IaC를 이용하여 AWS 클라우드에 wordpress 배포하기

목차

9. 오토 스케일링 그룹에 로드밸런서 연결하기
10. Output Value로 자주 참조하는 변수 처리하기
11. 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 Terraform를 이용하여 Terraform 프로바이더인 AWS 클라우드, Azure 클라우드 각각에 고가용성 wordpress를 배포해보는 프로젝트를 진행한다.

b. 프로젝트 요약

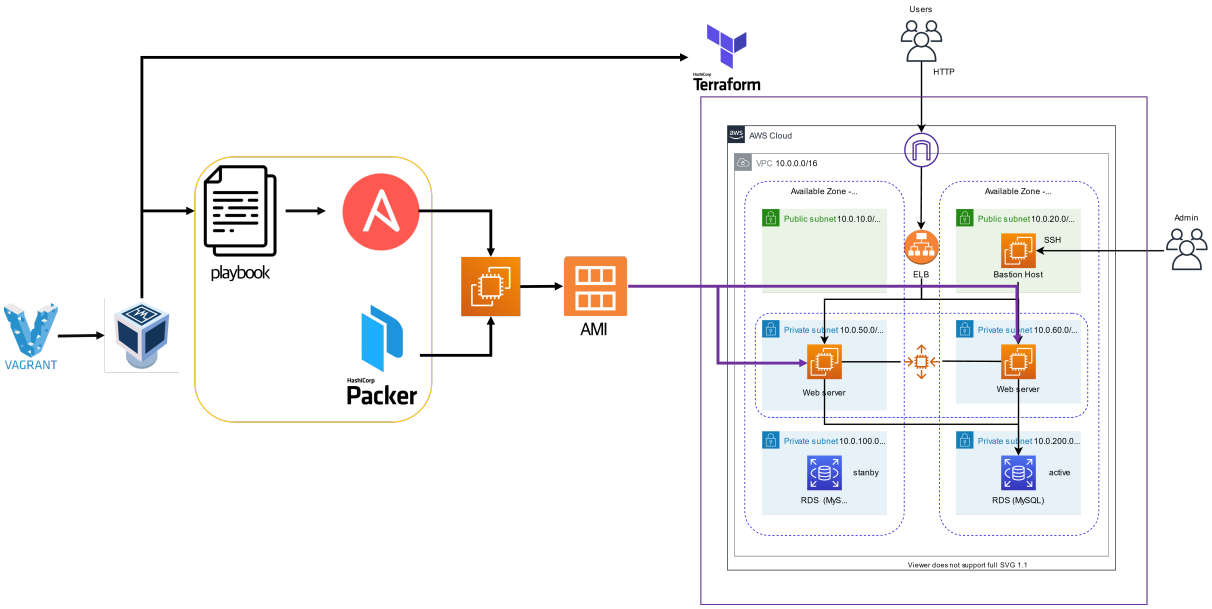
프로젝트 명	IaC를 이용하여 클라우드에 Wordpress 배포하기
프로젝트 기간	2022. 04. 26 ~ 2022. 05. 02
프로젝트 인원	3인
프로젝트 역할 분담	신소희 Terraform을 이용한 AWS 리소스 배포 markdown 문서 정리 발표 자료 제작 선우지훈 Terraform을 이용한 Azure 리소스 배포 markdown 문서 정리 홍성민 Ansible Playbook 작성 markdown 문서 정리

