

# How to Provision Cloud Infrastructure on Vultr using Terragrunt

Learn how to efficiently provision and manage cloud infrastructure on Vultr using Terragrunt. A step-by-step guide for automating your deployment workflow.

---

# Contents

01	Introduction	3
02	Prerequisites	3
03	Install Terragrunt	4
04	Set up the Terragrunt Modules Directory	5
05	Create Vultr Cloud Compute Instances using Terragrunt	8
06	Create Vultr Managed Databases with Terragrunt	16
07	Create Vultr Kubernetes Engine (VKE) Clusters with Terragrunt	23
08	Conclusion	29

# Introduction

---

Terraform is an open-source Infrastructure as Code (IaC) tool designed to automate the provisioning and management of cloud infrastructure resources. With Terraform, users can define infrastructure resources using a domain-specific language, enabling the creation, modification, and version control of infrastructure components across multiple cloud providers.

Terragrunt is a thin abstraction layer that runs on top of Terraform as an improved deployment platform with extra tools to deploy multiple infrastructure resources and services with a Don't Repeat Yourself (DRY) file structure. The [Vultr Terraform provider](#) is compatible with Terragrunt, and you can define configurations to create infrastructure resources such as databases, storage, compute instances, and VKE clusters.

This article explains how to provision cloud infrastructure resources on Vultr using Terragrunt. You will set up a DRY configuration file structure consisting of a root Terragrunt directory and parent-child directories to deploy resources using your Vultr API key.

## Prerequisites

---

Before you begin:

- Deploy a [Vultr Ubuntu server](#) to use as the management workstation.
- Enable your [Vultr API Key](#) and allow the workstation public IP address within the **Access Control** section to access your Vultr account with the API key.
- Access the server [using SSH](#) as a [non-root user with sudo privileges](#).
- Install Terraform:

CONSOLE

```
$ sudo snap install terraform --classic
```

## Install Terragrunt

1. Install the JSON processor `jq` utility to extract the latest Terragrunt version.

CONSOLE

```
$ sudo snap install jq
```

2. Extract the latest Terragrunt version using `jq` and store it in a new `VERSION` environment variable.

CONSOLE

```
$ export VERSION=$(curl -sL https://api.github.com/repos/gruntwork-io/terragrunt/releases/latest | jq -r '.tag_name | ltrimstr("v")')
```

3. Download the latest Terragrunt release package using the `wget` utility and the `VERSION` variable value.

CONSOLE

```
$ sudo wget -O terragrunt https://github.com/gruntwork-io/terragrunt/releases/download/v$VERSION/terragrunt_linux_amd64
```

4. Enable execute privileges on the `terragrunt` binary.

CONSOLE

```
$ sudo chmod a+x terragrunt
```

5. Move the Terragrunt binary to a system-wide directory within your user `$PATH`. For example, `/usr/local/bin` to activate Terragrunt as a system-wide command.

```
CONSOLE
```

```
$ sudo mv terragrunt /usr/local/bin/
```

6. View the Terragrunt version to verify that the command runs correctly

```
CONSOLE
```

```
$ terragrunt --version
```

Output:

```
terragrunt version v0.54.22
```

## Set up the Terragrunt Modules Directory

Terragrunt uses a hierarchical directory structure to access and execute your configuration files. To deploy resources on your Vultr account, Terragrunt uses the Terraform provider and resource definition configurations in your project directory with a parent-child module structure similar to the one below:

```
.
├── vultr-terragrunt
│   ├── Compute
│   │   ├── vultr_instance.tf
│   │   ├── terragrunt.hcl
│   │   ├── regular
│   │   │   └── terragrunt.hcl
│   │   ├── optimized
│   │   │   └── terragrunt.hcl
│   │   └── baremetal
│   │       └── terragrunt.hcl
│   ├── Databases
│   │   ├── vultr_database.tf
│   │   ├── terragrunt.hcl
│   │   ├── valkey
│   │   │   └── terragrunt.hcl
│   └── mysql
```

```
| | └─ terragrunt.hcl
| └─ postgresql
|   └─ terragrunt.hcl
└─ VKE
    └─ vke.tf
    └─ terragrunt.hcl
    └─ dev
        └─ terragrunt.hcl
    └─ prod
        └─ terragrunt.hcl
    └─ stage
        └─ terragrunt.hcl
```

Follow the steps below to set up a root Terragrunt project structure to replicate with multiple parent-child modules and deploy cloud resources to your Vultr account.

