opentofu.org/docs/language/meta-arguments/resource-provider/

[22] OpenTofu Project, "References to named values." 2026. Available: https://opentofu.org/docs/language/expressions/references/

[23] OpenTofu Project, "Arithmetic and logical operators." 2026. Available: https://opentofu.org/docs/language/expressions/operators/

[24] OpenTofu Project, "Conditional expressions." 2026. Available: https://opentofu.org/docs/language/expressions/conditionals/

[25] OpenTofu Project, "For expressions." 2026. Available: https://opentofu.org/docs/language/expressions/for/

[26] OpenTofu Project, "Splat expressions." 2026. Available: https://opentofu.org/docs/language/expressions/splat/

[27] OpenTofu Project, "Strings and templates." 2026. Available: https://opentofu.org/docs/language/expressions/strings/

[28] OpenTofu Project, "Function calls." 2026. Available: https://opentofu.org/docs/language/expressions/function-calls/