IaC를 이용하여 클라우드에 wordpress 배포하기

Cloud BootCamp 4기
3차 미니 팀 프로젝트

F조
김효진
선우지훈
신소희
홍성민

CONTENTS

01

프로젝트 개요

ANSIBLE

Python으로 구현된 오픈소스로써 서버의 프로비저닝, SW 배포 등 인프라를 구성하고 관리하는 것을 자동화할 수 있는 IaC 도구

HashiCorp Terraform

HashiCorp에서 오픈소스로 개발 중인 인프라스트럭쳐 구축 및 운영의 자동화를 지향하는 IaC 도구

Terraform을 이용해 Terraform Provider인 AWS와 Azure에 리소스를 생성한 뒤

Ansible을 이용해 wordpress를 배포해보는 프로젝트를 진행한다.

02

프로젝트 진행 요약

역할 분담

선우지훈

terraform을 이용한 Azure 리소스 배포
markdown 문서 정리

김효진

Ansible, Terraform 초기 세팅
markdown 문서 정리

신소희

terraform을 이용한 AWS 리소스 배포
markdown 문서 정리 및 발표 자료 제작

홍성민

Ansible Playbook 작성
markdown 문서 정리

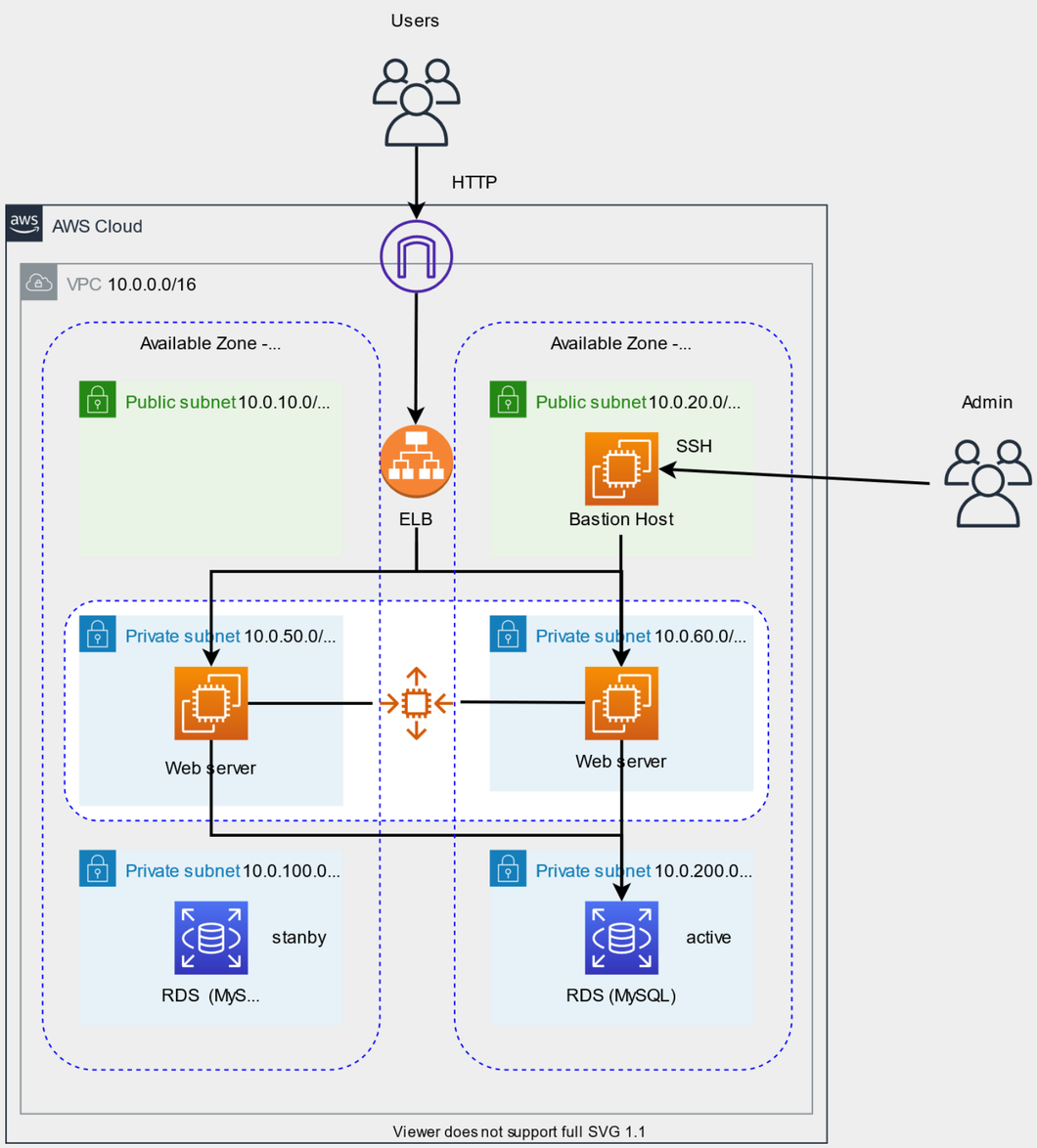
프로젝트 일정

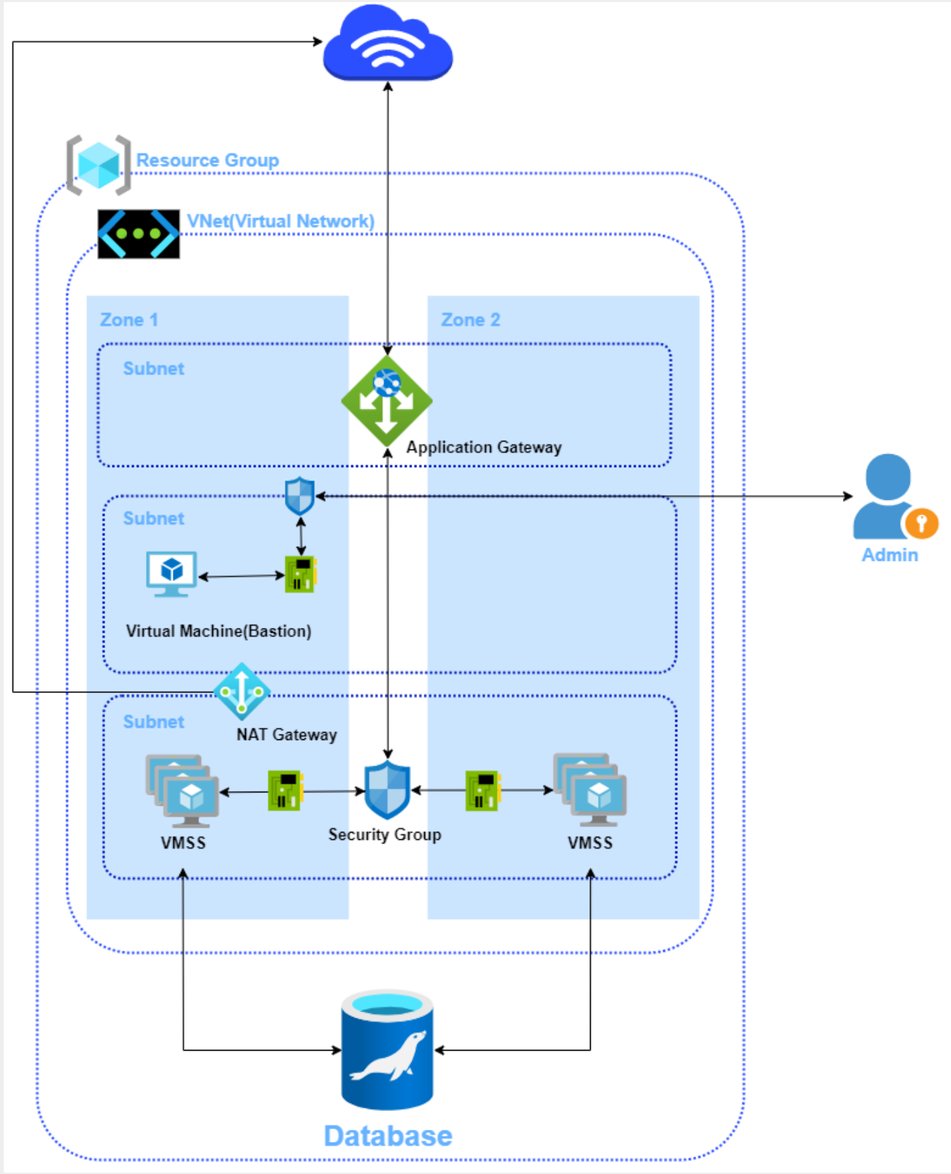
	4/26	4/27	4/28	4/29	4/30	5/1	5/2
아키텍처 설계							
초기 환경 세팅							
Playbook 작성							
Terraform 작성 (AWS)							
Terraform 작성 (Azure)							
Terraform 실행 및 동작 확인							
문서 작성 및 발표 자료 준비							