1. Switch to your user home directory.

```
CONSOLE
```

```
$ cd
```

2. Create a new directory to store your Terragrunt configurations.

```
CONSOLE
```

```
$ mkdir vultr-terragrunt
```

3. Switch to the new project directory.

```
CONSOLE
```

```
$ cd vultr-terragrunt
```

4. Create a new Terraform configuration file `provider.tf` to define your provider settings.

```
CONSOLE
```

```
$ nano provider.tf
```

5. Add the following contents to the file.

```
HCL

terraform {
  required_providers {
    vultr = {
      source = "vultr/vultr"
      version = "2.18.0"
    }
  }
}

provider "vultr" {
  api_key = var.VULTR_API_KEY
}

variable "VULTR_API_KEY" {}
```

Save and close the file.

The above Terraform provider configuration enables the Vultr Terraform provider version `2.18.0` as the default infrastructure source. The `provider` block authenticates using your Vultr API key stored in the `VULTR_API_KEY` variable to access the provider and deploy resources to your Vultr account.

6. Create a new Terragrunt-specific configuration file `terragrunt.hcl` to store your Vultr API key.

```
CONSOLE

$ nano terragrunt.hcl
```

7. Add the following contents to the file. Replace `example-api-key` with your actual Vultr API key.

```
HCL
```

```
inputs = {  
  VULTR_API_KEY = "example-api-key"  
}
```

Save and close the file.

The above configuration references the `VULTR_API_KEY` variable in your `provider.tf` configuration with an API key value to connect to your Vultr account.

8. List the Terragrunt directory files to verify that all necessary parent configurations are available.

```
CONSOLE
```

```
$ ls /home/user/vultr-terragrunt/
```

Output:

```
provider.tf  terragrunt.hcl
```

The above Terragrunt file structure uses the `provider.tf` configuration to invoke the Vultr API and execute the defined operations on your Vultr account. To deploy resources using Terragrunt, you should create child modules to store `terragrunt.hcl` configurations that include your infrastructure specifications.

## Create Vultr Cloud Compute Instances using Terragrunt

To create Vultr Cloud Compute instances using Terragrunt, set up a new standalone parent directory structure to create child modules that match your target environment or instance type. Depending on your desired instance specifications, Vultr supports the following cloud compute tags:

- `vc2`: Regular Cloud Compute
- `vdc`: Dedicated Cloud

- `vhf` : High-Frequency Compute
- `vhp` : High Performance
- `voc` : Optimized Cloud Compute
- `vcg` : Cloud GPU
- `vbm` : Bare Metal

For more information, run the following requests using the Curl utility to verify all supported values:

- List available plans:

**CONSOLE**

```
$ curl "https://api.vultr.com/v2/plans" \  
-X GET \  
-H "Authorization: Bearer ${VULTR_API_KEY}"
```

- List Vultr regions:

**CONSOLE**

```
$ curl "https://api.vultr.com/v2/regions" \  
-X GET \  
-H "Authorization: Bearer ${VULTR_API_KEY}"
```

- List operating system IDs:

**CONSOLE**

```
$ curl "https://api.vultr.com/v2/os" \  
-X GET \  
-H "Authorization: Bearer ${VULTR_API_KEY}"
```

In this section, create a regular Cloud Compute `vc2` instance with the `vc2-1c-1gb` plan which includes `1 vCPU` core and `1 GB` RAM in the Singapore `sgp` Vultr location as described in the steps below.

1. Create a new `compute` directory to use as a parent directory with underlying child modules.

```
CONSOLE
```

```
$ mkdir compute
```

2. Copy the parent configuration files `provider.tf`, `terragrunt.hcl` files to the `compute` directory.

```
CONSOLE
```

```
$ cp provider.tf terragrunt.hcl compute/
```

Terragrunt uses the DRY configuration file scheme, but requires parent configurations at the root of each directory. The above command copies root configurations to enable the `compute` directory as a standalone parent directory referenced by child modules.

3. Switch to the `compute` directory.

```
CONSOLE
```

```
$ cd compute
```

4. Create a new Terraform configuration file `vultr_instance.tf` to use as the base cloud compute instance definition file.

```
CONSOLE
```

```
$ nano vultr_instance.tf
```

5. Add the following contents to the file.

```
HCL
```

```
resource "vultr_instance" "vultr_compute_instance" {  
  label = var.LABEL  
  plan = var.PLAN  
  region = var.REGION  
  os_id = var.OS  
}
```

```
variable "LABEL" {}  
variable "PLAN" {}  
variable "REGION" {}  
variable "OS" {}
```

Save and close the file.

