



From Terraform Chaos to Terragrunt Order

Ceyda Düzgeç



Ceyda Düzgeç

Solution Architect | Platform Team



■■ About Me

Education:

- MS in Software Engineering @Boğaziçi
- BSc in Computer Science & Engineering @Sabancı

Certifications:

- AWS Solutions Architect Associate
- HashiCorp Terraform Associate

My Experience with Terragrunt:

- Working with multiple production level projects on Cloud with Terragrunt

Open Source:

- Maintainer of github.com/Hipo/university-domains-list



[More on LinkedIn](#)

Agenda

- 01 The Chaos: Pure Terraform Problems
- 02 The Orchestrator: What is Terragrunt?
- 03 The Order: 4 Key Solutions
- 04 The Result: Minimalistic Example
- 05 Q&A

Terraform Structure

```
1 my-infrastructure/
2   └── modules/          # Reusable logic (DRY)
3     └── app/
4       ├── main.tf
5       ├── variables.tf
6       └── outputs.tf
7     └── vpc/ ...
8
9   └── live/             # The "Chaos" begins here
10    └── prod/
11      └── app/
12        ├── main.tf      # Calls module
13        └── variables.tf
14        └── provider.tf  # ⚠ Duplicate 1
15        └── backend.tf   # ⚠ Duplicate 1
16      └── vpc/
17        ├── main.tf
18        └── variables.tf
19        └── provider.tf  # ⚠ Duplicate 2
20        └── backend.tf   # ⚠ Duplicate 2
21    └── stage/
22      └── app/
23        ├── provider.tf  # ⚠ Duplicate 3
24        └── backend.tf   # ⚠ Duplicate 3
25      └── ...
```

```
1 # live/prod/app/backend.tf
2 # WE COPY-PASTE THIS FILE EVERYWHERE ...
3
4 terraform {
5   backend "gcs" {
6     bucket  = "my-project-tf-state"
7     # 🚫 PAIN POINT: Hardcoded path manually typed per folder
8     prefix  = "prod/app/state"
9   }
10 }
```

Problems with Pure Terraform

01 Code Duplication Copy-pasting configurations across every environment folder

02 Monolith State Large state files that are risky to update and slow to plan

03 Dependency Hell Manually running terraform apply in specific orders (VPC first, then DB, then App)

04 Inconsistent Environments Drift between environments due to manual variable management

Terragrunt

This is an open source wrapper for Terraform that fills in some gaps in Terraform. You'll see how to use it a bit later in this chapter to deploy versioned Terraform code across multiple environments with minimal copying and pasting.

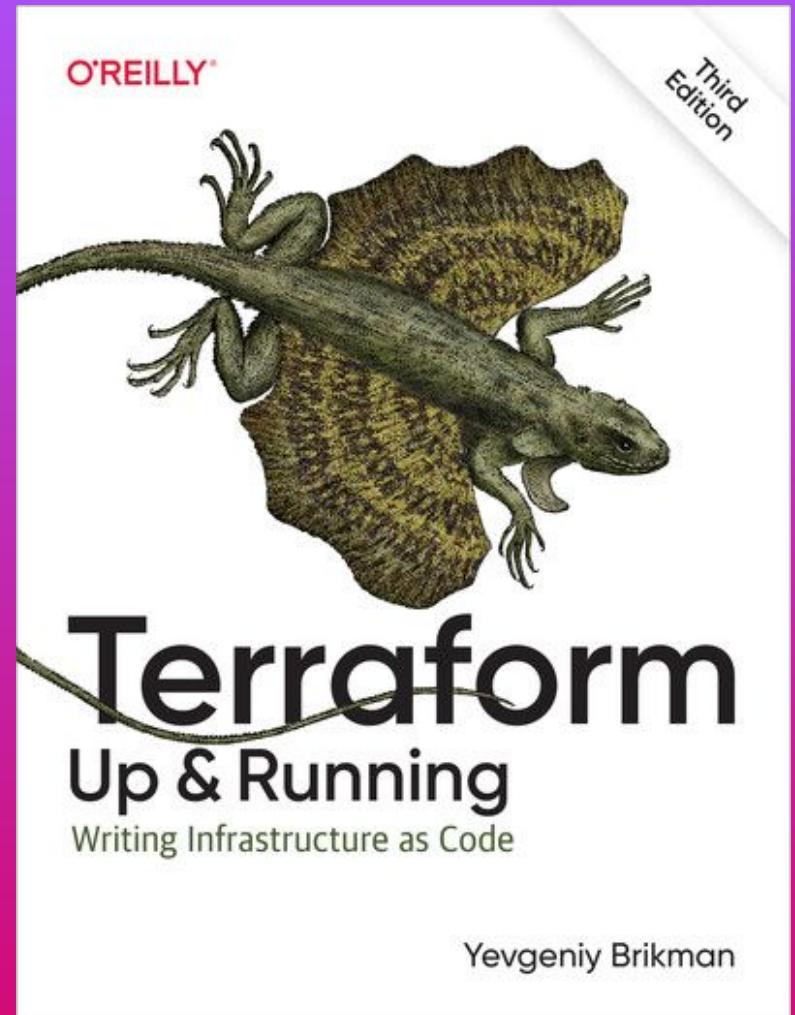
Another option for reducing copy-and-paste is to use [Terragrunt](#), an open source tool that tries to fill in a few gaps in Terraform. Terragrunt can help you keep your entire backend configuration DRY (Don't Repeat Yourself) by defining all the basic backend settings (bucket name, region, DynamoDB table name) in one file and automatically setting the `key` argument to the relative folder path of the module.

