

laC를 이용하여 Azure 클라우드에 wordpress 배포하기

목차

1. 프로젝트 개요
 - a. 프로젝트 목적
 - b. 프로젝트 요약
 - c. Azure 시스템 아키텍처
 - i. 시스템 동작 구조
 - ii. 시스템 아키텍처
2. 프로젝트 진행 과정
 - a. Ansible을 이용한 Playbook 작성 및 실행
 1. Ansible 사용을 위한 패키지 설치
 2. Ansible 파일 설정
 3. Ansible Playbook 작성하기
 4. Ansible Playbook 실행하기
 - b. Terraform으로 Azure 클라우드에 wordpress 배포하기
 1. Azure에 대한 Terraform 액세스 설정
 2. Terraform 환경 변수 구성
 3. Packer를 이용한 Wordpress 이미지 생성
 4. Terraform 리소스 그룹 생성
 5. 가상 네트워크 생성
 6. 서브넷 및 네트워크 인터페이스 생성
 7. DB 생성
 8. Wordpress Database 생성 및 방화벽 설정
 9. Application gateway 생성
 10. VMSS생성
 11. NAT Gateway 생성
 12. Output Value로 자주 참조하는 변수 처리하기
 13. terraform apply 및 생성된 리소스 작동 확인하기
3. 프로젝트 최종 결과
 - a. wordpress 서비스 확인
 - b. 느낀점

1. 프로젝트 개요

a. 프로젝트 목적

AWS 서비스와 IaC 도구인 Ansible과 Terraform의 이해를 돕기위해 Terraform 프로바이더인 AWS 클라우드, Azure 클라우드 각각에 고가용성 wordpress를 배포해보는 프로젝트를 진행한다.

b. 프로젝트 요약

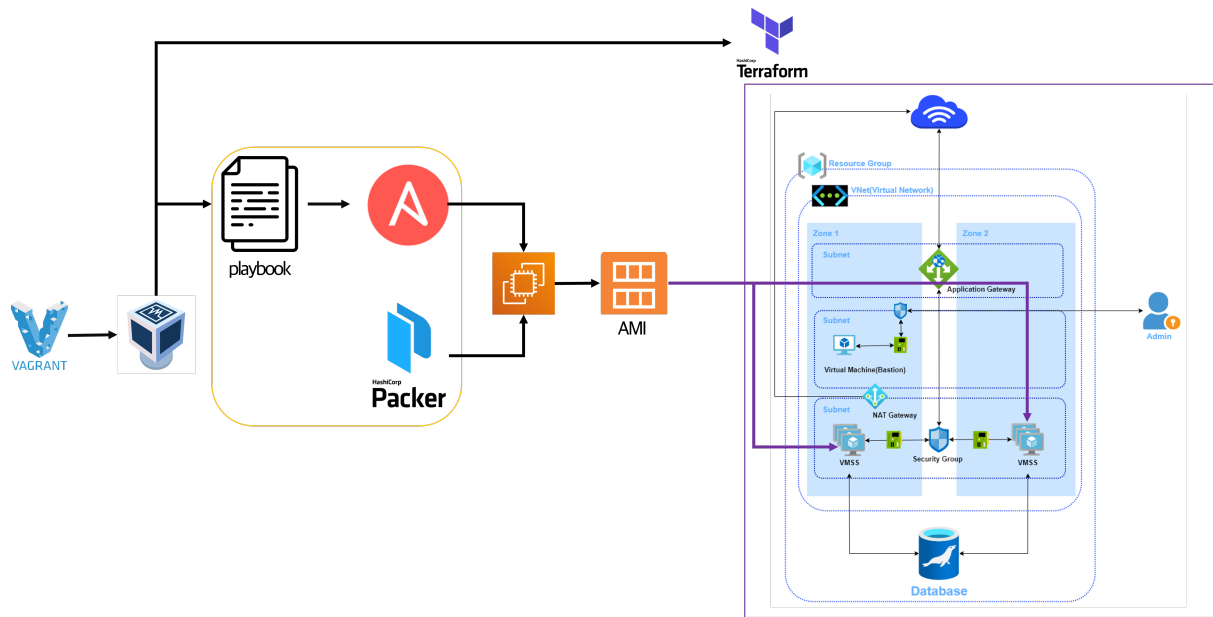
프로젝트 명	IaC를 이용하여 클라우드에 wordpress 배포하기
프로젝트 기간	2022. 04. 26 ~ 2022.05. 02
프로젝트 인원	4인
프로젝트 역할 분담	<p>김효진 Ansible, Terraform 초기 세팅 markdown 문서 정리</p> <p>선우지훈 terraform을 이용한 Azure 리소스 배포 markdown 문서 정리</p> <p>신소희 terraform을 이용한 AWS 리소스 배포 markdown 문서 정리 및 발표 자료 제작</p> <p>홍성민 Ansible Playbook 작성 markdown 문서 정리</p>

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							

c. Azure 시스템 아키텍처

시스템 동작 구조



전체 시스템 동작 구조는 다음과 같다.

가상머신에서 Packer를 이용해 Ansible Playbook을 실행하여 이미지를 만든다.

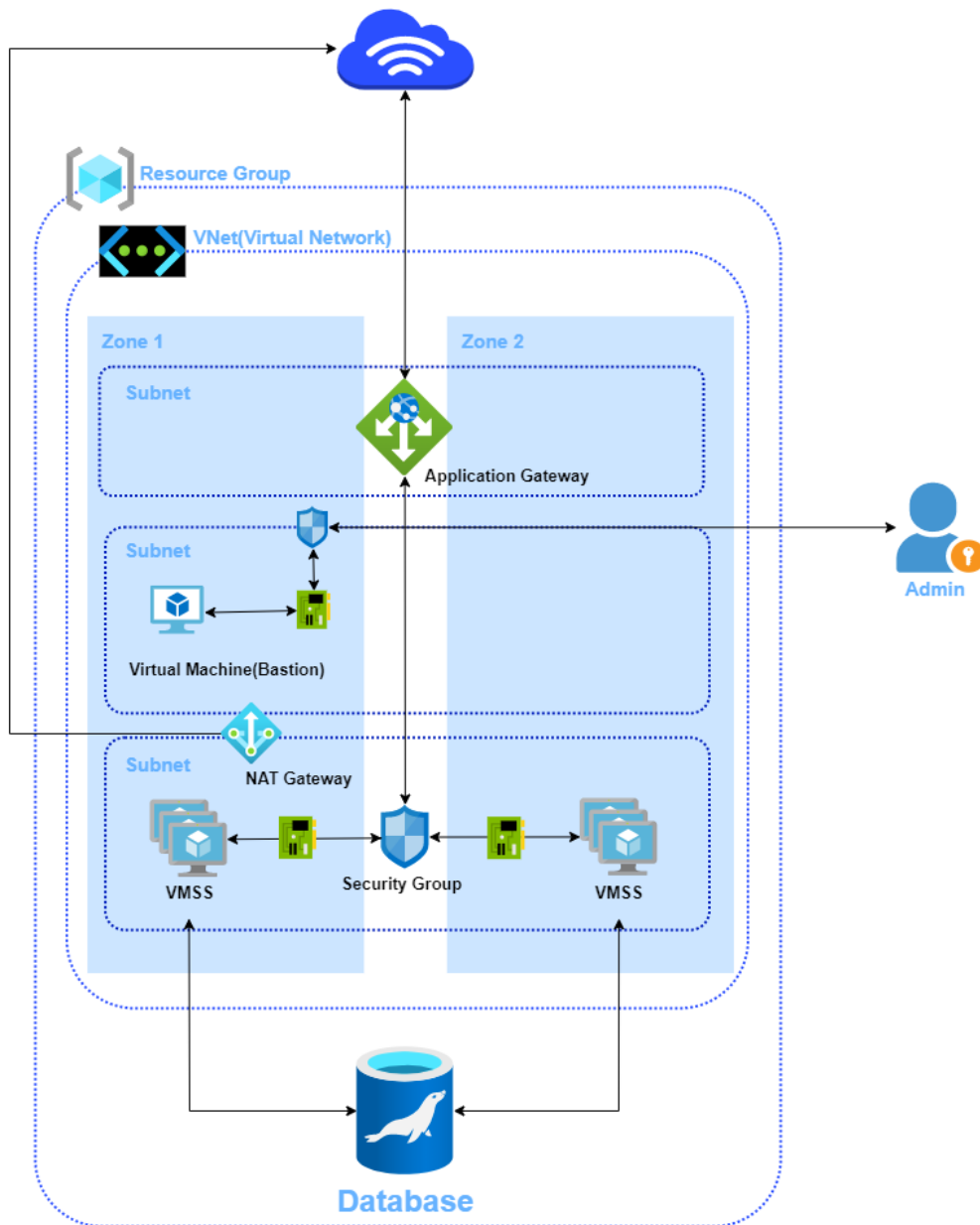
Terraform을 이용해 클라우드에 리소스를 생성할 때 VMSS 이미지에 해당 이미지를 사용한다.