03

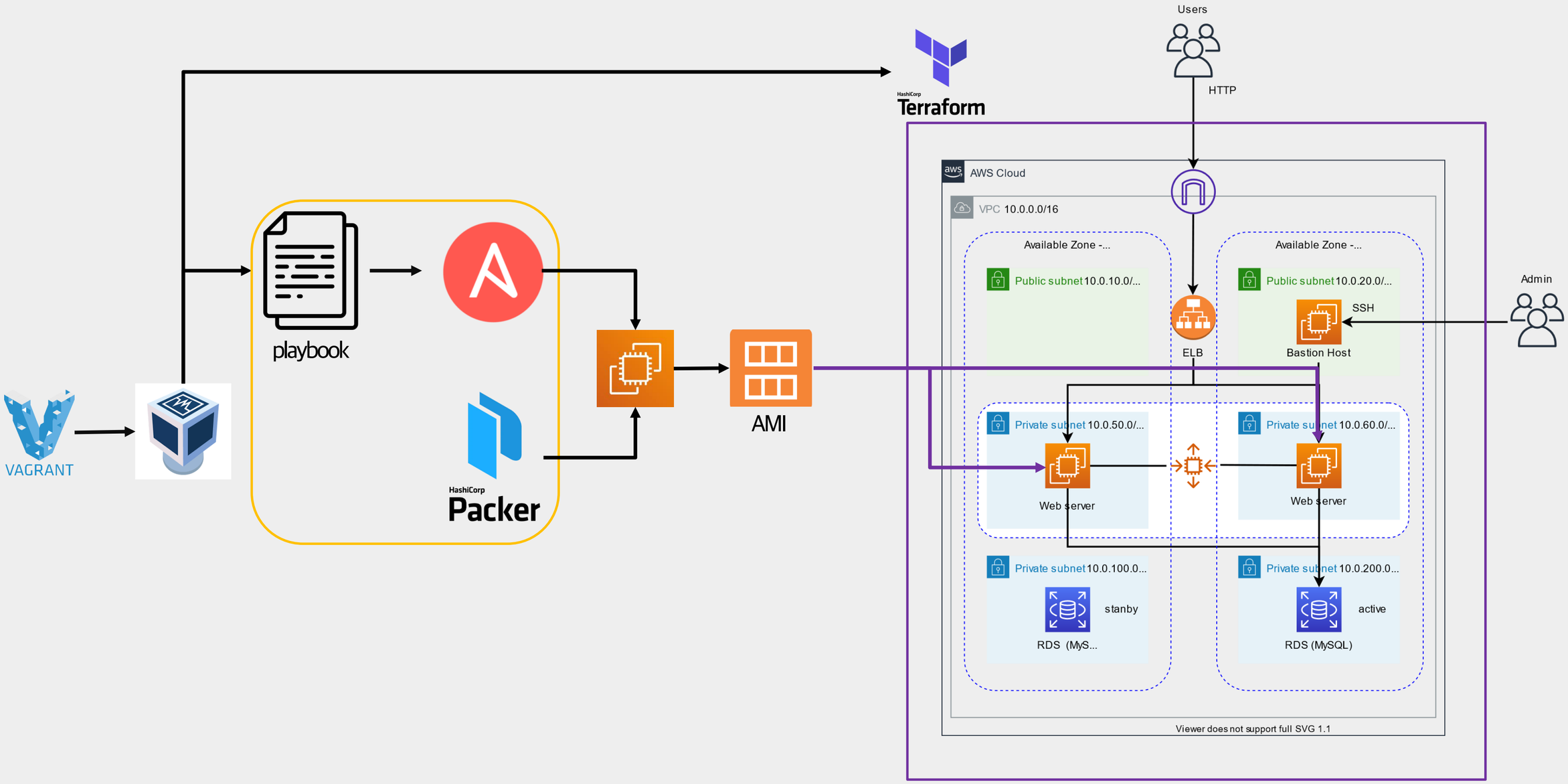
프로젝트 설계

시스템아키텍처


Users
HTTP
AWS Cloud
VPC 10.0.0.0/16
Available Zone -...
Available Zone -...
Public subnet 10.0.10.0/...
Public subnet 10.0.20.0/...
ELB
SSH
Bastion Host
Admin
Private subnet 10.0.50.0/...
Private subnet 10.0.60.0/...
Web server
Web server
Private subnet 10.0.100.0...
Private subnet 10.0.200.0...
stanby
active
RDS (MyS...
RDS (MySQL)
Viewer does not support full SVG 1.1


Resource Group
VNet(Virtual Network)
Zone 1
Zone 2
Subnet
Application Gateway
Subnet
Virtual Machine(Bastion)
Admin
Subnet
NAT Gateway
VMSS
Security Group
VMSS
Database

전체 시스템 동작 구조

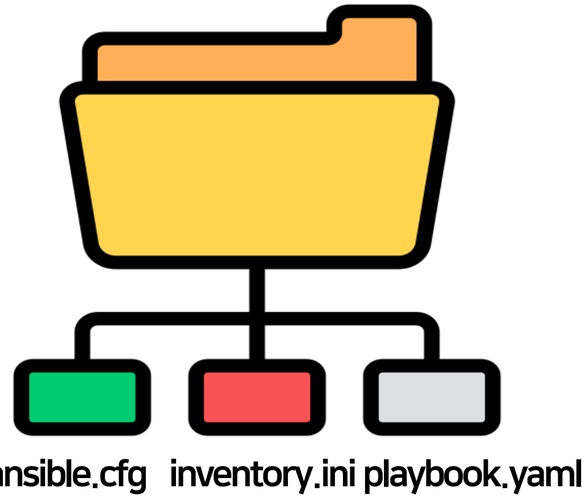

Users
HTTP
HashiCorp
Terraform
playbook
VAGRANT
HashiCorp
Packer
AMI
aws
AWS Cloud
VPC 10.0.0.0/16
Available Zone -...
Available Zone -...
Public subnet 10.0.10.0/...
Public subnet 10.0.20.0/...
ELB
SSH
Bastion Host
Admin
Private subnet 10.0.50.0/...
Private subnet 10.0.60.0/...
Web server
Web server
Private subnet 10.0.100.0...
Private subnet 10.0.200.0...
stanby
active
RDS (MyS...
RDS (MySQL)
Viewer does not support full SVG 1.1

04

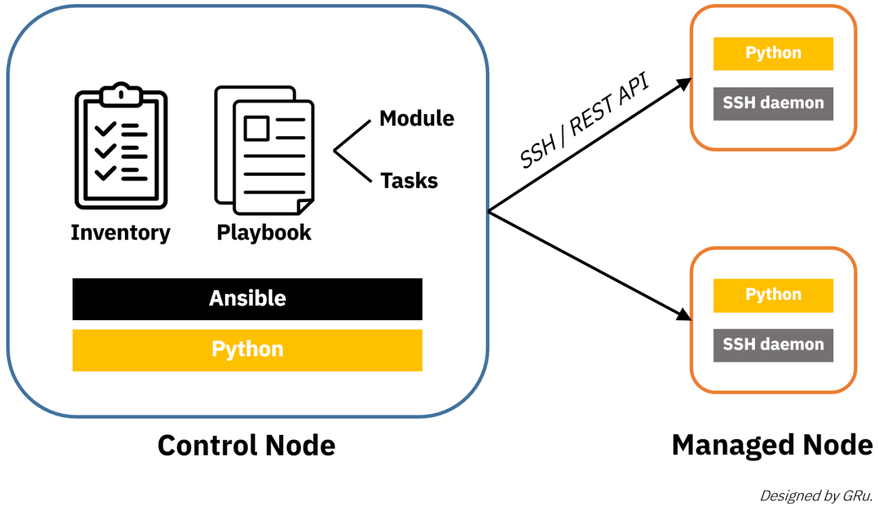
프로젝트 진행 과정

1. Ansible을 이용한 Playbook 작성 및 실행
2. Terraform으로 AWS 클라우드에 wordpress 배포하기
3. Terraform으로 Azure 클라우드에 wordpress 배포하기

Ansible 사용을 위한 패키지 설치



Ansible 파일 설정



Ansible Playbook 작성 및 실행

ansible-playbook 실행파일 tree 구조

```
.
├── ansible.cfg
├── httpd.yaml
├── inven_aws_ec2.yaml
└── roles
    └── common
        ├── defaults
        │   └── main.yml
        ├── handlers
        │   └── main.yml
        ├── meta
        │   └── main.yml
        ├── README.md
        ├── tasks
        │   └── main.yml
        ├── templates
        │   └── wp-config.php.j2
        └── vars
            └── main.yml
```

ansible-playbook 실행파일 tree 구조

```
.
├── ansible.cfg
├── azure_rm.yaml
├── httpd.yaml
└── roles
    └── common
        ├── defaults
        │   └── main.yml
        ├── handlers
        │   └── main.yml
        ├── meta
        │   └── main.yml
        ├── README.md
        ├── tasks
        │   └── main.yml
        ├── templates
        │   └── wp-config.php.j2
        └── vars
            └── main.yml
```

AWS	Azure
ansible.cfg : ansible 설정 파일	
httpd.yaml : 역할이 포함된 Playbook 실행 파일	
inven_aws_ec2.yaml 동적 인벤토리 파일	azure_rm.yaml 동적 인벤토리 파일
roles/common : 역할 파일	

ansible.cfg

```
[defaults]
inventory = inven_aws_ec2.yaml
remote_user = ec2-user
become = true
ask_pass = false
callback_whitelist = timer, profile_tasks, profile_roles
```

Ansible 설정 파일

inventory: 인벤토리 파일을 지정한다(inven_aws_ec2.yaml가 인스턴스 목록을 불러온다.).

remote_user: 로그인 유저를 지정한다(Amazon LInux 2: ec2-user, CentOS = centos).

become: sudo로 실행한다(true).

ask_pass: 키 기반으로 인증한다(false).

inven_aws_ec2.yaml

```
plugin: aws_ec2
regions:
- ap-northeast-2
aws_profile: "default"
```

인벤토리 모듈을 불러오는 파일

plugin: 인벤토리 모듈의 플러그인을 불러온다(aws_ec2).

regions: AWS 리전을 지정한다(ap-northeast-2, 서울리전).

aws_profile: 로그인할 프로파일을 지정한다(default, aws configure로 설정).

httpd.yaml

```yaml
---
- hosts: all                    # hostname
  become: True                  # sudo 권한
  roles:                        # 역할 파일 로드
    - common
```

Ansible-Playbook 실행 파일

hosts: inventory에 있는(모듈이 불러온) 모든 호스트를 실행한다.

become: sudo 권한으로 실행한다(true).

roles: 역할 파일을 로드한다.

roles

```
.
└── common
    ├── defaults
    │   └── main.yml
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
    │   └── main.yml
    ├── templates
    │   └── wp-config.php.j2
    └── vars
        └── main.yml
```

역할(roles) 디렉토리

tasks/main.yml : PlayBook에서 실제로 실행되는 작업을 저장한 파일

handler/main.yml : 핸들러 작업을 저장한 파일

template/main.yml : 템플릿 파일을 저장한 파일

vars/main.yml : 변수를 저장한 파일

1. Ansible을 이용한 Playbook 작성 및 실행

tasks/main.yaml

```yaml
---
- name: Repo Set_AWS # Amazon Linux 2 리포지토리 활성화
  block:
    - name: Install epel package # Amazon LInux 2 epel 패키지 활성화
      command:
        cmd: amazon-linux-extras install epel -y
    - name: Active PHP74 # Amazon LInux 2 php74 리포지토리 활성화
      command:
        cmd: amazon-linux-extras enable php7.4
  when: ansible_facts['distribution'] == "Amazon"
  tags:
    - wordpress
    - amazon_linux
    -
- name: Repo Set_CentOS
  block:
    - name: Install Remi Repo Package # CentOS Remi Repo 추가
      yum:
        name: '{{ repo_set["pkg"] }}'
        state: present
        validate_certs: no
    - name: Disable Remi php54 Repo # CentOS php54 리포지토리 비활성화
      yum_repository:
        name: '{{ repo_set["safe"]["name"] }}'
        file: '{{ repo_set["safe"]["name"] }}'
        mirrorlist: '{{ repo_set["safe"]["mirror"] }}'
        description: '{{ repo_set["safe"]["name"] }}'
        enabled: no
    - name: Enable Remi php74 Repo  # CentOS php74 리포지토리 활성화
      yum_repository:
        name: '{{ repo_set["php74"]["name"] }}'
        file: '{{ repo_set["php74"]["name"] }}'
        mirrorlist: '{{ repo_set["php74"]["mirror"] }}'
        description: '{{ repo_set["php74"]["name"] }}'
        enabled: yes
        gpgcheck: yes
        gpgkey: '{{ repo_set["php74"]["gpgkey"] }}'
  when: ansible_facts['distribution'] == "CentOS"
  tags:
    - wordpress
    - centos
```

artifact 변수를 이용하여 CentOS와 Amazon Linux를 구별하며
각 OS에 맞는 패키지 의존성으로 wordpress를 셋팅할 수 있도록 함

templates/main.yml/wp-config.php.j2

```
---------------------- 이전 생략 ------------------------

// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', '{{ database["name"] }}' );

/** Database username */
define( 'DB_USER', '{{ database["user"] }}' );

/** Database password */
define( 'DB_PASSWORD', '{{ database["pwd"] }}' );

/** Database hostname */
define( 'DB_HOST', '{{ database_source }}' );

---------------------- 이후 생략 ------------------------
```

jinja template을 이용해 데이터베이스 로그인 정보를 세팅한다.
이후 wp-config.php 파일로 해당 내용을 덮어씌운다.

