Terraform 워크스페이스

테라폼 기본 개념

프로비저닝(Provisioning)

어떤 프로세스나 서비스를 실행하기 위한 준비 단계를 프로비저닝이라고 이야기합니다. 프로비저닝에는 크게 네트워크나 컴퓨팅 자원을 준비하는 작업과 준비된 컴퓨팅 자원에 사이트 패키지나 애플리케이션 의존성을 준비하는 단계로 나뉘어집니다. 명확한 경계는 불분명하지만 테라폼은 전자를 주로 다루는 도구입니다.

프로바이더(Provider)

테라폼과 외부 서비스를 연결해주는 기능을 하는 모듈입니다. 예를 들어 테라폼으로 AWS 서비스의 컴퓨팅 자원을 생성하기 위해서는 `aws` 프로바이더를 먼저 셋업해야합니다. 프로바이더로는 AWS, 구글 클라우드 플랫폼(Google Cloud Platform), 마이크로소프트 애저(Microsoft Azure)와 같은 범용 클라우드 서비스를 비롯해 깃허브(Github), 데이터도그(Datadog), DNSimple과 같은 특정 기능을 제공하는 서비스, MySQL, 레빗MQ(RabbitMQ), 도커(Docker)와 같은 로컬 서비스 등을 지원합니다. 전체 목록은 테라폼 프로바이더 문서에서 찾아볼 수 있습니다.

리소스(자원:Resource)

리소스란 특정 프로바이더가 제공해주는 조작 가능한 대상의 최소 단위입니다. 예를 들어 AWS 프로바이더는 `aws_instance` 리소스 타입을 제공하고, 이 **리소스 타입**을 사용해 Amazon EC2의 가상 머신 **리소스**를 선언하고 조작하는 것이 가능합니다. EC2 인스턴스, 시큐리티 그룹, 키 페어 모두 `aws` 프로바이더가 제공해주는 리소스 타입입니다.

HCL(Hashicorp Configuration Language)

HCL은 테라폼에서 사용하는 설정 언어입니다. 테라폼에서 모든 설정과 리소스 선언은 HCL을 사용해 이루어집니다. 테라폼에서 HCL 파일의 확장자는 `.tf`를 사용합니다.

계획(Plan)

테라폼 프로젝트 디렉터리 아래의 모든 `.tf` 파일의 내용을 실제로 적용 가능한지 확인하는 작업을 계획이라고 합니다. 테라폼은 이를 `terraform plan` 명령어로 제공하며, 이 명령어를 실행하면 어떤 리소스가 생성되고, 수정되고, 삭제될지 계획을 보여줍니다.

적용(Apply)

테라폼 프로젝트 디렉터리 아래의 모든 `.tf` 파일의 내용대로 리소스를 생성, 수정, 삭제하는 일을 적용이라고 합니다. 테라폼은 이를 `terraform apply` 명령어로 제공합니다. 이 명령어를 실행하기 전에 변경 예정 사항은 `plan` 명령어를 사용해 확인할 수 있습니다. 적용하기 전에도 플랜의 결과를 보여줍니다.

terraform 디렉터리 구조

- minimal 예제

```
├── README.md
├── main.tf
├── variables.tf
└── outputs.tf
```

- complete 예제

```
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
├── ...
├── modules/
│   ├── nestedA/
│   │   ├── README.md
│   │   ├── variables.tf
│   │   ├── main.tf
│   │   └── outputs.tf
│   ├── nestedB/
│   ├── .../
├── examples/
```

│ ├── exampleA/

│ │ ├── main.tf

│ ├── exampleB/

│ ├── .../

- 여기에 추가로 커스텀하게 아래의 파일들을 붙히기도 합니다.

├── provider.tf

├── backend.tf

- main.tf : 테라폼 CLI를 사용하여 apply 명령어를 사용하면 가장 먼저 main소스 코드를 동작시킵니다. 말 그대로 main
- modules : 자바로 보면 하나의 클래스를 만드는 것과 비슷한 개념입니다. main에서 input 값을 지정하고 해당 모듈을 사용할 수 있습니다.
- backend.tf : 테라폼은 형상관리를 위해 .tfstate 파일을 생성합니다. 이 파일을 backup하고 형상관리 하기위한 설정을 정의합니다.
- provider.tf : 리소스를 어디서 제공하는지, 버전은 어떤것인지 등을 설정합니다.
- outputs.tf : 해당 파일에 설정을 통해서 소스코드에 대한 실행 결과를 출력할 수 있습니다.
- variables.tf : 소스코드에 사용할 변수들을 정의 합니다.