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							

c. AWS 시스템 아키텍처

시스템 동작 구조



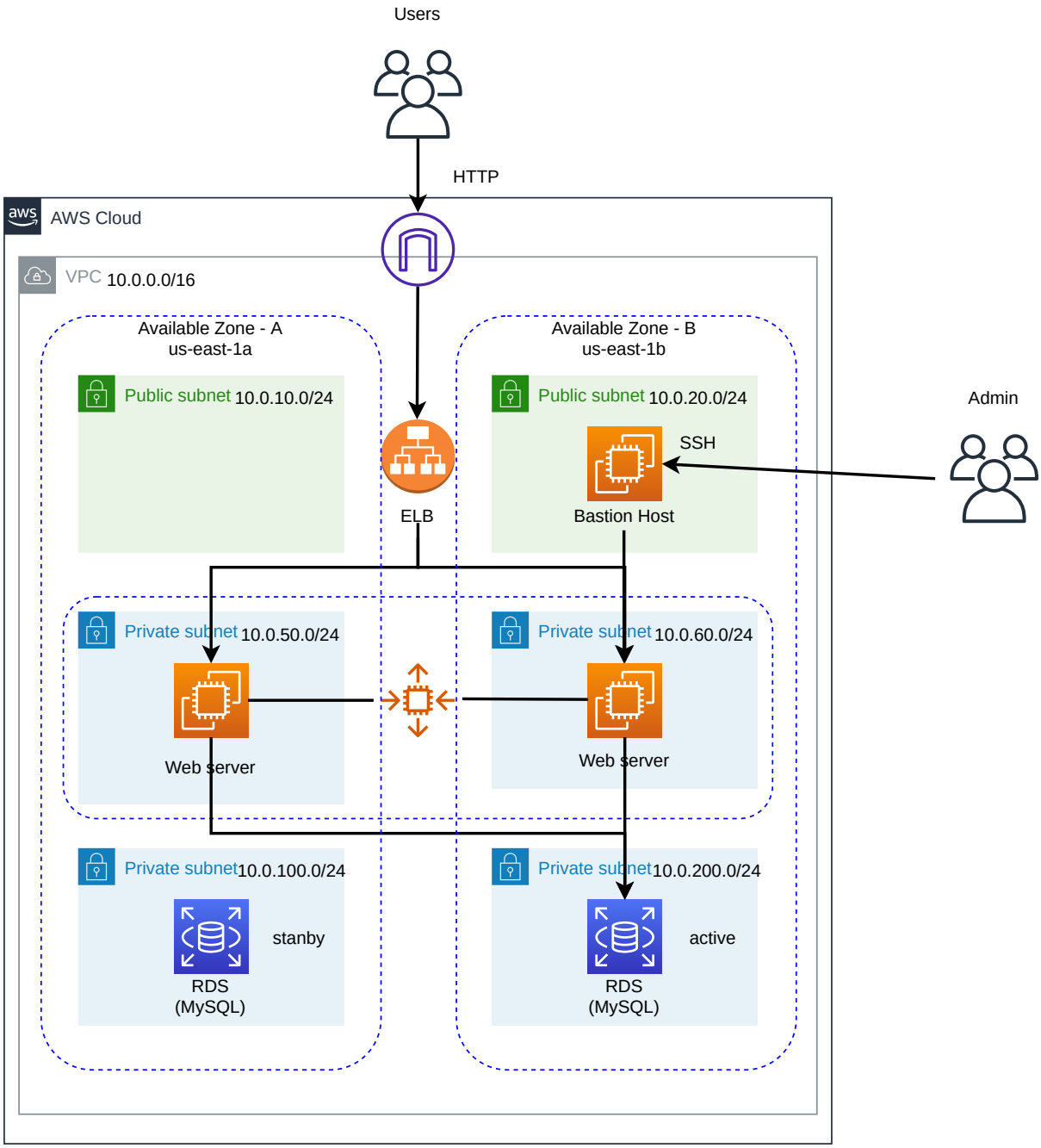
전체 시스템 동작 구조는 다음과 같다.
가상머신에서 Packer를 이용해 Ansible Playbook을 실행하여 AMI 이미지를 만든다.
Terraform을 이용해 클라우드에 리소스를 생성할 때 시작템플릿 구성에 해당 이미지를 사용한다.

사용한 서비스 목록

Vagrant	가상 시스템 환경을 구축하고 관리하기위한 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	인프라스트럭쳐 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

AWS 클라우드 시스템 아키텍처



전체 아키텍처 개요

가용영역 2개에 각각 퍼블릭 서브넷 1개 프라이빗 서브넷 2개를 둔다.

퍼블릭 서브넷에는 배스천 호스트를 두어 로드 밸런서를 통하지 않으면 wordpress 서버 내부에 접근할 수 없도록 한다.

프라이빗 서브넷 영역에는 wordpress를 배포한 ec2 인스턴스 웹 서버와 데이터가 저장되는 RDS 데이터베이스가 위치하는데 이 때 웹 서버에 장애가 발생해도 데이터에는 손상이 없도

록 하기위해 같은 프라이빗 영역에 배치하지 않았다.

DB 백업본을 다른 가용영역의 프라이빗 서브넷에 두어 DB 고가용성을 만족시킬 수 있다.

관리자는 SSH를 이용해 배스천 호스트를 통해 wordpress 서버, DB 서버에 직접 접속할 수 있으며

사용자는 ELB를 통해 외부에서 wordpress 서버에 접속할 수 있다.

ELB를 사용해 사용자 트래픽이 몰릴 경우 트래픽을 분배할 수 있으며 서버가 무너지는 경우를 대비해 오토 스케일링을 구성하였다.

사용한 AWS 서비스 목록

VPC	가상 네트워크 구축
EC2	가상 컴퓨터를 이용한 wordpress 웹 서비스, 서비스 관리를 위한 Bastion Host
RDS	wordpress DB 용 관리형 데이터베이스
ELB	wordpress EC2 인스턴스에 로드 밸런싱
Auto Scaling	wordpress EC2 인스턴스에 오토 스케일링

2. 프로젝트 진행 과정

a. Ansible을 이용한 Playbook 작성 및 실행

1. Ansible 사용을 위한 패키지 설치

```
sudo yum update
sudo yum -y install epel-release
sudo yum -y install python3
sudo yum -y install python3-pip
sudo yum install python-boto3
sudo pip3 install boto boto3
sudo yum install -y centos-release-ansible-29.noarch
sudo yum install -y ansible
```

2. Ansible 파일 설정

ansible.cfg 파일 설정

`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 파일을 지정한다.