tasks/main.yml

```yaml
- name: Configure Database for Wordpress # 워드프레스 데이터베이스 세팅
  block:
  - name: Copy Database Configure File for Wordpress # 데이터베이스 설정파일
    template:
      src: templates/wp-config.php.j2
      dest: /var/www/html/wordpress/wp-config.php
      owner: apache
      group: apache
  tags:
    - wordpress
    - amazon_linux
    - centos
```

1. Ansible을 이용한 Playbook 작성 및 실행

vars/main.yml

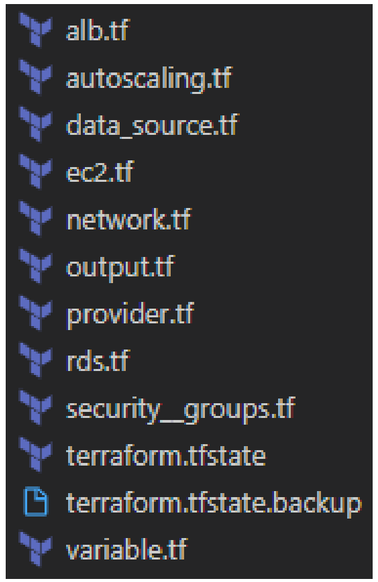
```yaml
---
# vars file for common
repo_set: # repo
  pkg: https://rpms.remirepo.net
  safe:
    name: remi-safe
    mirror: http://cdn.remirepo.
  php74:
    name: remi-php74
    mirror: http://cdn.remirepo.
    gpgkey: file:///etc/pki/rpm-

wordpress: # 웹서버 설치
  svc_port: 80
  package:
    packages_httpd: httpd,php,pl
PyMySQL

wordpress_version: 5.9.3 # 워드프
wordpress_filename: "wordpress-
wordpress_url: "https://wordpres

# Terraform으로 데이터베이스 삽입
database: # 데이터베이스 로그인
  svc_port: 3306
  name: wordpress
  user: admin
  pwd: adminpass

database_source: tmp_endpoint #
```

```
web-instance.amazon-ebs.project-wordpress-web: PLAY [all] *********************************************************************
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [Gathering Facts] *********************************************************
web-instance.amazon-ebs.project-wordpress-web: ok: [default]
web-instance.amazon-ebs.project-wordpress-web: [WARNING]: Platform linux on host default is using the discovered Python
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: interpreter at /usr/bin/python, but future installation of another Python
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Install epel package] ***************************************
web-instance.amazon-ebs.project-wordpress-web: interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
web-instance.amazon-ebs.project-wordpress-web: ce_appendices/interpreter_discovery.html for more information.
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Active PHP74] ***************************************************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Install Remi Repo Package] ***********************************
web-instance.amazon-ebs.project-wordpress-web: skipping: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Disable Remi php54 Repo] ***************************************
web-instance.amazon-ebs.project-wordpress-web: skipping: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Enable Remi php74 Repo] ***************************************
web-instance.amazon-ebs.project-wordpress-web: skipping: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Disable PHP54] ***************************************************
web-instance.amazon-ebs.project-wordpress-web: ok: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Install Apache, PHP, mariadb] ***********************************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Set httpd port] ***************************************************
web-instance.amazon-ebs.project-wordpress-web: ok: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Start httpd service] *****************************************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Download Wordpress System for Amzon Linux 2] ********************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Decompress Archive file for Amzon Linux 2] ********************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Download Wordpress System for CentOS 7] ***********************
web-instance.amazon-ebs.project-wordpress-web: skipping: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Decompress Archive file for CentOS 7] ***************************
web-instance.amazon-ebs.project-wordpress-web: skipping: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: TASK [common : Copy Database Configure File for Wordpress] ********************
web-instance.amazon-ebs.project-wordpress-web: changed: [default]
web-instance.amazon-ebs.project-wordpress-web:
web-instance.amazon-ebs.project-wordpress-web: PLAY RECAP *********************************************************************
web-instance.amazon-ebs.project-wordpress-web: default                    : ok=10   changed=7    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
web-instance.amazon-ebs.project-wordpress-web:
```

운로드 정보와 버전을 지정

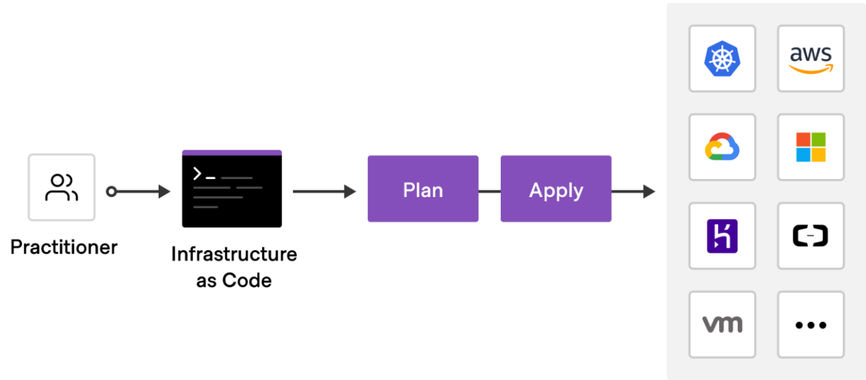
2. Terraform으로 AWS 클라우드에 wordpress 배포하기

Terraform 설치 및 초기 설정



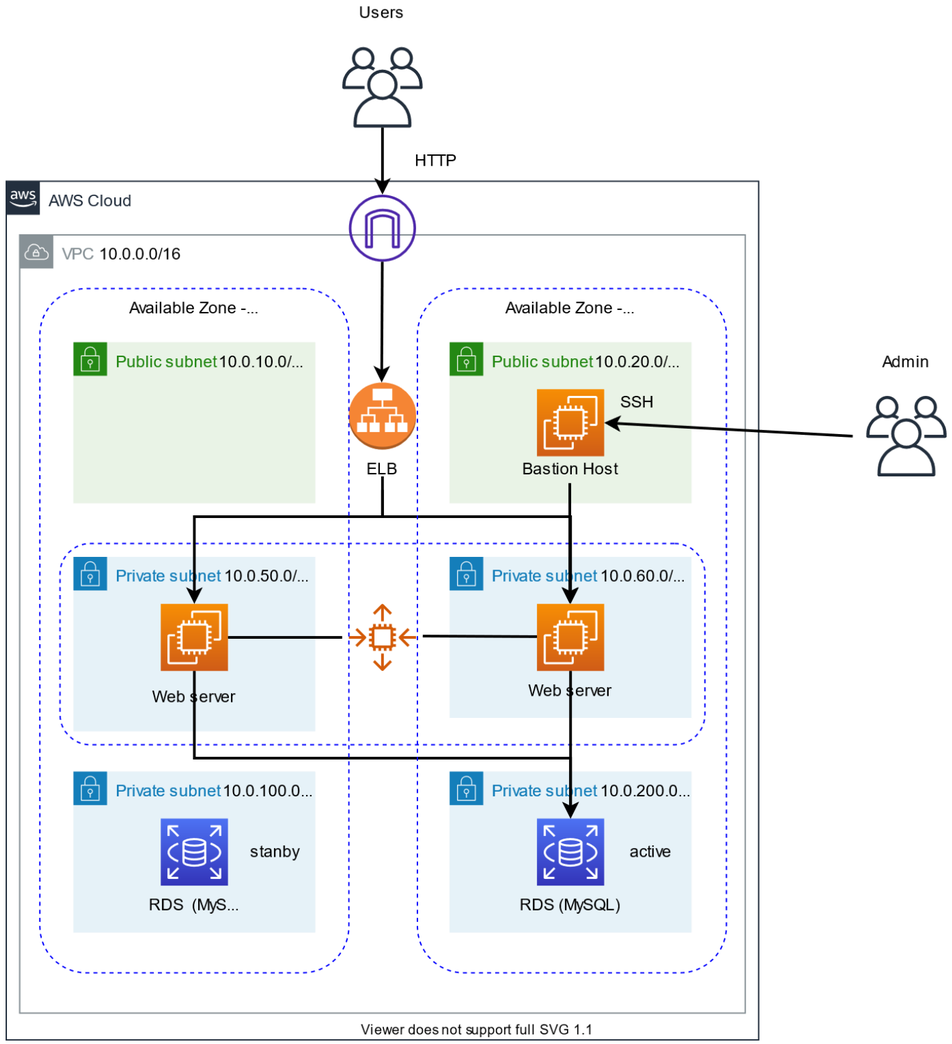
Terraform 리소스 작성하기

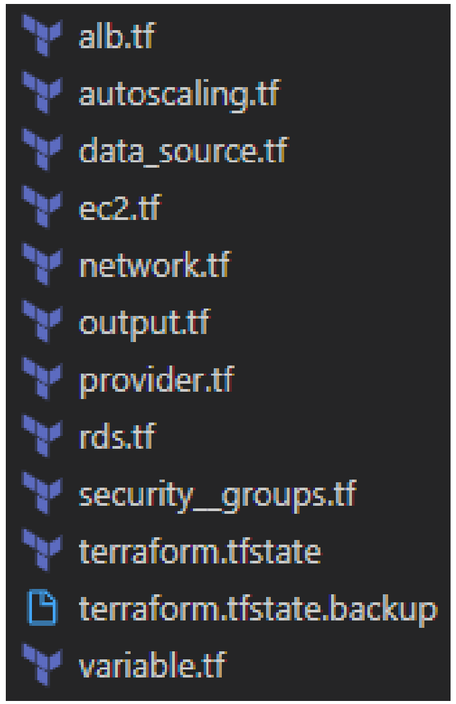
- Terraform Provider 설정
- 네트워크 구축
- Bastion Host 생성 및 보안 그룹 설정
- Packer를 사용하여 커스텀 인스턴스로 이미지 만들기
- wordpress 이미지로 시작 템플릿 구성
- RDS 데이터베이스 생성 및 보안 그룹 설정
- 시작 템플릿으로 오토 스케일링 그룹 구성
- 오토 스케일링 그룹에 로드밸런서 연결
- Output Value로 자주 참조하는 변수 처리



terraform apply 및 생성된 리소스 작동 확인

2. Terraform으로 AWS 클라우드에 wordpress 배포하기





alb.tf : 로드밸런서 생성 파일

autoscaling.tf : 오토스케일링 적용 파일

data_source.tf : data block 사용 파일

ec2.tf : 인스턴스 생성 파일

network.tf : VPC 구축 및 네트워크 연결 설정 파일

output.tf : 리소스 값 출력 변수 파일

provider.tf : 프로바이더 지정 파일

rds.tf : RDS 데이터베이스 생성 파일

security_groups.tf : 보안 그룹 설정 파일

variable.tf : 변수 파일

provider.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws" # 프로바이더 설치를 위한 경로를 지정
      version = "~> 3.0" # 프로바이더 버전을 지정
    }
  }
}

provider "aws" {
  region = "ap-northeast-2" # 리소스를 생성할 AWS 리전을 지정
}
```

provider 블록
aws API를 사용하기 위해
Terraform Provider Plugin을 선언한다.

network.tf

