Published in ITNEXT

Ali Abbas Jaffri Follow
Sep 1, 2022 · 5 min read · Listen

Save

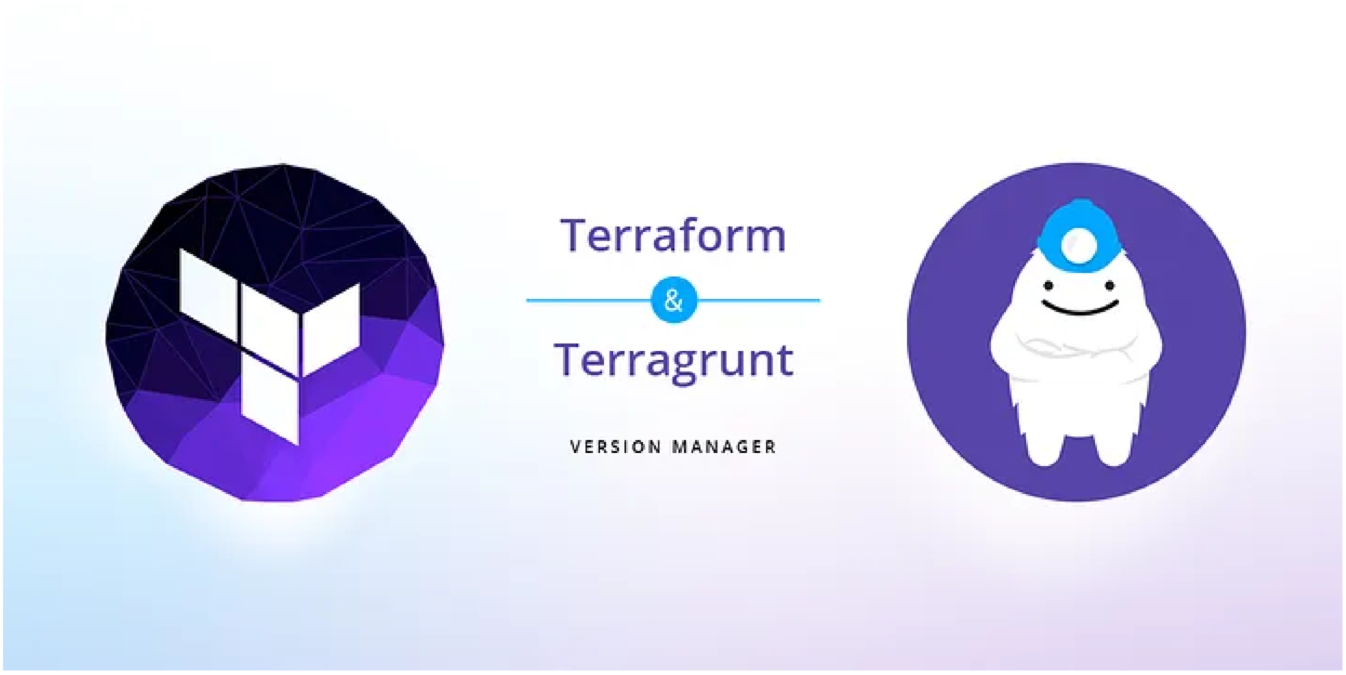

image credits: terraenv

Structuring terraform projects using terragrunt — Part I

I've been noticing that there is a reasonable gap regarding the wisdom on how to configure terraform projects. I decided to write a two part series on how to configure your terraform projects; ranging from small and medium sized projects, all the way to significantly large scale projects. I'll be discussing pros and cons of using terragrunt and some tips on how to get it working for your projects.

I recently read a very nice write up by founder of Gruntwork, Yevgeniy Brikman, in which he compares using terraform workspaces, multiple branches and terragrunt to configure different environments. It is a 4 part series and even though it might seem like a biased perspective coming from the founder himself, the author was quite on point with a lot of comparative aspects that he mentioned in his posts.

How to manage multiple environments with Terraform

A comparison of using workspaces, branches, and Terragrunt

blog.gruntwork.io

I would highly encourage you to take a moment and go through those series and understand why just relying on different branching structure or using variables using `.tfvars` file is not enough. Yevgeniy however didn’t cater terraspace in his write up, which in fact is a direct competitor of terragrunt, with more opinions on how to configure your terraform project. Another emerging tool in this space is pulumi which is removing the need to learn `.hcl` language and bringing the infrastructure and cloud configuration to every developer. Pulumi allows its users code in the language of their choosing to generate infrastructure for multiple cloud environments.