- 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).

inventory 파일 설정

`inven_aws_ec2.yaml`

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

인벤토리 모듈을 불러온다.

동적 인벤토리 모듈을 사용하여 AWS계정에서 가동 중인 EC2에 플레이북 적용 가능하도록 함 (이후 Packer를 사용하여 Test 에만 사용)

AWS일 경우, 사전에 AWS CLI 2가 설치되어야 한다.

- 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 살행 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,fliename,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 AWS 클라우드에 wordpress 배포하기

1. Terraform 설치 및 초기 설정

Terraform 설치 및 확인

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo <https://rpm.releases.hashicorp.com/RHEL/hashicorp.
repo>
sudo yum -y install terraform

[vagrant@controller ~]$ terraform --version
Terraform v1.1.9
on linux_amd64
```

aws cli 설치

```
curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip>" -o "awscliv2.zip"
unzip awscliv2.zip
sudo yum -y install unzip
```

curl 명령을 사용해 awscliv2.zip 파일을 다운받고 unzip을 통해 압축을 푼뒤 install 한다.

aws configure 설정

```
ls
cd aws/
sudo ./install
aws configure
```

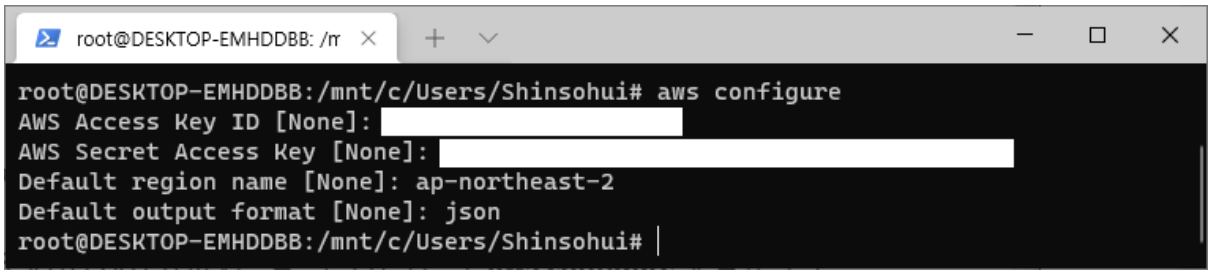
aws 디렉토리로 이동하여 `sudo ./insatll` 명령을 입력한뒤 aws configure 설정을 완료한다.

aws configure을 위해서는 aws IAM 유저를 생성해야 한다.

aws IAM 유저 생성을 완료하면 해당 화면과 함께 `.csv 다운로드` 버튼을 통해 자격 증명을 위한 .csv 파일을 다운 받을 수 있다.

다운받은 .csv 파일을 열게되면 IAM 사용자명, 콘솔 로그인 URL, 비밀번호를 확인할 수 있고 콘솔 로그인 URL에 접근하여 사용자 이름과 암호를 작성해서 로그인할 수 있다.

```
aws configure
```



명령을 입력하면 해당 프롬프트가 뜨게되며 `.csv` 파일 내용을 등록해주면 된다.

등록된 IAM 사용자 정보 확인

```
aws sts get-caller-identity
{
    "UserId": "",
    "Account": "",
```

```
    "Arn": ""
}
```

`aws sts get-caller-identity` 명령으로 등록된 사용자 정보를 확인할 수 있다.

---

2. Terraform Provider 설정하기

Terraform이 클라우드의 리소스를 생성하고 삭제할 때 필요한 Provider를 선언한다.

`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.

블록 헤더에 지정된 "aws"는 구성할 프로바이더의 로컬 이름이다.
이 프로바이더는 `required_providers` 블록에 포함되어 있어야 한다.

- `region` : 리소스를 생성할 AWS 리전을 의미한다. 서울 리전을 사용할 것이다.

`required_providers` 블록

- `source` : AWS 프로바이더 설치를 위한 경로를 지정한다.
- `version` : 프로바이더 버전을 지정한다.
  `~>` 의 의미는 특정 버전과 해당 버전의 가장 낮은 버전 대역의 최신 버전까지 지원한다는 것이다.

3. 네트워크 구축하기

`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
}
```

VPC를 구성한다. 편리한 VPC 구성을 위해 VPC 모듈을 사용한다.

- `source` : 모듈을 사용하기 위해 module이 저장되어있는 레포지토리 경로를 명시한다.

VPC의 이름, VPC의 cidr 블록, 서브넷을 지정할 가용 영역 및 퍼블릭 서브넷, 프라이빗 서브넷의 CIDR블록을 지정한다.

- `name` : vpc의 이름
- `cidr` : vpc의 cidr 블록
- `azs` : 가용 영역 지정
- `public_subnets` : 퍼블릭 서브넷으로 사용할 cidr 블록 지정
- `private_subents` : 프라이빗 서브넷으로 사용할 cidr 블록 지정

네트워크 통신을 위한 인터넷 게이트웨이, NAT 게이트웨이 사용을 지정한다.

