Terraform Developer Toolchain

5W1H Hands-On Lab Manual

Student Edition

Write -> Validate -> Test -> Lint -> Secure -> Document -> Cost -> Automate -> Govern

Framework	What | Why | When | How to Install | How to Use | Top 5 Commands
Scope	20 Terraform developer tools + Terraform CLI/Git foundation
Primary goal	Learn the workflow by running small, safe exercises
Cloud safety	Core labs require no cloud credentials; risky AWS examples are scan-only
Validated baseline	5 September 2026

Rule for this manual: understand the tool, run the smallest useful exercise, verify the result, then move on.

How to Use This Lab Manual

GOAL	Each lab is deliberately small. Students should be able to explain the tool in one minute and demonstrate its core value with one exercise.

Recommended learning path

VS Code / Terraform Extension -> terraform-ls -> Terraform MCP -> tenv -> Terraform CLI
-> terraform test -> TFLint -> Trivy -> terraform-docs -> Infracost
-> pre-commit -> Git -> GitHub Actions / GitLab CI -> HCP Terraform
-> optional scale/governance tools: Terragrunt, Checkov, OPA/Conftest, Sentinel, Atlantis, Renovate, tfupdate

Lab rules

- Run commands from the lab workspace unless the lab says otherwise.
- Do not paste cloud access keys into Terraform files, shell history, Git, CI variables, or screenshots.
- Never run terraform apply inside security-demo/ or cost-demo/. Those folders exist only for scanning and cost exercises.
- For hosted tools, use a training/sandbox organization and follow instructor-provided access.
- If a command is unavailable, run <tool> --help and verify your installed version before assuming the lab is wrong.

Prerequisites

Requirement	Minimum	Check
Terminal + shell	bash/zsh/PowerShell	echo $SHELL
Git	Recent stable	git --version
VS Code	Recent stable	code --version
Homebrew	Recommended for this manual	brew --version
Docker	Optional: Terraform MCP lab	docker --version
GitHub/GitLab account	Optional: CI lab	Sign in
HCP Terraform account	Optional: HCP lab	Sign in

Foundation Lab - Build the Safe Student Workspace

WHY	Every later lab reuses this tiny project. It creates only local resources, so students can practice Terraform without an AWS/Azure/GCP account.

1. Create the folders

```
mkdir -p terraform-toolchain-lab/tests
mkdir -p terraform-toolchain-lab/security-demo
mkdir -p terraform-toolchain-lab/cost-demo
cd terraform-toolchain-lab
```

2. Create versions.tf

```
terraform {
  required_version = ">= 1.7.0, < 2.0.0"

  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "~> 3.7"
    }
    local = {
      source  = "hashicorp/local"
      version = "~> 2.5"
    }
  }
}
```

3. Create main.tf

```
variable "environment" {
  type    = string
  default = "dev"

  validation {
    condition     = contains(["dev", "staging", "prod"], var.environment)
    error_message = "environment must be dev, staging, or prod"
  }
}

locals {
  app_name = "training-${var.environment}"
}

resource "random_pet" "suffix" {
  length = 2
}

resource "local_file" "summary" {
  filename = "${path.module}/generated.txt"
  content  = "app=${local.app_name}\nsuffix=${random_pet.suffix.id}\n"
}

output "app_name" {
  value = local.app_name
}

output "generated_file" {
  value = local_file.summary.filename
}
```

4. Create tests/basic.tftest.hcl

```
mock_provider "random" {}
mock_provider "local" {}

run "naming_is_correct" {
  command = plan
```

```
  variables {
    environment = "staging"
  }

  assert {
    condition     = output.app_name == "training-staging"
    error_message = "Unexpected application name"
  }
}
```

PASS	The folder contains versions.tf, main.tf, and tests/basic.tftest.hcl. No cloud credentials are required.

Foundation - Terraform CLI + Git

5W1H Quick View

What?	Terraform CLI is the IaC execution engine; Git records and reviews code changes.
Why?	Every tool in this manual either improves Terraform authoring, checks Terraform, or automates its Git workflow.
When?	Use Terraform CLI throughout development; use Git for every team change.
Install?	Terraform is installed through tenv in Lab 08. Git is normally preinstalled or available through your OS package manager.
How?	Initialize, format, validate, plan, then apply only in the approved context. Commit source and .terraform.lock.hcl - never state files.

Terraform - Top 5 Commands

Command / Action	Purpose
terraform init	Download providers/modules and initialize the working directory.
terraform fmt -recursive	Apply canonical Terraform formatting.
terraform validate	Check configuration validity.
terraform plan	Preview proposed changes.
terraform apply	Execute an approved plan; in this manual only the local training project.

Git - Top 5 Commands

Command / Action	Purpose
git status	See changed/untracked files.
git diff	Review your changes before commit.
git add .	Stage intended changes.
git commit -m "..."	Create a reviewable change.
git push	Send the branch to the remote repository.

SECURITY Never commit *.tfstate, .terraform/, saved plan files, credentials, tokens, or sensitive *.tfvars.

Lab 01 - VS Code + HashiCorp Terraform Extension

Terraform-aware editing: syntax, completion, formatting, navigation and diagnostics.