Ubuntu에 테라폼 설치

```
$ sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install terraform

# 테라폼 명령어 자동완성 기능 설정
# bash 설정
$ touch ~/.bashrc
$ terraform -install-autocomplete

# zsh 설정
$ touch ~/.zshrc
$ terraform -install-autocomplete

# 참고사이트 : https://www.terraform.io/cli/config/config-file
# workspace에 필요한 중복되는 plugin 하나만 설치
$ vim ~/.terraformrc
# Configuration File Syntax
plugin_cache_dir   = "$HOME/.terraform.d/plugin-cache"
disable_checkpoint = true

$ mkdir -p ~/.terraform.d/plugin-cache
```

테라폼 워크스페이스 생성

워크스페이스 : 테라폼을 관리하기 위한 하나의 프로젝트 단위

Terraform Registry

https://registry.terraform.io/providers/hashicorp/local/latest/docs/resources/file

terraform alias 지정

```
$ nvim ~/.zshrc
# 밑에 추가
alias "tf"="terraform"

$ source ~/.zshrc

또는

$ vim ~/.bashrc
# 밑에 추가
alias "tf"="terraform"

$ source ~/.bashrc
```

Local resource

```
$ vim main.tf

provider "local" {
}

# Infra resource
resource "local_file" "foo" {
    # path.module - String inter pdation terraform 변수
    # ${path.module} : string interpolation 문법
    # 파일이 위치한 디렉토리 경로, main.tf
    filename = "${path.module}/foo.txt"
    content = "Hello World!"
}

# 테라폼 초기화
$ terraform init
$ ls -al
drwxr-xr-x  3 ubuntu ubuntu 4096 Jan 21 09:40 .terraform
-rw-r--r--  1 ubuntu ubuntu 1078 Jan 21 09:40 .terraform.lock.hcl
-rw-r--r--  1 ubuntu ubuntu  124 Jan 21 09:40 main.tf

# 프로바이드의 데이터를 다운받아서 저장
```

```
$ tree .terraform/
.terraform
└── providers
    └── registry.terraform.io
        └── hashicorp
            └── local
                └── 2.2.3
                    └── linux_amd64
                        └── terraform-provider-local_v2.2.3_x5

# 협업에 필요한 데이터 저장
$ cat .terraform.locl.hcl
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/local" {
  version = "2.2.3"
  hashes = [
    "h1:aWp5iSUxBGgPv1UnV5yag9Pb0N+U1I0sZb38AXBFO8A=",
    "zh:04f0978bb3e052707b8e82e46780c371ac1c66b689b4a23bbc2f58865ab7d5c0",
    "zh:6484f1b3e9e3771eb7cc8e8bab8b35f939a55d550b3f4fb2ab141a24269ee6aa",
    "zh:78a56d59a013cb0f7eb1c92815d6eb5cf07f8b5f0ae20b96d049e73db915b238",
    "zh:78d5eefdd9e494defcb3c68d282b8f96630502cac21d1ea161f53cfe9bb483b3"
    ...
  ]
}

$ terraform plan
Terraform used the selected providers to generate the following
execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # local_file.foo will be created
  + resource "local_file" "foo" {
      + content              = "Hello World!"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "./foo.txt"
      + id                   = (known after apply)
    }

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

$ terraform apply
Terraform used the selected providers to generate the following
execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # local_file.foo will be created
  + resource "local_file" "foo" {
      + content              = "Hello World!"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "./foo.txt"
      + id                   = (known after apply)
    }

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

Do you want to perform these actions?
  Terraform will perform the actions described above.
```

```
  Only 'yes' will be accepted to approve.

  Enter a value: yes # yes 입력

local_file.foo: Creating...
local_file.foo: Creation complete after 0s [id=2ef7bde608ce5404e97d5f042f95f89f1c232871]

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

$ ls -al
drwxr-xr-x  3 ubuntu ubuntu 4096 Jan 21 09:48 .
drwxr-xr-x 15 ubuntu ubuntu 4096 Jan 21 09:49 ..
drwxr-xr-x  3 ubuntu ubuntu 4096 Jan 21 09:40 .terraform
-rw-r--r--  1 ubuntu ubuntu 1078 Jan 21 09:40 .terraform.lock.hcl
-rwxr-xr-x  1 ubuntu ubuntu   12 Jan 21 09:48 foo.txt
-rw-r--r--  1 ubuntu ubuntu  124 Jan 21 09:40 main.tf
-rw-r--r--  1 ubuntu ubuntu  829 Jan 21 09:48 terraform.tfstate

$ cat foo.txt
Hello World!

# 테라폼 상태 파일
$ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.1.4",
  "serial": 1,
  "lineage": "e43e603f-6a95-f4bd-6609-7dd20fae085f",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "foo",
      "provider": "provider[\"registry.terraform.io/hashicorp/local\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "Hello World!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "./foo.txt",
            "id": "2ef7bde608ce5404e97d5f042f95f89f1c232871",
            "sensitive_content": null,
            "source": null
          },
          "sensitive_attributes": [],
          "private": "bnVsbA=="
        }
      ]
    }
  ]
}
```