```
module "app_vpc" {
  source = "terraform-aws-modules/vpc/aws" # 모듈 레포지토리 명시

  name = "app_vpc" # vpc의 이름

  cidr = "10.0.0.0/16" # vpc의 cidr 블록

  azs             = ["ap-northeast-2a", "ap-northeast-2c"] # 가용 영역
  public_subnets  = ["10.0.10.0/24", "10.0.20.0/24"] # 퍼블릭 서브넷
  private_subnets = ["10.0.50.0/24", "10.0.60.0/24", "10.0.100.0/24",
"10.0.200.0/24"] # 프라이빗 서브넷

  create_igw = true # 인터넷 게이트웨이 활성화

  enable_nat_gateway = true # NAT 게이트웨이 활성화
  single_nat_gateway = true # 단일 NAT 게이트웨이 사용
}
```

편리한 VPC 구성을 위해 vpc 모듈을 사용

모듈을 사용하기 위해 module이 저장되어있는 레포지토리 경로를 명시

vpc의 이름, vpc의 cidr 블록, 서브넷을 만들 가용 영역 및 퍼블릭 서브넷, 프라이빗 서브넷의 cidr 블록을 지정

네트워크 통신을 위한 인터넷 게이트웨이, NAT 게이트웨이 사용을 지정

vpc module을 사용하면 네트워크 연결이 자동으로 이루어지므로 라우트 테이블을 생성하고 라우트 테이블 간의 연결을 따로 명시할 필요가 없다.

ec2.tf

```
# SSH 접속을 위한 공개키 설정
resource "aws_key_pair" "app_server_key" {
  key_name   = "app_server_key"
  public_key = file("/home/vagrant/.ssh/id_rsa.pub")
}

# Bastion Host 인스턴스 생성
resource "aws_instance" "bastionhostEC201" {
  ami                    = data.aws_ami.amazonLinux.id
  availability_zone      = module.app_vpc.azs[0] # ap-northeast-2a
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.bastionSG01.id]
  subnet_id              = module.app_vpc.public_subnets[0]
  key_name               = aws_key_pair.app_server_key.key_name

  # vagrant의 개인키를 이용해 서버에 접속
  connection {
    user        = "ec2-user"
    host        = self.public_ip
    private_key = file("/home/vagrant/.ssh/id_rsa") # 개인키로 접속
  }

  # file 프로비저너를 사용해 개인키 전달
  provisioner "file" {
    source      = "/home/vagrant/.ssh/id_rsa"
    destination = "/tmp/id_rsa" # 권한 문제로인해 임시 디렉토리로 이동
  }

  # 키를 ec2-user의 .ssh 디렉토리로 복사하고 권한을 변경
  provisioner "remote-exec" {
    inline = [
      "sudo cp /tmp/id_rsa /home/ec2-user/.ssh/id_rsa", # 키 복사하기
      "sudo chmod 400 /home/ec2-user/.ssh/id_rsa" # 권한 변경하기
    ]
  }
}
```

Bastion Host 인스턴스 생성

프라이빗 ip만을 가진 ec2 인스턴스로 jump host하기 위해서는
Bastion Host 인스턴스에 가상머신의 개인키가 있어야 한다.

이를 위해 connection 블록 및 provisioner 를 사용하여 연결을 설정한뒤
키 파일을 전송하고 키 파일의 권한을 변경한다.

2. Terraform으로 AWS 클라우드에 wordpress 배포하기

security_groups.tf

```
# Security Group

# Basion host로 접속하기 위한 보안 그룹
resource "aws_security_group" "bastionSG01" {
  name        = "bastion-SG"
  description = "Allow all SSH"
  vpc_id      = module.app_vpc.vpc_id

  ingress {
    cidr_blocks = ["0.0.0.0/0"] # 모든 ip 허용
    from_port   = 22
    protocol    = "tcp"
    to_port     = 22
  }

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

# Bastion Host에서 wordpress 인스턴스로 접속하기위한 보안 그룹
resource "aws_security_group" "bastion-to-private" {
  name        = "bastion-to-private-sg"
  description = "Allow SSH from Bastion Host"
  vpc_id      = module.app_vpc.vpc_id

  ingress {
    # Bastion Host의 ip만 접속을 허용한다.
    cidr_blocks = ["${aws_instance.bastionhostEC201.private_ip}/32"]
    from_port   = 22
    protocol    = "tcp"
    to_port     = 22
  }

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

Bastion Host를 위한 보안 그룹을 설정

가상머신에서 Bastion Host로 접속할 수 있도록
Bastion Host 인스턴스의 인바운드, 아웃바운드 규칙을 설정한다.

프라이빗 서브넷에 존재하는 wordpress 인스턴스로
jump host 하기 위한 보안 그룹도 설정한다.

Packer를 사용하여 커스텀 인스턴스로 이미지 만들기

```
.
├── httpd
│   ├── ansible.cfg
│   ├── httpd.yaml
│   ├── inven_aws_ec2.yaml
│   └── roles
│       └── common
│           ├── defaults
│           │   └── main.yml
│           ├── handlers
│           │   └── main.yml
│           ├── meta
│           │   └── main.yml
│           ├── README.md
│           ├── tasks
│           │   └── main.yml
│           ├── templates
│           │   └── wp-config.php.j2
│           └── vars
│               └── main.yml
└── project-ami.pkr.hcl
```

HashiCorp Packer

project-ami.pkr.hcl

```
packer {
  required_plugins {
    amazon = {
      version = "≥ 0.0.2"
      source  = "github.com/hashicorp/amazon"
    }
  }
}

source "amazon-ebs" "project-wordpress_web" {
  region  = "ap-northeast-2"
  profile = "default"

  ami_name      = "project-wordpress_web"
  instance_type = "t2.micro"
  source_ami_filter {
    filters = {
      name                = "amzn2-ami-hvm-2.0.*"
      root-device-type    = "ebs"
      virtualization-type = "hvm"
    }
    most_recent = true
    owners      = ["amazon"]

  }
  ssh_username = "ec2-user"
  force_deregister = true
}

build {
  name = "web-instance"
  sources = [
    "source.amazon-ebs.project-wordpress_web"
  ]
  provisioner "ansible" {
    playbook_file = "/home/vagrant/test/images/httpd/httpd.yaml"
    extra_arguments = [
      "--become",
    ]
    ansible_env_vars = [
    ]
  }
}
```

Packer 프로바이더를 설정
source "amazon-ebs" 블록에는 커스텀 이미지를 생성하기 위해 인스턴스를 만드는데 사용할 초기 인스턴스를 지정한다.

build 블록에서는 source "amazon-ebs" 블록에서 지정한 초기 인스턴스에 커스터마이징을 진행한다.

Packer에서는 ansible provisioner를 제공하므로 편리하게 Playbook을 실행할 수 있다.

extra_arguments를 통해 playbook을 실행 권한을 지정할 수 있다.

2. Terraform으로 AWS 클라우드에 wordpress 배포하기

ec2.tf

```
# wordpress가 구동될 EC2 인스턴스는 Auto Scaling을 통해 생성할 것이므로,
# 사용할 Launch Template을 작성한다.
resource "aws_launch_template" "project-launch-template" {
  # 명시적 의존성
  depends_on = [
    module.app_vpc.public_subnets,
    aws_db_subnet_group.testSubnetGroup,
    aws_db_instance.testDB
  ]

  name                                 = "project-launch-template"
  description                          = "for Auto Scaling"
  instance_type                        = "t2.micro"
  image_id                             = data.aws_ami.wordpressLinux.id
  instance_initiated_shutdown_behavior = "terminate"
  key_name                             = aws_key_pair.app_server_key.key_name
  vpc_security_group_ids               = [aws_security_group.privateEC2SG01.id,
aws_security_group.bastion-to-private.id]

  # DB 엔드포인트 수정을 위한 사용자 데이터를 작성한다.
  # sed 명령을 사용해 wp-config.php 내용을 수정한다.
  user_data = "${base64encode(
    <<-EOF
    #!/bin/bash
    sed -i 's/tmp_endpoint/${aws_db_instance.testDB.endpoint}/g'
/var/www/html/wordpress/wp-config.php
    systemctl restart httpd
    systemctl restart mariadb
    EOF
    )}"

  monitoring {
    enabled = true # 모니터링을 활성화한다.
  }

  placement {
    availability_zone = "ap-northeast-2"
  }

  tags = {
    "Name" = "project-ec2-template"
  }
}
```

Auto Scaling Group에서 wordpress 인스턴스를 생성하기 위해 Packer로 만든 이미지를 사용하여 시작 템플릿을 구성한다.

```
depends_on = [
  module.app_vpc.public_subnets,
  aws_db_subnet_group.testSubnetGroup,
  aws_db_instance.testDB
]
```

시작 템플릿에서 인스턴스가 생성될 때
사용자 데이터를 이용해 wordpress 설정 파일인 wp-config.php 파일에 데이터베이스 엔드포인트 값을 입력하기 때문에
RDS DB가 생성된 이후에 시작 템플릿을 구성해야 한다.

따라서 RDS 데이터베이스에 대한 명시적 의존성을 선언한다.

2. Terraform으로 AWS 클라우드에 wordpress 배포하기

rds.tf

```
# RDS가 위치할 데이터베이스용 프라이빗 서브넷을 지정한다.
resource "aws_db_subnet_group" "testSubnetGroup" {
  name = "test"
  subnet_ids = [
    module.app_vpc.private_subnets[2],
    module.app_vpc.private_subnets[3]
  ]

  tags = {
    "Name" = "test-subnet-group"
  }
}

# RDS DB 인스턴스를 생성한다.
resource "aws_db_instance" "testDB" {
  allocated_storage     = 20
  max_allocated_storage = 50
  availability_zone     = "ap-northeast-2a"
  db_subnet_group_name  = aws_db_subnet_group.testSubnetGroup.name
  engine                = "mariadb"
  engine_version        = "10.5"
  instance_class        = "db.t3.small"
  skip_final_snapshot   = true
  identifier            = "project-db"
  name                  = "wordpress"      # DB name
  username              = "admin"          # 사용자 이름
  password              = var.db_password # 패스워드 (adminpass로)
  port                  = "3306"
  vpc_security_group_ids = [
    aws_security_group.privateRDSSG01.id
  ]
}

# DB root 패스워드를 설정한다.
variable "db_password" {
  description = "RDS root user password"
  type        = string
  sensitive   = false # 프롬프트에 비밀번호를 입력할 때 확인할 수 있도록 한다.
  # default 값을 설정하기 않아 직접 프롬프트에 입력한 값을 사용한다.
}
```

```
resource "aws_security_group" "privateRDSSG01" {
  name        = "private-rds-sg-01"
  description = "Allow acceess from private web instance"
  vpc_id      = module.app_vpc.vpc_id

  ingress = [{
    cidr_blocks      = null
    description      = null
    from_port        = 3306
    ipv6_cidr_blocks = null
    prefix_list_ids  = null
    protocol         = "tcp"
    security_groups  = [aws_security_group.privateEC2SG01.id]
    self             = false
    to_port          = 3306
  }]

  egress = [{
    cidr_blocks      = ["0.0.0.0/0"]
    description      = null
    from_port        = 0
    ipv6_cidr_blocks = null
    prefix_list_ids  = null
    protocol         = "-1"
    security_groups  = null
    self             = false
    to_port          = 0
  }]
}
```

wordpress 인스턴스에서 사용할 RDS 데이터베이스 생성

데이터베이스는 보안이 매우 중요하므로 외부 접근을 차단하고 wordpress 인스턴스에서만 RDS 데이터베이스 인스턴스에 접속할 수 있도록 보안 그룹을 설정한다.

autoscaling.tf

```
# wordpress 인스턴스들을 Auto Scaling하기 위한 Auto Scaling Group 생성
resource "aws_autoscaling_group" "project-ASG" {
  launch_template {
    id = aws_launch_template.project-launch-template.id
  }

  desired_capacity = 2 # 원하는 인스턴스의 개수 2개
  min_size         = 2 # 최소 인스턴스 개수 2개
  max_size         = 4 # 최대 인스턴스 개수 4개

  health_check_type         = "ELB"
  health_check_grace_period = 180 # 3분 (default는 300초)
  force_delete              = true
  vpc_zone_identifier       = [module.app_vpc.private_subnets[0],
module.app_vpc.private_subnets[1]]

}
```

시작 템플릿을 사용하여 오토 스케일링 그룹을 구성하고 로드 밸런서를 생성하여 오토 스케일링 그룹에 연결한다.