- `create_igw = true` : 인터넷 게이트웨이를 사용한다는 의미이다.
- `enable_nat_gateway = true` : NAT 게이트웨이를 사용한다는 의미이다.
- `single_nat_gateway = true` : NAT 게이트웨이를 하나만 사용한다는 의미이다.

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

네트워크 IP address 설계는 다음과 같다.

VPC CIDR	10.0.0.0/16	
us-east-1a	public subnet	10.0.10.0/24
	private subnet	10.0.50.0/24
	private subnet (for RDS)	10.100.0./24
us-east-1b	public subnet	10.0.20.0/24
	private subnet	10.0.60.0/24
	private subnet (for RDS)	10.200.0.0/24

4. Bastion Host 생성 및 보안 그룹 설정

Bastion Host 인스턴스 생성

`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
    ]
  }
}
```

wordpress를 구성할 인스턴스는 프라이빗 서브넷에 위치하므로 퍼블릭 ip를 갖지 않는다.

안전한 관리를 위해서는 Bastion Host를 사용해야 하며 가상머신에서 wordpress 인스턴스로 jump host 해야한다.

`aws_key_pair` 리소스 블록

jump host SSH 접속을 위해 사용할 공개키를 설정하기 위한 리소스 블록이다.

- `key_name` : 사용할 key의 이름을 지정한다.
- `public_key` : 접속의 주체가되는 가상머신의 공개키를 지정한다.

`aws_instance` 리소스 블록
ec2 인스턴스를 생성하기 위한 리소스 블록이다.

- `ami` : 인스턴스를 생성하기 위한 이미지를 지정한다. `data` 블록의 filter를 사용해 이미지를 가져온다.
- `availability_zone` : 가용 영역을 지정한다.
- `instance_type` : 사용할 인스턴스의 타입을 지정한다.
- `vpc_security_group_ids` : 인스턴스에 적용할 보안 그룹을 지정한다.
- `subnet_id` : 인스턴스가 생성될 서브넷을 지정한다.
- `key_name` : Bastion Host에 접속하기 위한 공개키를 지정한다.

프라이빗 ip만을 가진 ec2 인스턴스로 jump host하기 위해서는 Bastion Host 인스턴스에 키를 갖다놔야 한다. 이를 위해 `connection` 블록 및 `provisioner` 를 사용하여 연결을 설정한 뒤 키 파일을 전송하고 키 파일의 권한을 변경한다.

`connection` 블록
`provisioner`를 사용하기 위해 SSH 연결을 설정하는 블록이다.

- `user` : 접속할 사용자명을 지정한다.
- `host` : 연결할 인스턴스의 주소를 지정한다.
  현재 `connection` 블록은 `aws_instance` 리소스 블록 내부에 선언되어 있기 때문에 `self.public_ip`는 Bastion Host 인스턴스의 퍼블릭 ip를 의미한다.
- `private_key` : 접속을 위한 개인키를 지정한다. 접속 주체가 되는 가상머신의 개인키를 사용한다.

`provisioner "file"` 블록
출발지 가상머신에서 목적지 인스턴스로 파일을 전송하기 위한 블록이다.
file 프로비저너를 사용하여 접속 주체 가상머신의 개인키를 Bastion Host 인스턴스로 전송한다.

- `source` : 전송할 파일을 명시한다.
- `destination` : 파일이 전송될 목적지이다.

`provisioner "remote-exec"` 블록
목적지 인스턴스에 명령을 실행하기 위한 블록이다.
inline 에 실행할 명령어를 나열한다.

Bastion Host를 위한 보안 그룹을 설정한다.

`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
  }
}
```

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

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

`ingress` 블록의 argument는 다음과 같다.

- `cidr_block` : 인바운드 규칙의 소스를 지정한다. 0.0.0.0/0 은 모든 ip를 허용함을 의미한다.
- `from_port` : 인바운드 규칙에서 사용할 프로토콜의 포트 번호를 지정한다. SSH 프로토콜의 포트 번호는 ‘22’이다.
- `to_port` : 아웃바운드 규칙에서 사용할 프로토콜의 포트 번호를 지정한다. SSH 프로토콜의 포트 번호는 ‘22’이다.

- `protocol` : 사용할 프로토콜을 명시한다. SSH는 TCP 기반의 프로토콜이므로 TCP를 입력한다.

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

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

wordpress 인스턴스는보안을 위해 Bastion Host에 의해서만 관리될 수 있도록 인바운드 규칙의 `cidr_blocks`을 Bastion Host의 ip로만 한정한다.

5. Packer를 사용하여 커스텀 인스턴스로 이미지 만들기

자동화된 이미지 빌더인 Packer를 사용하여 wordpress 인스턴스로 이미지를 만든다. Packer는 ansible provisioner를 지원하기 때문에 ansible playbook 사용에 있어 매우 편리하다.