사용한 서비스 목록

Vagrant	가상 시스템 환경을 구축하고 관리하기 위한 도구
Virtual Machine	컴퓨터 시스템을 에뮬레이션하는 소프트웨어
Ansible	오픈 소스 소프트웨어 프로비저닝, 구성 관리, 애플리케이션 전개 IaC 도구
Packer	클라우드 이미지 및 컨테이너 이미지를 자동으로 빌드하는 도구
Terraform	인프라스트럭처 구축 및 운영의 자동화를 지향하는 오픈 소스 IaC 도구
Visual Studio Code	디버깅 지원과 Git 제어, 구문 강조 기능등이 포함된 소스 코드 편집기

시스템 아키텍처

Azure 클라우드 시스템 아키텍처



전체 아키텍처 개요

Terraform을 이용한 Azure Wordpress 아키텍처는 **Korea Central Region**에 위치하고 있다.

WordpressResourcegroup Resource Group으로 전체 아키텍처를 구성하는 모든 **Instance**를 관리한다.

하나의 Resource Group 안에 **가상 네트워크 서비스**를 이용하여 개별 네트워크를 구성함으로써 DB서버와 격리된 환경을 제공한다.

Zone1, Zone2 두 곳의 **Availability Zone**을 지정하고 **가상 머신 확장 집합(VMSS)**을 통해 Instance를 관리하도록 구성하여 **Single Point of Failure**를 방지하고 **HA**를 확보하기 위해 설계되었다.

가상 머신 확장 집합 VMSS는 두 곳의 가용성 영역에 각각 **2개씩 총 4개의 VM**을 관리하도록 설계하였다. 여기서 사용된 VM 이미지는 **Packer**와 **Ansible Playbook**을 활용하여 만든 사용자 VM 이미지를 활용한다.

사용한 Azure 서비스 목록

Region	Korea Central
Resource Group	리소스 그룹

Vnet	가상 네트워크
Availability Zone	가용성 영역
Application Gateway	애플리케이션 게이트웨이
NSG	네트워크 보안 그룹
NAT Gateway	NAT 게이트웨이
VM	Bastion Host Virtual Machine
VMSS	가상 머신 확장 집합
Azure Database for MariaDB Server	Azure 관리형 데이터베이스 서비스(Maria DB)

2. 프로젝트 진행 과정

a. Ansible을 이용한 Playbook 작성 및 실행

1. Ansible 사용을 위한 패키지 설치

```
# python3.9 설치
$ sudo yum groupinstall -y "Development Tools"
$ sudo yum install -y openssl-devel libffi-devel bzip2-devel
$ sudo yum install -y wget
$ wget <https://www.python.org/ftp/python/3.9.10/Python-3.9.10.tgz>
$ tar xvf Python-3.9.10.tgz
$ cd Python-3.9.10/
$ ./configure --enable-optimizations
$ sudo make altinstall
$ /usr/local/bin/python3.9 -m pip install --upgrade pip

# ansible설치
$ pip3.9 install --upgrade setuptools
$ pip3.9 install setuptools-rust
$ pip3.9 install --upgrade pip
$ pip3.9 install ansible
```

2. Ansible 파일 설정

ansible.cfg 파일 설정

ansible.cfg

```
[defaults]
inventory = azure_rm.yaml
remote_user = azureuser
become = true
ask_pass = false
```

Ansible 설정 파일을 지정한다.

- inventory: 인벤토리 파일을 지정한다(azure_rm.yaml가 인스턴스 목록을 불러온다.).
- remote_user: 로그인 유저를 지정한다. (Azure VM : azureuser)
- become: sudo로 실행한다(true).
- ask_pass: 키 기반으로 인증한다(false).

inventory 파일 설정

azure_rm.yaml

```
plugin: azure.azcollection.azure_rm
include_vm_resource_groups:
  - mytestweb_group
auth_source: auto
```

동적 인벤토리 모듈을 사용하여 Azure 계정에서 가동 중인 VM에 플레이북 적용 가능하도록 함 (이후 Packer를 사용하여 Test 에만 사용)

- plugin: 인벤토리 모듈의 플러그인을 불러온다(aws_ec2).
- regions: AWS 리전을 지정한다(ap-northeast-2, 서울리전).
- aws_profile: 로그인할 프로파일을 지정한다(default, aws configure로 설정).

3. Ansible Playbook 작성하기

httpd.yaml

```
---
- hosts: all                # playbook 실행 호스트 지정
  become: True              # sudo 권한
  roles:                    # 역할 파일 로드
    - common
```

Ansible-Playbook을 실행하는 파일이다.

- hosts: inventory에 있는(모듈이 불러온) 모든 호스트를 실행한다(all).
- become: sudo 권한으로 실행한다(true).
- roles: 역할 파일을 로드한다.

Ansible 파일 및 역할 디렉토리 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
```

tasks

Playbook에서 실제로 실행되는 작업을 저장한 디렉토리이다.

main.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
```

Amazon Linux 2일 경우 PHP 7.4버전을 설치할 수 있게 리포지토리를 세팅한다.
기본적으로는 5.4버전이 설치되어 있다.
amazon-linux-extras는 대응되는 모듈이 없어 command로 입력한다.

```
- 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
```

CentOS일 경우 PHP 7.4를 설치하기 위해 Remi 리포지토리를 설치한다.
Remi의 기본 버전은 PHP 5.4이기에 7.4로 바꾸어준다.

```
- name: Flush Handlers # 핸들러 즉시 활성화
meta: flush_handlers
tags:
  - wordpress
  - amazon_linux
  - centos
```

핸들러를 알림 받을 때 실행할 수 있게 세팅한다.

```
- name: Install Web Service # 웹서버 설치
block:
  - name: Disable PHP54 # PHP54 삭제(Amazon Linux 2 Only)
    yum:
      name: php
      state: absent
when: ansible_facts['distribution'] == "Amazon"
  - name: Install Apache, PHP, mariadb # 아파치, PHP74, DB 클라이언트 설치
    yum:
      name: '{{ wordpress["centos"]["packages_httpd"] }}'
      state: present
  - name: Set httpd port # httpd 포트 변경
    lineinfile:
      path: /etc/httpd/conf/httpd.conf
      regexp: '^Listen'
      line: 'Listen {{ wordpress["svc_port"] }}'
    notify: Restart httpd
  - name: Start httpd service # httpd 서비스 시작
    service:
      name: httpd
      state: started
      enabled: yes
tags:
  - wordpress
  - amazon_linux
  - centos
```

웹서버(Apache2)와 PHP7.4, SQL 클라이언트를 설치한다.
 Amazon Linux 2는 PHP5.4가 사전 설치되어 있기에 제거 후 설치한다.
 웹서버 포트를 변경할 경우, 핸들러를 작동시켜 웹서버를 재시작해준다.

```
- name: Active Wordpress System for Amzon Linux 2 # 워드프레스 설치
block:
  - name: Download Wordpress System for Amzon Linux 2 # 워드프레스 다운로드
    get_url:
      url: '{{ wordpress_url }}'
      dest: /home/ec2-user
  - name: Decompress Archive file for Amzon Linux 2 # 압축 해제
    unarchive:
      src: '/home/ec2-user/{{ wordpress_filename }}'
      remote_src: yes
      dest: /var/www/html/
      owner: apache
      group: apache
when: ansible_facts['distribution'] == "Amazon"
tags:
  - wordpress
  - amazon_linux
```

Amazon Linux 2에 워드프레스를 홈 디렉토리에 다운로드하고 HTML 디렉토리에 압축을 해제한다.

```
- name: Active Wordpress System for CentOS 7 # 워드프레스 설치
block:
  - name: Download Wordpress System for CentOS 7 # 워드프레스 다운로드
    get_url:
      url: '{{ wordpress_url }}'
      dest: /home/centos
  - name: Decompress Archive file for CentOS 7 # 압축 해제
    unarchive:
      src: '/home/centos/{{ wordpress_filename }}'
      remote_src: yes
      dest: /var/www/html/
      owner: apache
      group: apache
when: ansible_facts['distribution'] == "CentOS"
tags:
  - wordpress
  - centos
```

CentOS에 워드프레스를 홈 디렉토리에 다운로드하고 HTML 디렉토리에 압축을 해제한다.

