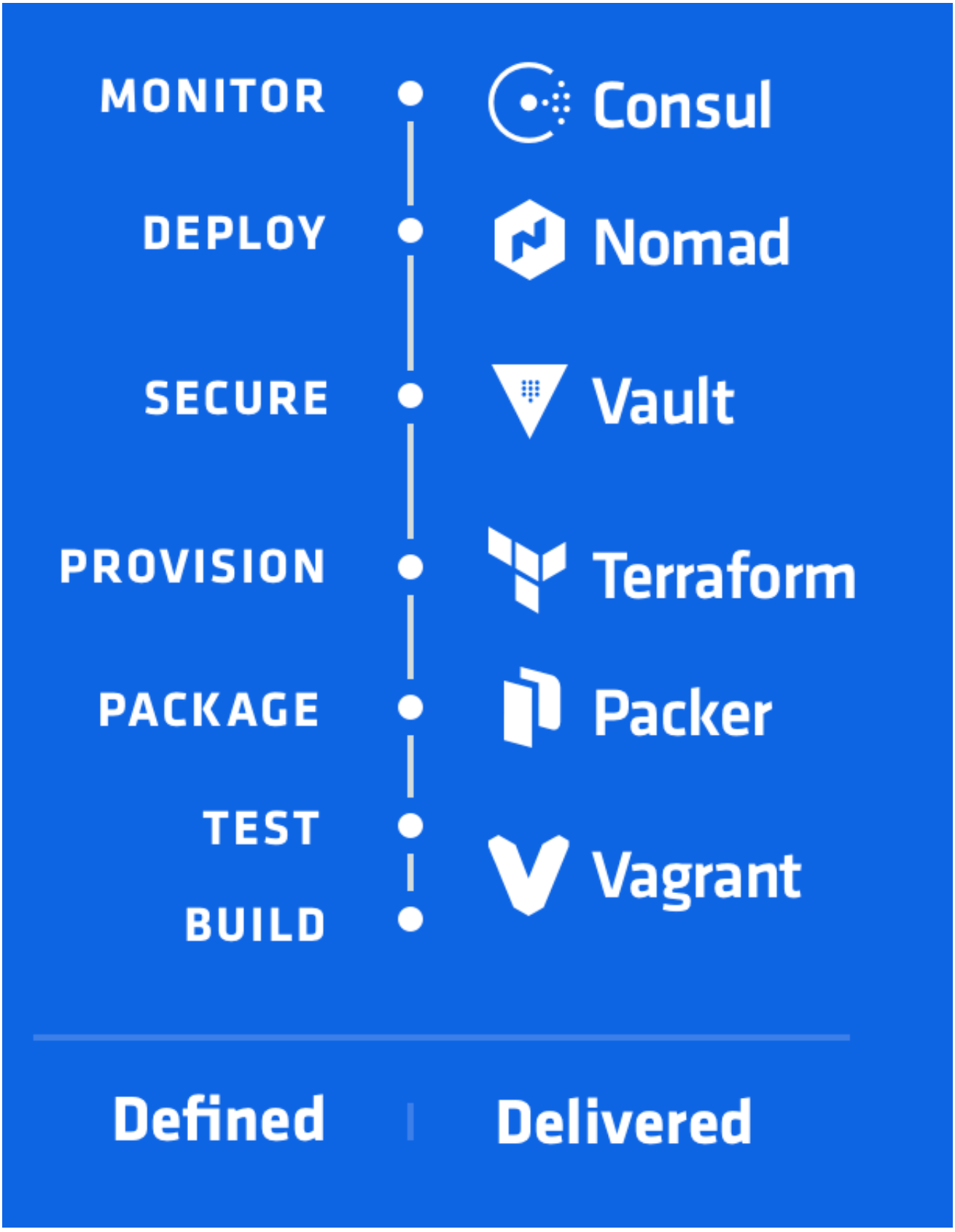
# Terraform

## Table of Contents



---

Hashicorp가 정의한 DevOps 단계 (https://www.hashicorp.com/devops.html)에 따르면 Provision 단계에 속하는 도구