80 번 포트로 들어오는 트래픽을 오토 스케일링 그룹에 의해 관리되는 인스턴스 2대에 부하 분산한다.

alb.tf

```
# application loadbalancer를 사용하기 위해 선언
resource "aws_alb" "project-elb" {
  name                       = "project-alb"
  internal                   = false # internet facing 설정
  load_balancer_type         = "application"
  security_groups            = [aws_security_group.publicSG01.id]
# 외부에서 들어오는 HTTP 허용
  subnets                    = [module.app_vpc.public_subnets[0],
module.app_vpc.public_subnets[1]]
  enable_cross_zone_load_balancing = true
}

# alb에 연결할 Auto Scaling 타겟 그룹을 지정한다.
resource "aws_alb_target_group" "project-elb-tg" {
  name     = "tset-alb-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id = module.app_vpc.vpc_id
}

# autoscaling 그룹으로 관리할 타겟 그룹을 지정한다.
resource "aws_autoscaling_attachment" "wp-atsg-attach" {
  autoscaling_group_name = aws_autoscaling_group.project-ASG.name
  alb_target_group_arn   = aws_alb_target_group.project-elb-tg.arn
}

# alb의 80번 포트로 들어오는 트래픽을 오토 스케일링 타겟 그룹으로 제어
resource "aws_alb_listener" "project-elb-listener" {
  load_balancer_arn = aws_alb.project-elb.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_alb_target_group.project-elb-tg.arn
  }
}
```

```
## public Security Group
resource "aws_security_group" "publicSG01" {
  name        = "public-SG-01"
  description = "Allow all HTTP"
  vpc_id      = module.app_vpc.vpc_id

  ingress {
    cidr_blocks = ["0.0.0.0/0"]
    from_port   = 80
    protocol    = "tcp"
    to_port     = 80
  }

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

```
## private Security Group
resource "aws_security_group" "privateEC2SG01" {
  name        = "private-ec2-sg-01"
  description = "Allow HTTP from ALB"
  vpc_id      = module.app_vpc.vpc_id

  ingress = [{
    cidr_blocks      = null
    description      = null
    from_port        = 80
    ipv6_cidr_blocks = null
    prefix_list_ids  = null
    protocol         = "tcp"
    security_groups  = [aws_security_group.publicSG01.id]
    self             = false
    to_port          = 80
  }]

  egress = [{
    cidr_blocks      = ["0.0.0.0/0"]
    description      = null
    from_port        = 0
    ipv6_cidr_blocks = null
    prefix_list_ids  = null
    protocol         = "-1"
    security_groups  = null
    self             = false
    to_port          = 0
  }]
}
```

로드 밸런서를 위한 보안 그룹을 설정한다.

publicSG01 : 로드 밸런서에 모든 ip로부터의 HTTP 접속을 허용하는 보안 그룹

privateEC2SG01 : 로드 밸런서로부터 wordpress 인스턴스로 전달된 HTTP 접속을 허용하는 보안 그룹

output.tf

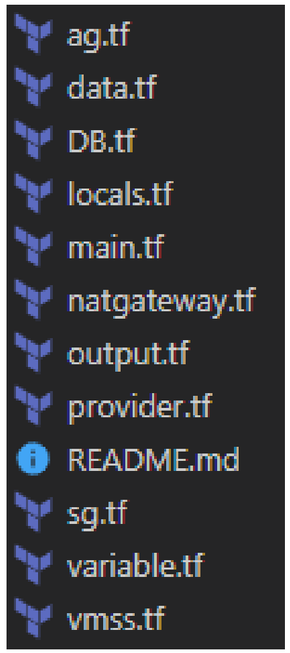
```
# Packer로 만든 이미지의 id 출력
output "pakcer-image" {
  value = data.aws_ami.wordpressLinux.id
}

# db 엔드포인트 출력
output "wordpress_db_endpoint" {
  value = aws_db_instance.testDB.endpoint
}

# Bastion Host 프라이빗 ip 출력
output "bastion-instance-private" {
  value = aws_instance.bastionhostEC201.private_ip
}

# Bastion Host 퍼블릭 ip 출력
output "bastion-instance-public" {
  value = aws_instance.bastionhostEC201.public_ip
}

# 로드밸런서 도메인 출력
output "alb_domain" {
  value = aws_alb.project-elb.dns_name
}
```

```
Apply complete! Resources: 35 added, 0 changed, 0 destroyed.

Outputs:

alb_domain = "project-alb-1367632546.ap-northeast-2.elb.amazonaws.com"
bastion-instance-private = "10.0.10.125"
bastion-instance-public = "3.35.216.156"
pakcer-image = "ami-03a46dd884bd20a55"
wordpress_db_endpoint = "project-db.cicxdwwu5hr0.ap-northeast-2.rds.amazonaws.com:3306"
```

리소스를 생성 완료 후 자주 사용해야 하는 리소스 값들을
output value로 선언하여
리소스 생성 완료 시 터미널 상에서 확인할 수 있도록 한다.

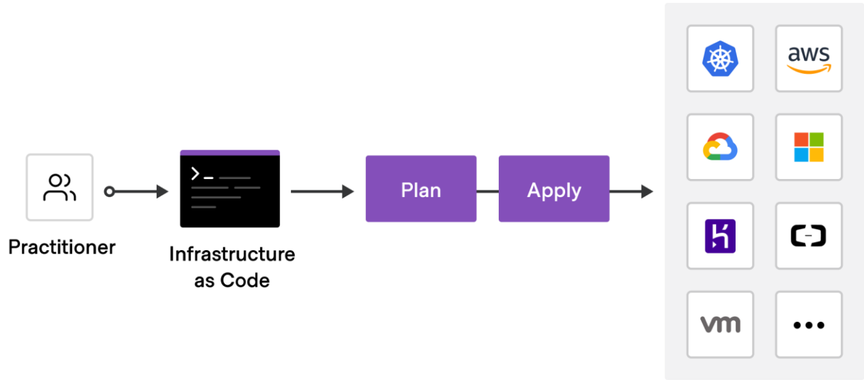
2. Terraform으로 Azure 클라우드에 wordpress 배포하기

Terraform 설치 및 초기 설정



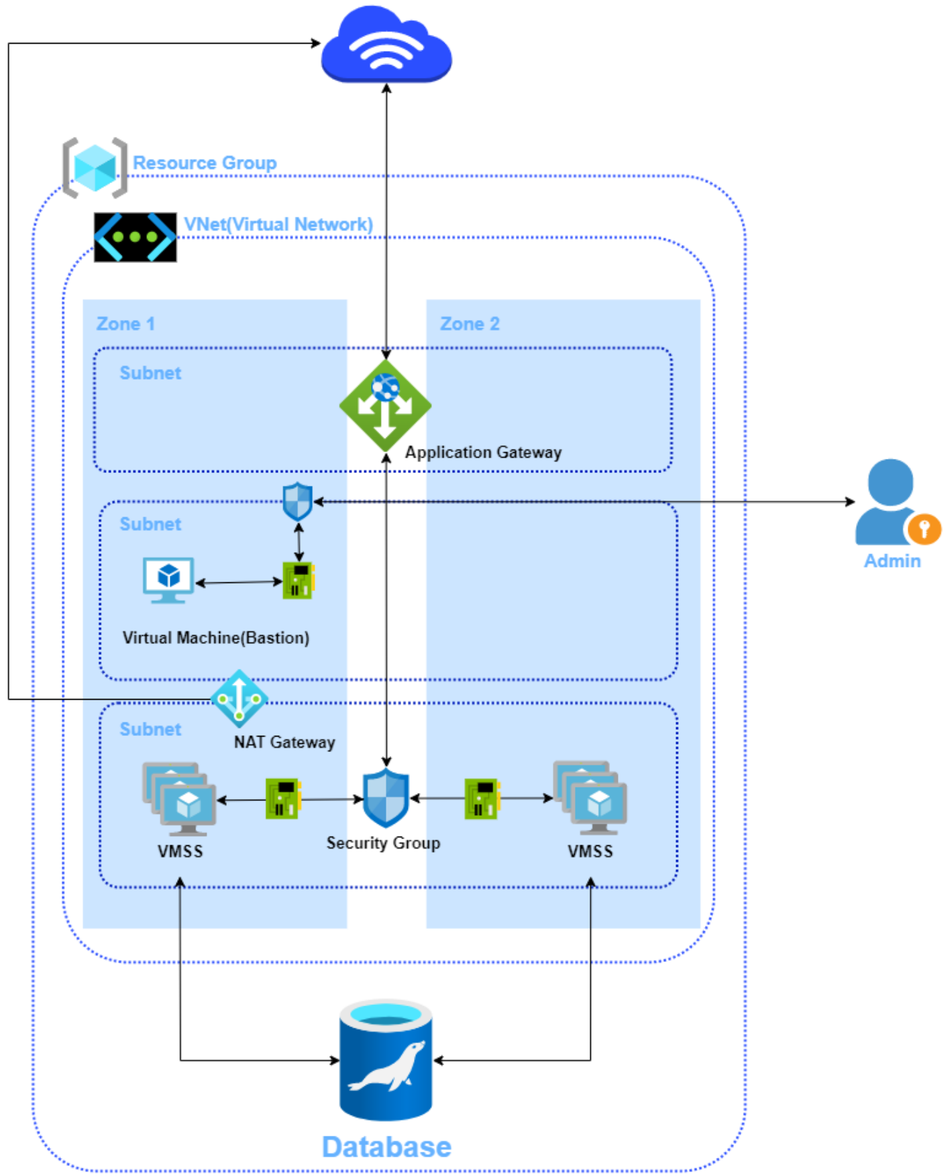
Terraform 리소스 작성하기

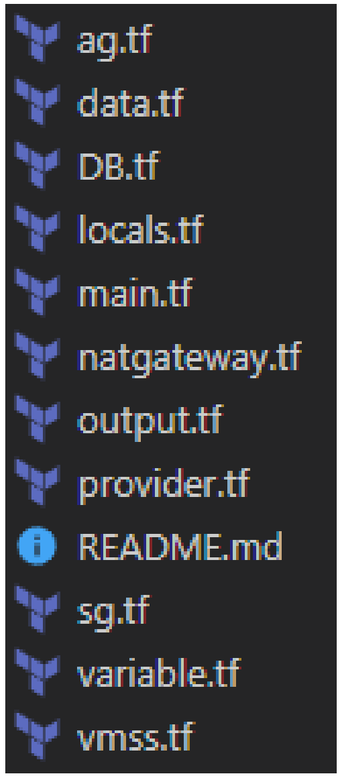
- Azure에 대한 Terraform 액세스 설정
- Terraform 환경 변수 구성
- Packer를 이용한 Wordpress 이미지 생성
- Terraform 리소스 그룹 생성
- 가상 네트워크 생성
- 서브넷 및 네트워크 인터페이스 생성
- DB 생성
- Wordpress Database 생성 및 방화벽 설정
- Application gateway 생성
- VMSS생성



terraform apply 및 생성된 리소스 작동 확인

3. Terraform으로 Azure 클라우드에 wordpress 배포하기





ag.tf : 애플리케이션 게이트웨이 생성 파일

data.tf : data block 사용 파일

DB.tf : 데이터베이스 서버, 데이터베이스 생성 파일

locals.tf : 로컬 변수 처리 파일

main.tf : 서브넷 생성 및 Bastion Host 생성 파일

networkgateway.tf : 네트워크 연결 설정 파일

output.tf : 리소스 값 출력 변수 파일

provider.tf : 프로바이더 지정 파일

sg.tf : 보안 그룹 설정 파일

variable.tf : 변수 파일

vmss : 가상 머신 확장 집합 설정 파일

Packer를 이용한 Wordpress 이미지 생성

- az cli로 my-image 리소스 그룹 생성하기