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

템플릿을 사용하여 wp-config.php에 데이터베이스 로그인 정보를 입력한다.

handlers

핸들러 작업을 저장한 디렉토리이다.

main.yaml

```
---
# handlers file for common
- name: Restart httpd # httpd 서비스 재시작
```



```
service:
  name: httpd
  state: restarted
```

알림을 받으면 웹서버의 서비스를 재시작한다.

template

템플릿 파일을 저장한 디렉토리이다.

`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 -----
```

데이터베이스 로그인 정보를 세팅한다.

- `define('DB_NAME', '{{ database["name"] }}');`
데이터베이스의 이름을 지정한다.
- `define('DB_USER', '{{ database["user"] }}');`
데이터베이스의 로그인할 사용자를 지정한다.
- `define('DB_PASSWORD', '{{ database["pwd"] }}');`
데이터베이스의 로그인할 사용자의 암호를 지정한다.
- `define('DB_HOST', '{{ database_source }}');`
데이터베이스로의 접근 경로를 지정한다.

vars

변수를 저장한 디렉토리이다.

```
---
# vars file for common

repo_set: # repo
  pkg: <https://rpms.remirepo.net/enterprise/remi-release-7.rpm>
  safe:
    name: remi-safe
    mirror: <http://cdn.remirepo.net/enterprise/7/safe/mirror>
  php74:
    name: remi-php74
    mirror: <http://cdn.remirepo.net/enterprise/7/php74/mirror>
  gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-remi

wordpress: # 웹서버 설치
  svc_port: 80
  package:
    packages_httpd: httpd,php,php-mysqlnd,mariadb,mariadb-server,python2-PyMySQL

wordpress_version: 5.9.3 # 워드프레스 다운로드
wordpress_filename: "wordpress-{{ wordpress_version }}.tar.gz"
wordpress_url: "<https://wordpress.org/>{{ wordpress_filename }}"

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

database_source: tmp_endpoint # 데이터베이스 소스
```

repo_set: CentOS의 Remi Repo를 지정한다.
wordpress: 패키지 설치를 위한 정보를 저장한다.
wordpress_version,filename,url: 워드프레스 다운로드 정보와 버전을 지정한다.
database: 로그인할 데이터베이스 정보를 지정한다.
사전에 수동으로 지정하거나 Terraform으로 수정한다.

Ansible Playbook 실행하기

```
cd /home/사용자명/playbook 위치 # ex) cd /home/vagrant/test/images/httpd
ansible-playbook httpd.yaml
```

Playbook을 실행한다.

b. Terraform으로 Azure 클라우드에 wordpress 배포하기

1. Azure에 대한 Terraform 액세스 설정

Terraform에서 Azure로 리소스를 프로비전할 수 있도록 Azure AD 서비스 사용자를 만든다.
서비스 사용자는 Azure 구독에서 리소스를 프로비전하는 권한을 Terraform 스크립트에 부여한다.

선택한 구독을 사용하려면 `az account set` 명령을 사용하여 이 세션에 대한 구독을 설정한다.
사용하려는 구독에서 반환된 SUBSCRIPTION_ID 필드 값을 보유하도록 id 환경 변수를 설정한다.

```
$ az account set --subscription="0e337d72-xxxx-xxxx-xxxx-9968126bxxxx"
```

이제 Terraform에 사용할 서비스 사용자를 만들 수 있다.
다음과 같이 `az ad sp create-for-rbac`를 사용하고, 범위를 구독으로 설정한다.

```
$ az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/2e5d848e-xxxx-xxxx-xxxx-fd25ae915bcd"
```

- `appId`, `password`, `sp_name` 및 `tenant`가 반환된다. `appId` 및 `password`를 기록해 둔다.

2. Terraform 환경 변수 구성

Azure AD 서비스 사용자를 사용하도록 Terraform을 구성하려면 `provider.tf` 파일에 다음 변수를 설정하고, `variable.tf` 파일에 생성된 정보들을 변수로 저장하면 Azure에 대한 Terraform 액세스 설정이 끝난다.

```
provider "azurerm" {
  features {}

  subscription_id = var.subscription_id
  client_id       = var.client_id
  client_secret   = var.client_secret
  tenant_id       = var.tenant_id
}
```

Terraform의 버전 정보를 확인한다.

```
# terraform version
Terraform v1.1.9
on linux_amd64
```

3. Packer를 이용한 Wordpress 이미지 생성

Packer를 이용해서 사용자 인스턴스 이미지를 생성할 수 있다. 이번 프로젝트에서는 Ansible playbook을 통해 wordpress가 배포된 이미지를 Azure의 가상 머신 확장 집합에 사용할 것이다.

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.

- 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"
}
```

- 리소스 그룹이 성공적으로 생성된 것을 알 수 있다.
- 다음으로 Packer가 설치된 로컬 Vagrant VM에서 빌드 파일을 작성한다.

- `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": []
  ]}
```

```
    }
  }
}
```

- 파일 상단에 `variables` 블록에 변수를 정의하고 `builders` 블록에 이미지 빌드에 필요한 정보들을 작성한다. 여기서 핵심은 `provisioners` 블록인데, `ansible` 프로비저너를 이용해서 플레이북을 실행한 후 빌드된 결과를 이미지로 저장하는 방식이다.
- `ansible` playbook이 존재하는 디렉토리의 구조는 다음과 같다.

- Ansible 디렉토리 구조

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

`Role` 를 이용해 코드의 구성을 체계화하고 모듈화 한 디렉토리이다.

`ansible.cfg` 에 인벤토리의 정보, 관리자 권한 여부 접속 방식에 대한 정보들이 작성되어 있다.

`tasks/main.yml` 파일에는 `wordpress` 배포에 필요한 수행 작업들을 모듈을 이용해 실행하도록 하는 코드가 작성되어 있다.

`templates`과 `vars`를 사용해 설정 파일에 대한 동적인 변수 설정이 가능하다.

이번 프로젝트에서는 `wordpress`의 `wp-config.php` 파일에 `jinja` 템플릿을 적용해 DB 정보를 관리한다.

- `vars.main.yml`

```
# Terraform으로 데이터베이스 삽입
database: # 데이터베이스 로그인
  svc_port: 3306
  name: wordpress
  user: adminx
  pwd:

database_source: wp-mariadb-server.mariadb.database.azure.com # 데이터베이스 소스
database_collate: utf8_unicode_ci
```

- `wp-config.php.j2`

```
// ** 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 }}' );

/** Database charset to use in creating database tables. */
```

```
define( 'DB_CHARSET', 'utf8' );

/** The database collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '{{ database_collate }}' );
```

- vars 폴더의 `main.yml` 에 변수를 설정하고 `wp-config.php.j2` 에 적용하면 설정 파일을 쉽게 관리할 수 있다.
- 빌드 파일 작성 후 실행하기 전 프로비저너를 초기화 한 후 `packer validate` 명령어로 파일 검증을 진행한다.

- Packer 초기화 및 검증

```
packer init .

packer validate centos.json
The configuration is valid.
```

- 빌드 파일에 문제가 없다면 `packer build` 명령어로 배포한다.

- Packer 프로비저닝

```
packer build centos.json

azure-arm: output will be in this color.