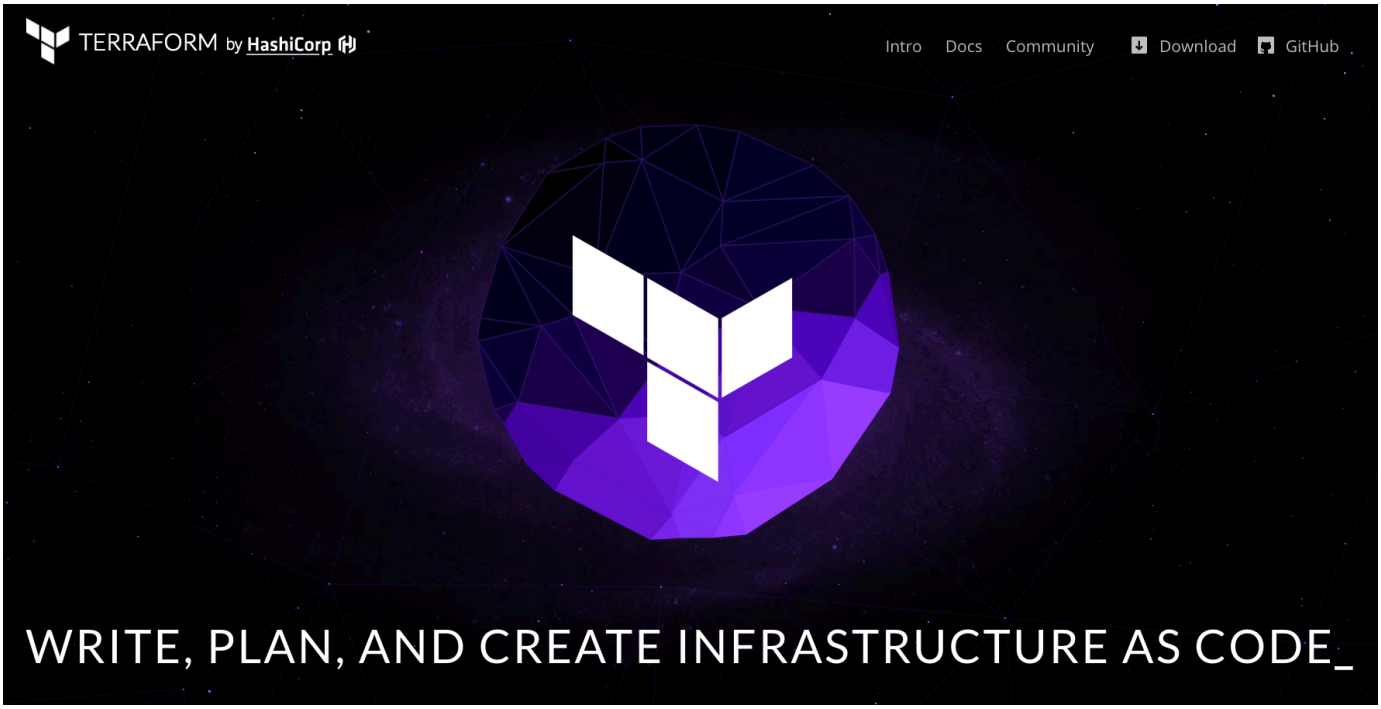

Provisionning

> *Config Management is part of provisioning. Basically, that's using a tool like Chef, Puppet or Ansible to configure your server. "Provisioning" often implies it's the first time you do it. Config management usually happens repeatedly. - What's Deployment Versus Provisioning Versus Orchestration? (http://codefol.io/posts/deployment-versus-provisioning-versus-orchestration)*

## 왜 Terraform을 보게 되었는가?

Apex로 AWS Lambda를 이용할 때 Apex가 Terraform을 사용하고 있어서 사용하게 됨. 다른 인프라와 달리 Lambda와 연동하는 API Gateway나 S3, 리소스 권한 등은 Lambda와 생명주기를 같이 해야 하고 AWS에서 연동된 서비스를 한꺼번에 볼 수 있는 화면이 없기 때문에 Terraform을 이용한 인프라 관리가 필요하다고 생각됨.

## Terraform이란?




### Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.

- Infrastructure as Code
- Execution Plans
- Resource Graph
- Change Automation

원하는 Provider (https://www.terraform.io/docs/providers/)에 인프라를 생성하고 관리할 수 있다.