```
az group create --name my-image --location koreacentral
```

- 리소스 그룹 확인

```
az group show --name my-image
{
  "id": "/subscriptions/0e337d72-xxxx-xxxx-xxxx-
9968126bxxxx/resourceGroups/my-image",
  "location": "koreacentral",
  "managedBy": null,
  "name": "my-image",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": {},
  "type": "Microsoft.Resources/resourceGroups"
}
```

centos.json (Packer v1.5.1)

```
{
    "variables": {
        "managed_image_resource_group_name": "my-image",
        "managed_image_name": "wordpress",
        "ssh_username": "azureuser",
        "location": "koreacentral",
        "playbook_file": "/home/vagrant/httpd_Azure/httpd.yaml"
    },

    "builders": [{
      "type": "azure-arm",

      "client_id": "",
      "client_secret": "",
      "tenant_id": "",
      "subscription_id": "",

      "managed_image_resource_group_name": "{{user
`managed_image_resource_group_name`}}",
      "managed_image_name": "{{user `managed_image_name`}}",
      "ssh_username": "{{user `ssh_username`}}",

      "os_type": "Linux",
      "image_publisher": "OpenLogic",
      "image_offer": "CentOS",
      "image_sku": "7_9",

      "location": "{{user `location`}}",
      "vm_size": "Standard_DS2_v2"
    }],
    "provisioners": [{
      "type": "ansible",
      "playbook_file": "{{user `playbook_file`}}",
      "extra_arguments": ["--become"],
      "ansible_env_vars": []
    }]
}
```

Azure의 이미지를 생성하려면 개별 리소스 그룹이 필요하다.
Terraform을 통해 리소스 그룹을 생성할 수 있지만 Terraform은 같은 디렉토리에 있는 tf파일을 취합하여 실행하기 때문에 프로비저닝의 순서를 장담할 수 없다.
따라서 의존성의 문제를 방지하고자 az cli를 이용하여 이미지 빌드 전용 리소스 그룹을 생성하여 진행한다.

Terraform으로 리소스 그룹 생성 및 가상 네트워크 생성

```
# 리소스 그룹 생성
resource "azurerm_resource_group" "wp_rg" {
  name     = "WordpressResourcegroup"
  location = var.location
}


variable "location" {
  type        = string
  description = "리소스 영역"
  default     = "koreacentral"
}
```

Azure는 리소스 그룹이 있어야 여러가지 리소스를 생성할 수 있다.
azurerm_resource_group 블록을 이용해 리소스 그룹을 생성한다.

```
resource "azurerm_virtual_network" "wp_network" {
  name                = "Wordpress-vnet"
  address_space       = ["10.0.0.0/16"]
  location            = var.location
  resource_group_name = azurerm_resource_group.wp_rg.name
}
```

가상 네트워크를 생성한다.

서브넷 및 네트워크 인터페이스 생성

Bastion Host Subnet

```
resource "azurerm_subnet" "wp-bastion-subnet" {
  name                 = "Bastion-subnet"
  resource_group_name  = azurerm_resource_group.wp_rg.name
  virtual_network_name = azurerm_virtual_network.wp_network.name
  address_prefixes     = ["10.0.10.0/24"]
}
```

Bastion Host Public IP

```
resource "azurerm_public_ip" "wp-bastion-public-ip" {
  name                = "Bastion-public-ip"
  location            = var.location
  resource_group_name = azurerm_resource_group.wp_rg.name
  allocation_method   = "Static"
}
```

Bastion Host Network Interface

```
resource "azurerm_network_interface" "wp-bastion-network-interface" {
  name                = "Bastion-nic"
  location            = var.location
  resource_group_name = azurerm_resource_group.wp_rg.name

  ip_configuration {
    name                          = "Bastion_IPConfiguration"
    subnet_id                     = azurerm_subnet.wp-bastion-subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.wp-bastion-public-ip.
  }
}
```

가상 네트워크의 생성이 완료되었다면 서브넷을
Azure에서는 서브넷이 생성되면 모든 가용영역에
또한 Public과 Private의 개념이 없어서 만약 Pri
싶다면 Public IP를 할당하지 않으면 된다.

Bastion Host 생성

Bastion Host VM 생성

```
resource "azurerm_virtual_machine" "wp-bastion-vm" {
  name                  = "Bastion-vm"
  location              = var.location
  resource_group_name   = azurerm_resource_group.wp_rg.name
  network_interface_ids = [azurerm_network_interface.wp-bastion-network-
interface.id]
  vm_size               = "Standard_DS1_v2"

  delete_os_disk_on_termination = true # VM 삭제시 자동삭제

  storage_image_reference {
    publisher = var.linux_vm_image_publisher
    offer     = var.linux_vm_image_offer
    sku       = var.centos_7_sku
    version   = "latest"
  }

  storage_os_disk {
    name              = "Bastion-osdisk"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

  os_profile {
    computer_name  = var.bastion_computer_name
    admin_username = var.admin_user
  }

  os_profile_linux_config {
    disable_password_authentication = true
    ssh_keys {
      path     = "/home/${var.admin_user}/.ssh/authorized_keys"
      key_data = file("~/.ssh/id_rsa.pub")
    }
  }
}
```

```
bnet

subnet" "wp-web-subnet" {
      = "wp-web-subnet"
ame   = azurerm_resource_group.wp_rg.name
name  = azurerm_virtual_network.wp_network.name
      = ["10.0.50.0/24"]
```

```
Interface

ublic_ip" "wp-bastion-public-ip" {
    = "Bastion-public-ip"
    = var.location
me  = azurerm_resource_group.wp_rg.name
    = "Static"
```

rivate으로 운영해야 하기 때문에
하지 않는다.
장 집합 VMSS를 통해 VM을 관리할
것이기 때문에 따로 VM 이미지는 생성하지 않는다.

DB 서버, DB 생성 및 방화벽 설정

Azure Database for MariaDB Server

```
resource "azurerm_mariadb_server" "mariadb-server" {
  name                = "wp-mariadb-server"
  location            = azurerm_resource_group.wp_rg.location
  resource_group_name = azurerm_resource_group.wp_rg.name

  administrator_login          = var.mariadb-admin-login
  administrator_login_password = var.mariadb-admin-password

  sku_name = var.mariadb-sku-name
  version  = var.mariadb-version        # Maria DB 서버의 버전(10.2)

  storage_mb        = var.mariadb-storage
  auto_grow_enabled = true      # 자동 확장 기능

  backup_retention_days        = 7      # 백업 데이터 보존 기간
  geo_redundant_backup_enabled = false # geo 기반 백업 x
  ssl_enforcement_enabled = false       # ssl 접속 옵션 해제
```

```
resource "azurerm_mariadb_database" "mariadb-db" {
  name                = "wordpress"
  resource_group_name = azurerm_resource_group.wp_rg.name
  server_name         = azurerm_mariadb_server.mariadb-server.name
    charset             = "utf8"      # character set
  collation           = "utf8_unicode_ci"   # 데이터 베이스에 대한 데이터 정렬 지
정
}
```

```
resource "azurerm_mariadb_firewall_rule" "mariadb-fw-rule" {
  name                = "mariadbOfficeAccess"
  resource_group_name = azurerm_resource_group.wp_rg.name
  server_name         = azurerm_mariadb_server.mariadb-server.name
  start_ip_address    = ""    # IP 주소 범위 설정 (시작 주소)
  end_ip_address      = ""    # IP 주소 범위 설정 (끝 주소)
  depends_on          = [azurerm_mariadb_server.mariadb-server,
azurerm_mariadb_database.mariadb-db]
}
```

Azure의 데이터베이스 서비스는 가상 네트워크와는 격리되고 리소스 그룹안에 할당된다.
따라서 가상 네트워크 주소를 지정해 줄 필요가 없다.
DB의 정보들은 외부 사용자가 접근하면 안되기 때문에
Public 접근을 허용하지 않도록 설정한다.
또한 작업의 편의상 SSL 연결은 사용하지 않았다.

워드프레스에서 사용할 Wordpress DB를 생성한다.
데이터베이스 서버에 사용할 방화벽 규칙을 알맞게 지정해 주고
DB 서버와 DB보다 방화벽이 먼저 생성되면 의존성 충돌이 발생하므로
depends_on 옵션을 활용하여 의존성 설정을 해준다.

application gateway 생성

- Application gateway 배포 순서

1. Public IP 생성
2. Subnet 생성
3. Application gateway 리소스 생성
   - Sku 설정
   - Autoscale_configuration 설정
   - Application Gateway 설정
   - 프론트엔드 설정
     - 포트 설정
     - IP 설정
   - 백엔드 설정
     - 백엔드 풀 지정(vmss)
     - 백엔드 http 설정
   - http 리스너 설정
   - 라우팅 규칙 설정

L7 계층에서 부하 분산을 담당할 애플리케이션 게이트웨이를 생성한다.
애플리케이션 게이트웨이의 배포 순서는 다음과 같다.

application gateway 생성

- Public IP와 Subnet을 생성