==> azure-arm: Running builder ...
==> azure-arm: Getting tokens using client secret
==> azure-arm: Getting tokens using client secret
    azure-arm: Creating Azure Resource Manager (ARM) client ...
```

- 프로비저닝이 진행되는 과정을 로그로 확인할 수 있는데 wordpress 배포에 필요한 ansible playbook task들이 잘 실행되는 것을 알 수 있다.

- Tasks 실행 과정

```
azure-arm: TASK [Gathering Facts] *****
azure-arm: ok: [default]
azure-arm:
azure-arm: TASK [common : Install Remi Repo Package] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Disable Remi php54 Repo] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Enable Remi php74 Repo] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Flush Handlers] *****
azure-arm:
azure-arm: TASK [common : Install Apache, PHP, mariadb] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Set httpd port] *****
azure-arm: ok: [default]
azure-arm:
azure-arm: TASK [common : Start httpd service] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Download Wordpress System for CentOS 7] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Decompress Archive file for CentOS 7] *****
azure-arm: changed: [default]
azure-arm:
azure-arm: TASK [common : Copy Database Configure File for Wordpress] *****
```

- 이미지 빌드가 성공적으로 진행 되면 이미지 정보를 반환해준다.

```
==> Wait completed after 5 minutes 48 seconds

==> Builds finished. The artifacts of successful builds are:
--> azure-arm: Azure.ResourceManagement.VMImage:

OSType: Linux
ManagedImageResourceGroupName: my-image
ManagedImageName: wordpress
ManagedImageId: /subscriptions/0e337d72-f370-4cb8-9749-9968126b71df/resourceGroups/my-image/providers/Microsoft.Compute/images/wordpress
ManagedImageLocation: koreacentral
```

4. Terraform으로 리소스 그룹 생성

Terraform을 이용해 리소스 그룹을 생성한다.

Azure는 리소스 그룹이 있어야 여러가지 인스턴스를 생성할 수 있다.

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

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

- azurerm의 resource_group 리소스를 이용하여 리소스 그룹을 생성할 수 있다. 영역은 "koreacentral"로 지정하여 `variable.tf` 파일에 기록한다.

5. 가상 네트워크 생성

리소스 그룹의 생성이 완료되면 가상 네트워크를 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
}
```

- CIDR은 10.0.0.0/16로 설정하여 리소스 그룹과 같은 영역에 배치하고 할당한다.

6. 서브넷 및 네트워크 인터페이스 생성

가상 네트워크의 생성이 완료되었다면 서브넷을 생성하여 할당할 수 있다.

AWS와 다르게 Azure는 특이한 Subnet의 개념을 가지고 있다.

AWS에서는 하나의 VPC의 가용 영역에 Public, Private 서브넷을 배치한다면, Azure에서는 서브넷이 생성되면 모든 가용영역에 적용된다. 또한 Public과 Private의 개념이 없어서 만약 Private 서브넷을 생성하고 싶다면 Public IP를 할당하지 않으면 된다.

- 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"]
}

```

- 먼저 Bastion Host로 사용할 Subnet을 생성한다.

Subnet의 네트워크 인터페이스를 할당하고 이 네트워크 인터페이스에 Public IP를 할당하면 Public subnet의 기능을 할 수 있는 서브넷이 생성된다.

- 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 네트워크 인터페이스에 할당할 Public IP 주소를 생성한다.

여기서 중요한 것은 IP에도 서비스 영역을 설정할 수 있고 zones 옵션을 활용하면 해당 가용영역에 IP 주소가 생성되어 가용 영역별로 IP주소를 관리 할 수 있다.

- 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.id
  }
}

```

- Public IP와 Subnet을 생성했다면 네트워크 인터페이스를 생성해서 할당 시켜주면 Subnet이 완성된다.

- 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")
}
}
```

- Bastion 서비스를 사용할 수 있지만 Private 서버에 접속하기 위한 절차가 복잡하기 때문에 역할을 대신할 VM 이미지를 생성한다.

- Web server Host Subnet(VMS)

```
resource "azurerm_subnet" "wp-web-subnet" {
  name = "wp-web-subnet"
  resource_group_name = azurerm_resource_group.wp_rg.name
  virtual_network_name = azurerm_virtual_network.wp_network.name
  address_prefixes = ["10.0.50.0/24"]
}
```

- Web server Host도 Bastion Host의 생성 과정과 똑같이 생성해준다.

7. DB 생성

Wordpress의 DataBase를 생성한다. mariadb 10.2 버전을 사용하고 VM 인스턴스에 직접 설정하는 것이 아닌 Azure의 관리형 데이터베이스 서비스인 Azure Database for MariaDB Server를 이용한다.

- 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
```

- 앞서 생성했던 인스턴스들과 마찬가지로 "WordpressResourcegroup" 리소스 그룹에 koreacentral 영역에 배치한다.
- Azure의 데이터베이스 서비스는 가상 네트워크와는 격리되고 리소스 그룹안에 할당된다. 따라서 가상 네트워크 주소를 지정해 줄 필요가 없다.
- 관리자 이름과 패스워드와 같은 민감한 정보들은 변수 처리해두었고, 패스워드의 경우 sensitive = true 옵션을 설정하여 관리한다.
- 백업 데이터의 보유 기간은 7일이고 geo_redundant_backup는 사용하지 않는다.
- DB의 정보들은 외부 사용자가 접근하면 안되기 때문에 Public 접근을 허용하지 않도록 설정했다. 또한 작업의 편의상 SSL 연결은 사용하지 않았다.

8. Wordpress Database 생성 및 방화벽 설정

```
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 지정
}
```



```
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]
}
```

- 데이터베이스 서버를 생성했다면 워드프레스에서 사용할 **wordpress** DB를 생성한다.
- 마지막으로 데이터베이스 서버에 사용할 방화벽 규칙을 알맞게 지정해 주고 앞서 두 과정보다 방화벽이 먼저 생성되면 의존성 충돌이 발생하므로 `depends_on` 옵션을 활용하여 의존성 설정을 해준다.

9. Application gateway 생성

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

- 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(vms)
    - 백엔드 http 설정
  - http 리스너 설정
  - 라우팅 규칙 설정
```

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.

- 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"]
}
```

- Application gateway 리소스 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 비정상상으로 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 선회도 활성화 여부
    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
}
}

```

```

locals {
    backend_address_pool_name = "${azurerm_virtual_network.wp_network.name}-beap" # 백엔드 풀 주소 이름
    frontend_port_name        = "${azurerm_virtual_network.wp_network.name}-feport" # 프론트엔드 포트 이름
    frontend_ip_configuration_name = "${azurerm_virtual_network.wp_network.name}-feip" # 프론트엔드 ip_config 이름
    http_setting_name         = "${azurerm_virtual_network.wp_network.name}-be-htst" # http 세팅 이름
    listener_name              = "${azurerm_virtual_network.wp_network.name}-httplstn" # http 리스너 이름
    request_routing_rule_name  = "${azurerm_virtual_network.wp_network.name}-rqrt" # 요청 라우팅 규칙의 이름
    redirect_configuration_name = "${azurerm_virtual_network.wp_network.name}-rdrcfg" # redirect_configuration_name
    backend_http_probe         = "${azurerm_virtual_network.wp_network.name}-httpprobe" # 백엔드 상태프로브 이름
}

```

< 프론트 엔드 설정 >

- `sku` : 애플리케이션에서 사용할 sku를 지정한다. Standard_v2를 사용해야 AZ 영역 확장 설정이 가능하다.
- `autoscale_configuration` : 애플리케이션 게이트웨이에 auto scaling을 사용할때 최소 인스턴스 수, 최대 인스턴스 수를 지정할 수 있다.
- `gateway_ip_configuration` : Subnet에 애플리케이션 게이트웨이를 할당한다.
- `frontend_port` : 프론트엔드 포트를 설정한다. 기본 값은 80번 포트다.
- `frontend_ip_configuration` : 프론트엔드 IP를 설정한다.

< 백엔드 설정 >

- `backend_address_pool` : 백엔드 풀을 설정한다. 이번 프로젝트에서는 VMSS를 지정한다. 해당 블록에서는 백엔드 풀에 대한 이름 정보만 기입하고 연동 설정은 `VMSS.tf` 파일에서 진행한다.
- `backend_http_settings` : 백엔드 http를 설정한다. cookie_based_affinity 옵션을 사용하면 쿠키 기반 선호도를 활성화 할 수 있다.

< http 리스너 및 라우팅 규칙 생성 >

- `http_listener` : http 리스너를 설정한다. 이름과 프로토콜을 지정할 수 있다.
- `request_routing_rule` : 라우팅 규칙(회람 규칙)를 설정한다. rule_type에서 라우팅 규칙 타입을 지정할 수 있다.

10. vmss생성

- `data block`