Packer 설치

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo <https://rpm.releases.hashicorp.com/RHEL/hashicorp.
repo>
sudo yum -y install packer
```

Packer 소스

이미지를 만들기 위해 사용한 Packer 소스 파일의 구조는 다음과 같다.

```
.
├── 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
```

httpd 디렉토리의 `project-ami.pkr.hcl` 파일을 제외하고 모두 ansible playbook 실행을 위한 파일 및 디렉토리이다.

Packer로 이미지를 빌드하기 위해 사용되는 가장 핵심적인 파일이다.

`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 = [
    ]
  }
}
```

Packer도 Terraform과 마찬가지로 프로바이더를 설정해야 한다.
`source "amazon-ebs"` 블록에는 커스텀 이미지를 생성하기 위해 인스턴스를 만드는데 사용할 초기 인스턴스를 지정한다.

- `force-deregister : true` : 같은 이름의 이미지는 덮어씌우겠다는 의미이다.

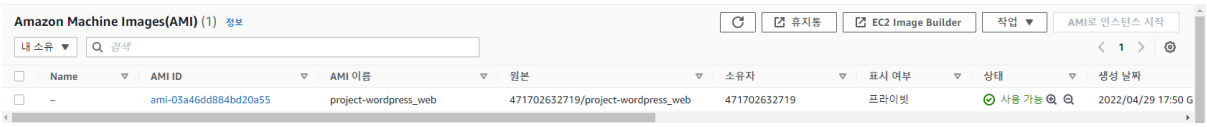
`build` 블록에서는 `source "amazon-ebs"` 블록에서 지정한 초기 인스턴스에 커스터마이징을 진행한다.
Packer에서는 ansible provisioner를 제공하므로 편리하게 Playbook을 실행할 수 있다.

- `extra_arguments`를 통해 playbook을 실행 권한을 지정할 수 있다.

Packer 이미지 빌드

```
packer init . # packer 플러그인 설치
packer fmt . # packer 소스코드 포맷 자동 정렬
packer validate . # packer 소스코드 유효성 확인
packer build . # packer로 이미지 빌드
```

Packer 명령어를 사용해 이미지를 빌드한다.
. 은 현재 디렉토리를 의미한다.



빌드에 성공한뒤 AWS console로 접속하여 AMI 탭으로 이동하면 생성한 Packer 이미지를 확인할 수 있다.

6. wordpress 이미지로 시작 템플릿 구성하기

5.에서 Packer를 이용해 wordpress 커스텀 이미지를 만들었다.
Auto Scaling Group에서 wordpress 인스턴스를 생성하기 위해 해당 이미지를 사용하여 시작 템플릿을 구성한다.

```
# 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_se
curity_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/wordpre
ss/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"
  }
}

# wordpress가 구동될 인스턴스를 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 # 3분으로 지정
  force_delete              = true
  vpc_zone_identifier       = [module.app_vpc.private_subnets[0], module.app_vpc.priva
te_subnets[1]]
}
```

Auto Scaling Group에 사용할 Launch Template을 작성한다.

`aws_launch_template` 리소스 블록

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

`depends_on` 을 통해 의존성을 명시한다.

시작 템플릿을 구성하기 위해
VPC의 퍼블릭 서브넷, 데이터베이스 서브넷 그룹, 데이터베이스 인스턴스 리소스가 모두 생성된 후 시작 템플릿 리소스를 생성하도록 명시적 의존성을 지정한다.

```
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_secu
rity_group.bastion-to-private.id]
```

- `name` : 시작 템플릿 이름 지정
- `description` : 설명
- `instance_type` : 인스턴스의 타입 지정

- `image_id` : 사용할 이미지의 id 값을 지정한다.
- `instance_initiated_shutdown_behavior` : 인스턴스의 종료 동작을 지정한다. default 값은 stop이다.
- `key_name` : 공개키 지정
- `vpc_security_group_ids` : 인스턴스에 적용할 보안 그룹 지정

7. RDS 데이터베이스 생성 및 보안 그룹 설정

wordpress 인스턴스에서 사용할 RDS 데이터베이스를 생성한다.

```
# 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.
}
```

`aws_db_subnet_group` 블록
RDS 인스턴스의 서브넷 그룹으로 사용할 서브넷을 지정한다.

- `name` : 서브넷 그룹의 이름을 지정한다.
- `subnet_ids` : DB 서브넷으로 사용할 cidr 블록을 지정한다.

`aws_db_instance` 블록
데이터베이스에서 사용할 엔진 정보, 인스턴스 정보, 초기 데이터베이스명, 사용자명, 사용자 패스워드 등을 지정한다.

