Terraform Infrastructure Documentation on VPS Platform

10/09/2025

Haseef Ahamed

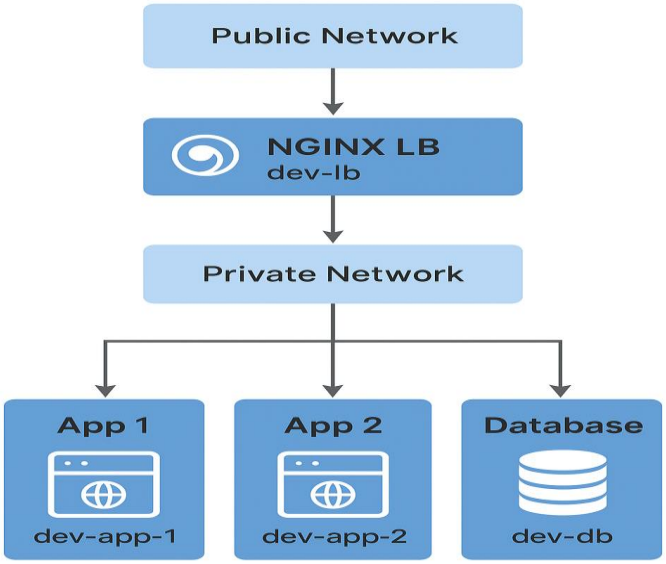
Contents

1. Project Overview

This project deploys a multi-container environment using **Terraform** and **Docker** on a VPS. The architecture includes:

- **Load Balancer (LB)**: NGINX container to distribute traffic to backend app containers.
- **Application (App)**: Multiple HTTPD containers serving web apps.
- **Database (DB)**: MySQL container for persistent storage.
- **Network**: Public and private Docker networks for container isolation and communication.

2. Architecture Diagram



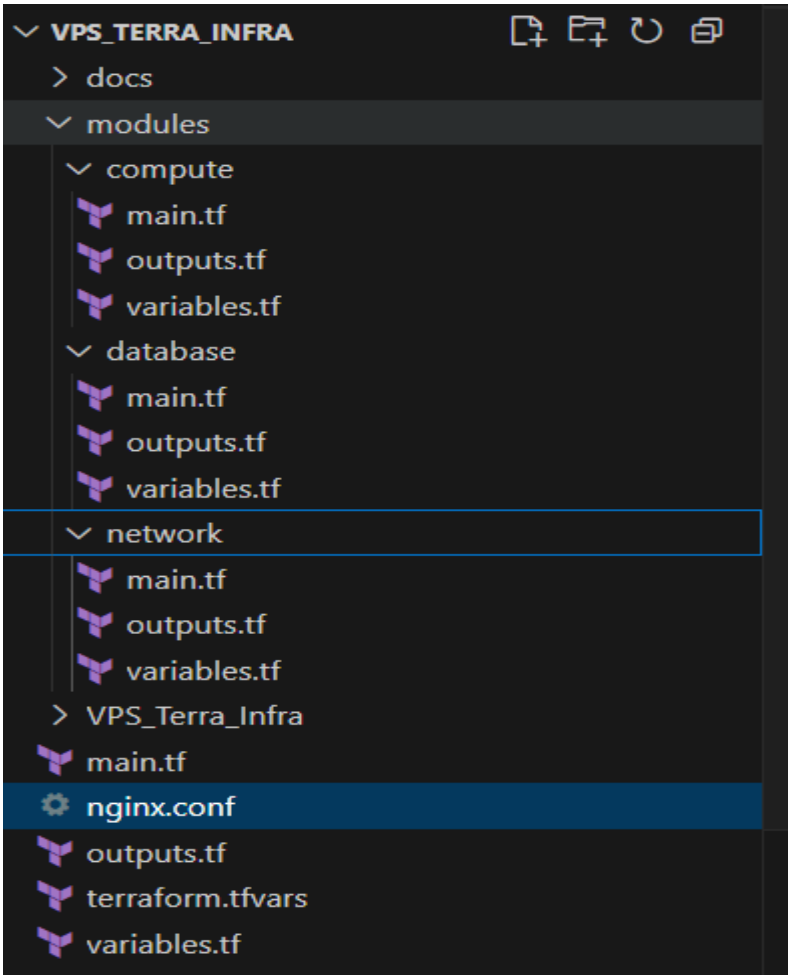
3. Terraform Modules

Module	Description	Resources
network	Manages Docker networks	docker_network.public, docker_network.private
compute	Creates application containers	docker_image.app, docker_container.app
database	Creates database container	docker_image.mysql, docker_container.db
main.tf	Manages load balancer and provider configuration	docker_image.nginx, docker_container.lb