```
data "azurerm_resource_group" "image" {
  name = var.packer_resource_group_name
  depends_on = [azurerm_resource_group.wp_rg] # 의존성
} # 이미지가 생성될 리소스 그룹 설정

data "azurerm_image" "image" {
  name = var.packer_image_name # Packer로 생성된 이미지 이름
  resource_group_name = data.azurerm_resource_group.image.name # 이미지가 존재하는 리소스 그룹의 이름
  depends_on = [azurerm_resource_group.wp_rg] # 의존성
}
```

Packer로 생성한 wordpress 이미지에 대한 정보를 Data source 블록을 통해 참조한다.

tf 파일 작성에 필요한 인스턴스 타입 ID, 이미지 ID 등은 콘솔에서 확인하거나, Terraform이 제공하는 Data sources를 이용하여 가져올 수 있다.

Data sources는 tf 파일 안에 작성하며, 가져온 정보는 수정할 수 없고 오직 참조만 가능하다.

- `VMSS.tf`

```
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 프로파일 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}" # appl
    }
}
}
}

```

- `source_image_id` : vmss에서 사용할 VM이미지를 지정할 수 있다. Data Source 블록에서 가져온 Packer 이미지의 정보를 참조하도록 설정한다.
- `user_data` : base64code를 사용하여 생성되는 이미지에 shell 명령어를 입력할 수 있다. SELinux를 비활성화 하고 httpd와 mariadb를 재시작하는 명령어를 사용하도록 설정한다.
- `zones` : 고가용성을 위한 가용 영역을 지정한다.
- `overprovision` : VMSS를 오버 프로비전 해야 하는지 여부를 지정한다.
- `ip_configuration` : vmss의 ip 정보를 설정한다. `application_gateway_backend_address_pool_ids` 옵션을 활용하여 앞에서 생성했던 애플리케이션 게이트웨이 백 엔드 풀에 연결하도록 설정한다.

11. 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
}
```

```
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
}
```