#### 제공하는 프로바이더

- AWS
- BitBucket
- Chef

- CloudFlare
- Consul
- DigitalOcean
- Docker
- GitHub
- Google Cloud
- Grafana
- InfluxDB
- Heroku
- Microsoft Azure
- MySQL
- PostgreSQL

## HashiCorp configuration language

- Terraform의 설정 파일 (https://www.terraform.io/docs/configuration/index.html)은 Terraform 형식( `.tf` )과 JSON( `.tf.json` ) 둘다 사용 가능하다.
  - Terraform 형식은 HashiCorp configuration language(HCL) (https://github.com/hashicorp/hcl)의 문법을 따른다.
- 지정한 폴더의 `.tf` , `.tf.json` 를 알파벳순으로 로드한다.
  - 오버라이드 파일은 다른 파일을 모두 로드한 뒤에 알파벳순으로 로드한다.
    - 오버라이드 파일은 파일명이 `override` 이거나 파일명이 `_override` 로 끝나야 한다.
  - 정의된 변수나 리소드의 정의된 순서는 상관없다.

*sample.tf*

```JS
# An AMI (1)
variable "ami" {
description = "the AMI to use" (2) (3)
}

/* A multi
line comment. */ (4)
resource "aws_instance" "web" {
ami = "${var.ami}" (5)
count = 2
source_dest_check = false
description = <<EOF
...
...
EOF (6)

connection {
user = "root"
}

}
```

1. 주석은 `#` 를 사용
2. 값 할당은 `key = value` 형식으로 하고 value에는 문자열, 숫자, 불리언, 리스트, 맵이 모두 가능하다.
3. 문자열은 쌍따옴표를 사용
4. 멀티라인 주석은 `/* */` 를 사용
5. 스트링 인터폴레이션은 `${}`를 사용
6. 멀티라인 문자열은 here doc 스타일로 ``<<EOF`와 `EOF`를 사용

## 리소스 설정

```JS
resource TYPE NAME {
CONFIG ... (1)
[count = COUNT]
[depends_on = [RESOURCE NAME, ...]]
[provider = PROVIDER]

[LIFECYCLE]

[CONNECTION]
[PROVISIONER ...]
}
```

1. CONFIG는 `KEY = VALUE` 이거나 `KEY { CONFIG }` 가 된다.

예시:

```JS
resource "aws_instance" "web" {
ami = "ami-408c7f28"
instance_type = "t1.micro"
}
```

## 데이터 소스

Terraform 설정외의 다른 곳에서 데이터를 가져올 수 있다.

```JS
data "aws_ami" "web" {
filter {
name = "state"
values = ["available"]
}
filter {
name = "tag:Component"
values = ["web"]
}
most_recent = true
}
```

`Component = web` 태그가 붙은 최신 AMI를 조회한다.

## Provider

리소스의 라이프 사이클을 관리할 책임을 진다. 테라폼에서 모든 리소스는 접두사로 프로바이더에 매칭된다. 예를 들어 `aws_instance`는 `aws` 프로바이더와 매칭된다.

```
provider NAME {
CONFIG ...
[alias = ALIAS]
}
```

예시:

```
provider "aws" {
access_key = "foo"
secret_key = "bar"
region = "us-east-1"
}
```

## 변수

변수는 CLI에게 덮어쓸 수 있다.

```
variable NAME {
[type = TYPE]
[default = DEFAULT]
[description = DESCRIPTION]
}
```

예시:

```
variable "key" {
type = "string"
}

variable "images" {
type = "map"

default = {
us-east-1 = "image-1234"
us-west-2 = "image-4567"
}
}

variable "zones" {
default = ["us-east-1a", "us-east-1b"]
}
```

## 모듈

리소스 그룹을 모듈화하고 캡슐화한다.

```
module NAME {
source = SOURCE_URL (1)

CONFIG ...
}
```

(1) source는 필수값으로 모듈을 다운로드 받을 경로를 지정한다.