4. Key Terraform Concepts

- **Providers**: kreuzwerker/docker is used for Docker container management.
- **Backend**: Local backend (terraform.tfstate) stores state file on the VPS.
- **Absolute Paths**: Required for Docker volume mounts.
- **Networks**: Containers communicate using Docker networks (private_net_id and public_net_id).
- **Outputs**:
    - lb_access_url: URL to access the NGINX load balancer.
    - app_ips: IP addresses of app containers.
    - db_endpoint: Database connection endpoint.

5. File Structure



6. Root Configuration (main.tf)

Providers and Backend

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
  backend "local" {
    path = "terraform.tfstate"
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}
```

- Uses the Docker provider to manage Docker resources on your VPS.
- Stores Terraform state locally (terraform.tfstate).

Modules

Network Module

```
module "network" {
  source = "./modules/network"
  env    = var.env
}
```

- Creates public and private Docker networks.
- Output: public_net_id, private_net_id.

Compute Module

```
module "compute" {
  source         = "./modules/compute"
  env            = var.env
  private_net_id = module.network.private_net_id
  app_count      = var.app_count
}
```

- Deploys multiple app containers (httpd:latest) on private network.
- Output: app_ips.

Database Module

```
module "database" {
  source         = "./modules/database"
  env            = var.env
  private_net_id = module.network.private_net_id
  db_password    = var.db_password
  db_name        = var.db_name
}
```

- Deploys MySQL container on private network.
- Output: db_endpoint.

Load Balancer

```
resource "docker_image" "nginx" {
  name = "nginx:latest"
}

resource "docker_container" "lb" {
  name  = "${var.env}-lb"
  image = docker_image.nginx.name

  networks_advanced {
    name = module.network.public_net_id
  }
  networks_advanced {
    name = module.network.private_net_id
  }

  ports {
    internal = 80
    external = 8082
  }

  volumes {
    host_path      = abspath("${path.module}/nginx.conf")
    container_path = "/etc/nginx/conf.d/default.conf"
    read_only      = true
  }
}
```

- Nginx container bridges **public and private networks**.
- Uses nginx.conf to route traffic to app containers.
- Exposes port 8082 on VPS.

7. Modules Explained

a. Network Module

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}

resource "docker_network" "public" {
  name   = "${var.env}-public-net"
  driver = "bridge"
}

resource "docker_network" "private" {
  name     = "${var.env}-private-net"
  driver   = "bridge"
  internal = true
}
~
```

- public network allows external access.
- private network is internal only for apps and DB communication.

b. Compute Module

```
terraform-user@srv878597:~/VPS_Terra_Infra/modules/compute$ vi main.tf
terraform-user@srv878597:~/VPS_Terra_Infra/modules/compute$ cat main.tf
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}


resource "docker_image" "app" {
  name = "httpd:latest"
}

resource "docker_container" "app" {
  count = var.app_count
  name  = "${var.env}-app-${count.index + 1}"
  image = docker_image.app.name

  networks_advanced {
    name = var.private_net_id
  }

  ports {
  internal = 80
  external = 8081
  }

}
```

- Deploys var.app_count HTTP app containers.

- Connected to the private network.
- count = var.app_count -> number of app containers.
- Outputs IPs for LB to use.

c. Database Module

```
terraform-user@srv878597:~/VPS_Terra_Infra/modules$ cd database/
terraform-user@srv878597:~/VPS_Terra_Infra/modules/database$ cat main.tf
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}

resource "docker_image" "mysql" {
  name = "mysql:8.0"
}

resource "docker_container" "db" {
  name  = "${var.env}-db"
  image = docker_image.mysql.name

  networks_advanced {
    name = var.private_net_id
  }

  env = [
    "MYSQL_ROOT_PASSWORD=${var.db_password}",
    "MYSQL_DATABASE=${var.db_name}",
    "MYSQL_REQUIRE_SECURE_TRANSPORT=ON"
  ]

  ports {
    internal = 3306
    external = 3307  # Free port on VPS
  }
}
```

- MySQL container on private network.
- Environment variables configure DB credentials.
- Exposes external port 3307 for optional access.