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

12. Output Value로 자주 사용하는 변수 참조하기

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

- `output.tf`

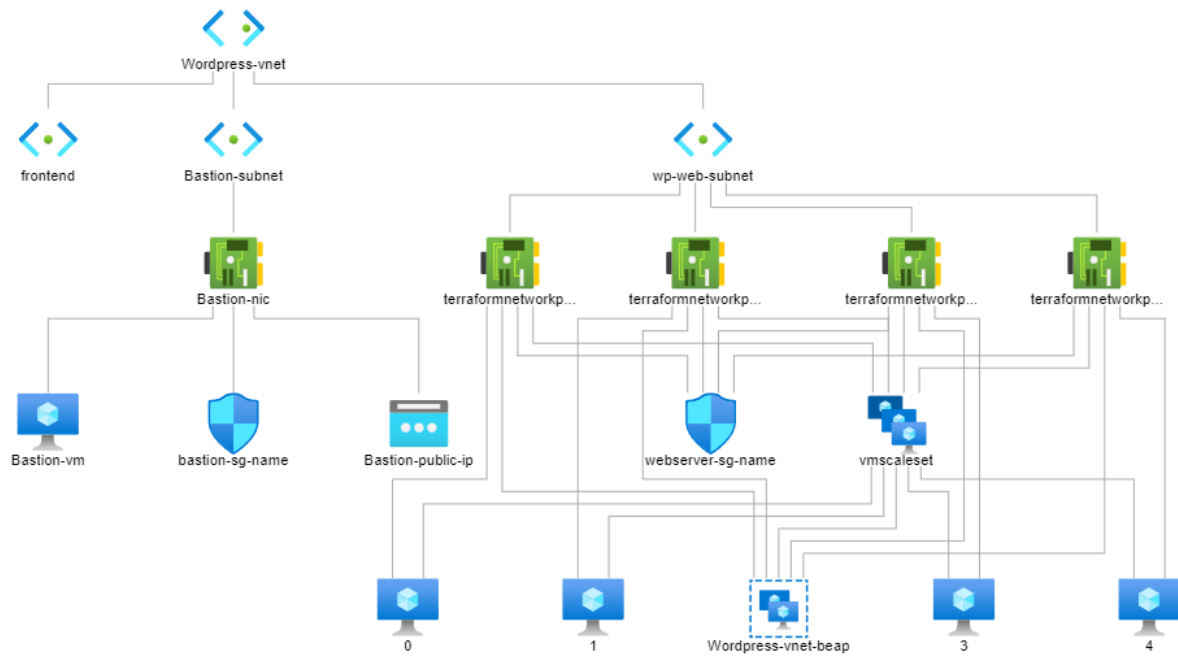
```
# 애플리케이션 게이트웨이 아이피
output "app_ip" {
  value = azurerm_public_ip.wp-app-gateway-ip
}

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

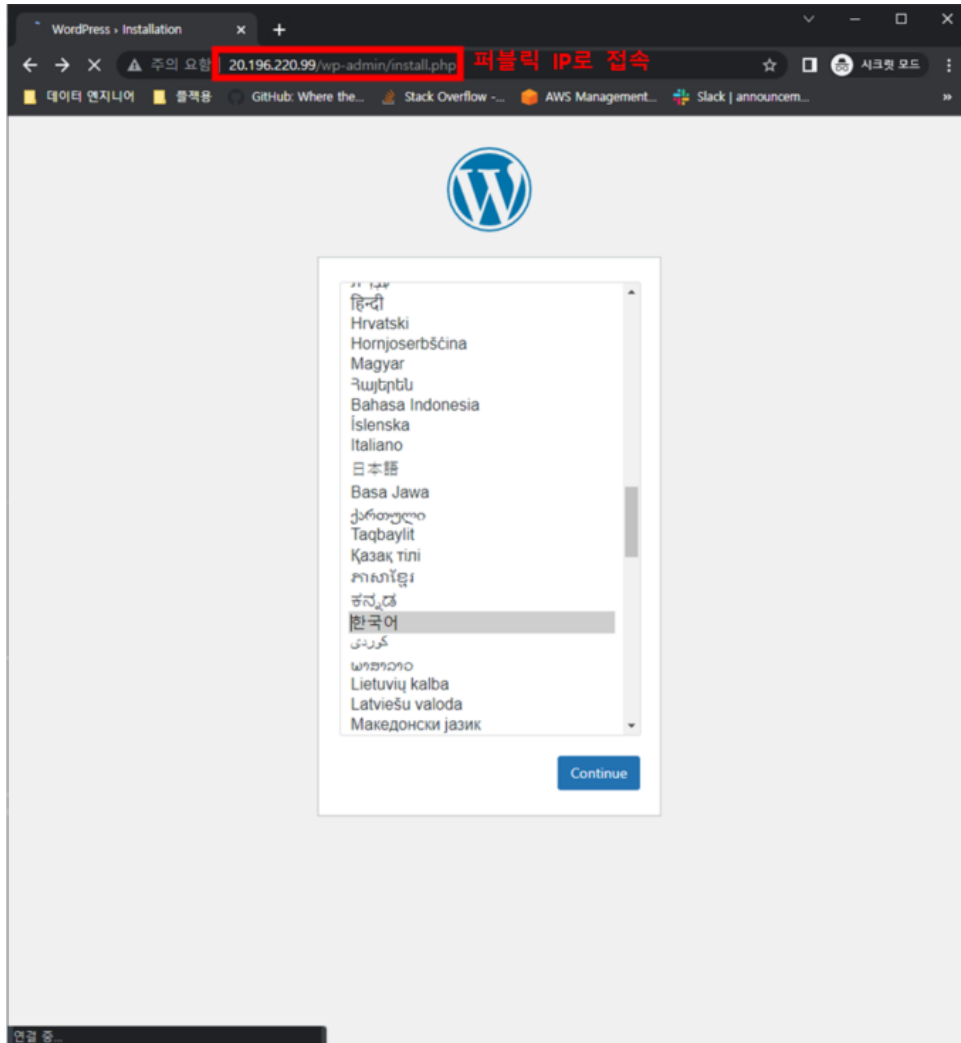
13. terraform apply 및 생성된 리소스 작동 확인하기

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		

Terraform으로 생성한 모든 리소스들을 리소스 그룹에서 확인할 수 있다.



Azure에서 제공하는 네트워크 토폴로지 시각화 서비스를 확인해보면 처음 설계한 아키텍처대로 모든 리소스들이 연결되어 있는 것을 확인할 수 있다.

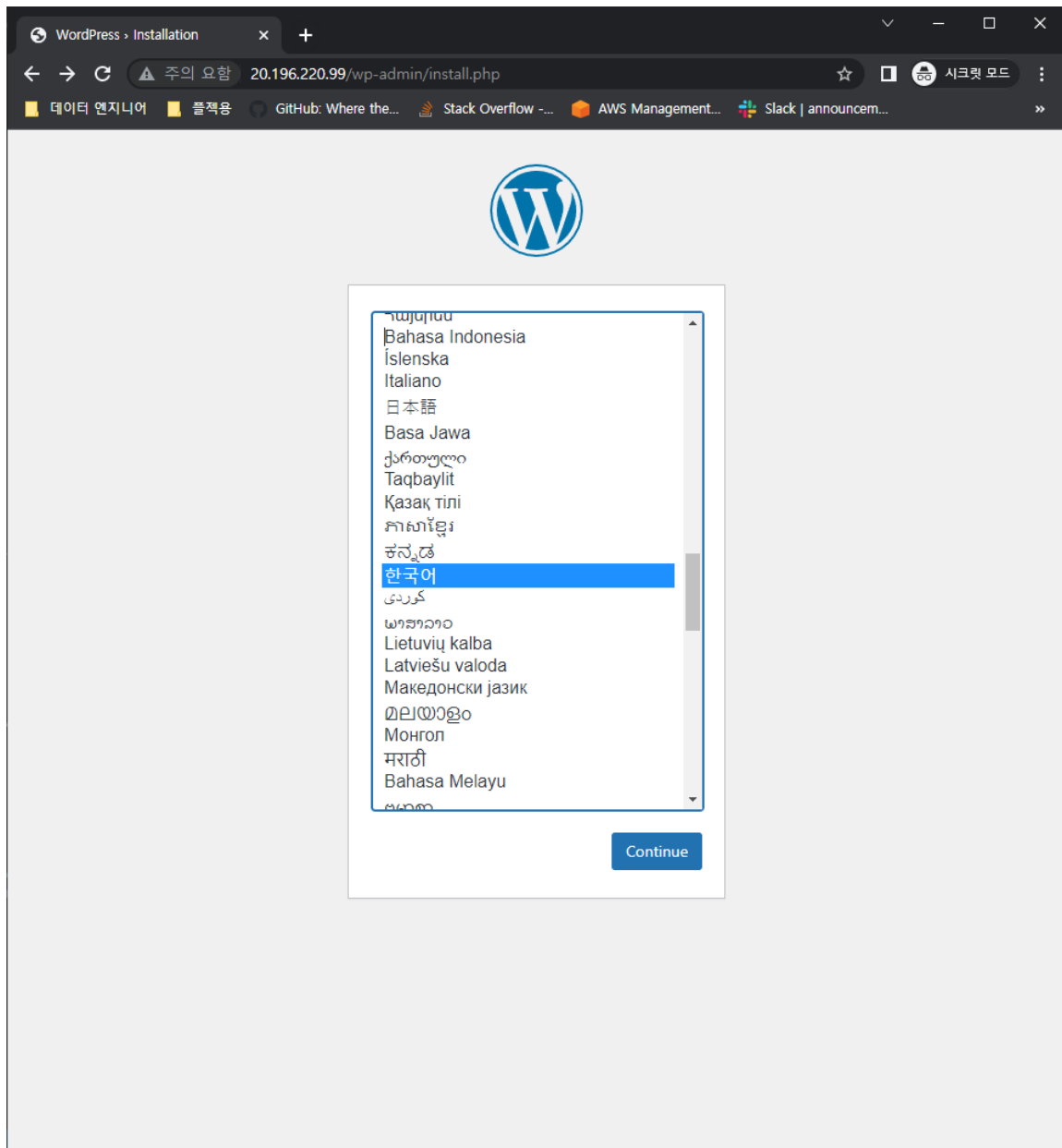


어플리케이션 게이트웨이의 퍼블릭 IP를 통해 접속했을 때 워드프레스가 정상적으로 서비스 된다.

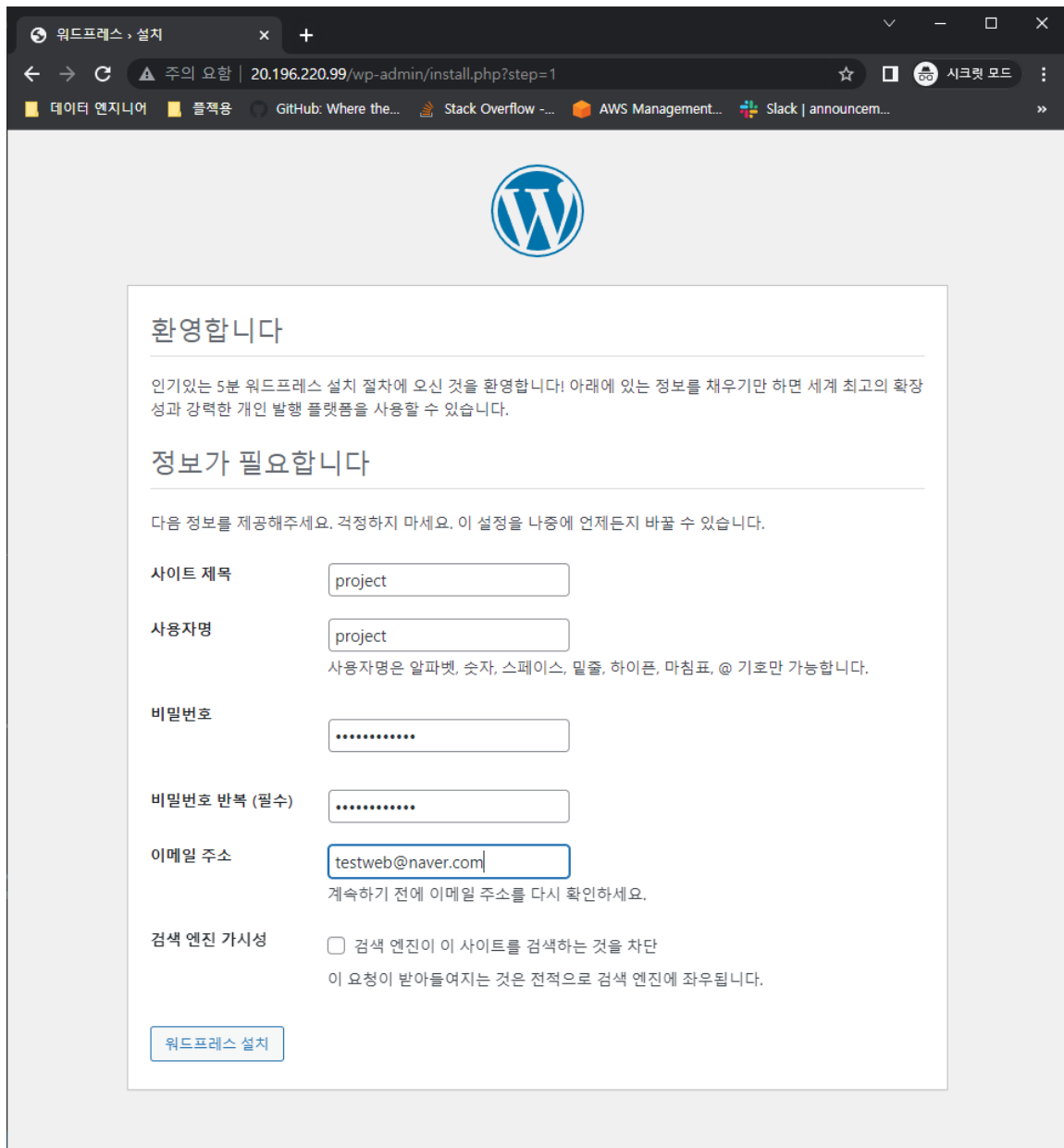
3. 프로젝트 최종 결과

a. wordpress 서비스 확인