Priority: 5/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	The official Terraform extension turns VS Code into a Terraform-aware editor and uses the Terraform language server.
Why?	It catches simple mistakes while you type and makes provider/module code easier to navigate.
When?	Use it whenever authoring or reviewing .tf, .tfvars, .tftest.hcl, Terraform Stacks/Search/Policy files.
Install?	VS Code Extensions -> search "HashiCorp Terraform", or run: code --install-extension HashiCorp.terraform
How?	Open the project folder, edit main.tf, use autocomplete, Format Document, Go to Definition, and the Problems panel.

Top 5 Commands / Actions

Command / Action	Purpose
code --install-extension HashiCorp.terraform	Install the official extension.
code .	Open the current Terraform project.
Format Document	Run terraform-aware formatting from the editor.
F12 - Go to Definition	Jump to variables, modules, resources, outputs.
Problems panel	Review syntax/diagnostic issues from the language server.

Student Exercise

1. Install the extension and open terraform-toolchain-lab as a folder.
2. Open main.tf and type "var." inside a temporary local expression; confirm autocomplete suggests environment.
3. Intentionally misalign an attribute, run Format Document, and confirm formatting changes.
4. Use F12 on var.environment or local.app_name.
5. Remove the temporary edit and save the file.

PASS - Expected: Terraform files have syntax highlighting and autocomplete; formatting and navigation work from the editor. Pass when: You can demonstrate autocomplete + Format Document + Go to Definition without using a terminal command.

Lab 02 - terraform-ls

Language Server Protocol backend behind Terraform IDE intelligence.

Priority: 5/5 **Lab mode:** Hands-on / background service **Outcome:** Run + verify

5W1H Quick View

What?	HashiCorp's Terraform language server provides completion, diagnostics, hover and navigation through LSP.
Why?	Editors need a project-aware process that understands Terraform configuration and schemas.
When?	Normally every time the Terraform editor extension is active.
Install?	VS Code normally bundles a compatible terraform-ls. Manual install is mainly for other LSP-capable editors: brew install hashicorp/tap/terraform-ls
How?	Usually let the editor start it automatically. Manual users run terraform-ls serve.

Top 5 Commands / Actions

Command / Action	Purpose
terraform-ls version	Show the installed language-server version.
terraform-ls serve	Start the LSP server manually.
which terraform-ls	Find the binary on macOS/Linux.
code .	Open a project so the extension can start/index the server.
View -> Output -> HashiCorp Terraform	Inspect language-server startup and diagnostics.

Student Exercise

1. In VS Code, open View -> Output and select HashiCorp Terraform.
2. Confirm the output shows a language server starting for the workspace.
3. If you installed terraform-ls separately, run terraform-ls version.
4. Open main.tf and confirm hover/completion/navigation still work.

PASS - Expected: The editor launches terraform-ls automatically; manual installation is not required for normal VS Code usage. Pass when: You can explain that terraform-ls is a background service, not a replacement for terraform CLI.

Lab 03 - TFLint

Lint Terraform and detect provider-specific mistakes, deprecated patterns and quality issues.

Priority: 5/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	TFLint is a pluggable Terraform linter with Terraform and cloud-provider rulesets.
Why?	terraform validate checks validity; TFLint catches additional suspicious or non-standard configuration.
When?	Run after validate, before commit, and again in CI.
Install?	macOS/Homebrew: brew install terraform-linters/tap/tflint
How?	Create .tflint.hcl, initialize plugins, then lint the project.

Top 5 Commands / Actions

Command / Action	Purpose
tflint --init	Download/initialize configured ruleset plugins.
tflint	Lint the current directory.
tflint --recursive	Lint Terraform directories recursively.
tflint --format=compact	Use concise CI-friendly output.
tflint --fix	Apply fixes for supported fixable findings; always review the diff.

Student Exercise

```
# .tflint.hcl
plugin "terraform" {
  enabled = true
  preset  = "recommended"
}

rule "terraform_documented_variables" {
  enabled = true
}

rule "terraform_documented_outputs" {
  enabled = true
}
```

1. Install TFLint and create the configuration shown below in the lab root.
2. Run tflint --init.
3. Run tflint.
4. Read every finding; do not blindly suppress warnings.
5. Run git diff after any automatic fix.

PASS - Expected: TFLint completes and reports either no findings or clear lint findings tied to Terraform source. Pass when: tflint --init succeeds and tflint runs against the lab directory.

Lab 04 - Trivy

Scan Terraform for IaC security misconfiguration and optionally repository secrets.

Priority: 5/5 **Lab mode:** Hands-on - scan only **Outcome:** Run + verify

5W1H Quick View

What?	Trivy is a unified security scanner; trivy config scans Terraform/IaC misconfiguration.
Why?	Valid Terraform can still create publicly exposed, unencrypted or overly permissive infrastructure.
When?	Run locally before commit and enforce selected severities in CI.
Install?	macOS/Homebrew: brew install trivy
How?	Scan source with trivy config. Use trivy fs when you also want secret scanning.

Top 5 Commands / Actions

Command / Action	Purpose
`trivy config .`	Scan IaC configuration recursively.
`trivy config --severity HIGH,CRITICAL .`	Focus on serious findings.
`trivy config --exit-code 1 .`	Return failure for detected findings - useful in CI.
`trivy config --tf-vars values.tfvars .`	Evaluate source using supplied Terraform variables.
`trivy fs --scanners misconfig,secret .`	Scan repository files for misconfigurations and secrets.