예시:

```
module "consul" {
source = "github.com/hashicorp/consul/terraform/aws"
servers = 5
}
```

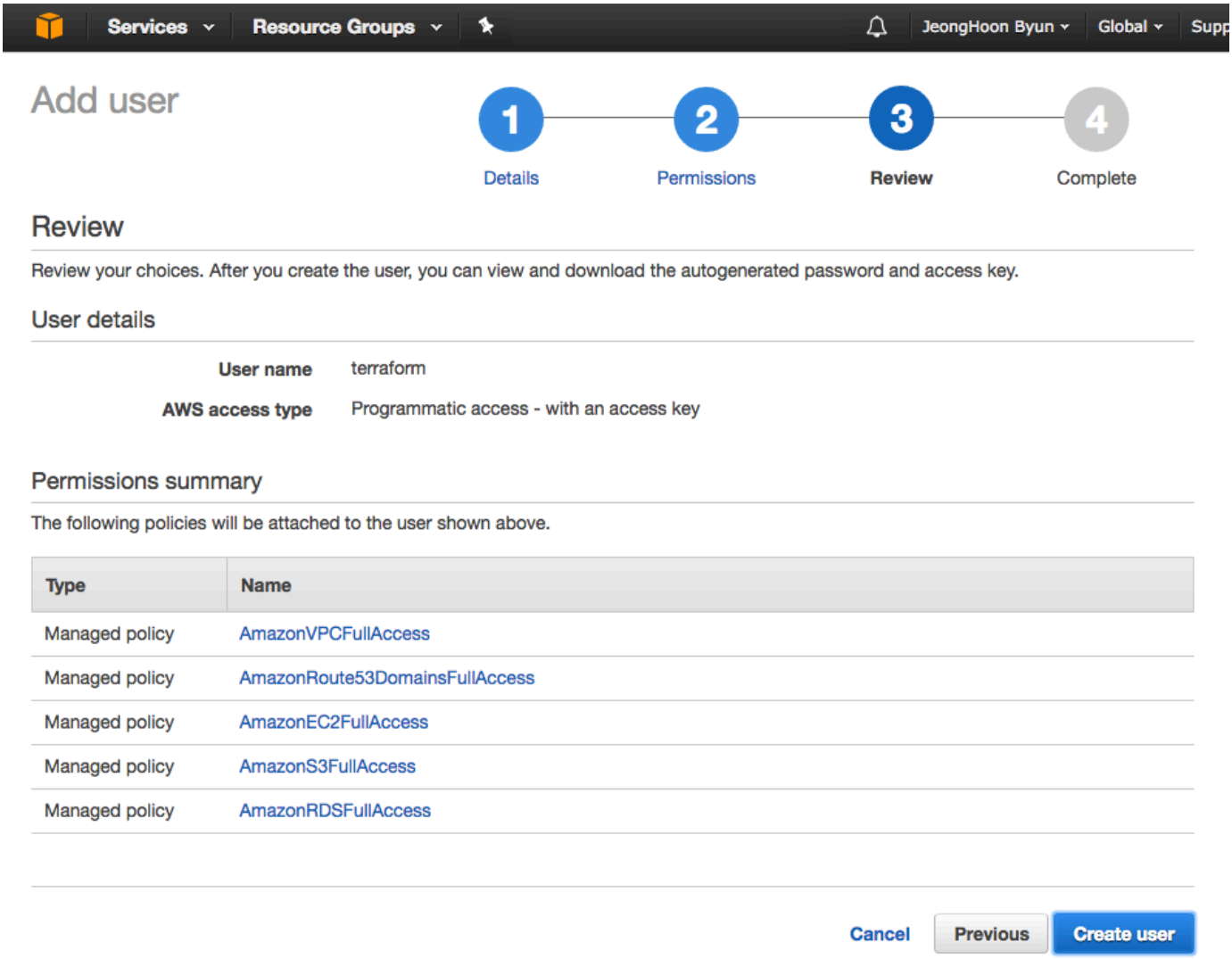
# Terraform 설정

```
$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
apply Builds or changes infrastructure
destroy Destroy Terraform-managed infrastructure
fmt Rewrites config files to canonical format
get Download and install modules for the configuration
graph Create a visual graph of Terraform resources
import Import existing infrastructure into Terraform
init Initializes Terraform configuration from a module
output Read an output from a state file
plan Generate and show an execution plan
push Upload this Terraform module to Atlas to run
refresh Update local state file against real resources
remote Configure remote state storage
show Inspect Terraform state or plan
taint Manually mark a resource for recreation
untaint Manually unmark a resource as tainted
validate Validates the Terraform files
version Prints the Terraform version

All other commands:
state Advanced state management
```

# Terraform으로 AWS 환경설정 하기

AWS에 웹서비스 환경을 프로비저닝하자.