The above instance definition file creates the necessary variables to use with configuration files in the child modules for deployment to your Vultr account. Within the file:

- `vultr_instance`: Defines a Vultr instance resource with the custom label `vultr_compute_instance`.
- `label`: Sets the compute instance using your `LABEL` variable as the descriptive value.
- `plan`: Specifies the compute instance specification using the `PLAN` variable value.
- `region`: Specifies the compute instance region using your `REGION` variable to deploy the instance.
- `os_id`: Sets the operating system to deploy with the compute instance using your `OS` variable value.

6. Create a new child directory `regular` to set up a test regular Cloud Compute instance.

```
CONSOLE
```

```
$ mkdir regular
```

7. Switch to the directory.

```
CONSOLE
```

```
$ cd regular
```

8. Create a new resource definition file `terraform.hcl`.

```
CONSOLE
```

```
$ nano terragrunt.hcl
```

9. Add the following contents to the file.

```
HCL

include {
  path = find_in_parent_folders()
}

terraform {
  source = "../"
}

inputs = {
  LABEL = "dev-server"
  PLAN = "vc2-1c-1gb"
  REGION = "sgp"
  OS = "387"
}
```

Save and close the file.

The above configuration creates a new Vultr Cloud Compute instance with the following specifications:

- **TYPE:** Vultr Regular Cloud Compute `vc2`
- **vCPUS:** 1 vCPU `1c`
- **Memory:** 1 GB `1gb`
- **Location:** Singapore `sgp`
- **OS:** 387 - `Ubuntu 20.04`

Replace the above instance values to match your desired specifications. Within the configuration:

- `path = find_in_parent_folders()`: Scans the parent directory for configurations referenced by the `inputs={}` values within the file.
- `source = "../"`: Defines the parent `compute` directory to use the necessary Terraform provider configuration. `../` defines one step back from the configuration's working directory.

- `inputs ={`: Sets the Vultr Cloud Compute instance specifications to match and activate with the `vultr_instance.tf` file variables.

10. View your `compute` directory structure to verify the available configuration files.

CONSOLE

```
$ ls -F -R /home/user/vultr-terragrunt/compute/
```

Output:

```
/home/linuxuser/vultr-terragrunt/compute/:  
provider.tf  regular/  terragrunt.hcl  vultr_instance.tf  
  
/home/linuxuser/vultr-terragrunt/compute/regular:  
terragrunt.hcl
```

Within the above output, Terragrunt runs the `regular` child module specifications as variable values to the parent directory `compute` instance configuration. Then, it authenticates and deploys the defined Cloud Compute resource to your Vultr account using the parent Terraform provider configuration.

11. Test the Terragrunt configuration for errors.

CONSOLE

```
$ terragrunt validate
```

Output:

```
Success! The configuration is valid.
```

12. Initialize the Terragrunt configuration.

CONSOLE

```
$ terragrunt init
```

When successful, your output should be similar to the one below:

```
Initializing the backend...

Initializing provider plugins...

- Finding vultr/vultr versions matching "2.18.0"...
- Installing vultr/vultr v2.18.0...
- Installed vultr/vultr v2.18.0 (signed by a HashiCorp partner, key ID
853B1ED644084048)

.....

Terraform has been successfully initialized!
```

13. View the Terraform project summary to verify your configuration changes.

CONSOLE

```
$ terraform plan
```

Your output should be similar to the one below:

```
Terraform will perform the following actions:

# vultr_instance.vultr_compute_instance will be created

+ resource "vultr_instance" "vultr_compute_instance" {
+   allowed_bandwidth = (known after apply)
+   app_id             = (known after apply)
+   backups           = "disabled"
+   date_created      = (known after apply)
+   ddos_protection   = false
+   default_password  = (sensitive value)
+   disk              = (known after apply)
+   enable_ipv6       = true
+   features          = (known after apply)
+   firewall_group_id = (known after apply)
+   gateway_v4        = (known after apply)
+   hostname          = (known after apply)
+   id                = (known after apply)
```

```
+ image_id           = (known after apply)
+ internal_ip       = (known after apply)
+ kvm               = (known after apply)
+ label            = "dev-server"
+ main_ip          = (known after apply)
+ netmask_v4       = (known after apply)
+ os               = (known after apply)
+ os_id            = 387
+ plan             = "vc2-1c-1gb"
+ power_status     = (known after apply)
+ private_network_ids = (known after apply)
+ ram              = (known after apply)
+ region           = "sgp"
+ reserved_ip_id   = (known after apply)
+ script_id        = (known after apply)
+ server_status    = (known after apply)
+ snapshot_id      = (known after apply)
+ status           = (known after apply)
+ user_data        = (known after apply)
+ v6_main_ip       = (known after apply)
+ v6_network        = (known after apply)
+ v6_network_size  = (known after apply)
+ vcpu_count       = (known after apply)
+ vpc_ids          = (known after apply)
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

#### 14. Apply your Terraform configurations to your Vultr account.

CONSOLE