Student Exercise

CAUTION This file is intentionally insecure. It exists only to produce scanner findings. Never run terraform apply in security-demo/.

```
# security-demo/main.tf - TRAINING ONLY, DO NOT APPLY
resource "aws_security_group" "insecure_demo" {
  name = "training-insecure-demo"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

1. Create security-demo/main.tf from the sample below.
2. From the lab root run: trivy config security-demo.
3. Run again with --severity HIGH,CRITICAL.
4. Explain why an open SSH ingress rule is risky.
5. Do not apply the security-demo configuration.

PASS - Expected: Trivy flags the intentionally risky Terraform configuration or reports the security checks it evaluated. Pass when: You can identify the insecure setting and explain the remediation: restrict the CIDR/remove public SSH.

Lab 05 - terraform-docs

Generate accurate Terraform module input/output/provider documentation.

Priority: 5/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	terraform-docs generates documentation directly from Terraform module source.
Why?	Manually maintained input/output tables drift quickly as modules change.
When?	Use for reusable modules and run automatically before commit/CI.
Install?	macOS/Homebrew: brew install terraform-docs
How?	Add BEGIN_TF_DOCS / END_TF_DOCS markers to README.md and inject generated Markdown.

Top 5 Commands / Actions

Command / Action	Purpose
terraform-docs markdown table .	Print Markdown table documentation.
terraform-docs markdown table --output-file README.md --output-mode inject .	Inject generated docs into README.
terraform-docs markdown document .	Generate Markdown document format.
terraform-docs json .	Generate JSON metadata.
terraform-docs --output-check .	Check whether configured generated output is current.

Student Exercise

```
# Terraform Toolchain Lab

<!-- BEGIN_TF_DOCS -->
<!-- END_TF_DOCS -->
```

1. Create README.md with the marker block shown below.
2. Run the inject command.
3. Open README.md and locate Requirements, Providers, Inputs and Outputs.
4. Change the environment variable description in main.tf and regenerate docs.
5. Use git diff to see exactly what changed.

PASS - Expected: README.md contains an automatically generated Terraform reference between the markers. Pass when: Changing Terraform input/output metadata and rerunning terraform-docs updates README without manual table editing.

Lab 06 - pre-commit-terraform

Run Terraform quality/security/documentation checks automatically before Git commits.

Priority: 5/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	A collection of pre-commit hooks specialized for Terraform and related tools.
Why?	Developers get fast feedback before pushing broken or unformatted Terraform.
When?	Use on every Terraform repository; CI must still re-run authoritative checks.
Install?	Install pre-commit first: brew install pre-commit. Hooks are fetched from the repository config.
How?	Create .pre-commit-config.yaml, install the Git hook, run all files once.

Top 5 Commands / Actions

Command / Action	Purpose
pre-commit install	Install the Git pre-commit hook.
pre-commit run --all-files	Run every configured hook now.
pre-commit run terraform_fmt --all-files	Run one specific hook.
pre-commit autoupdate	Update hook revisions; review resulting changes.
pre-commit clean	Clear cached hook environments when troubleshooting.

Student Exercise

```
repos:
  - repo: https://github.com/antonbabenko/pre-commit-terraform
    rev: REV
    hooks:
      - id: terraform_fmt
      - id: terraform_validate
      - id: terraform_tflint
      - id: terraform_trivy
      - id: terraform_docs
```

1. Create the minimal config below.
2. Replace REV with the instructor-approved current pre-commit-terraform release.
3. Run pre-commit install.
4. Run pre-commit run --all-files.
5. Fix any failure and re-run until all hooks pass.

PASS - Expected: One command runs formatting, validation, TFLint, Trivy and terraform-docs in a repeatable local workflow. Pass when: pre-commit run --all-files finishes successfully after you resolve findings.

Lab 07 - Infracost

Show cost impact before Terraform changes are applied.

Priority: 4/5 **Lab mode:** Hands-on / login may be required **Outcome:** Run + verify

5W1H Quick View

What?	Infracost analyzes IaC and estimates cloud cost plus FinOps/policy findings.
Why?	A technically correct plan can still introduce unexpected monthly spend.
When?	Use during development and PR review for cost-bearing resources.
Install?	macOS/Homebrew: brew install infracost
How?	Use the current CLI workflow: setup/auth -> scan -> inspect; price is useful for one-off snippets.

Top 5 Commands / Actions

Command / Action	Purpose
infracost setup	Interactive first-time configuration.
infracost auth login	Authenticate the CLI.
infracost scan ./cost-demo	Scan IaC and calculate costs/policy results.
infracost inspect --summary	Summarize results from the latest scan.
infracost price < cost-demo/main.tf	Quickly price Terraform piped on stdin.

Student Exercise

CAUTION Do not apply cost-demo/. The AMI is only a placeholder and the example exists to demonstrate cost feedback.