```
Outputs:
Bastion_ip_address = "20.196.219.212"
application_ip_address = "20.196.220.99"
terraform >
```



Terraform으로 리소스를 생성하고 출력한 Application Gateway의 주소를 통해 접속하면 wordpress 화면이 뜨는 것을 볼 수 있다.



The image shows a web browser window with the URL `20.196.220.99/wp-admin/install.php?step=1`. The page features the WordPress logo at the top center. Below it, the heading "환영합니다" (Welcome) is followed by a paragraph: "인기있는 5분 워드프레스 설치 절차에 오신 것을 환영합니다! 아래에 있는 정보를 채우기만 하면 세계 최고의 확장성과 강력한 개인 발행 플랫폼을 사용할 수 있습니다." (Welcome to the popular 5-minute WordPress installation process! Fill out the information below to use the world's best scalable and powerful personal publishing platform.)

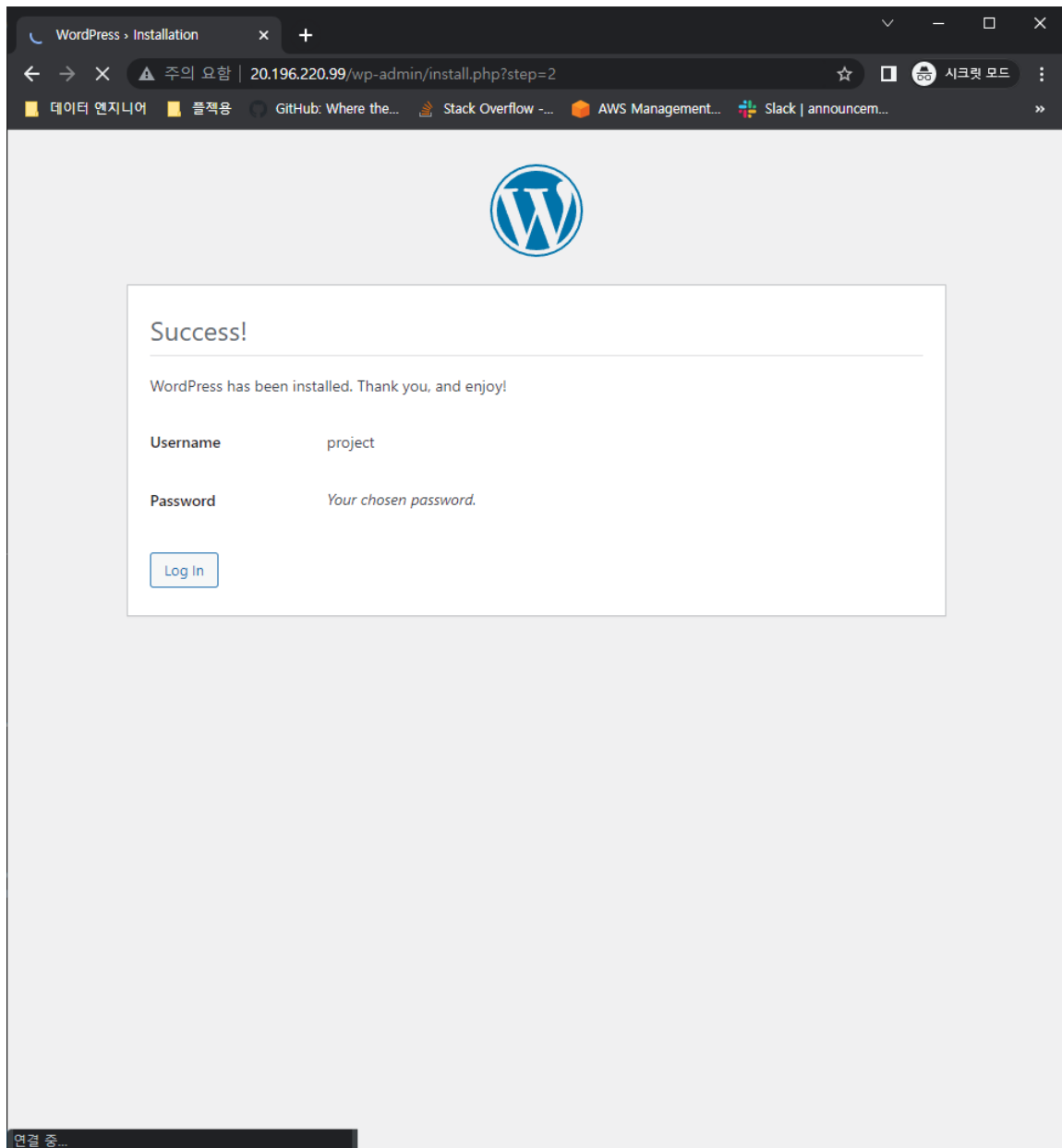
The next section is titled "정보가 필요합니다" (We need some information). It includes a sub-instruction: "다음 정보를 제공해주세요. 걱정하지 마세요. 이 설정을 나중에 언제든지 바꿀 수 있습니다." (Please provide the following information. Don't worry, you can change these settings at any time later.)

The form contains the following fields and options:

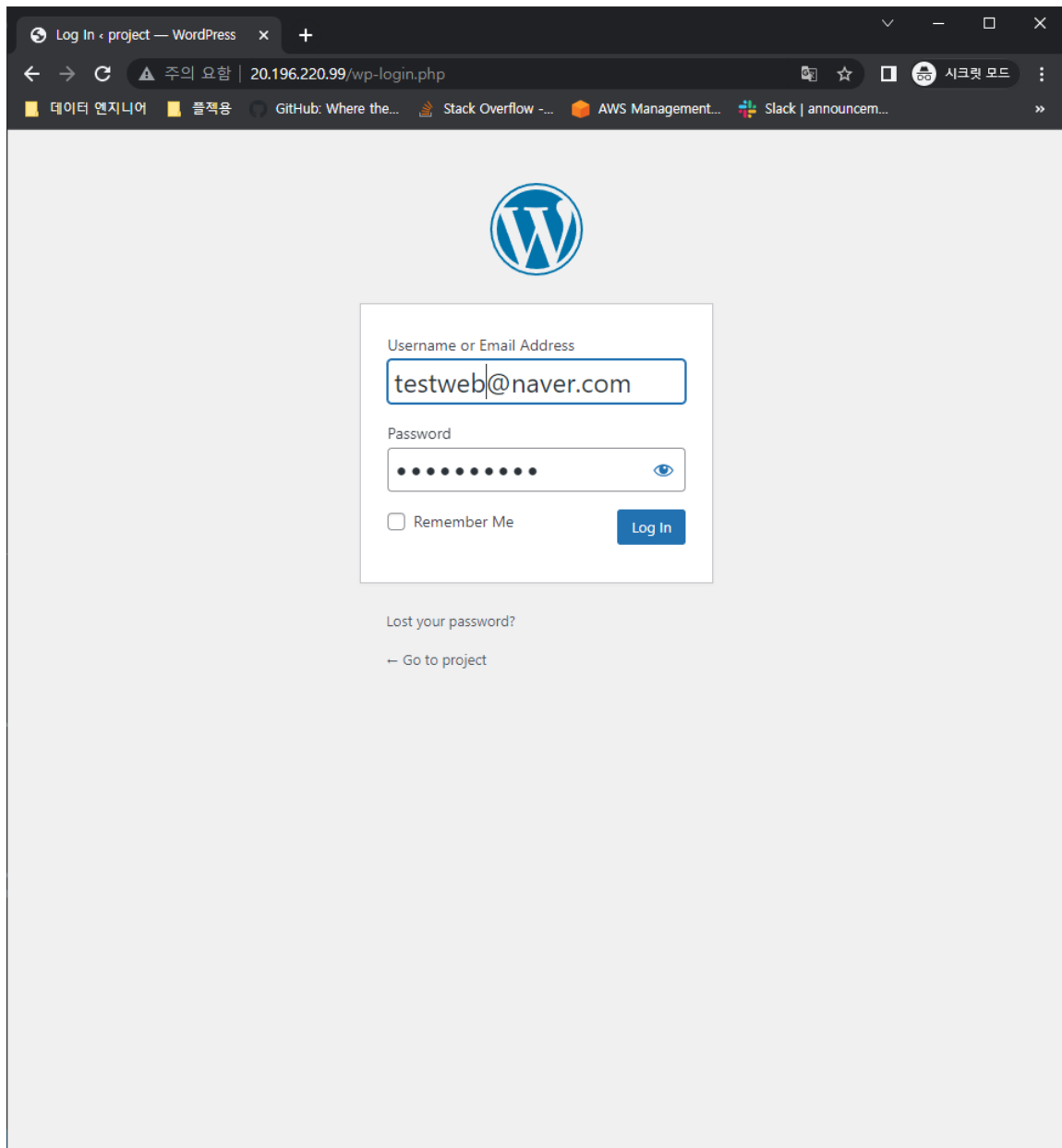
- 사이트 제목** (Site Title): A text box containing the word "project".
- 사용자명** (Username): A text box containing the word "project". Below it, a note states: "사용자명은 알파벳, 숫자, 스페이스, 밑줄, 하이픈, 마침표, @ 기호만 가능합니다." (Username can only contain letters, numbers, spaces, underscores, hyphens, and periods.)