```
$ terraform apply
```

When prompted, enter **yes** and press Enter to approve the Terraform actions.

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
```

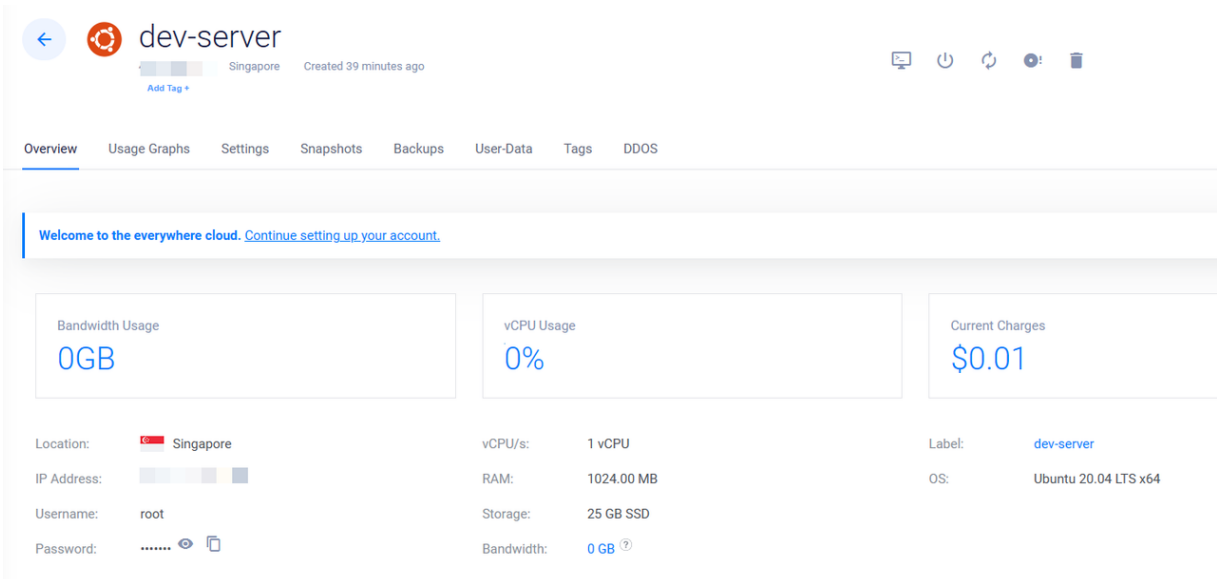
Enter a value:

When the deployment is successful, your output should be similar to the one below:

```
  vultr_instance.vultr_compute_instance: Creating...
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

15. Open the [Vultr customer portal](#).

16. Navigate to **Products -> Compute** and verify that the new Cloud Compute instance is available.



The screenshot displays the Vultr customer portal interface for a newly created Cloud Compute instance named "dev-server". The instance is located in Singapore and was created 39 minutes ago. The interface shows various tabs for management: Overview, Usage Graphs, Settings, Snapshots, Backups, User-Data, Tags, and DDOS. A welcome message is displayed, along with three key metrics: Bandwidth Usage (0GB), vCPU Usage (0%), and Current Charges (\$0.01). Below these metrics, the instance's configuration is detailed, including its location (Singapore), IP Address, Username (root), Password, vCPU/s (1 vCPU), RAM (1024.00 MB), Storage (25 GB SSD), and Bandwidth (0 GB).

## Create Vultr Managed Databases with Terragrunt

To create Vultr Managed Databases with Terragrunt, set up a new parent directory to use with your provider configurations and specify your database resource configurations to deploy on your Vultr account. Depending on your target database engine and plan, Vultr supports the following values:

- Database Engines: `MySQL`, `PostgreSQL`, `Valkey`
- View the list of available plans per Vultr location:

## CONSOLE