- `availability_zone` : DB를 사용할 리전 정보를 입력한다.
- `db_subnet_group_name` : DB 서브넷 그룹을 지정한다.
- `engine` : DB 엔진을 지정한다.
- `engine_version` : DB 엔진의 버전을 지정한다.
- `instance_class` : DB에 사용할 인스턴스 유형을 지정한다.
- `identifier` : DB 인스턴스를 식별하기 위한 id 값을 지정한다.
- `name` : DB 초기 데이터베이스명을 지정한다.
- `username` : 사용자명을 지정한다.
- `password` : 로그인 시 사용할 패스워드를 지정한다.
- `port` : DB 서비스 포트 번호를 지정한다.
- `vpc_security_group_ids` : 보안 그룹을 지정한다.

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

```
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
  }]
}
```

8. 시작 템플릿으로 오토 스케일링 그룹 구성하기

6.에서 구성한 시작 템플릿을 사용하여 오토 스케일링 그룹을 구성한다.

```
# 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.priva
te_subnets[1]]

}
```

- `launch_template` : 오토 스케일링 시작 시 사용할 시작 템플릿을 지정한다. 6.에서 만들어 놓은 시작 템플릿의 id 값을 지정하면 된다.

원하는 인스턴스의 개수 및 최소, 최대 인스턴스 개수를 지정한다. 지정한 수대로 오토 스케일링 그룹이 인스턴스를 생성하고 health check를 통해 최대, 최소 범위에서 인스턴스 수를 조정할 것이다.

- `desired_capacity` : 초기 시작 시 원하는 인스턴스의 개수를 입력한다.

- `min_size` : 인스턴스의 최소 개수를 지정한다.
- `max_size` : 인스턴스의 최대 개수를 지정한다.

health check 수행 방식을 선택하고 인스턴스가 서비스를 시작한 후 상태를 확인하기 전의 시간을 지정한다.

- `health_check_type` : "EC2" 또는 "ELB"로 지정할 수 있으며 상태 확인이 수행되는 방식을 지정한다.
  - EC2 health check 방식 : EC2 상태 확인 결과를 사용하여 인스턴스 상태를 확인하는 방식이다. 인스턴스가 running 이외의 상태이면 해당 인스턴스를 비정상으로 간주하여 교체한다.
  - ELB health check 방식 : 인스턴스의 지정된 TCP 포트가 연결을 수락하는지 또는 지정된 웹 페이지가 2xx 코드를 반환하는지 확인한다. 따라서 ELB 상태 확인은 인스턴스만 작동하는지 확인하는 대신 실제 앱이 작동하는지 확인하기 더 쉽다.
- `health_check_grace_period` : 인스턴스가 서비스를 시작한 후 상태를 확인하기 전의 시간(초)를 지정한다.
- `force_delete` argument는 오토 스케일링 풀의 모든 인스턴스가 종료될 때까지 기다리지 않고 Auto Scaling 그룹을 삭제하도록 지정할 수 있다.
  즉 Auto Scaling이 리소스를 확장하는 과정에 있더라도 강제로 삭제하도록 할 수 있다.
- `vpc_zone_identifier` : 인스턴스가 시작될 서브넷을 지정한다.

9. 오토 스케일링 그룹에 로드밸런서 연결하기

오토 스케일링 그룹에 로드 밸런서를 연결하여 오토 스케일링 그룹으로 관리되는 인스턴스에 부하 분산을 적용한다.

```
# 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_vp
c.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
}

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
}

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
  }
}
```

`aws_alb` 리소스 블록
사용할 로드밸런서의 옵션을 지정한다.

- `internal` : default 값은 true이며 false 값은 internet-facing을 사용한다는 의미이다.
  - internal : internal 로드 밸런서는 프라이빗 서브넷에 할당되며 퍼블릭 IP가 없다. VPC에 있지 않은 클라이언트는 이를 가리키는 Route53 레코드를 생성하더라도 액세스할 수 없다. 클라이언트가 VPC에 없는 로드 밸런서에 연결할 수 있도록 하려면 인터넷 연결 로드 밸런서를 설정해야 한다.
  - internet-facing : 인터넷에 연결되는 로드밸런서를 뜻한다. 인터넷으로부터 요청을 받아서 각기 다른 EC2로 분배해주는 역할을 한다.
- `load_balancer_type` : 생성할 로드 밸런서의 유형이다. default는 application이다.
- `security_groups` : 보안 그룹을 지정한다.
- `subnets` : 로드 밸런서를 연결할 인스턴스가 있는 서브넷을 지정한다.
- `enable_cross_zone_load_balancing` : default 값은 false이며 true로 지정시 로드 밸런서의 교차 영역 로드 밸런싱이 활성화된다.

`aws_alb_targat_group` 리소스 블록
alb에 연결할 Auto Scaling 타겟 그룹을 지정한다.

- `port` : alb를 거쳐 인스턴스가 수신받는 서비스 포트를 지정한다.
- `protocol` : 트래픽을 대상으로 라우팅하는 데 사용할 프로토콜을 지정한다.

`aws_autoscaling_attachment` 리소스 블록
오토 스케일링 그룹과 타겟 그룹을 연결한다.