You'll see an example of how to use Terragrunt in [Chapter 10](#).

Resource dependencies

Breaking the code into multiple folders makes it more difficult to use resource dependencies. If your app code was defined in the same Terraform configuration files as the database code, that app code could directly access attributes of the database using an attribute reference (e.g., access the database address via `aws_db_instance.foo.address`). But if the app code and database code live in different folders, as I've recommended, you can no longer do that.

Solution: One option is to use dependency blocks in Terragrunt, as you'll see in [Chapter 10](#). Another option is to use the `terraform_remote_state` data source, as described in the next section.



Yevgeniy Brikman



What is Terragrunt?

A **thin wrapper** that provides extra tools for keeping your configurations **DRY**, working with **multiple modules**, and managing **remote state**.

- **✓ DRY Backend:** Define state config once in the root
- **✓ DRY CLI Flags:** Auto-retry, timeouts, and var-files
- **✓ Auto-Init:** Creates GCS buckets automatically
- **✓ Dependency Management:** run-all apply executes graphs

Terragrunt Structure

```
1 my-infrastructure/
2 |
3 └── modules/                      # Unchanged (Pure Terraform)
4   ├── app/ ...
5   └── vpc/ ...
6 └── terragrunt.hcl                  # 🌐 The Root Parent (Config inherited from here)
7 └── live/                          # The "Order"
8   ├── prod/
9     ├── app/
10    |   └── terragrunt.hcl          # Replaces all 4 .tf files!
11    └── vpc/
12      └── terragrunt.hcl
13 └── stage/
14   ├── app/
15   |   └── terragrunt.hcl
16   └── vpc/
17     └── terragrunt.hcl
```

Terraform

```
1 my-infrastructure/
2   ├── modules/          # Reusable logic (DRY)
3   |   └── app/
4   |       ├── main.tf
5   |       ├── variables.tf
6   |       └── outputs.tf
7   └── vpc/ ...
8
9   └── live/             # The "Chaos" begins here
10    └── prod/
11        └── app/
12            ├── main.tf      # Calls module
13            ├── variables.tf
14            ├── provider.tf  # ⚠ Duplicate 1
15            └── backend.tf   # ⚠ Duplicate 1
16
17    └── vpc/
18        ├── main.tf
19        ├── variables.tf
20        ├── provider.tf  # ⚠ Duplicate 2
21        └── backend.tf   # ⚠ Duplicate 2
22
23   └── stage/
24       └── app/
25           ├── provider.tf # ⚠ Duplicate 3
26           └── backend.tf  # ⚠ Duplicate 3
27
28   └── ...
29
```

Terragrunt

```
1 my-infrastructure/
2
3   └── modules/          # Unchanged (Pure Terraform)
4       └── app/ ...
5       └── vpc/ ...
6
7   └── terragrunt.hcl     # 🌃 The Root Parent (Config inherited from here)
8
9   └── live/              # The "Order"
10    └── prod/
11        └── app/
12            └── terragrunt.hcl # Replaces all 4 .tf files!
13
14    └── vpc/
15
16    └── stage/
17        └── app/
18            └── terragrunt.hcl
19
20        └── vpc/
21
22        └── terragrunt.hcl
23
24
25
```