```
resource "azurerm_public_ip" "wp-app-gateway-ip" {
  name                = "wp-app-gateway-ip"
  resource_group_name = azurerm_resource_group.wp_rg.name
  location            = var.location
  allocation_method   = "Static"
  sku                 = "Standard"
}

resource "azurerm_subnet" "frontend" {
  name                 = "frontend"
  resource_group_name  = azurerm_resource_group.wp_rg.name
  virtual_network_name = azurerm_virtual_network.wp_network.name
  address_prefixes     = ["10.0.70.0/24"]
}
```

애플리케이션 게이트웨이는 단독 Subnet과 Public IP를 사용한다.
서브넷에 다른 인스턴스가 존재한다면 애플리케이션 게이트웨이를 생성할 수 없기 때문에 먼저 애플리케이션 게이트웨이에 사용할 Public IP와 Subnet을 생성한다.

application gateway 생성

```
resource "azurerm_application_gateway" "wp-app-gateway" {
  name                = "wp-app-gateway"
  resource_group_name = azurerm_resource_group.wp_rg.name
  location            = var.location


  # Application Gateway에서 사용할 SKU
  sku {
    name = "Standard_v2" # AZ 영역 확장 설정은 V2에서만 가능
    tier = "Standard_v2"
  }

# 상태체크 프로브
  probe {
    interval                                  = 30  # 다음 상태 프로브가 전송되기 전에 대시가는 시간(초)
    minimum_servers                           = 2   # 최소 서버 0 -> 2로 변경
    name                                      = local.backend_http_probe  # 프로브 이름
    path                                      = "/"   # 경로
    pick_host_name_from_backend_http_settings = true
    protocol                                  = "Http"  # 프로토콜
    timeout                                   = 30  # 타임아웃 시간
    unhealthy_threshold                       = 3   # 노드가 비정상으로 간주되기전에 시도해야하는 재시도 횟수 (비정상 임계값)
  }

  # 오토스케일링 설정
  autoscale_configuration {
    min_capacity = 4  # 최소 갯수
    max_capacity = 8  # 최대 갯수
  }

  # Application Gateway 설정
  gateway_ip_configuration {
    name      = "my-gateway-ip-configuration"
    subnet_id = azurerm_subnet.frontend.id    # Application gateway의 서브넷
  }
```

```
  # 프론트엔드 포트
  frontend_port {
    name = local.frontend_port_name
    port = 80   # 80/tcp
  }

  # 프론트엔드 IP 설정
  frontend_ip_configuration {
    name                 = local.frontend_ip_configuration_name   # 프론트엔드 IP 구성의 이름
    public_ip_address_id = azurerm_public_ip.wp-app-gateway-ip.id # Application gateway에 사용할 공용 IP주소의 ID
  }

  # 백엔드 풀 지정 - 연결대상(vmss)
  backend_address_pool {
    name = local.backend_address_pool_name
  }

  # 백엔드 http 설정
  backend_http_settings {
    name                  = local.http_setting_name
    cookie_based_affinity = "Disabled" # 쿠키 기반 선호도 활성화 여부
    # path                  = "/path1/"
    port            = 80     # 백엔드 HTTP 설정에서 사용하는 포트
    protocol        = "Http" # 프로토콜
    request_timeout = 120     # 요청 제한시간 (초)
    probe_name                          = local.backend_http_probe    # 상태체크 프로브 이름
    pick_host_name_from_backend_address = true
  }

  # http 리스너
  http_listener {
    name                           = local.listener_name
    frontend_ip_configuration_name = local.frontend_ip_configuration_name
    frontend_port_name             = local.frontend_port_name
    protocol                       = "Http" # 프로토콜
  }

  # 라우팅 규칙
  request_routing_rule {
    name                       = local.request_routing_rule_name # 요청 라우팅 규칙의 이름
    rule_type                  = "Basic"                         # 라우팅 유형
    http_listener_name         = local.listener_name
    backend_address_pool_name  = local.backend_address_pool_name
    backend_http_settings_name = local.http_setting_name
  }
}
```

VMSS 생성

```
# 가상 머신 확장 집합 (Packer wordpress 이미지)
resource "azurerm_virtual_machine_scale_set" "vmss" {
  name                = "vmscaleset"
  location            = var.location
  resource_group_name = azurerm_resource_group.wp_rg.name
  upgrade_policy_mode = "Automatic" # VMSS의 가상머신에 대한 업그레이드 모드 지정

  zones = ["1", "2"]  # VMSS의 zone

  sku {
    name     = "Standard_DS1_v2"
    tier     = "Standard"
    capacity = 4
  }

  # 리눅스 머신의 구성정보
  storage_profile_image_reference {
    id = data.azurerm_image.image.id
  }
```

```
  # 스토리지 프로필 OS 디스크 블록
  storage_profile_os_disk {
    name              = ""             # 이름을 유연하게 작성하지 않으면 여러 그룹이 생기면서 충돌할수있다.
    caching           = "ReadWrite"    # 캐싱 요구사항을 지정한다. (None, ReadOnly, ReadWrite)
    create_option     = "FromImage"    # 데이터 디스크를 생성하는 방법
    managed_disk_type = "Standard_LRS" # 생성할 관리 디스크의 유형 지정
  }

  # 스토리지 프로필 데이터 디스크 블록
  storage_profile_data_disk {
    lun           = 0
    caching       = "ReadWrite"
    create_option = "Empty"
    disk_size_gb  = 10
  }

  # OS에 대한 정보
  os_profile {
    computer_name_prefix = var.web_computer_name
    admin_username       = var.admin_user
    custom_data          = file("azure-user-data.sh") # cloud init(sudo setenforce 0 / sudo systemctl restart httpd)
  }

  # OS가 리눅스 머신인 경우 설정
  os_profile_linux_config {
    disable_password_authentication = true # 패스워드 로그인 차단
    ssh_keys {
      path     = "/home/${var.admin_user}/.ssh/authorized_keys"
      key_data = file("~/.ssh/id_rsa.pub")
    }
  }

  network_profile {
    name                      = "terraformnetworkprofile" # 네트워크 인터페이스 구성의 이름
    primary                   = true                      # 네트워크 인터페이스 구성에서 생성된 네트워크 인터페이스가 vm의 기본 NIC인지 여부
    network_security_group_id = azurerm_network_security_group.webserver-sg.id
    ip_configuration {
      name                                         = "IPConfiguration"              # ip 구성의 이름
      subnet_id                                    = azurerm_subnet.wp-web-subnet.id # 적용할 서브넷
      primary                                      = true                           # 이 ip config가 기본 구성인지?
      application_gateway_backend_address_pool_ids = "${azurerm_application_gateway.wp-app-gateway.backend_address_pool[*].id}" # application gateway 백엔드 풀 연결
    }
  }
}
```

NAT gateway 생성

Public IP 생성

```
resource "azurerm_public_ip" "wp-ng-ip" {
  name                = "nat-gateway-publicIP"
  location            = var.location
  resource_group_name = azurerm_resource_group.wp_rg.name
  allocation_method   = "Static"    # IP 할당 방식
  sku                 = "Standard"
  zones               = ["1"]   # IP 주소의 zone
}
```

NAT Gateway에서 사용할 Public IP를 생성한다.
기존의 생성 방법과는 다르게 NAT Gateway가 배치될 가용 영역에
Public IP 주소를 배치 해야한다.
NAT Gateway와 같은 Zone 1에 배치한다.

NAT Gateway.tf

```
resource "azurerm_nat_gateway" "wp-ng" {
  name                    = "nat-Gateway"
  location                = var.location    # 영역
  resource_group_name     = azurerm_resource_group.wp_rg.name # 리소스 그룹 이름
  sku_name                = "Standard"    # sku 이름
  idle_timeout_in_minutes = 10    # TCP 연결에 대한 유효시간 초과 지정(분)
  zones                   = ["1"] # NAT gateway zone
}
```

가용 영역 Zone1에 NAT Gateway를 생성한다.
표준 sku를 사용하고 TCP 연결에 대한 유효 시간은 10분으로 지정한다.
생성이 완료되었다면 NAT Gateway를 사용할 Subnet을 지정해준다.

```
resource "azurerm_nat_gateway_public_ip_association" "public-ip-ng-
connect" {
  nat_gateway_id       = azurerm_nat_gateway.wp-ng.id     # NAT gateway의 이름
  public_ip_address_id = azurerm_public_ip.wp-ng-ip.id    # NAT gateway의 아이피
}

resource "azurerm_subnet_nat_gateway_association" "wp-ng-connect" {
  subnet_id      = azurerm_subnet.wp-web-subnet.id    # 할당할 서브넷
  nat_gateway_id = azurerm_nat_gateway.wp-ng.id   # NAT gateway의 이름
}
```

Output Value로 자주 사용하는 변수 참조하기

output.tf

```
# 애플리케이션 게이트웨이 아이피
output "app_ip" {
  value = azurerm_public_ip.wp-app-gateway-ip
}