8. Outputs

- lb_access_url → Access load balancer via VPS public IP.
- db_endpoint → DB IP:port for internal or external access.
- pp_ips → List of all app container IPs in private network.

9. Nginx Config

```
terraform-user@srv878597:~/VPS_Terra_Infra$ cat nginx.conf
    upstream backend {
        server dev-app-1:80;
        server dev-app-2:80;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }
```

- Routes traffic from public network to private app containers.

10. Step-by-Step Workflow

a. Initialize Terraform

```
terraform-user@srv878597:~/VPS_Terra_Infra$ terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of kreuzwerker/docker from the dependency lock file
- Using previously-installed kreuzwerker/docker v3.6.2

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
terraform-user@srv878597:~/VPS_Terra_Infra$
```

- Downloads Docker provider (kreuzwerker/docker).
- Prepares local backend (terraform.tfstate).

b. Check What Will Happen

```
terraform-user@srv878597:~/VPS_Terra_Infra$ terraform plan
module.compute.docker_image.app: Refreshing state... [id=sha256:65005131d37e90347c3259856d51f35c505d260c308f2b7d0fc020a841dd1220httpd:latest]
module.network.docker_network.private: Refreshing state... [id=2f5d95b317a2bf9f0bb1b66d82f2f2d5c10973e95d0873500bcfa3b48d9fd6c1]
module.database.docker_image.mysql: Refreshing state... [id=sha256:6f19538dd7d235eca1d38163bdb4124d5b52a7bcae4ed20c8dd35af64dbd1434mysql:8.0]
module.network.docker_network.public: Refreshing state... [id=f335d8596410aa3de4c143fd3635f843f665e3cefd90ca01b75eef70ff38bc26]
docker_image.nginx: Refreshing state... [id=sha256:2cd1d97f893f70cee86a38b7160c30e5750f3ed6ad86c598884ca9c6a563a501nginx:latest]
module.compute.docker_container.app[1]: Refreshing state... [id=9e7ef816b591bdfa03ab4775b518ff943e340c84519a418567f31f201a73f439]
module.compute.docker_container.app[0]: Refreshing state... [id=1691fac133a42a1f12716bf7a02bd2fc64f8c97b1c8e4044a80e015a5868ece6]
docker_container.lb: Refreshing state... [id=bf988585208bb32f2f0f8458c5ff4ac4c03b0c75f75d30da7d9719096f9fb66a]
module.database.docker_container.db: Refreshing state... [id=136e2e18fec451bceeed0ca33e5b0e1a7eead50b53f855e99199220bb90236d4]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # module.compute.docker_container.app[0] must be replaced
-/+ resource "docker_container" "app" {
      + bridge                                      = (known after apply)
      ~ command                                     = [
          - "httpd-foreground",
        ] -> (known after apply)
      + container_logs                              = (known after apply)
      - cpu_shares                                  = 0 -> null
      - dns                                         = [] -> null
      - dns_opts                                    = [] -> null
      - dns_search                                  = [] -> null
      ~ entrypoint                                  = [] -> (known after apply)
      ~ env                                         = [] -> (known after apply)
      + exit_code                                   = (known after apply)
      - group_add                                   = [] -> null
      ~ hostname                                    = "1691fac133a4" -> (known after apply)
```

Terraform **simulates** what it will create:

- Networks → public & private
- App containers → 2 by default
- Database container
- Nginx load balancer

➢ It's like **drawing a blueprint** before building.

c. Apply Terraform

terraform apply

```
module.database.docker_container.db: Creating...
module.compute.docker_container.app[0]: Creation complete after 6s
module.compute.docker_container.app[1]: Creation complete after 5s
module.database.docker_container.db: Creation complete after 4s [i

Apply complete! Resources: 3 added, 0 changed, 3 destroyed.

Outputs:

app_ips = [
  "172.23.0.2",
  "172.23.0.3",
]
db_endpoint = "172.23.0.4:3306"
lb_access_url = "http://194.164.151.129:8082"
```

- Terraform actually **creates the resources**.

d. Check Outputs

```
terraform-user@srv878597:~/VPS_Terra_Infra$  terraform output
app_ips = [
  "172.23.0.2",
  "172.23.0.3",
]
db_endpoint = "172.23.0.4:3306"
lb_access_url = "http://194.164.151.129:8082"
```