출력 파일 지정

```
$ vim main.tf
provider "local" {
}
```

```
resource "local_file" "foo" {
  filename = "${path.module}/foo.txt"
  content = "Hello World!"
}
# 데이터파일 지정
data "local_file" "bar" {
  filename = "${path.module}/bar.txt"
}

$ cat > bar.txt
Hello DevOps!
^C
$ cat bar.txt
Hello DevOps!

$ terraform apply
local_file.foo: Refreshing state... [id=2ef7bde608ce5404e97d5f042f95f89f1c232871]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.

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


$ vim main.tf
   ....
# 데이터 출력파일 지정
output "file_bar" {
    value = data.local_file.bar
}

$ terraform apply
local_file.foo: Refreshing state... [id=2ef7bde608ce5404e97d5f042f95f89f1c232871]

# 인프라는 변경이 없지만 Output가 변경
Changes to Outputs:
  + file_bar = {
      + content        = <<-EOT
            Hello DevOps!
        EOT
      + content_base64 = "SGVsbG8gRGV2T3BzIQo="
      + filename       = "./bar.txt"
      + id             = "7397c8d706f70c34e3c27f634b6e7e437997992b"
    }

You can apply this plan to save these new output values to the
Terraform state, without changing any real infrastructure.

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

  Enter a value: yes


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

Outputs:

file_bar = {
  "content" = <<-EOT
  Hello DevOps!

  EOT
  "content_base64" = "SGVsbG8gRGV2T3BzIQo="
```

```
  "filename" = "./bar.txt"
  "id" = "7397c8d706f70c34e3c27f634b6e7e437997992b"
}
```

완성된 main.tf

```
provider "local" {
}

resource "local_file" "foo" {
    filename = "${path.module}/foo.txt"
    content = "Hello World!"
}

data "local_file" "bar" {
  filename = "${path.module}/bar.txt"
}

output "file_bar" {
  value = data.local_file.bar
}
```

AWS Provider

Terraform Registry

https://registry.terraform.io/providers/hashicorp/aws/latest/docs

```
# AWS CLI 설치는 여기로 > 링크
$ aws sts get-caller-identity
{
    "UserId": "551675860467",
    "Account": "551675860467",
    "Arn": "arn:aws:iam::551675860467:root"
}
```

```
$ vim main.tf
# 하나의 리전은 하나의 프로바이드에 연결
provider "aws" {
  region = "ap-northeast-2"
}

# VPC 생성
resource "aws_vpc" "vpc-std22" {
  cidr_block = "10.0.0.0/16"
```

```
  tags = {
    "Name" = "busanit-std22"
  }
}
```

Terraform Registry

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/vpcs

AWS VPC 수정

```
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_vpc" "vpc-std22" {
  cidr_block = "10.0.0.0/16"
}

data "aws_vpcs" "vpc--std22" {
  tags = {
    service = "production"
  }
}

output "vpc-std22" {
    value = aws_vpc.vpc-std22.ids
}
```

main.tf

```
provider "aws" {
  region = "ap-northeast-2"
}

# VPC 생성
resource "aws_vpc" "vpc-std22" {
  cidr_block = "10.0.0.0/16"
}

data "aws_vpcs" "this" {}

output "vpc-std22" {
    value = data.aws_vpcs.this
}
```