```
# cost-demo/main.tf - COST ANALYSIS ONLY, DO NOT APPLY
resource "aws_instance" "training_cost" {
  ami           = "ami-00000000000000000"
  instance_type = "m5.4xlarge"
}
```

1. Create cost-demo/main.tf from the sample below.
2. Authenticate/configure Infracost if your training environment requires it.
3. Run infracost scan ./cost-demo.
4. Run infracost inspect --summary and identify monthly cost.
5. Change instance_type to a smaller type and scan again. Compare the result.

PASS - Expected: Students can see a price/cost estimate before any infrastructure is created. Pass when: You can explain the cost difference between two instance sizes without running terraform apply.

Lab 08 - tenv

Manage Terraform, OpenTofu and Terragrunt versions per project.

Priority: 4/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	tenv is a version manager for Terraform/OpenTofu/Terragrunt and related IaC tools.
Why?	Different projects may require different Terraform versions. One global binary causes version drift.
When?	Use whenever teams maintain multiple Terraform repositories or CI must reproduce local versions.
Install?	macOS/Homebrew: brew install tofuutils/tap/tenv
How?	Install a compatible Terraform version, select it, and let tenv detect project requirements.

Top 5 Commands / Actions

Command / Action	Purpose
tenv tf install latest-allowed	Install the newest Terraform allowed by required_version.
tenv tf use latest-allowed	Select an allowed installed version.
tenv tf detect	Show which Terraform version the project resolves to.
tenv tf list	List installed Terraform versions.
tenv tf list-remote --stable	List stable remote versions available to install.

Student Exercise

1. Install tenv.
2. From the lab root run tenv tf install latest-allowed.
3. Run tenv tf use latest-allowed.
4. Run terraform version.
5. Run tenv tf detect and compare the resolved version with versions.tf.

PASS - Expected: Terraform runs with a version compatible with the project required_version constraint. Pass when: tenv tf detect and terraform version agree on an allowed Terraform release.

Lab 09 - Terragrunt

Reduce repeated configuration and orchestrate larger multi-environment Terraform layouts.

Priority: 4/5 **Lab mode:** Hands-on - optional architecture tool **Outcome:** Run + verify

5W1H Quick View

What?	Terragrunt is a thin orchestration/configuration layer around Terraform/OpenTofu.
Why?	It can reduce repeated backend/provider/environment configuration and coordinate multiple units.
When?	Use when repetition/orchestration is a real problem across many accounts, regions or environments.
Install?	macOS/Homebrew: brew install terragrunt
How?	Create terragrunt.hcl that points at a Terraform module/source and supplies environment-specific inputs.

Top 5 Commands / Actions

Command / Action	Purpose
terragrunt plan	Plan one Terragrunt unit.
terragrunt apply	Apply one unit; use only in approved environments.
terragrunt run --all plan	Plan all discovered units using current Terragrunt 1.x style.
terragrunt run --all validate	Validate all units.
terragrunt hcl fmt	Format Terragrunt HCL.

Student Exercise

```
# terragrunt-demo/dev/terragrunt.hcl
terraform {
  source = "../.."
}

inputs = {
  environment = "dev"
}
```

1. Create terragrunt-demo/dev/terragrunt.hcl as shown below.
2. Run terragrunt hcl fmt inside terragrunt-demo/dev.
3. Run terragrunt plan.
4. Identify the environment input passed to the Terraform root module.
5. Do not add Terragrunt to a real repository merely because this lab demonstrates it.

PASS - Expected: Terragrunt invokes the Terraform source and passes environment="dev". Pass when: You can explain one real problem Terragrunt solves and one reason a simple Terraform-only repo may not need it.

Lab 10 - terraform test

Native Terraform tests using .tftest.hcl, assertions and mocked providers.

Priority: 4/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	terraform test is Terraform's native test runner for modules and root configurations.
Why?	validate proves configuration validity; tests prove intended behavior through assertions.
When?	Use for module contracts, input validation, outputs and planned configuration behavior.
Install?	No separate install - it is part of modern Terraform CLI.
How?	Put *.tftest.hcl in tests/, define run blocks/assertions, then run terraform test.

Top 5 Commands / Actions

Command / Action	Purpose
terraform test	Run all Terraform tests.
terraform test -verbose	Show plan/state details for each run.
terraform test -filter=tests/basic.tftest.hcl	Run one test file.
terraform test -json	Emit machine-readable test output.
terraform test -junit-xml=test-results.xml	Write a JUnit XML report for CI systems.

Student Exercise

1. Confirm tests/basic.tftest.hcl exists from the foundation setup.
2. Run terraform init.
3. Run terraform test.
4. Change the expected value in the assertion so the test fails; read the failure.
5. Restore the correct assertion and rerun until PASS.

PASS - Expected: The naming_is_correct test passes with environment=staging and expected output training-staging. Pass when: You can intentionally make the test fail, explain why, fix it, and return to a passing test suite.

Lab 11 - terraform console

Interactively test Terraform expressions, collections, functions and CIDR calculations.

Priority: 4/5 **Lab mode:** Hands-on **Outcome:** Run + verify

5W1H Quick View

What?	An interactive expression console using the current Terraform configuration context.
Why?	It is faster than editing/applying code just to test an expression or function.
When?	Use while developing locals, for expressions, maps/lists, conditionals and CIDR math.
Install?	No separate install - included with Terraform CLI.
How?	Run terraform console, evaluate expressions, then type exit or press Ctrl-D.

Top 5 Commands / Actions

Command / Action	Purpose
`terraform console`	Start the interactive console.
`upper("prod")`	Test a string function.
`contains(["dev","prod"], "prod")`	Test collection membership.
`[for x in ["api","worker"] : "prod-${x}"]`	Test a for-expression.
`cidrsubnet("10.0.0.0/16", 8, 1)`	Calculate a subnet CIDR.

Student Exercise

1. Run terraform console in the lab root.
2. Evaluate local.app_name.
3. Run each of the four example expressions above.
4. Predict each result before pressing Enter.
5. Exit the console.

PASS - Expected: Students can prototype Terraform language expressions without changing resources. Pass when: You can explain at least one expression result and recreate it in Terraform code if needed.

Lab 12 - Checkov

IaC security/compliance scanning and Terraform plan-policy analysis.

Priority: 4/5	**Lab mode:** Hands-on - optional alternative/complement to Trivy	**Outcome:** Run + verify

5W1H Quick View

What?	Checkov statically scans Terraform and other IaC against security/compliance policies.
Why?	It gives broad policy coverage and can also evaluate Terraform plan JSON.
When?	Use when your organization standardizes on Checkov/Prisma policies or needs policy coverage not provided by your primary scanner.
Install?	Simple Python isolation: pipx install checkov. Homebrew is also available.
How?	Scan a directory or file, filter frameworks/checks, and optionally scan plan JSON in trusted CI.

Top 5 Commands / Actions

Command / Action	Purpose
`checkov -d security-demo`	Scan a directory.
`checkov -f security-demo/main.tf`	Scan one file.
`checkov -d . --framework terraform`	Limit scan to Terraform.
`checkov -l`	List available checks.
`checkov -f tfplan.json`	Scan Terraform plan JSON.

Student Exercise

1. Install Checkov.
2. Run checkov -d security-demo.
3. Compare its findings with Trivy from Lab 04.
4. Identify overlapping findings and any unique checks.
5. Decide whether your hypothetical team needs Trivy, Checkov, or both - and state one reason.

PASS - Expected: Checkov reports policy results against the intentionally insecure sample. Pass when: You can explain why running two scanners with identical findings may add noise rather than value.

Lab 13 - Conftest / OPA

Write custom vendor-neutral policy-as-code checks with Rego.

Priority: 3/5 **Lab mode:** Hands-on - policy fundamentals **Outcome:** Run + verify

5W1H Quick View

What?	OPA is a general policy engine; Conftest is a convenient CLI for applying Rego policies to configuration files.
Why?	Built-in scanner rules cannot express every organization-specific rule.
When?	Use for custom rules such as approved environments, regions, tags, resource classes and network policies.
Install?	macOS/Homebrew: brew install opa conftest
How?	Write a Rego deny rule, test an input with conftest, and unit-test policy logic with OPA/Conftest.

Top 5 Commands / Actions

Command / Action	Purpose
conftest test input.json	Evaluate input against policy/.
conftest test -p policy input.json	Use an explicit policy directory.
conftest verify --policy policy	Run Rego policy unit tests.
opa eval -d policy -i input.json "data.training.deny"	Evaluate a Rego decision directly.
opa test policy	Run OPA test_* rules.

Student Exercise