01 Code Duplication

Problem: Copy-pasting Backend/Provider configs

Solving Duplication: Inheritance

```
1 # root/terragrunt.hcl
2
3 # We define the configuration ONCE here.
4 # (e.g., Common variables, remote state logic)
5
6 inputs = {
7   org_name = "devfest-gcp"
8   region   = "us-east-1"
9 }
```

```
1 # live/prod/app/terragrunt.hcl
2
3 # 🔗 The Include Block
4 include "root" {
5   path = find_in_parent_folders()
6 }
7
8 terraform {
9   source = "../..../modules/app"
10 }
11
12 # 🤝 No backend.tf needed!
13 # It is inherited automatically.
```

02 Monolith State

Problem:

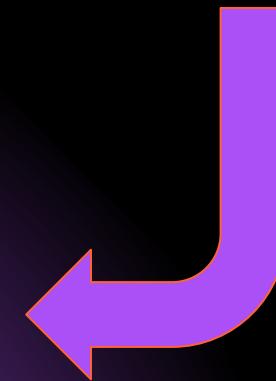
Managing state files manually,
Monolith risks

**Solving Management:
Dynamic Paths**

```
1 # root/terragrunt.hcl
2
3 remote_state {
4   backend = "gcs"
5   config = {
6     bucket  = "my-project-tf-state"
7     project = "my-gcp-project-id"
8     location = "us-east1"
9
10  # ✎ SOLUTION: Dynamic State Paths
11  # Automatically sets prefix to: "prod/app"
12  # based on the folder structure.
13  prefix    = "${path_relative_to_include()}"
14 }
15 }
```

```
1 └── gs://my-company-tf-state
2   └── prod
3     └── default.tfstate (⚠ All Resources: VPC + DB + App)
```

```
1 └── gs://my-project-tf-state
2   └── prod
3     ├── vpc
4     │   └── default.tfstate
5     ├── database
6     │   └── default.tfstate
7     └── app
8       └── default.tfstate
```



03 Dependency Hell

Problem:

Manual deployment orders
(VPC -> DB -> App)

Solving Order:

Explicit Dependencies

Command:

terragrunt run-all apply

```
1 # live/prod/app/terragrunt.hcl
2
3 # 🔗 Explicit Dependency
4 dependency "vpc" {
5   config_path = "../vpc"
6 }
7
8 inputs = {
9   # Terragrunt grabs the output from VPC
10  # and passes it to the App automatically.
11  network = dependency.vpc.outputs.network_self_link
12 }
```

04 Inconsistent Environments

Problem:

Inconsistency between environment versions

Solving Inconsistency:

Code Generation

```
1 # root/terragrunt.hcl
2
3 # 🏛 Generates provider.tf in EVERY folder
4 generate "provider" {
5   path      = "provider.tf"
6   if_exists = "overwrite_terraform"
7   contents  = <<EOF
8 provider "google" {
9   project = "my-gcp-project-id"
10  region  = "us-east1"
11
12  # 🔒 Pinning the version globally!
13  version = "~> 5.0"
14 }
15 EOF
16 }
```

The Result: Minimalistic Example

```
1 # live/prod/app/terragrunt.hcl
2
3 # ① INHERITANCE
4 include "root" {
5   path = find_in_parent_folders()
6 }
7
8 # ② SOURCE
9 terraform {
10   source = "tfr:///terraform-google-
11   modules/vm/google//modules/compute_instance?version=7.0.0"
12 }
13 # ③ DEPENDENCY
14 dependency "vpc" {
15   config_path = "../vpc"
16 }
18 # ④ INPUTS
19 inputs = {
20   # ▾ INHERITED FROM ROOT (Automatic Merge)
21   # region    = "europe-west1" ← No need to repeat this!
22
23   # ▾ CHILD SPECIFIC
24   machine_type = "e2-medium"
25
26   # ▾ DYNAMIC DEPENDENCY OUTPUTS
27   network      = dependency.vpc.outputs.network_name
28   subnetwork   = dependency.vpc.outputs.subnetwork_name
29
30   tags = ["http-server"]
31 }
```

Summary

Feature	The Chaos (Pure Terraform)	The Order (Terragrunt)
Configuration	Duplicated in every folder	Inherited from Root (DRY)
State Management	Hardcoded & Monolithic	Dynamic & Isolated
Deployments	Manual (Readme-driven)	Automated (run-all apply)
Consistency	Drifting Provider Versions	Generated & Identical

Q&A

Any questions?



Slides



Linkedin