# Bastion 아이피
output "Bastion_ip_address" {
  value = azurerm_public_ip.wp-bastion-public-ip
}
```

리소스를 생성할 때 자주 참조해야 하는 변수들을 output value로 선언하여 리소스 생성 완료 시 터미널 상에서 확인할 수 있도록 한다.

05

최종 결과물

1. AWS 리소스 생성 및 동작 확인
2. Azure 리소스 생성 및 동작 확인

2. AWS 리소스 생성 및 동작 확인

최종 결과물 05

인스턴스 ID	인스턴스 상태	인스턴스 유형	상태 검사	경보 상태	가용 영역	퍼블릭 IPv4 DNS	퍼블릭 IPv4 ...
i-069cc3228dd15d71a	실행 중	t2.micro	2/2개 검사 통과	경보 없음	ap-northeast-2a	–	3.36.87.171
i-0720c86de9d6c34b8	실행 중	t2.micro	2/2개 검사 통과	경보 없음	ap-northeast-2a	–	–
i-08b67fb7944c3c88d	실행 중	t2.micro	2/2개 검사 통과	경보 없음	ap-northeast-2c	–	–

Amazon Machine Images(AMI) (1) 정보

휴지통 | EC2 Image Builder

내 소유 | 검색

Name	AMI ID	AMI 이름	원본	소유자	표시 여부
–	ami-03a46dd884bd20a55	project-wordpress_web	471702632719/project-wordpress_web	471702632719	프라이빗

project-alb	project-alb-1960054389.ap-n...	활성	vpc-0c63542ad79d10ae0	ap-northeast-2a, ap-northeast-2c

Auto Scaling 그룹 (1)

편집 | 삭제 | Auto Scaling 그룹 생성

Auto Scaling 그룹 검색

1

이름	시작 템플릿/구성	인스턴스	상태	원하는 용량	최...	최...	가용 영역
terraform-20220502042621568400000000	project-launch-template | 버전 -	2	-	2	2	4	ap-northeast-2a, ap-northeast-2c

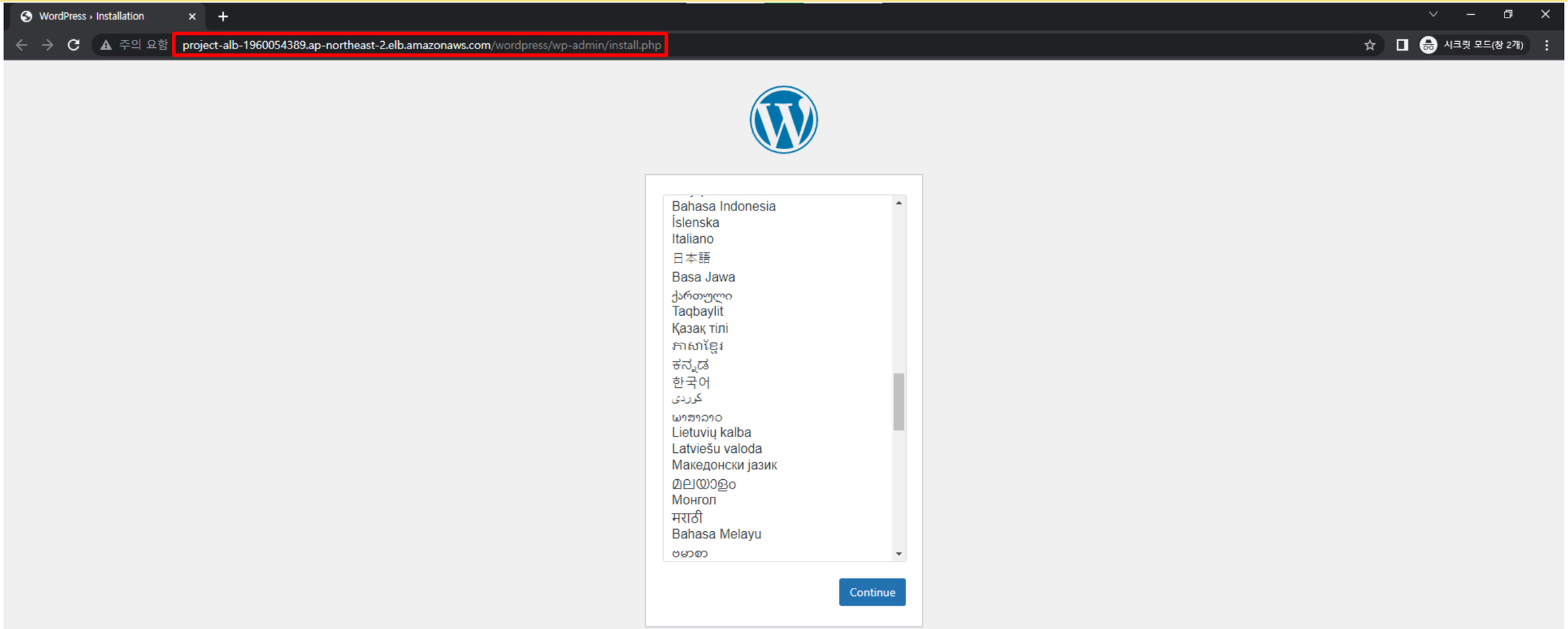
데이터베이스

데이터베이스을(를) 기준으로 필터링

DB 식별자	역할	엔진	리전 및 AZ
project-db	인스턴스	MariaDB	ap-northeast-2a

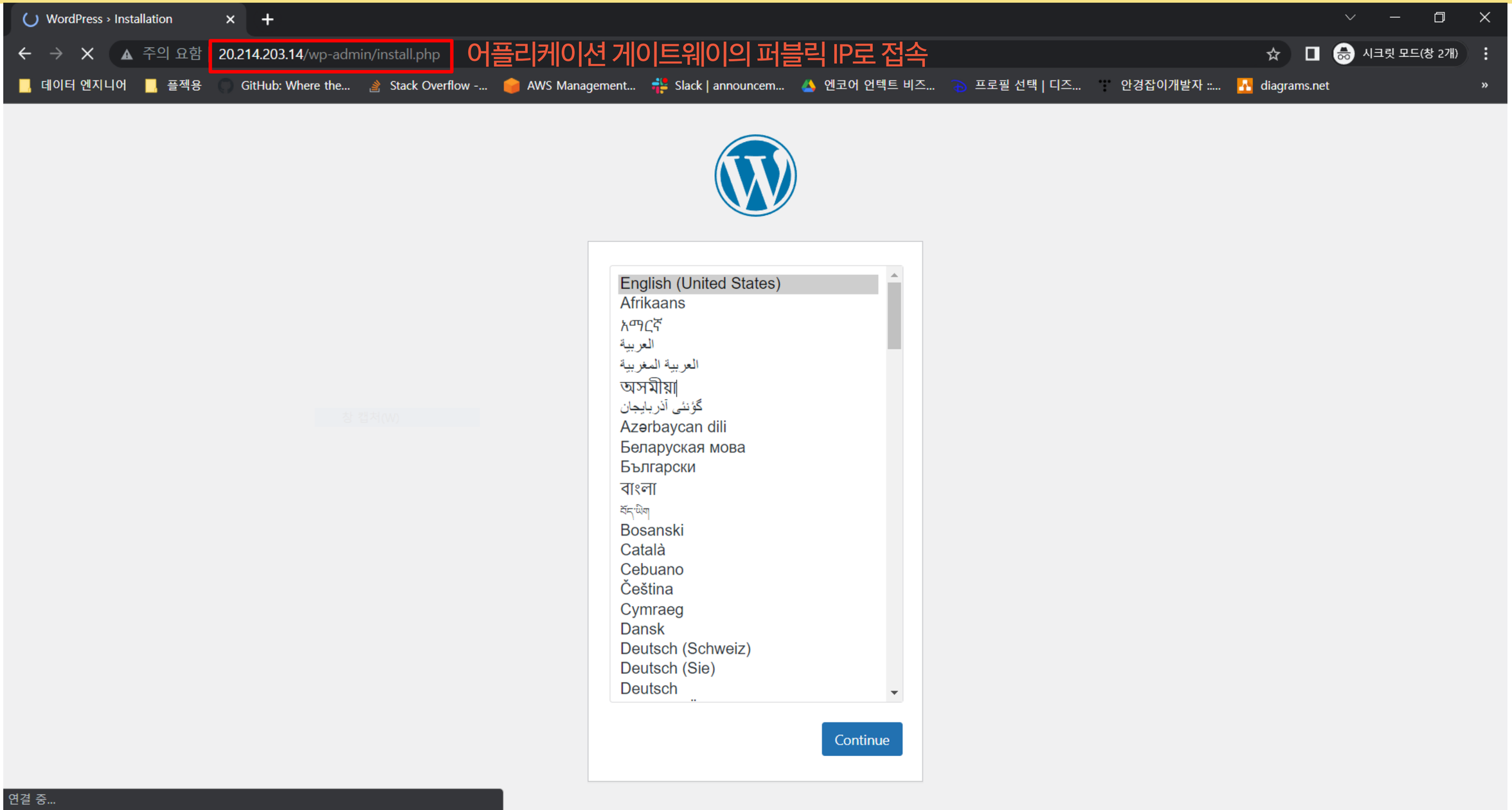
2. AWS 리소스 생성 및 동작 확인

최종 결과물 05



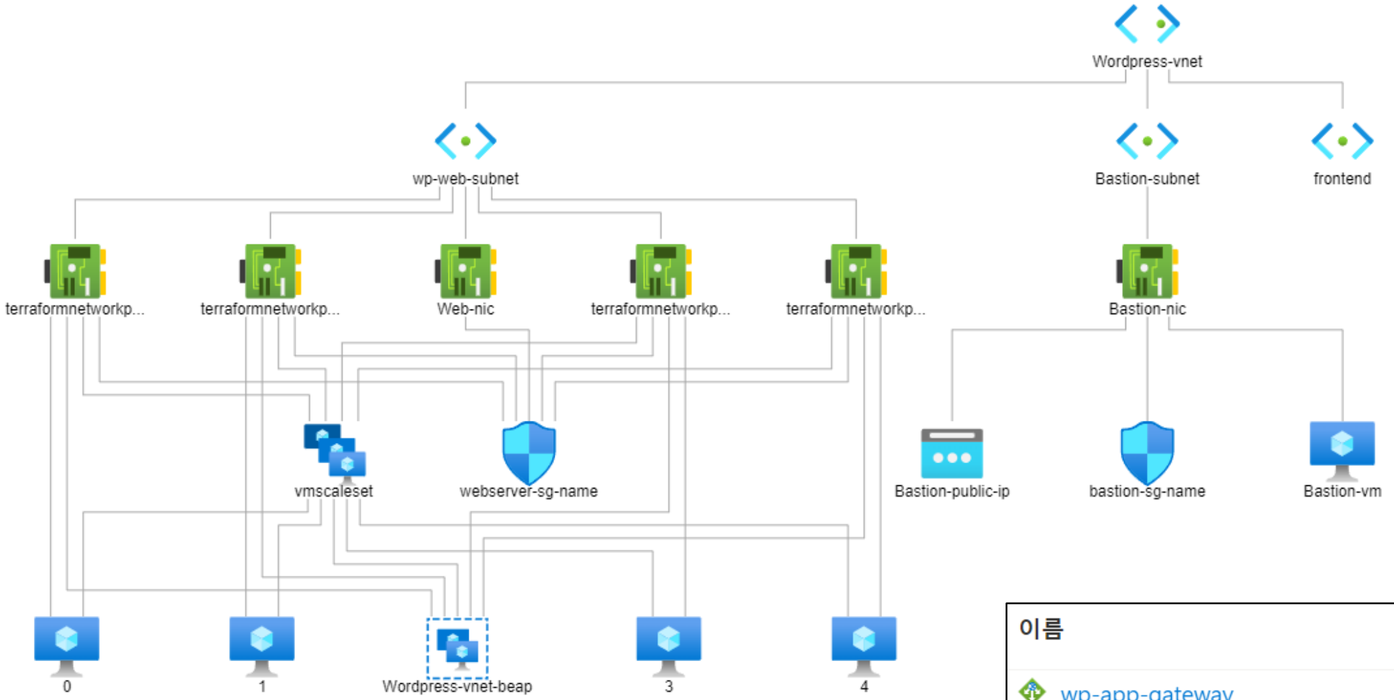
1. Azure 리소스 생성 및 동작 확인

최종 결과물 05



1. Azure 리소스 생성 및 동작 확인

최종 결과물 05



이름	유형	위치	리소스 그룹
wp-app-gateway	애플리케이션 게이...	Korea Central	WordpressResource...
vmscaleset	가상 머신 확장 집합	Korea Central	WordpressResource...
webserver-sg-name	네트워크 보안 그룹	Korea Central	WordpressResource...
my-image	리소스 그룹	Korea South	my-image
wp-app-gateway-ip	공용 IP 주소	Korea Central	WordpressResource...
Bastion-vm	가상 머신	Korea Central	WordpressResource...
Web-nic	일반 네트워크 인터...	Korea Central	WordpressResource...
Wordpress-vnet	가상 네트워크	Korea Central	WordpressResource...
playdata40	구독		

감사합니다 !