```
# policy/input.json
{"environment":"prod","public_cidr":"0.0.0.0/0"}

# policy/network.rego
package training

deny contains msg if {
  input.environment == "prod"
  input.public_cidr == "0.0.0.0/0"
  msg := "Production must not allow a public CIDR"
}
```

1. Create policy/input.json and policy/network.rego using the sample below.
2. Run conftest test -p policy policy/input.json.
3. Change public_cidr to 10.0.0.0/8 and rerun.
4. Explain why the first input should fail and the second should pass.
5. Relate the rule to a Terraform plan-policy gate.

PASS - Expected: The policy denies a production configuration with public CIDR 0.0.0.0/0 and passes the restricted CIDR. Pass when: You can change only input data and observe policy behavior without changing the Rego rule.

Lab 14 - Sentinel

HashiCorp policy-as-code for HCP Terraform / Terraform Enterprise governance.

Priority: 3/5 **Lab mode:** Local CLI + guided HCP integration **Outcome:** Run + verify

5W1H Quick View

What?	Sentinel is HashiCorp's policy-as-code language/runtime integrated with HCP Terraform/TFE.
Why?	It can enforce organization-wide rules between plan and apply.
When?	Use when your HCP/TFE governance model standardizes on Sentinel policy sets.
Install?	Install the Sentinel CLI from HashiCorp releases; verify with sentinel --help.
How?	Develop/format/test policy locally, then publish it through the organization policy workflow.

Top 5 Commands / Actions

Command / Action	Purpose
sentinel --help	List CLI commands.
sentinel version	Show runtime version.
sentinel fmt policy.sentinel	Format Sentinel source.
sentinel apply policy.sentinel	Evaluate a policy locally.
sentinel test	Run local Sentinel policy tests.

Student Exercise

```
# policy.sentinel
main = rule {
  true
}
```

1. Create policy.sentinel from the minimal example below.
2. Run sentinel fmt policy.sentinel.
3. Run sentinel apply policy.sentinel and confirm success.
4. Change true to false and confirm the policy fails.
5. Restore the passing policy. Instructor demonstrates how a real plan policy set is attached in HCP Terraform/TFE.

PASS - Expected: Students understand the local Sentinel development loop and that HCP/TFE policy integration is a separate governance layer. Pass when: You can make a policy pass/fail locally and explain advisory versus blocking enforcement conceptually.

Lab 15 - HCP Terraform

Managed remote state, remote runs, collaboration, registry, policy and governance.

Priority: 5/5 **Lab mode:** Guided / HCP account required **Outcome:** Run + verify

5W1H Quick View

What?	HashiCorp's managed Terraform platform for state, runs, workspaces/projects, variables, registry and governance.
Why?	Teams need centralized state, locking, repeatable execution, RBAC and audit instead of laptop-owned production workflows.
When?	Use for team/enterprise Terraform when managed remote execution and governance fit the operating model.
Install?	No separate HCP client is required; use Terraform CLI and an HCP Terraform account.
How?	terraform login, add a cloud block, terraform init, then plan/apply through the configured HCP workspace.

Top 5 Commands / Actions

Command / Action	Purpose
terraform login	Authenticate interactively to app.terraform.io.
terraform init	Connect the local configuration to HCP Terraform settings.
terraform plan	Start/view a remote plan in CLI-driven workflow.
terraform apply	Start an approved remote apply.
HCP UI: Runs / States / Variables	Inspect centralized run, state and workspace context.

Student Exercise

CAUTION Use only an instructor-provided sandbox workspace. Do not point this lab at production state or credentials.