- `autoscaling_group_name` : 연결할 오토 스케일링 그룹을 지정한다.
- `alb_target_group_arn` : 오토 스케일링 그룹의 arn을 지정한다.

`aws_alb_listener` 리소스 블록
로드밸런서 리스너 리소스를 제공한다.

- `load_balancer_arn` : 로드밸런서의 arn을 지정한다.
- `default_action` : 기본 작업에 대한 구성 블록이다.
  - `type` : 라우팅 작업의 유형을 지정한다.
  - `target_group_arn` : 트래픽을 라우팅할 대상 그룹의 arn을 지정한다.

로드 밸런서를 위한 보안 그룹을 설정한다.

```
## 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
  }]
}
```

- `publicSG01` : 로드 밸런서에 모든 ip로부터의 HTTP 접속을 허용하는 보안 그룹이다.
- `privateEC2SG01` : 로드 밸런서로부터 wordpress 인스턴스로 전달된 HTTP 접속을 허용하는 보안 그룹이다.

---

10. Output Value로 자주 참조하는 변수 처리하기

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

`output.tf`

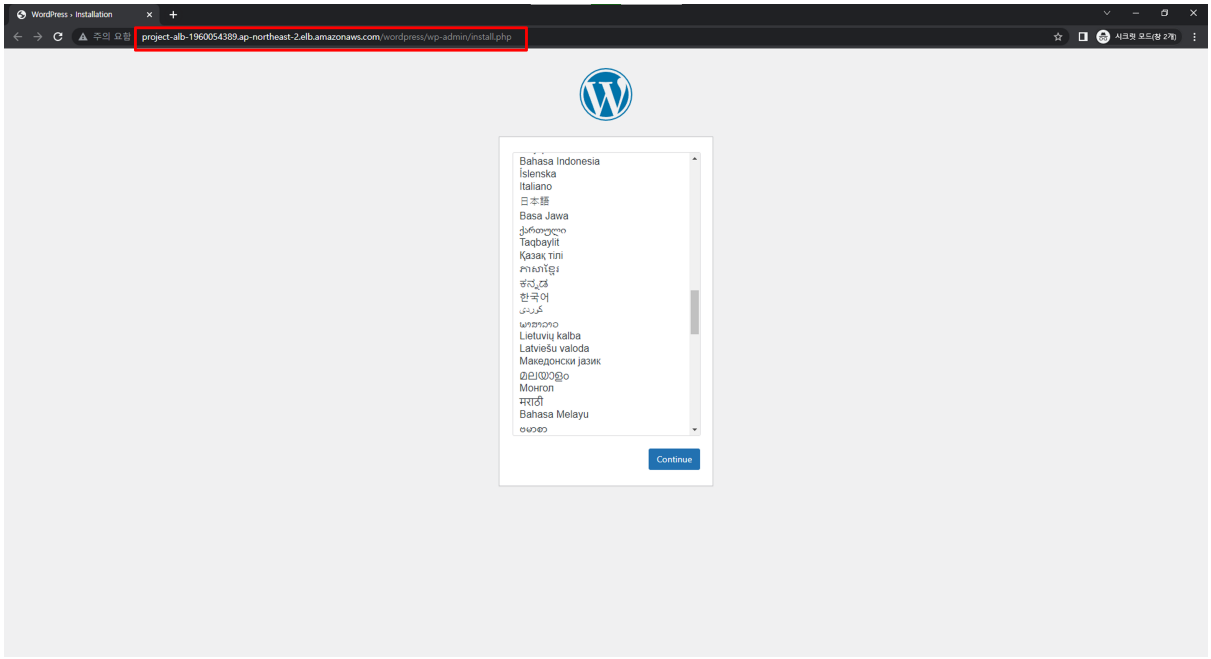
```
# Packer로 만든 이미지의 id 출력
output "pakcer-image" {
  value = data.aws_ami.wordpressLinux.id
}

# 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
}
```

11. terraform apply 및 생성된 리소스 작동 확인하기

Terraform으로 생성한 리소스를 확인할 수 있다.
Bastion Host를 위한 퍼블릭 IP가 있는 인스턴스 1대와 오토 스케일링 그룹으로부터 생성된 wordpress 인스턴스를 확인할 수 있다.

AMI 서비스에서는 Packer를 사용하여 만든 커스텀 이미지를 확인할 수 있다.

로드 밸런싱 및 오토 스케일링 그룹이 잘 설정된 것을 확인할 수 있다.



