Terragrunt Configuration Blocks

- terraform block
- remote_state block
- include block
- locals block
- dependency block
- dependencies block
- generate block

Terragrunt Configuration Attributes

- inputs
- download_dir
- prevent_destroy
- skip
- iam_role
- iam_assume_role_duration
- iam_assume_role_session_name
- terraform_binary
- terraform_version_constraint
- terragrunt_version_constraint
- retryable_errors

Terraform block

- Terraform block helps terragrunt to communicate for resource management on target provider
- With terraform block, it can find Terraform configuration files
- We can define arguments that can be passed for terraform CLI
- Define hooks (pre and post) to run before and after Terraform
- Arguments for Terraform block:
  - source: to locate terraform configuration files stored in VCS (git) etc.,
  - extra_arguments: to define additional arguments to be passed for terraform CLI
  - hooks (before_hook, after_hook, and error_hook): to run commands before and after terraform commands
  - include_in_copy: to define files that terragrunt will copy the code defined in source to scratch folder
  - terragrunt-read-config: to run actions we can define this argument only after hook(after_hook)
  - init-from-module and init:
    - to run hook when terragrunt us using “go-getter” to download remote configurations, use “init-from-module”
    - If we want to run hook after terragrunt is using terraform init for auto-init

```
terraform {
    # It pulls terraform configuration files from "dubareddy/terragrunt-aws-demo-modules" repo, under
    # subdirectory "modules/ec2-demo" for tag "v1.0.0"
    source = "git::git@github.com:dubareddy/terragrunt-aws-demo-modules.git//modules/ec2-demo?ref=v1.0.0"

    extra_arguments "custom_vars" {
        commands = [
            "apply",
            "plan",
            "refresh"
        ]
        required_var_files = ["${get_parent_terragrunt_dir()}/common.tfvars"]
    }

    include_in_copy = [
        ".command_values.json",
        "*.yaml",
    ]

    before_hook "run_before_checks" {
        commands = ["apply", "plan"]
        execute  = ["echo", "Lets execute before plan and apply hook!!!"]
    }

    after_hook "run_after_checks" {
        commands = ["apply", "plan"]
        execute  = ["echo", "Now lets execute after plan and apply hook!!!"]
        on_errors = [
            ".*",
        ]
    }
}
```

Sample terragrunt configuration file with terraform block

Remote_state block

- remote_state block adds remote state configuration for our terraform code
- For example, we can add remote state storage for AWS target provider to store terraform state file in s3 bucket from terragrunt configuration file with “remote_state” block
- To update/add backend resource block in terraform configuration files, we use generate block which will create a file to add backend resource block in the working directory
- We can add same block to multiple modules with the help of “find_in_parent_folders()” to pull settings
- remote_state block adds remote state and locking resources for Terraform. For example: AWS with s3 bucket for remote state and DynamoDB for state locking
- Terragrunt can control S3 backend as per requirement with config sub block. For example, if we need versioning to be enabled which is not enabled on bucket by default and Terragrunt will do the job for you
- Arguments for remote_state block:
  - backend
  - disable_init
  - disable_dependency_optimization
  - generate
  - config

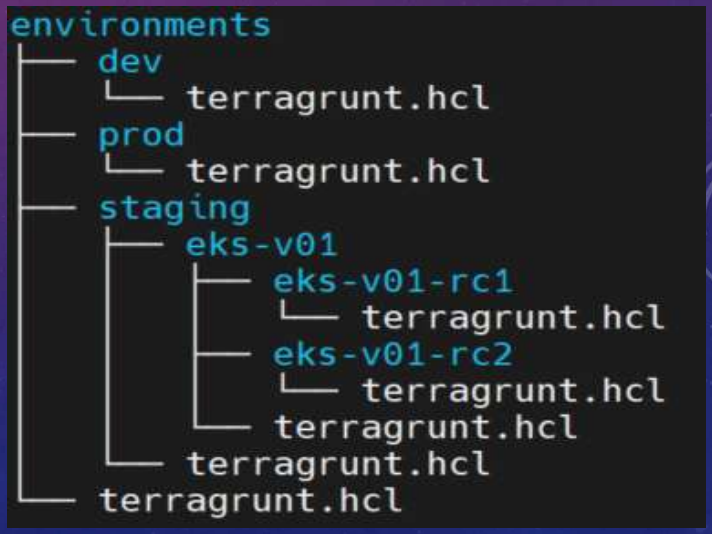
Sample terragrunt configuration file with remote_state for s3 backend

```
remote_state {
    backend = "s3"
    generate = {
        path      = "s3-backend.tf"
        if_exists = "overwrite"
    }
    config = {
        bucket         = "terraform-state-files"
        key            = "${path_relative_to_include()}/dev-terraform.tfstate"
        region         = "us-east-1"
        encrypt        = true
        dynamodb_table = "terraform-state-lock"
        # Specify below parameter, if you don not want terragrunt to automatically apply changes
        #disable_bucket_update = true
    }
    #config = {
    #skip_bucket_versioning         = true # use only if the object store does not support versioning
    #skip_bucket_ssencryption       = true # use only if non-encrypted Terraform State is required and/or the object store does
    #skip_bucket_root_access        = true # use only if the AWS account root user should not have access to the remote state b
    #skip_bucket_enforced_tls       = true # use only if you need to access the S3 bucket without TLS being enforced
    #enable_lock_table_ssencryption = true # use only if non-encrypted DynamoDB Lock Table for the Terraform State is required
    #accesslogging_bucket_name      = <string> # use only if you need server access logging to be enabled for your terraform st
    #accesslogging_target_prefix    = <string> # use only if you want to set a specific prefix for your terraform state S3 buck

    #shared_credentials_file     = "/path/to/credentials/file"
    #skip_credentials_validation = true
    #skip_metadata_api_check     = true
    #force_path_style            = true
    #}
}
```