```
$ curl "https://api.vultr.com/v2/databases/plans" -X GET  
-H "Authorization: Bearer ${VULTR_API_KEY}" | jq
```

Follow the steps below to create a sample Vultr Managed Database for Caching using the `vultr-dbaas-hobbyist-cc-hp-intel-1-11-1`, `1 GB` RAM and `1 vCPU` plan in the Singapore `sgp` Vultr location. When deploying a different database engine, replace the `valkey` configuration values with your target Vultr Managed Database specifications.

1. Switch to the root Terraform project directory.

## CONSOLE

```
$ cd /home/user/vultr-terraform/
```

2. Create a new parent directory `Databases` to store your resource configurations.

## CONSOLE

```
$ mkdir Databases
```

3. Copy the `provider.tf` and `terraform.hcl` files to the new parent directory.

## CONSOLE

```
$ cp provider.tf terraform.hcl Databases/
```

4. Switch to the `Databases` directory.

## CONSOLE

```
$ cd Databases
```

5. Create a new configuration file `vultr_valkey.tf` to define the database specification values.

CONSOLE

```
$ nano vultr_database.tf
```

6. Add the following contents to the file.

HCL

```
resource "vultr_database" "vultr_db" {
  database_engine = var.DB_ENGINE
  database_engine_version = var.DB_VERSION
  region = var.DB_LOCATION
  plan = var.DB_PLAN
  label = var.DB_LABEL
}

variable "DB_PLAN" {}
variable "DB_LABEL" {}
variable "DB_ENGINE" {}
variable "DB_VERSION" {}
variable "DB_LOCATION" {}
```

Save and close the file.

The above configuration creates a new database resource that uses variable values from the child directory `Terragrunt.hcl` file. Within the configuration:

- `resource "vultr_database"`: Sets Vultr Managed Database as the resource type with a custom descriptive label `vultr_db`.
- `database_engine`: Specifies the Vultr Managed Database engine using the `DB_Engine` variable value.
- `database_engine_version`: Sets the target Vultr Managed Database version using the `DB_VERSION` variable value.
- `region`: Sets the target Vultr location to deploy a Vultr Managed Database using the `DB_LOCATION` variable value.
- `plan`: Sets the Vultr Managed Database plan to use as the backend infrastructure.
- `label`: Assigns a descriptive label to the Vultr Managed Database using the `DB_LABEL` variable value.

- `variable`: Declares the configuration variables for use within the child specification values.

7. Create a new child directory to store the Vultr Managed Database resource values. For example, `valkey`.

CONSOLE

```
$ mkdir valkey
```

8. Switch to the directory.

CONSOLE

```
$ cd valkey
```

9. Create a new resource configuration file `terragrunt.hcl`.

CONSOLE

```
$ nano terragrunt.hcl
```

10. Add the following contents to the file.

HCL

```
include {
  path = find_in_parent_folders()
}

terraform {
  source = "../"
}

inputs = {
  DB_ENGINE = "valkey"
  DB_VERSION = "7"
  DB_LOCATION = "sgp"
  DB_LABEL = "prod-valkey-db"
  DB_PLAN = "vultr-dbaas-hobbyist-cc-hp-intel-1-11-1"
}
```

Save and close the file.

The above configuration creates a new Vultr Managed Database for Caching with the following values:

- Database Type : valkey
- Vultr location : Singapore sgp
- Database Label : prod-valkey-db
- Database infrastructure plan : 1 vCPU, 1 GB RAM, HP Intel-powered backend server

Within the configuration:

- `path = find_in_parent_folders()` : Scans the specified source path for all Terragrunt configurations referenced from the file.
- `source = "../"` : Sets the location of the Terraform provider and other configurations.
- `input = {}` : Sets all variable values to match the Terragrunt instance configuration defined in the parent directory.

## 11. Test the Terragrunt configuration for errors.

```
CONSOLE
$ terragrunt validate
```

Output:

```
Success! The configuration is valid.
```

## 12. View the Terragrunt plan to verify all changes to apply.