```
terraform {
 cloud {
  organization = "YOUR_TRAINING_ORG"

  workspaces {
   name = "terraform-toolchain-lab-YOURNAME"
  }
 }
}
```

1. Create a training workspace in the instructor-provided HCP organization.
2. Run terraform login.
3. Add cloud.tf using the example below and your assigned organization/workspace.
4. Run terraform init, then terraform plan.
5. Open the HCP workspace and locate the run, variables and state areas. Apply only if the instructor explicitly enables it.

PASS - Expected: The CLI is connected to an HCP workspace and a plan is visible in both terminal and HCP UI. Pass when: You can explain why HCP Terraform workspaces are deployment/state/RBAC boundaries, not merely local CLI workspaces.

Lab 16 - GitHub Actions / GitLab CI

Automate Terraform validation, testing and security checks on every change.

Priority: 5/5 **Lab mode:** Hands-on with GitHub; GitLab equivalent **Outcome:** Run + verify

5W1H Quick View

What?	Hosted CI engines that run repeatable checks from repository events.
Why?	Local hooks can be skipped; CI becomes the authoritative, reviewable quality gate.
When?	Use for every team Terraform repository, especially before plan/apply.
Install?	No runner install is needed for basic hosted GitHub/GitLab CI. Add workflow YAML to the repository.
How?	Trigger on pull requests, install pinned Terraform, run fmt/validate/test/security, then add OIDC-based plan/apply jobs when cloud access is needed.

Top 5 Commands / Actions

Command / Action	Purpose
git push	Trigger CI for the pushed branch.
gh workflow list	List GitHub Actions workflows.
gh workflow run <workflow>	Manually dispatch a supported workflow.
gh run list	List workflow runs.
gh run view <id> --log	Read a run log from GitHub CLI.

Student Exercise

```
on: [pull_request]
jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
      - run: terraform fmt -check -recursive
      - run: terraform init -backend=false -input=false
      - run: terraform validate
      - run: terraform test
```

1. Create .github/workflows/terraform-ci.yml using the sample below.
2. Commit and push to a training repository.
3. Open a pull request.
4. Confirm fmt, validate and test execute in CI. For real cloud plans, use OIDC/workload identity - never static cloud keys.

PASS - Expected: The pull request receives an automated Terraform quality result without any cloud secret in the workflow. Pass when: CI fails when formatting/tests fail and passes after the issue is fixed.

Lab 17 - Atlantis

PR-comment-driven Terraform plans/applies with project locking and approval workflow.

Priority: 4/5 **Lab mode:** Guided - shared Atlantis server required **Outcome:** Run + verify

5W1H Quick View

What?	Atlantis listens to pull requests and runs Terraform plan/apply based on comments and repository configuration.
Why?	It makes Terraform execution visible and conversational inside the pull request.
When?	Use when your operating model wants PR-driven Terraform and you are prepared to run/secure Atlantis.
Install?	Students do not need to install a server for this lab; use an instructor-provided Atlantis sandbox.
How?	Open/change a PR, let autoplan run or comment atlantis plan, obtain approval, then atlantis apply if permitted.

Top 5 Commands / Actions

Command / Action	Purpose
atlantis help	Show supported PR commands.
atlantis plan	Plan modified projects.
atlantis plan -p project1	Plan one configured project.
atlantis apply	Apply available approved plans.
atlantis unlock	Remove Atlantis locks/discard plans for the PR.

Student Exercise

CAUTION Atlantis apply is a real infrastructure action. In this lab it is allowed only in the instructor-provided sandbox.

1. Open the instructor-provided training PR.
2. Comment: atlantis plan.
3. Read the returned Terraform plan.
4. Observe the PR/project lock behavior.
5. Only if the instructor has configured a safe sandbox and approval requirement, run atlantis apply.

PASS - Expected: Plan output is posted back to the PR and is tied to the PR/project context. Pass when: You can explain how Atlantis differs from generic CI and from HCP Terraform.

Lab 18 - Renovate

Automatically create dependency-update PRs for Terraform, providers, modules and CI actions.

Priority: 4/5 **Lab mode:** Hands-on config + guided bot **Outcome:** Run + verify

5W1H Quick View

What?	Renovate is a dependency update automation bot with Terraform-aware managers.
Why?	Small, continuous version updates are safer than large, infrequent upgrade jumps.
When?	Use on actively maintained Terraform repositories with CI/review gates.
Install?	For local config validation use Node/npm (npx). For production, install the Renovate GitHub/GitLab app or self-host the bot.
How?	Create renovate.json, validate it, enable the bot, then review dependency dashboard/update PRs.

Top 5 Commands / Actions

Command / Action	Purpose
npx renovate-config-validator	Validate Renovate configuration.
npx renovate --help	Show self-hosted CLI options.
Dependency Dashboard	See pending/blocked dependency updates.
Update PR	Review Terraform/provider/module/action changes.
Automerge rule	Automatically merge only low-risk updates after required checks.

Student Exercise