Include block

- To inherit parent terraform configuration file to child configuration file, we use include block
- It will process the data from parent to child in current configuration file, if include block defined
- Terragrunt will process only one include block, it will throw error if additional block specified in file
- Arguments for include block:
  - name: label to the include block
  - path: path to fetch/define
  - expose: true or false (to be parsed or not and exposed as a variable)
  - merge_strategy: shallow (default), deep, no_merge



```
include "region" {
    name           = region_data
    path           = find_in_parent_folders("region.hcl") #find_in_parent_folders()
    expose         = true
    merge_strategy = "no_merge" # shallow (default), deep
}
```

Locals block

- locals block provides a way to define alias within the configuration file
- It would like same as Terraform locals variables
- To call locals variables to the configuration we use #local.ARG_NAME

```
locals {
  aws_region  = "us-east-1"
}

inputs = {
    region = local.aws_region
    name   = "${local.aws_region}-bucket"
}
```

Dependency block

- Dependency block is used to configure module dependencies
- It helps to pass the resource details created under dependency module throughout the configuration file as inputs
- For example, VPC module will provide VPC id, Subnet id output as inputs
- Arguments in dependency block:
  - name: label for dependency block section when we have multiple entries
  - config_path: path to the terragrunt module as a dependency
  - skip_outputs: when true, skip “terragrunt output”
  - mock_outputs: map of arbitrary key-value pair to use as outputs attribute when no outputs are available from target module
  - mock_outputs_allowed_terraform_commands: list terraform commands for which mock_outputs are allowed
  - mock_outputs_merge_strategy_state: (mock_outputs_merge_with_state was deprecated)
  - mock_outputs_merge_strategy_with_state: how many existing state should be merged into the mocks (no_merge → default, shallow, deep_map_only)

```
# Run `terragrunt output` on the module at the relative path `../vpc` and expose them under the
#attribute `dependency.vpc.outputs`
dependency "vpc" {
  config_path = "../vpc"

  # Configure mock outputs for the `validate` command that are returned when there are no outputs
  # available (e.g the module hasn't been applied yet)
  mock_outputs_allowed_terraform_commands = ["validate"]
  mock_outputs = {
    vpc_id = "vpc-f34fgy65ew"
  }
}

# Another dependency, available under the attribute `dependency.rds.outputs`
dependency "mysql_db" {
  config_path = "../rds/mysql"
}

inputs = {
  vpc_id = dependency.vpc.outputs.vpc_id
  db_url = dependency.mysql_db.outputs.db_url
}
```

Dependencies block

- Dependencies block is used to apply the module dependencies one by one
- This strategy will be applied while using “run-all” command
- Arguments for dependencies block:
  - paths

```
# When applying this terragrunt config in an `run-all` command, make sure the modules at "../vpc"
# and "../rds/mysql" are handled first.
dependencies {
  paths = ["../vpc", "../rds/mysql"]
}
```

Generate block

- Generate help terragrunt to create and add data in the terragrunt working directory
- This will be used only when Terragrunt calls Terraform for certain actions
- We use generate block to adding provider, backend etc., blocks dynamically for terraform
- Arguments for generate block:
  - name
  - path
  - if_exists
  - comment_prefix
  - disable_signature
  - contents

```
# When using this terragrunt config, terragrunt will generate the file "provider.tf"
# with the aws provider block before calling to terraform.
# Note that this will overwrite the `provider.tf` file if it already exists.
generate "provider" {
  path      = "provider.tf"
  if_exists = "overwrite"
  contents = <<EOF
provider "aws" {
  region      = "us-east-1"
  profile     = "eks-cluster"
  access_key  = "account-access-key"
  secret-key  = "account-secret-key"
  #shared_config_files      = ["/Users/sudheer/.aws/conf"]
  #shared_credentials_files = ["/Users/sudheer/.aws/creds"]
}
EOF
}
```

What we have learnt and understood?