- 비밀번호** (Password): A text box with masked characters (dots).
- 비밀번호 반복 (필수)** (Repeat Password): A text box with masked characters (dots).
- 이메일 주소** (Email Address): A text box containing "testweb@naver.com". Below it, a note states: "계속하기 전에 이메일 주소를 다시 확인하세요." (Double-check your email address before continuing.)
- 검색 엔진 가시성** (Search Engine Visibility): A checkbox labeled "검색 엔진이 이 사이트를 검색하는 것을 차단" (Prevent search engines from indexing this site). Below it, a note states: "이 요청이 받아들여지는 것은 전적으로 검색 엔진에 좌우됩니다." (Whether this request is accepted is entirely up to the search engine.)

At the bottom left of the form is a button labeled "워드프레스 설치" (Install WordPress).

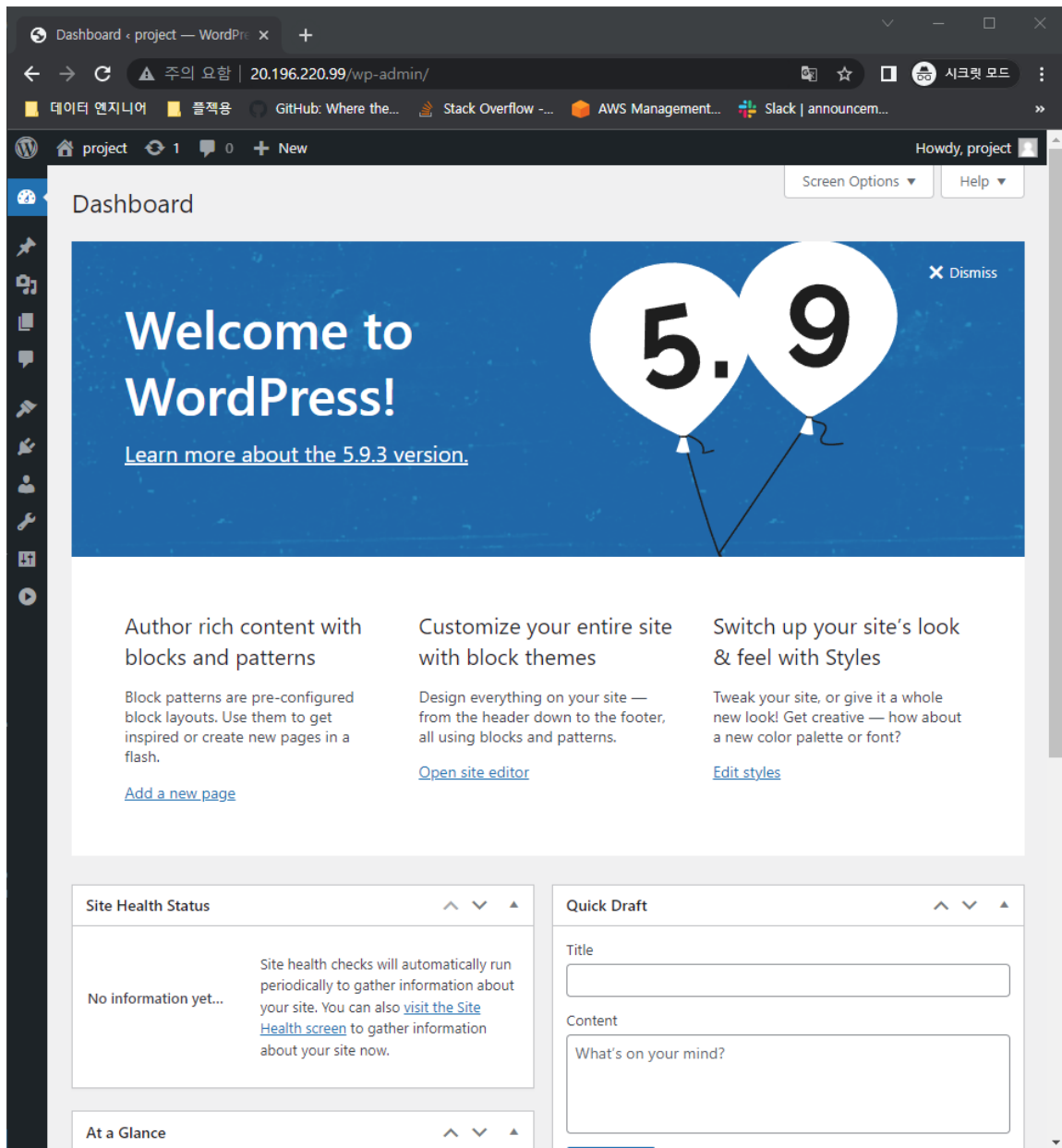
회원가입을 위한 정보를 입력하고 워드프레스 설치 버튼을 클릭한다.



회원가입과 워드프레스 설치가 완료됐다는 화면이 뜬다.



가입한 회원 정보를 입력하고 로그인 버튼을 클릭한다.



워드프레스 서비스 화면이 정상적으로 뜨는 것을 확인할 수 있다.

b. 느낀점

Azure 서비스는 처음 사용해봐서 모든 것이 새로웠다. AWS와 Azure의 서비스의 차이에 대해 이해하는데 많은 시간이 걸렸다 특히 Azure의 Subnet의 개념이 너무 헷갈렸었다.

Azure cli를 이용하기 위한 로그인 설정에 4시간이 걸릴 정도로 모든 것이 어려웠지만 결국 성공해냈을 때 기분이 너무 좋았다.

리소스 생성 시 Azure Portal을 이용하면 쉬웠던 것들이 Terraform 코드를 이용할 땐 서비스 간의 의존 관계도 고민해야 하는 등 여러 문제점들이 많았고, 이 문제점들을 해결하기 위해 공식 문서와 Github를 참고해서 코드를 작성해 가는 과정이 쉽지는 않았다.

그럼에도 포기하지 않고 끝까지 시도해서 결국 좋은 결과물을 만들어 낼 수 있어서 다행이고 뿌듯하다. 이 과정에서 팀원이 많은 도움을 주었고, 혼자 진행했다면 절대 해내지 못했을 것이라 생각한다.

팀 프로젝트를 오랜만에 경험해봤는데 짧은 시간이었지만 한 단계 성장할 수 있었던 좋은 경험이었다고 생각한다.