```
CONSOLE
$ terragrunt plan
```

Output:

Terraform will perform the following actions:

```
# vultr_database.vultr_valkey will be created
+ resource "vultr_database" "vultr_valkey" {
  + cluster_time_zone      = (known after apply)
  + database_engine       = "valkey"
  + database_engine_version = "7"
  + date_created          = (known after apply)
  + dbname                = (known after apply)
  + credentials           = (known after apply)
  + host                  = (known after apply)
  + id                    = (known after apply)
  + label                  = "dev-valkey"
  + latest_backup         = (known after apply)
  + maintenance_dow      = (known after apply)
  + maintenance_time     = (known after apply)
  + password              = (known after apply)
  + plan                  = "vultr-dbaas-hobbyist-cc-hp-intel-1-11-1"
  + plan_disk             = (known after apply)
  + plan_ram              = (known after apply)
  + plan_replicas         = (known after apply)
  + plan_vcpus            = (known after apply)
  + port                  = (known after apply)
  + public_host           = (known after apply)
  + valkey_eviction_policy = (known after apply)
  + region                 = "sgp"
  + status                 = (known after apply)
  + user                  = (known after apply)
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

### 13. Apply your Terraform configurations.

CONSOLE

```
$ terraform apply
```

### 14. When prompted, enter `yes` and press Enter to approve the changes to your Vultr account.

```
Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.
```

```
Enter a value:
```

15. Wait for a few minutes for the database creation process to complete, when successful, your output should be similar to the one below:

```
vultr_database.vultr_db: Still creating... [4m30s elapsed]  
vultr_database.vultr_db: Still creating... [4m40s elapsed]  
vultr_database.vultr_db: Still creating... [4m50s elapsed]  
vultr_database.vultr_db: Creation complete after 5m0s [id=87example88fd]
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

16. Access the Vultr Customer Portal and verify that your new Vultr Managed Database for Caching is available.

The screenshot displays the Vultr Customer Portal interface for a Redis database instance named 'prod-redis-db'. The instance is located in Singapore and has the following specifications: 0 Replica Nodes, 1024 MB RAM, and 1 vCPU. The cluster status is 'Running'. The interface includes a navigation menu with options like Overview, Usage Graphs, Service Logs, Users, Migration, and Settings. Key metrics shown are vCPU Usage at 0%, Network Usage at 0/0 bps, and Current Charges at \$0.03. A promotional banner encourages migrating from DigitalOcean. The 'General Information' section lists a monthly price of \$18.00, a node plan of 'vultr-dbaas-hobbyist-cc-hp-intel-1-11-1', a cluster created 7 minutes ago, a maintenance window on Saturday at 9 AM UTC, a Redis DB engine, and no latest backup.

### Note

To create additional Vultr Managed Databases using Terraform, create new child directories with your target database specifications to apply to your Vultr account. To destroy a resource, run `terraform destroy` within the target directory and verify the changes to apply to your Vultr account.

# Create Vultr Kubernetes Engine (VKE) Clusters with Terraform

To create a Vultr Kubernetes Engine (VKE) cluster with Terraform, set up a new child directory to store the cluster configurations. Depending on your cluster specification, each resource configuration file includes a node pools section that specifies your backend VKE cluster structure. Follow the steps below to create a 3-node Vultr Kubernetes Engine (VKE) cluster using the regular Cloud Compute `vc2`, `2 vCPUs`, `4 GB RAM` plan per node.

1. Navigate to your root Terraform project directory.

```
CONSOLE
```

```
$ cd /home/user/vultr-terraform/
```

2. Create a new parent directory `vke` to store your cluster definition files.

```
CONSOLE
```

```
$ mkdir vke
```

3. Copy the `provider.tf` and `terraform.hcl` files to the parent directory.

```
CONSOLE
```

```
$ cp provider.tf terraform.hcl vke/
```

4. Switch to the directory.

```
CONSOLE
```

```
$ cd vke
```

5. Create a new Terraform configuration file `vke.tf` to define the cluster specifications.

```
CONSOLE
$ nano vke.tf
```

6. Add the following contents to the file.