```
{
  "extends": ["config:recommended"],
  "labels": ["dependencies"],
  "packageRules": [
    {
      "matchUpdateTypes": ["major"],
      "dependencyDashboardApproval": true
    }
  ]
}
```

1. Create renovate.json using the sample below.
2. Run npx renovate-config-validator.
3. Fix any validation errors.
4. In a training repository with Renovate enabled, locate the Dependency Dashboard or an update PR.
5. Explain why major provider/Terraform upgrades should usually require explicit review.

PASS - Expected: The configuration validates and students understand how dependency-update PRs enter the normal CI/review workflow. Pass when: You can identify which Terraform dependencies Renovate can keep current and state an appropriate major-upgrade policy.

Lab 19 - tfupdate

Update Terraform/provider/module version constraints from the command line.

Priority: 3/5 **Lab mode:** Hands-on - niche helper **Outcome:** Run + verify

5W1H Quick View

What?	tfupdate rewrites Terraform/OpenTofu core, provider and module version constraints and can update lock files.
Why?	It is useful for scripted bulk version edits across many Terraform directories.
When?	Use for focused command-line upgrade automation; Renovate is usually broader for continuous dependency PRs.
Install?	macOS/Homebrew: brew install minamijoyo/tfupdate/tfupdate
How?	Point tfupdate at a file/directory and choose the dependency/version constraint to rewrite.

Top 5 Commands / Actions

Command / Action	Purpose
`tfupdate --help`	List commands.
`tfupdate terraform -v "~> 1.16.0" versions.tf`	Update Terraform required_version.
`tfupdate provider random -v "~> 3.7" versions.tf`	Update a provider constraint.
`tfupdate module -v <ver> <source> <path>`	Update a module constraint.
`tfupdate lock .`	Update dependency lock information where applicable.

Student Exercise

1. Copy versions.tf to versions.tf.bak.
2. Run tfupdate terraform with an instructor-selected compatible constraint.
3. Run git diff versions.tf.
4. Run terraform init -upgrade only if the instructor asks you to test dependency resolution.
5. Restore versions.tf if the change was only for demonstration.

PASS - Expected: tfupdate changes only the intended version constraint and the diff is easy to review. Pass when: You can explain when tfupdate is useful and why Renovate is normally the broader long-term dependency workflow.

Lab 20 - Terraform MCP Server

Give AI coding agents live Terraform Registry/provider/module/policy context.

Priority: 5/5 **Lab mode:** Hands-on with Docker + AI client **Outcome:** Run + verify

5W1H Quick View

What?	HashiCorp's MCP server exposes current Terraform Registry information to MCP-capable AI tools and can integrate with HCP Terraform/TFE.
Why?	AI models can hallucinate or use outdated provider/module attributes; MCP grounds them in current Terraform data.
When?	Use when approved AI coding assistants generate or explain Terraform.
Install?	HashiCorp recommends Docker for getting started: hashicorp/terraform-mcp-server. A precompiled binary or Go install is also available.
How?	Configure the AI client to start the server over stdio, then ask Terraform questions that require current Registry context.

Top 5 Commands / Actions

Command / Action	Purpose
`docker pull hashicorp/terraform-mcp-server`	Fetch the local MCP server image.
`docker run -i --rm hashicorp/terraform-mcp-server`	Run the container over stdio for a client.
`go install github.com/hashicorp/terraform-mcp-server/cmd/terraform-mcp-server@latest`	Alternative source install.
`which terraform-mcp-server`	Find the locally installed binary.
`terraform-mcp-server stdio`	Run the binary in stdio transport mode.

Student Exercise

```
{
  "mcpServers": {
    "terraform": {
      "command": "docker",
      "args": ["run", "-i", "--rm", "hashicorp/terraform-mcp-server"]
    }
  }
}
```

1. Confirm Docker is available.
2. Pull the Terraform MCP server image.
3. Add the minimal MCP configuration below to your instructor-approved MCP client.
4. Restart/refresh the client and confirm the terraform MCP server is connected.
5. Ask: "Using current Registry information, explain the required arguments for a Terraform provider resource selected by the instructor."

PASS - Expected: The AI client can call Terraform MCP tools and ground answers in current Registry information. Pass when: You can explain that MCP improves context but does not replace fmt, validate, lint, security scan, test, plan or human review.

Capstone - Run the Complete Local Quality Chain

OBJECTIVE Use the tools as one workflow rather than isolated utilities. This is the key lesson of the course.

Run from the lab root

