# CH04 Terraform Practices

- Believe Everyone Have Learned How to Leverage Terraform to Manage AWS Resource

- Now I Want to Share My Some Experience About Developing Terraform

## Objectives

- What is Terragrunt?

- Modularize Everything Easily

- Create AWS Resource in Multiple Region

## Terraform Repository Fold Structure

```
practices/
├── account_a
│   ├── ap-northeast-1
│   │   └── dev
│   │       └── frontend
│   └── us-west-2
│       └── dev
│           └── frontend
└── modules
    └── kubernetes
```

## What You Need In A Terraform Folder At Least Before?

```
frontend/
├── Makefile          (Common Command)
├── env               (Environment Variable)
├── asg.tf            (Cloud Provider Resources)
├── lb.tf             (Cloud Provider Resources)
├── operations        (Helper Shell Script)
├── terraform.tfvars (Predefined Variable Value)
├── ...
└── variables.tf      (Variable Definition)
```

## After Using Terraform a Long Time...

- Have Multiple AWS Accounts

- Deploy Service Within Multiple Regions

- Trust Me, The Terraform Repository Will Become Mess, Spend More Time to Maintain It!

- Not to Mention Co-Working with Other Team Members

## What is Terragrunt?

- Terragrunt is a Thin `Wrapper` for Terraform

- Provides Extra Tools for Keeping Your Terraform Configurations DRY (Working with Multiple Terraform `Modules`, and Managing `Remote State`)

- Keep your Terraform code DRY

## What is Terragrunt?

**It's A Tool to Save Your Time, Force You to Produce Clean Code**

## What It Looks Like After Using Terragrunt

```
practices/
├── account_a
│   ├── ap-northeast-1
│   │   └── dev
│   │       ├── env.tfvars
│   │       ├── frontend
│   │       │   └── terraform.tfvars
│   │       └── terraform.tfvars
│   └── us-west-2
│       └── dev
│           ├── env.tfvars
│           ├── frontend
│           │   └── terraform.tfvars
│           └── terraform.tfvars
└── modules
    └── kubernetes
```

## Exercise I

### - Try to Create A Fountend Server Group in Tokyo...

### - Edit `terraform.tfvars` in practices/account_a/ap-northeast-1/dev, Change The `bucket` Value

```
~$ cd aws/ch04/practices
~$ cd account_a/ap-northeast-1/dev/frontend

~$ terragrunt init
~$ terragrunt apply
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroye

Outputs:

frontend_lb_dns_name = dev-frontend-443143937.ap-northeas
ubuntu_ami_id = ami-06c43a7df16e8213c
```

## Exercise II

### - If I Want to Achieve the Same Thing in Oregon...

### - Edit `terraform.tfvars` in practices/account_a/us-west-2/dev, Change The `bucket` Value (The Same as Previous One)

```
~$ cd ch04/practices
~$ cd account_a/us-west-2/dev/frontend

~$ terragrunt init
~$ terragrunt apply
```

```
Apply complete! Resources: 15 added, 0 changed, 0 destroy

Outputs:

frontend_lb_dns_name = dev-frontend-1834646447.us-west-2.
ubuntu_ami_id = ami-0e32ec5bc225539f5
```

## What You Have Done Just Now?

- Create two Frontend Server Groups in Two Different Regions

- And Without Write Any Extra Terraform Code

- Let Us Go Through What Terragrunt Do!

- Edit `terraform.tfvars` in practices/account_a/us-west-2/dev

## dev/terraform.tfvars

## - Define Remote State Backend

```
terragrunt = {
  remote_state {
    backend = "s3"

    config {
      encrypt         = true
      bucket          = "taipei-hug-workshop"
      key             = "account_a/ap-northeast-1/dev/${pa
      region          = "us-west-2"
    }
  }
...
}
```

## `dev/terraform.tfvars`

### - Define Enviornment Variable, Command

```
terragrunt = {
...
  # Configure root level variables that all resources can
  terraform {
    extra_arguments "bucket" {
      commands = ["${get_terraform_commands_that_need_var

      required_var_files = [
        "${get_parent_tfvars_dir()}/env.tfvars",
      ]
    }
  }
}
```

## dev/frontend/terraform.tfvars

### - Define Module Source From terraform-aws-frontend

```
terragrunt = {
  # Terragrunt will copy the Terraform configurations spe
  # working directory, into a temporary folder, and execu
  terraform {
    source = "github.com/Taipei-HUG/terraform-aws-fronten
  }

  # Include all settings from the root terraform.tfvars f
  include = {
    path = "${find_in_parent_folders()}"
  }
}

...
```

## dev/frontend/terraform.tfvars

### - Define the Variable Pass to Module terraform-aws-frontend

```
...

asg_config = {
  instance_count    = "1"
  instance_type     = "t3.small"
  root_volume_iops = "0"
  root_volume_size = "40"
  root_volume_type = "gp2"
}
```

## Not Finish Yet...

### We Have Not Understood Module Frontend Yet...

```
modules/
└── frontend
    ├── provision
    │   └── user_data
    ├── ami.tf
    ├── asg.tf
    ├── lb.tf
    ├── main.tf
    ├── outputs.tf
    ├── variables.tf
    └── vpc.tf
```

## What the File main.tf Include?

```
provider "aws" {
  region = "${var.aws_region}"
  version = "1.35"
}

terraform {
  # The configuration for this backend will be filled in
  backend "s3" {}
  required_version = ">= 0.11.8"
}

provider "template" {
  version = "1.0.0"
}
```

## If You Want to Create Something Afterward, Just ...

1. Develop/Find Module

2. Create Folder and *.tfvars Files

3. Execute terragrunt !

## How to test Terraform?

- Kitchen (Reference)

- Terratest (Reference)

## Key Takeaways

- Learned How to Use `Terragrunt`

- `Include/Retrive` Module from `GitHub`

- Create AWS Resource in Multiple Region, `But Not Writing Any Terraform Code`

# Destroy Resource Created by Exercise

```
~$ cd ch04/practices
~$ cd account_a/ap-northeast-1/dev/frontend
~$ terragrunt destroy
```

```
...
aws_default_vpc.default: Destruction complete after 0s

Destroy complete! Resources: 15 destroyed.
```

```
~$ cd ch04/practices
~$ cd account_a/us-west-2/dev/frontend
~$ terragrunt destroy
```

```
...
aws_default_vpc.default: Destruction complete after 0s

Destroy complete! Resources: 15 destroyed.
```