```
HCL
resource "vultr_kubernetes" "vke_cluster" {
  region = var.REGION
  label  = var.NAME
  version = var.VERSION

  node_pools {
    node_quantity = var.NODES
    plan          = var.PLAN
    label         = var.LABEL
    auto_scaler   = var.SCALING
    min_nodes     = var.MIN-NODES
    max_nodes     = var.MAXNODES
  }
}

variable "REGION" {}
variable "NAME" {}
variable "VERSION" {}
variable "NODES" {}
variable "PLAN" {}
variable "SCALING" {}
variable "MIN-NODES" {}
variable "MAXNODES" {}
```

Save and close the file.

The above configuration defines a new Vultr Kubernetes Engine (VKE) cluster configuration with the following specifications:

- `region`: Sets the Vultr location to deploy the VKE cluster using the `REGION` variable value.

- `label`: Specifies the VKE cluster descriptive label using the `NAME` variable value.
- `version`: Sets the target Kubernetes version to deploy with the cluster using the `VERSION` variable value.
- `node_pools`: Defines the VKE worker node specifications. Within the section:
  - `node_quantity`: Sets the number of nodes to attach to the VKE cluster.
  - `plan`: Defines the compute plan to deploy with the cluster nodes. To view the list of available Vultr Cloud Compute plans, visit the [API plans list](#).
  - `label`: Sets a descriptive label for all VKE nodes.
  - `auto-scaler = true`: Enables auto-scaling of VKE cluster nodes.
  - `min_nodes`: Sets the minimum number of nodes to support in the VKE node pool.
  - `max_nodes`: Sets the maximum number of nodes the VKE node pool can scale to when auto-scaling is enabled.

7. Create a new directory `prod` to store the cluster configuration values.

```
CONSOLE
$ mkdir prod
```

8. Switch to the directory.

```
CONSOLE
$ cd prod
```

9. Create a new configuration file `terraform.hcl` to set the VKE cluster values.

```
CONSOLE
$ nano terraform.hcl
```

10. Add the following contents to the file.

HCL

```
include {
  path = find_in_parent_folders()
}

terraform {
  source = "../"
}

inputs = {
  REGION = "ewr"
  NAME = "prod-vke-cluster"
  VERSION = "v1.29.1+1"
  NODES = "3"
  PLAN = "vc2-2c-4gb"
  SCALING = "true"
  MIN-NODES = "3"
  MAXNODES = "6"
}
```

Save and close the file.

The above configuration defines a VKE cluster with the following values:

- **Deployment location:** New Jersey ( `ewr` )
- **Cluster Label:** `prod-vke-cluster`
- **Kubernetes Version:** `v1.29.1+1`
- **Worker Nodes:** 3
- **Nodes compute plan:** Vultr regular cloud compute `vc2` , 2 vCPUs , 4 GB memory ( `vc2-2c-4gb` ) per node
- **Auto scaling:** yes
- **Minimum number of nodes:** 3
- **Maximum number of scaling nodes:** 6

Within the configuration:

- `path = find_in_parent_folders()`: Scans the parent directory to use the referenced configuration values.
- `source = "../"`: Sets the parent directory to use the defined provider configuration. The value `../` instructs Terragrunt to scan up one level

in the directory tree. In the configuration, `../` points to the previous directory `vke` as the parent directory to scan for all required configurations.

- `inputs = {}`: Defines the values to match each variable referenced in the parent directory resource specification file.

## 11. Test the Terragrunt configuration for errors.

CONSOLE

```
$ terragrunt validate
```

Output:

```
Success! The configuration is valid.
```

## 12. View the Terragrunt plan to verify the changes to apply to your Vultr account.

CONSOLE

```
$ terragrunt plan
```

Output:

```
Terraform will perform the following actions:
```

```
# vultr_kubernetes.vke_cluster will be created
+ resource "vultr_kubernetes" "vke_cluster" {
+ client_certificate      = (sensitive value)
+ client_key             = (sensitive value)
+ cluster_ca_certificate = (sensitive value)
+ cluster_subnet         = (known after apply)
+ date_created           = (known after apply)
+ enable_firewall        = false
+ endpoint               = (known after apply)
+ firewall_group_id      = (known after apply)
+ ha_controlplanes        = false
+ id                    = (known after apply)
+ ip                    = (known after apply)
```

```
+ kube_config      = (sensitive value)
+ label            = "app1-prod-vke"
+ region           = "ewr"
+ service_subnet   = (known after apply)
+ status           = (known after apply)
+ version          = "v1.27.7+2"

+ node_pools {
  + auto_scaler    = true
  + date_created   = (known after apply)
  + date_updated   = (known after apply)
  + id             = (known after apply)
  + label          = "prod-cluster-nodes"
  + max_nodes      = 3
  + min_nodes      = 1
  + node_quantity  = 2
  + nodes          = (known after apply)
  + plan           = "vc2-2c-4gb"
  + status         = (known after apply)
  + tag            = (known after apply)
}
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

### 13. Apply the Terraform configurations to your Vultr account.

CONSOLE

```
$ terraform apply
```

When prompted, enter `yes` and press Enter to approve the deployment actions.

