HCL-Kern

Bausteine

resource: verwaltetes Objekt (Create/Update/Delete durch Terraform). data: Read-only-Lookup bestehender Objekte. variable: Eingabe pro Root-/Child-Modul. locals: benannte Zwischenwerte, DRY fuer Ausdruecke. output: Rueckgabewerte, Schnittstelle zwischen Modulen und nach aussen.

Typen, Validation, sensitive

Typen explizit setzen: `string`, `number`, `bool`, `list(...)`, `map(...)`, `object({...})` – faengt Fehler beim Plan statt beim Apply.
validation-Block prueft Eingaben mit `condition` + `error_message`.
`sensitive = true` maskiert Werte im CLI-Output, entfernt sie aber nicht aus State oder Plan-Datei.

```
variable "env" {
  type = string
  validation {
    condition     = contains(["dev","prod"], var.env)
    error_message = "env muss dev oder prod sein."
  }
}
locals {
  name_prefix = "app-${var.env}"
}
output "lb_dns" {
  value       = aws_lb.this.dns_name
  description = "Public LB-Endpoint"
}
```

Provider & Versionen

Version-Pinning

`required_version` pinnt die CLI, `required_providers` pinnt Provider (`source` + `version`).
Pessimistic-Operator `~>` erlaubt nur Patch-/Minor-Updates innerhalb der Grenze.
`.terraform.lock.hcl` einchecken – exakte Provider-Versionen + Hashes fuer alle Umgebungen. `terraform init -upgrade` ist eine bewusste Aktion, kein Default.

Provider-Alias

Mehrere Regionen/Accounts: gleicher Provider mit `alias`, Zuordnung per `provider =` am Resource- bzw. `providers =`-Map am Modul-Block.

```
terraform {
  required_version = "≥ 1.10, < 2.0"
  required_providers {
    aws = { source = "hashicorp/aws",
            version = "~> 6.0" }
  }
}
provider "aws" { region = "eu-central-1" }
provider "aws" {
  alias  = "dr"
  region = "eu-west-1"
}
resource "aws_s3_bucket" "replica" {
  provider = aws.dr
  bucket   = "${local.name_prefix}-replica"
}
```

State

Remote Backend, Locking

State gehoert in ein Remote Backend (S3, azurerm, gcs, …): gemeinsamer Stand + Locking gegen parallele Applies.
S3: natives Locking via `use_lockfile = true` (seit 1.10, GA in 1.11); die DynamoDB-Argumente sind seit 1.11 deprecated.

```
terraform {
  backend "s3" {
    bucket       = "tf-state-prod"
    key          = "network/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true   # natives S3-Locking
    encrypt      = true
  }
}
```

State ist sensibel

Der State enthaelt Resource-Attribute im Klartext – auch Passwoerter und Keys. Zugriff wie auf ein Secret beschraenken, Verschluesselung at rest aktivieren.
Werte aus dem State heraushalten: ephemeral Values/Resources (1.10+) und write-only Attribute (1.11+) werden nicht persistiert.

State-Kommandos, Drift

`state list` / `state show`: Inventar inspizieren. `state mv` / `state rm`: imperative Chirurgie – fuer geplantes Refactoring besser `moved`/`removed`-Bloecke.
Drift: `plan -refresh-only` zeigt Abweichungen der Realitaet vom State, `apply -refresh-only` uebernimmt sie in den State (ohne Infrastruktur-Aenderung).

```
terraform state list
terraform state show aws_s3_bucket.logs
terraform state mv aws_iam_role.a aws_iam_role.b
terraform state rm aws_iam_role.orphan
terraform plan -refresh-only   # Drift-Report
```

Module

Struktur

Standard-Layout: `main.tf`, `variables.tf`, `outputs.tf` (+ `README`). Kleine, fachlich geschnittene Module; Komposition flach im Root-Modul. Tiefe Modul-in-Modul-Ketten machen Plans traege und Schnittstellen unklar.

Quellen, Versionierung

Registry-Quelle `<ns>/<name>/<provider>` + `version` (nur Registry-Quellen unterstuetzen `version`).
Git-Quelle ueber `?ref=` auf Tag/Commit pinnen – nie auf einen beweglichen Branch.