Terraspace | The Terraform Framework

The Terraform Framework

The Terraform Frameworkterraspace.cloud

Pulumi - Universal Infrastructure as Code

All architectures welcome Choose from over 60 cloud providers, including public, private, and hybrid architectures…

www.pulumi.com

The structuring process is quite straight forward for small to medium sized terraform projects. I would take the example of a recent post that i published; creating AWS ECS Fargate cluster for containerized application on AWS using terraform and terragrunt.

Creating an ECS Fargate service for container applications using terraform and terragrunt

A step by step procedure on how to deploy a container on ECS Fargate using terraform and terragrunt

itnext.io

In my experience, Terraform projects need to be structured correct right from the beginning. It gets relatively harder to adopt drastic changes as the project develops as you cannot make subtle changes without messing up a significant chunk of infrastructure. Its is also important to think about the different environments you'd be creating in your cloud so that the terragrunt configuration can well reflect that. A subtle point to note here is that this needs to be well thought through since the beginning so that you can seamlessly progress as the project matures.

The very first ingredient of effective structuring is to understand the layering of your infrastructure and how all the structures are going to build on top of each other. As I was setting a clean environment for the ECS project, i identified that there are few components, such as AWS VPC, AWS ECS cluster and AWS Application Load Balancer, which would be set up in the beginning and everything else would be built on top on them. I created terraform code for those components and started to configure the terragrunt configuration in a way i would want my infrastructure to layer on top of each other. The base layer components, in this case `VPC` , `ECS` cluster and `ALB` , were part of `base_infrastructure` which was to be set up in the beginning, and the containerized application would then be layered on top of the `base_infrastructure` . Extra care should be put in when identifying and configuring components in the `base_infrastructure` ; base layer has to be stable and well configured right from the beginning so that we don't introduce lots of changes later, which can possibly lead to recreation of these components, which, most of the times, would not be possible if layered components are dependent on them.

Once the requirements have been broken down in `base_infrastructure` and `applications` components, you can start with writing some code. Start with creating two base folders; `terraform` and `terragrunt`. All `.tf` code goes in `terraform` folder whereas all `.hcl` configuration goes in `terragrunt` folder. Modules in `terraform` folder can have a flat hierarchy, where all the components exist at the same level, ideally in their own folders. The `terragrunt` configuration exists in hierarchy in `terragrunt` folder, which is further split into `base_infrastructure` and `applications` sub-hierarchy. Another thing to cater would be the different environments that you'd be creating your infrastructure for; namely `development`, `staging` or `production` environments. You'd need introduce further sub-hierarchy within the folders under `terragrunt` to cater for `dev` , `staging` and `prod` environments. Create the modules that belong to that hierarchy of the infrastructure once those environment folders are in place.

There is no official guideline on this, but over the past years with multiple infrastructure projects, i've realised with experience that this is quite logical way so far to distribute your infrastructure components so that there is less impact of changes on other components in the architecture. This setup definitely has its own pros and cons; this would be an ideal setup in case your project is not very extensive, and it doesn't take 15 minutes to go through the entire folder structure to find the `terragrunt` file you need to configure or edit. The separation of configuration for modules allow you to define dependencies so that the base modules are configured before the dependent modules. The project structure is extensibility friendly; if you need to add a new module to your infrastructure, you just need to create a new `terraform` module and add the `terragrunt` configuration in the respective folder.

Configuration Blocks and Attributes

The Terragrunt configuration file uses the same HCL syntax as Terraform itself in terragrunt.hcl. Terragrunt also…

terragrunt.gruntwork.io

```
.
├── LICENSE
├── README.md
├── terraform
```

```
│   ├── aws_alb
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── ecs_application
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── ecs_cluster
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── service_role
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   └── vpc_subnet_module
│       ├── main.tf
│       ├── outputs.tf
│       └── variables.tf
└── terragrunt
    ├── applications
    │   ├── dev
    │   │   ├── ecs_application
    │   │   │   └── terragrunt.hcl
    │   │   └── stage.hcl
    │   └── staging
    ├── base-infrastructure
    │   ├── dev
    │   │   ├── aws_alb
    │   │   │   └── terragrunt.hcl
    │   │   ├── ecs_cluster
    │   │   │   └── terragrunt.hcl
    │   │   ├── stage.hcl
    │   │   └── vpc_subnet_module
    │   │       └── terragrunt.hcl
    │   └── staging
    ├── root-config.hcl
    └── terragrunt.hcl
```

project-structure.sh hosted with ❤ by **GitHub** **view raw**

Open in app ↗

Get unlimited access

H