## IAM에 계정 생성




## 프로바이더 설정

*sample.tf*

```JS
provider "aws" {
access_key = "AKIAJF5ZRZWTPVCPZECQ"
secret_key = "a83wGkwATrMgl6hZTLqNUnNKpFrKisqfWVKW45MF"
region = "ap-northeast-1"
}
```

AWS를 사용하려면 AWS Provider 문서 (https://www.terraform.io/docs/providers/aws/)를 참고해야 한다.

프로비전이 가능한 리소스

- API Gateway
- App Autoscaleing
- CloudFomation
- CloudFront
- CloudTrail
- CloudWatch
- CodeCommit
- CodeDeploy

- Directory Service
- DynamoDB
- EC2
- ECS
- EFS
- ElasticCache
- Elastic Beanstalk
- Elastic Map Reduce
- ElasticSearch
- Glacier
- IAM
- Kinesis
- Kinesis Firehose
- KMS
- Lambda
- OpsWorks
- RDS
- RedShift
- WAF
- Route53
- S3
- SES
- SimpleDB
- SNS
- SSM
- SQS
- VPC

## VPC 설정

*aws-vpc.tf*

```JS
resource "aws_vpc" "study" {
cidr_block = "172.10.0.0/20"
tags {
Name = "study"
}
}

resource "aws_subnet" "study-a" {
vpc_id = "${aws_vpc.study.id}"
cidr_block = "172.10.0.0/24"
availability_zone = "ap-northeast-1a"
}

resource "aws_subnet" "study-c" {
vpc_id = "${aws_vpc.study.id}"
cidr_block = "172.10.1.0/24"
availability_zone = "ap-northeast-1c"
}
```

*aws-security-group.tf*

```JS
resource "aws_security_group" "study-allow-all" {
name = "study-allow_all"
description = "Allow all inbound traffic"
vpc_id = "${aws_vpc.study.id}"

ingress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}

egress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}
}
```

## AWS 인프라 적용

`terraform plan`으로 미리 적용되는 프로비전을 테스트해볼 수 있다.

```BASH
$ terraform plan
```

plan에 이상이 없으면 `terraform apply`로 적용한다.

```BASH
$ terraform apply
```

적용되면 적용된 내용이 `terraform.tfstate`에 저장되고 이후 변경하면 `terraform.tfstate.backup`하나 더 상태이다. 이후 수정하려면 기존 상태가 필요하므로 이 파일도 저장하고 있어야 한다.

`terraform show`로 적용된 상태를 확인할 수 있다.

```bash
$ terraform show
```

## EC2 설정

key-pair는 현재 새로 생성은 할 수 없고 AWS에 설정된 것을 불러올 수만 있다 (https://www.terraform.io/docs/providers/aws/r/key_pair.html). AWS에 이미 있는 리소스를 불러올 때는 import를 사용한다. `import`를 사용할때는 환경변수를 제공해야 한다.

```bash
$ AWS_ACCESS_KEY_ID=AKIAJF5ZRZWTPVCPZECQ \
AWS_SECRET_ACCESS_KEY=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 \
AWS_DEFAULT_REGION=ap-northeast-1 \
terraform import aws_key_pair.mykey study
```

Import를 하면 설정이 만들어지지는 않고 상태만 바뀌므로 설정은 직접 만들어야 한다. `key_pair` 같은 경우는 public key가 없어서 어떻게 쓰라는 건지 알수가 없다. 차라리 변수를 쓰는게 나아보임.

*aws-ec2.tf*

```js
variable "key_pair" {
default = "study"
}
```

*aws-ec2.tf*

```js
data "aws_ami" "ubuntu" {
most_recent = true
filter {
name = "name"
values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
}
filter {
name = "virtualization-type"
values = ["hvm"]
}
owners = ["099720109477"] # Canonical
}

resource "aws_instance" "study-server" {
ami = "${data.aws_ami.ubuntu.id}"
instance_type = "t2.micro"
subnet_id = "${aws_subnet.study-a.id}"
vpc_security_group_ids = ["${aws_security_group.study-allow-all.id}"]
key_name = "${var.key_pair}"
count = 3
tags {
Name = "example"
}
}
```

# 인프라 리소스 그래프 보기

```bash
$ terraform graph
```

## 인프라 삭제

```bash
$ terraform plan -destroy
```

```bash
$ terraform destroy
```

Last updated 2016-12-14 11:47:54 KST