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
```

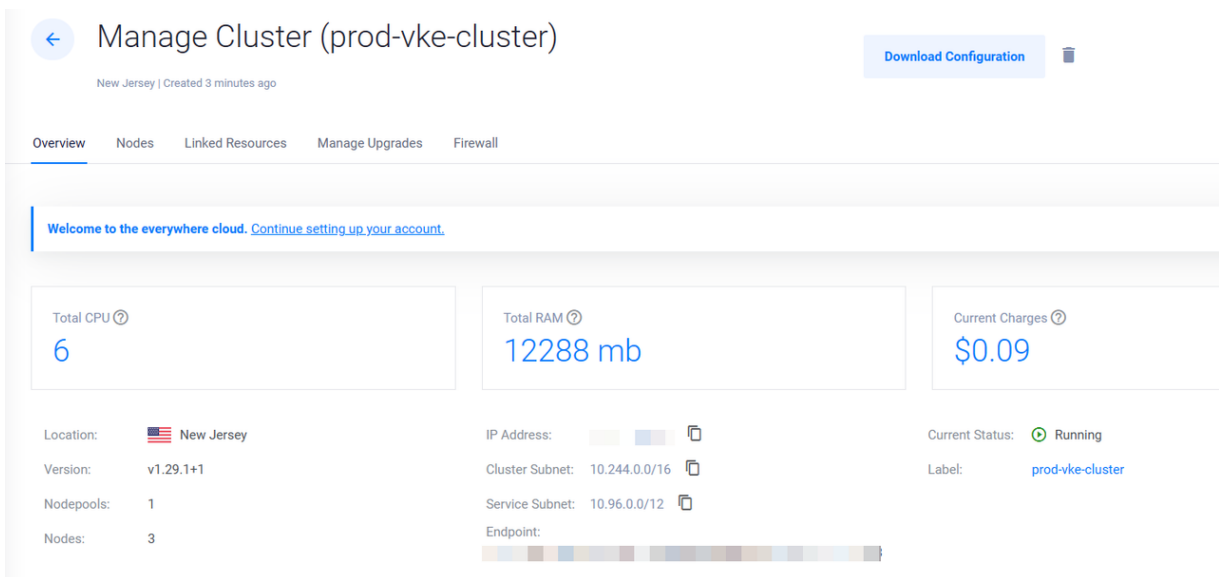
Enter a value:

Wait for at least `3` minutes for the cluster creation process to complete, when successful, your output should be similar to the one below:

```
vultr_kubernetes.vke_cluster: Still creating... [1m50s elapsed]
vultr_kubernetes.vke_cluster: Still creating... [2m0s elapsed]
vultr_kubernetes.vke_cluster: Still creating... [2m10s elapsed]
vultr_kubernetes.vke_cluster: Creation complete after 2m16s [id=f3bb-example88-
d72]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

14. Access the Vultr Customer Portal and verify that your VKE cluster is available.



The screenshot shows the 'Manage Cluster' interface for a production VKE cluster. The cluster is named 'prod-vke-cluster' and is located in New Jersey. It was created 3 minutes ago. The interface includes a 'Download Configuration' button and a navigation menu with tabs for Overview, Nodes, Linked Resources, Manage Upgrades, and Firewall. A welcome message is displayed: 'Welcome to the everywhere cloud. Continue setting up your account.' The main dashboard features three summary cards: Total CPU (6), Total RAM (12288 mb), and Current Charges (\$0.09). Below these cards, the cluster details are listed: Location (New Jersey), Version (v1.29.1+1), Nodepools (1), and Nodes (3). Network information includes IP Address, Cluster Subnet (10.244.0.0/16), Service Subnet (10.96.0.0/12), and Endpoint. The current status is 'Running' and the label is 'prod-vke-cluster'.

## Conclusion

You have set up Terragrunt and deployed resources on your Vultr account. Depending on your cloud infrastructure needs, create a standardized parent-child directory structure to correctly use Terragrunt with a DRY (Don't Repeat Yourself) format to organize and deploy resources using your Vultr API Key. To destroy any resources with Terragrunt, run `terraform destroy` within the target child directory. For more information, visit the official [Terragrunt documentation](#).



VULTR