```
tenv tf detect
terraform fmt -recursive
terraform init
terraform validate
terraform test
tflint --init
tflint
trivy config .
terraform-docs markdown table --output-file README.md --output-mode inject .
infracost scan ./cost-demo
infracost inspect --summary
pre-commit run --all-files
git status
git diff
```

Student explanation challenge

Stage	Question	Expected idea
Format	What does fmt prove?	Only canonical formatting.
Validate	What does validate prove?	Terraform configuration validity, not security or cloud success.
Lint	Why TFLint after validate?	Additional Terraform/provider quality checks.
Security	Why Trivy/Checkov?	Valid code can still be insecure.
Test	Why terraform test?	Assert intended behavior/contracts.
Docs	Why generate docs?	Keep module interface reference synchronized.
Cost	Why Infracost?	Review spend before deployment.
Automation	Why pre-commit + CI?	Fast local feedback + authoritative central enforcement.
Governance	Why HCP/Policy?	Controlled state/runs/RBAC/policy for teams.

Student Completion Checklist

PASS RULE	A lab is complete only when the student can both run the exercise and explain the tool's place in the workflow.

#	Tool	Evidence	Pass
1	VS Code + HashiCorp Terraform Extension	You can demonstrate autocomplete + Format Document + Go to Definition without using a terminal command.	☐
2	terraform-ls	You can explain that terraform-ls is a background service, not a replacement for terraform CLI.	☐
3	TFLint	tflint --init succeeds and tflint runs against the lab directory.	☐
4	Trivy	You can identify the insecure setting and explain the remediation: restrict the CIDR/remove public SSH.	☐
5	terraform-docs	Changing Terraform input/output metadata and rerunning terraform-docs updates README without manual table editing.	☐
6	pre-commit-terraform	pre-commit run --all-files finishes successfully after you resolve findings.	☐
7	Infracost	You can explain the cost difference between two instance sizes without running terraform apply.	☐
8	tenv	tenv tf detect and terraform version agree on an allowed Terraform release.	☐
9	Terragrunt	You can explain one real problem Terragrunt solves and one reason a simple Terraform-only repo may not need it.	☐
10	terraform test	You can intentionally make the test fail, explain why, fix it, and return to a passing test suite.	☐
11	terraform console	You can explain at least one expression result and recreate it in Terraform code if needed.	☐
12	Checkov	You can explain why running two scanners with identical findings may add noise rather than value.	☐

Student Completion Checklist - Continued

#	Tool	Evidence	Pass
13	Conftest / OPA	You can change only input data and observe policy behavior without changing the Rego rule.	☐
14	Sentinel	You can make a policy pass/fail locally and explain advisory versus blocking enforcement conceptually.	☐
15	HCP Terraform	You can explain why HCP Terraform workspaces are deployment/state/RBAC boundaries, not merely local CLI workspaces.	☐
16	GitHub Actions / GitLab CI	CI fails when formatting/tests fail and passes after the issue is fixed.	☐
17	Atlantis	You can explain how Atlantis differs from generic CI and from HCP Terraform.	☐
18	Renovate	You can identify which Terraform dependencies Renovate can keep current and state an appropriate major-upgrade policy.	☐
19	tfupdate	You can explain when tfupdate is useful and why Renovate is normally the broader long-term dependency workflow.	☐
20	Terraform MCP Server	You can explain that MCP improves context but does not replace fmt, validate, lint, security scan, test, plan or human review.	☐

One-Page Tool Decision Guide

Question	Primary tool	Companion	Do not confuse with
Is the code formatted?	terraform fmt	pre-commit	Validation
Is Terraform configuration valid?	terraform validate	CI	Security scanning
Any lint/provider quality issue?	TFLint	terraform validate	Security policy
Any IaC security issue?	Trivy	Checkov if needed	Linting
Does the module behave as intended?	terraform test	Terratest later if needed	validate
Are docs current?	terraform-docs	pre-commit	Manual interface tables
What will it cost?	Infracost	FinOps review	Cloud invoice
Which Terraform version?	tenv	required_version	Provider version
Need multi-env orchestration?	Terragrunt if justified	HCP Terraform	Mandatory Terraform layer
Need custom policy?	OPA/Conftest or Sentinel	HCP policy framework	Security scanner only
Need centralized runs/state?	HCP Terraform	CI	Local-only workflow
Need PR-comment Terraform?	Atlantis	CI	HCP Terraform platform
Need dependency updates?	Renovate	tfupdate	terraform init -upgrade everywhere
Need current AI Terraform context?	Terraform MCP Server	Registry docs	Blind AI generation

Official Reference Sources

The command examples and recommendations in this manual were checked against current official/vendor documentation on 5 September 2026. Always verify the installed tool version with --version / --help in long-lived training material.

- **Terraform CLI and test:** https://developer.hashicorp.com/terraform/
- **VS Code Terraform Extension:** https://marketplace.visualstudio.com/items?itemName=HashiCorp.terraform
- **terraform-ls:** https://github.com/hashicorp/terraform-ls
- **Terraform MCP Server:** https://developer.hashicorp.com/terraform/mcp-server
- **tenv:** https://github.com/tofuutils/tenv
- **TFLint:** https://github.com/terraform-linters/tflint
- **Trivy Terraform scanning:** https://trivy.dev/docs/
- **terraform-docs:** https://terraform-docs.io/
- **pre-commit-terraform:** https://github.com/antonbabenko/pre-commit-terraform
- **Infracost:** https://www.infracost.io/docs/
- **Terragrunt:** https://terragrunt.gruntwork.io/
- **Checkov:** https://www.checkov.io/
- **OPA:** https://www.openpolicyagent.org/docs/
- **Conftest:** https://www.conftest.dev/
- **Sentinel:** https://developer.hashicorp.com/sentinel
- **HCP Terraform:** https://developer.hashicorp.com/terraform/cloud-docs
- **Atlantis:** https://www.runatlantis.io/
- **Renovate:** https://docs.renovatebot.com/
- **tfupdate:** https://github.com/minamijoyo/tfupdate

FINAL	The goal is not to install every tool in production. The goal is to know which problem each tool solves, avoid duplicate tooling, and build a clear Terraform engineering workflow.