- Open browser → http://194.164.151.129:8082 → traffic is routed to one of the app containers.
- Use mysql -h 172.23.0.4 -P 3306 -u root -p → connect to DB (inside private network).

```
terraform-user@srv878597:~/VPS_Terra_Infra$ mysql -h 172.23.0.4 -P 3306 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.43 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Verify the Containers status:

```
terraform-user@srv878597:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED             STATUS
          PORTS                                     NAMES
136e2e18fec4   mysql:8.0                            "docker-entrypoint.s…"   18 minutes ago      Up 18 minutes
                                                    dev-db
9e7ef816b591   httpd:latest                         "httpd-foreground"       18 minutes ago      Up 18 minutes
                                                    dev-app-2
1691fac133a4   httpd:latest                         "httpd-foreground"       18 minutes ago      Up 18 minutes
                                                    dev-app-1
bf988585208b   nginx:latest                         "/docker-entrypoint.…"   About an hour ago   Up About an hour
          0.0.0.0:8082->80/tcp                      dev-lb
```

```
terraform-user@srv878597:~/VPS_Terra_Infra$ terraform providers

Providers required by configuration:
.
├── provider[registry.terraform.io/kreuzwerker/docker] ~> 3.0
├── module.network
│   └── provider[registry.terraform.io/kreuzwerker/docker] ~> 3.0
├── module.compute
│   └── provider[registry.terraform.io/kreuzwerker/docker] ~> 3.0
└── module.database
    └── provider[registry.terraform.io/kreuzwerker/docker] ~> 3.0

Providers required by state:

    provider[registry.terraform.io/kreuzwerker/docker]
```

11.Buggs and Solution

A. Terraform Provider Issue

Problem:

When running terraform init, you got:

```
Could not retrieve the list of available versions for provider hashicorp/docker
```

Cause:

- Terraform was trying to use the **wrong provider** (hashicorp/docker) which does not exist in the registry.
- Your modules were implicitly depending on hashicorp/docker.

Solution:

- Explicitly specify the correct provider in terraform block:

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}
provider "docker" {
  host = "unix:///var/run/docker.sock"
}
```

B. Unsupported Arguments in docker_container\

Problem:

Running terraform plan gave errors:

```
An argument named "cpu_count" is not expected here.
An argument named "type/source/target" is not expected here.
```

Cause:

- kreuzwerker/docker provider v3+ does **not use cpu_count**.
- volumes { type, source, target } was the old syntax from hashicorp/docker.

Solution:

- Use the **current provider syntax**:

```
volumes {
  host_path      = abspath("${path.module}/nginx.conf")
  container_path = "/etc/nginx/conf.d/default.conf"
  read_only      = true
}
```

- Remove unsupported fields (cpu_count, type, source, target).

C. Path Issue for Volume Mount

Problem:

```
'./nginx.conf' must be an absolute path
```

Cause:

- Docker provider requires **absolute paths** for host volume mounts.

Solution:

- Use abspath():

```
host_path = abspath("${path.module}/nginx.conf")
```

D. LB Container Exited Immediately

Problem:

```
Error: container exited immediately
```

- Even after terraform apply, LB container kept crashing.

Cause:

- Your nginx.conf included:

```
events {}
http { ... }
```

- When mounted to /etc/nginx/conf.d/default.conf, the extra http {} caused **nested http blocks**, which NGINX rejects.
- Also, LB might not have been on the same network as app containers, so dev-app-1/dev-app-2 could not be resolved.

Solution:

- Correct nginx.conf:

```
upstream backend {
    server dev-app-1:80;
    server dev-app-2:80;
}

server {
    listen 80;

    location / {
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

  - Mount to /etc/nginx/conf.d/default.conf.
  - Ensure LB is connected to the **same private network** as app containers.

E. App Not Accessible in Browser

Problem:

- Containers were running, but visiting http:// 194.164.151.129:8082 returned nothing.

Cause:

- NGINX LB was failing to start (due to nginx.conf errors).
- App containers were on private network, LB was not able to resolve them.

Solution:

- Fixed nginx.conf as above.
- LB connected to **both private and public networks**:

```
networks_advanced {
  name = module.network.public_net_id
}
networks_advanced {
  name = module.network.private_net_id
}
```

  - Exposed LB port 8082 to VPS.