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 6.0"
  name    = local.name_prefix
  cidr    = "10.20.0.0/16"
}
module "baseline" {
  source =
     "git::https://git.example.com/tf/baseline.git?ref=v2.3.1"
}
```

Meta-Argumente

for_each, count, depends_on

`for_each` ueber Map/Set: stabile Keys als Adresse – Eintraege loeschen verschiebt keine Nachbarn.
`count`: nur fuer identische Kopien oder das An/Aus-Muster (`count = var.enabled ? 1 : 0`). Bei benannten Objekten drohen Index-Shifts.
`depends_on`: nur wenn die implizite Abhaengigkeit ueber Referenzen nicht ausreicht (z.B. IAM-Policy vor Nutzung).

lifecycle

`create_before_destroy`: Ersatz zuerst hochziehen. `prevent_destroy`: Plan schlaegt fehl, statt z.B. eine Datenbank zu loeschen. `ignore_changes`: einzelne Attribute extern verwalten lassen – chirurgisch einsetzen. `replace_triggered_by`: Ersatz an fremde Aenderung koppeln.

```
resource "aws_iam_user" "team" {
  for_each = toset(["ci", "audit", "ops"])
  name     = "svc-${each.key}"
}
resource "aws_db_instance" "main" {
  # ...
  lifecycle {
    prevent_destroy = true
    ignore_changes  = [password]
  }
}
```

dynamic blocks

Wiederholte Nested Blocks

`dynamic "<block>"` + `for_each` + `content` erzeugt wiederholte verschachtelte Bloecke; Iterator heisst wie das Block-Label. Sparsam einsetzen: verschachtelte dynamics sind schwer lesbar – oft ist die ausgeschriebene Liste klarer.

```
resource "aws_security_group" "web" {
  name = "${local.name_prefix}-web"
  dynamic "ingress" {
    for_each = var.service_ports   # z.B. {http=80,
     https=443}
    content {
      from_port   = ingress.value
      to_port     = ingress.value
      protocol    = "tcp"
      cidr_blocks = ["10.0.0.0/8"]
    }
  }
}
```

Plan / Apply / Destroy

Kern-Workflow

`fmt -check` + `validate` als schnelles Gate, dann `plan -out=tfplan` und `apply tfplan` – es wird exakt der reviewte Plan angewandt.
`plan -detailed-exitcode`: 0 = keine Aenderung, 2 = Aenderungen – sauber fuer Pipelines. `-replace=<addr>` erzwingt gezielte Neuerstellung. `destroy` nur fuer ganze (Wegwerf-)Umgebungen.

-target ist ein Notfall-Werkzeug

-target schneidet den Abhaengigkeitsgraphen ab – das Ergebnis entspricht nicht mehr der Konfiguration. Nur zur Incident-Reparatur, danach sofort ein voller Plan ohne **-target**, bis er leer ist.

```
terraform fmt -check -recursive
terraform validate
terraform plan -out=tfplan -detailed-exitcode
terraform apply tfplan
terraform apply -replace=aws_instance.worker[2]
```

Import & Refactoring

import-Block (1.5+)

Bestehende Infrastruktur deklarativ adoptieren: plannbar, reviewbar, im normalen Plan/Apply-Zyklus. **-generate-config-out=<datei>** erzeugt HCL fuer importierte Objekte ohne Config. **for_each** am import-Block seit 1.7.

moved / removed

moved (1.1+): Adresse geaendert (Rename, Modul-Umzug) – Terraform migriert den State beim Plan, kein **state mv**.
removed (1.7+): Objekt aus Config + State entfernen, ohne es zu zerstoeren (**destroy = false**).

```
import {
  to = aws_s3_bucket.logs
  id = "legacy-logs-bucket"
}
moved {
  from = aws_instance.app
  to   = module.compute.aws_instance.app
}
removed {
  from = aws_s3_bucket.archiv
  lifecycle { destroy = false }
}
```

Workspaces & Tests

Workspaces und ihre Grenzen

Ein Workspace = eigener State bei gleicher Config und gleichem Backend. Gut fuer kurzlebige Varianten (Feature-Test, Review-Umgebung).
Grenze: Prod/Non-Prod-Trennung – gleiche Credentials, gleicher State-Bucket, **terraform.workspace**-Verzweigungen im Code. Dafuer getrennte Root-Module + Backends verwenden.

```
terraform workspace new review-142
terraform workspace select review-142
terraform workspace list
# im Code: name = "app-${terraform.workspace}"
```

terraform test (1.6+)

Tests in ***.tftest.hcl**: **run**-Bloecke fuehren **command = plan** (schnell, ohne echte Ressourcen) oder **apply** aus und pruefen **assert**-Bedingungen. Seit 1.7: **mock_provider** + Overrides – Provider-Antworten simulieren statt echte Infrastruktur anzulegen.

```
# tests/naming.tftest.hcl
run "prod_prefix" {
  command = plan
  variables { env = "prod" }
  assert {
    condition     = local.name_prefix == "app-prod"
    error_message = "Prefix-Schema verletzt"
  }
}
```

CI/CD-Patterns

Plan-Artefakt, Approval

Build-Job: **plan -out=tfplan**, Plan-Artefakt + lesbaren **show**-Output an den Review haengen. Nach Approval wendet der Apply-Job dasselbe Artefakt an – kein zweiter Plan zwischen Review und Apply, keine unbemerkte Differenz.

Pipeline-Hygiene

init -input=false (keine interaktiven Prompts), **fmt -check** + **validate** als fruehes Gate, Lock-Datei committed.
State-Zugriff nur fuer die Pipeline-Identitaet; kurzlebige Cloud-Credentials statt statischer Keys.

```
terraform init -input=false
terraform plan -input=false -out=tfplan
terraform show -no-color tfplan > plan.txt  # Review
# --- nach Approval ---
terraform apply -input=false tfplan
```

Version & Lizenz

1.x-Release-Linie, BUSL

Aktuelle stabile Linie: 1.15 (Stand Juli 2026).
Seit 1.6 steht Terraform unter der Business Source License 1.1 (Change License: MPL 2.0 nach vier Jahren); die Einschraenkung zielt auf konkurrierende Angebote. Versionen bis 1.5.x bleiben MPL 2.0. OpenTofu ist ein MPL-2.0-lizenzierter Community-Fork auf Basis von Terraform 1.5.

Anti-Patterns

Was du nicht tun solltest

-target als Routine: zerschnittener Graph, Code und Realitaet divergieren – nur im Notfall.
State von Hand editieren: JSON-Chirurgie korrumpiert Serial/Lineage – **state**-Kommandos oder **moved**/**removed** nutzen.
count fuer benannte Objekte: Loeschen in der Mitte verschiebt Indizes – Nachbarn werden zerstoert/neu erzeugt. **for_each** nehmen.
Provider/Module unpinned: naechstes **init** zieht ein Major-Update in den Apply – pinnen + Lock-Datei committen.
Secrets als Klartext-Variable: landen in State und Plan-Datei – ephemeral/write-only nutzen, State-Zugriff beschraenken.
Workspaces als Prod-Trennung: ein Backend, eine Credential- Grenze fuer alles – getrennte Root-Module + Backends.
Apply ohne Plan-Artefakt in CI: zwischen Review und Apply entsteht ein anderer Plan – immer **apply tfplan**.
Monolith-Root-Modul: ein State fuer alles = maximaler Blast-Radius und traege Plans – nach Lebenszyklus/Team schneiden.
ignore_changes als Dauerpflaster: verdeckt Drift und echte Ownership-Fragen – klaeren, wer das Attribut verwaltet.

terraform-cheatsheet.de

Terraform Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

IaC-Betrieb & Terraform-Modernisierung

OMNI52™ hilft Plattform-Teams, Terraform fuer regulierte Enterprise-Umgebungen sauber aufzustellen.

State-Migration auf Remote-Backends mit Locking, Modul-Architektur mit Versionierung und Lock-Dateien, CI/CD mit Plan-Artefakt und Approval-Gates, Refactoring ueber moved/removed statt State-Chirurgie, Secrets-Handling ohne Klartext-State. Output landet als Git-Repo bei euch: Module, Pipelines, Runbooks. Euer Team operiert weiter, nicht wir.

OMNI52 GmbH, Ebern · istio-consulting.de · kostenloses 30-min Initial-Assessment

Aus der OMNI52 Cheatsheet-Fabrik

Verwandte Sheets: vault-cheatsheet.de (Secrets-Management) · flux-cheatsheet.de (GitOps mit Flux) · argocd-cheatsheet.de (GitOps mit Argo CD) · rancher-cheatsheet.de (Cluster-Management).

Lizenz & Weiterverteilung

Trademarks