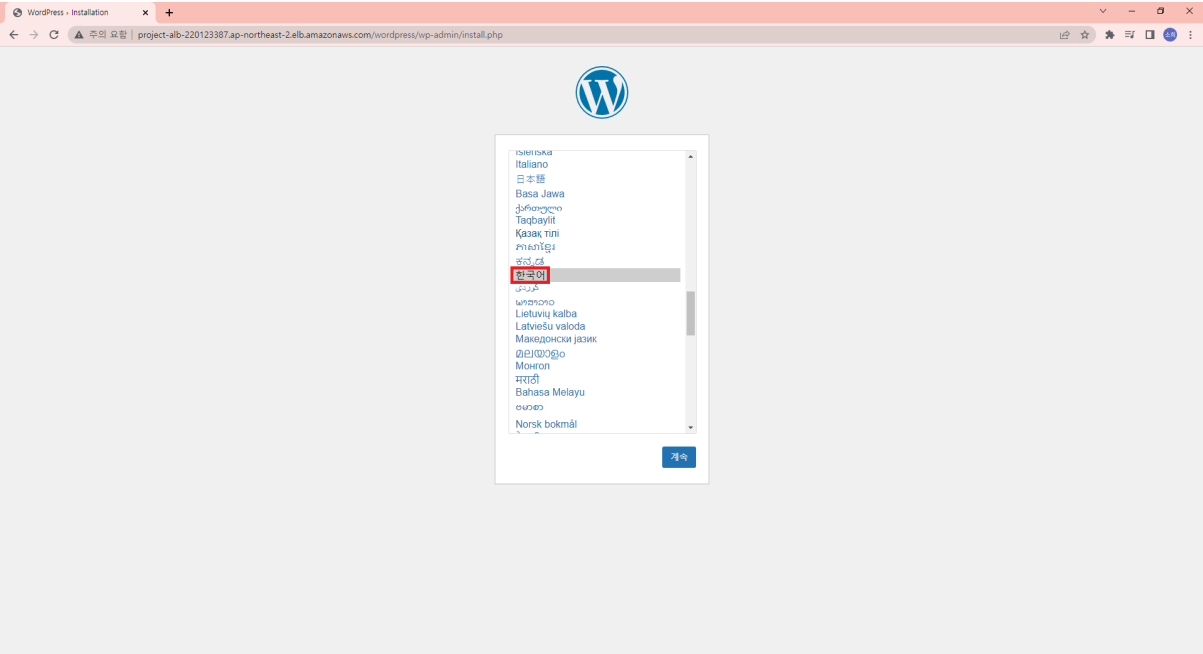
로드 밸런서의 DNS 통해 접속하면 워드프레스가 정상적으로 서비스되는 것을 확인할 수 있다.

---

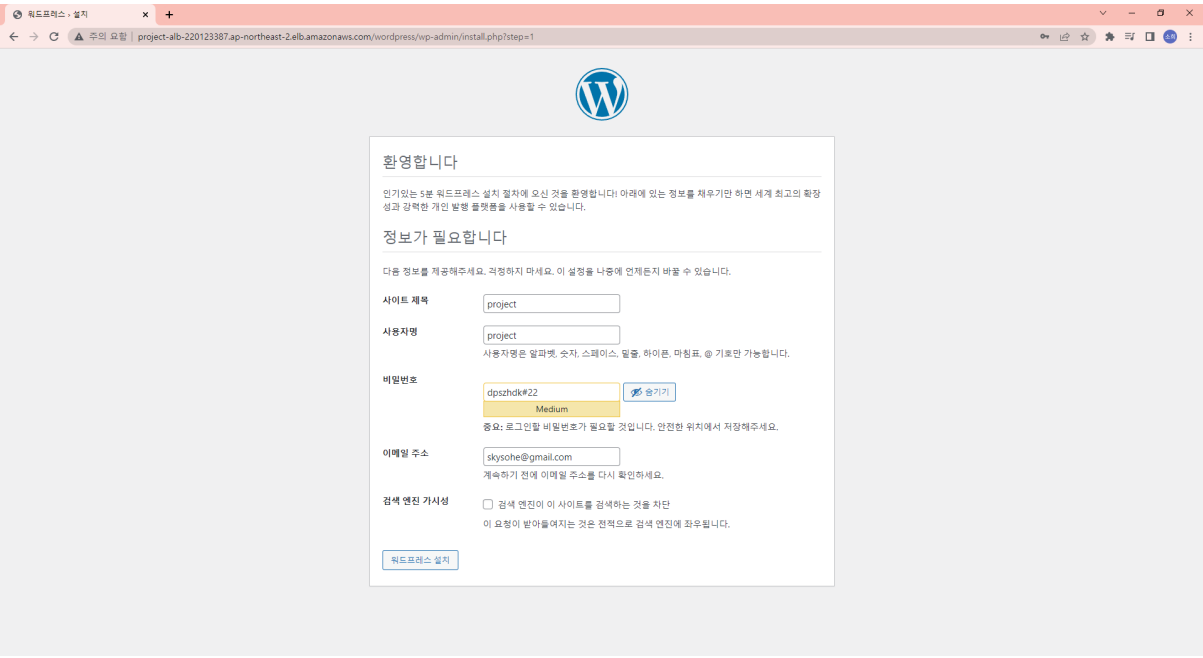
3. 프로젝트 최종 결과

a. wordpress 서비스 확인

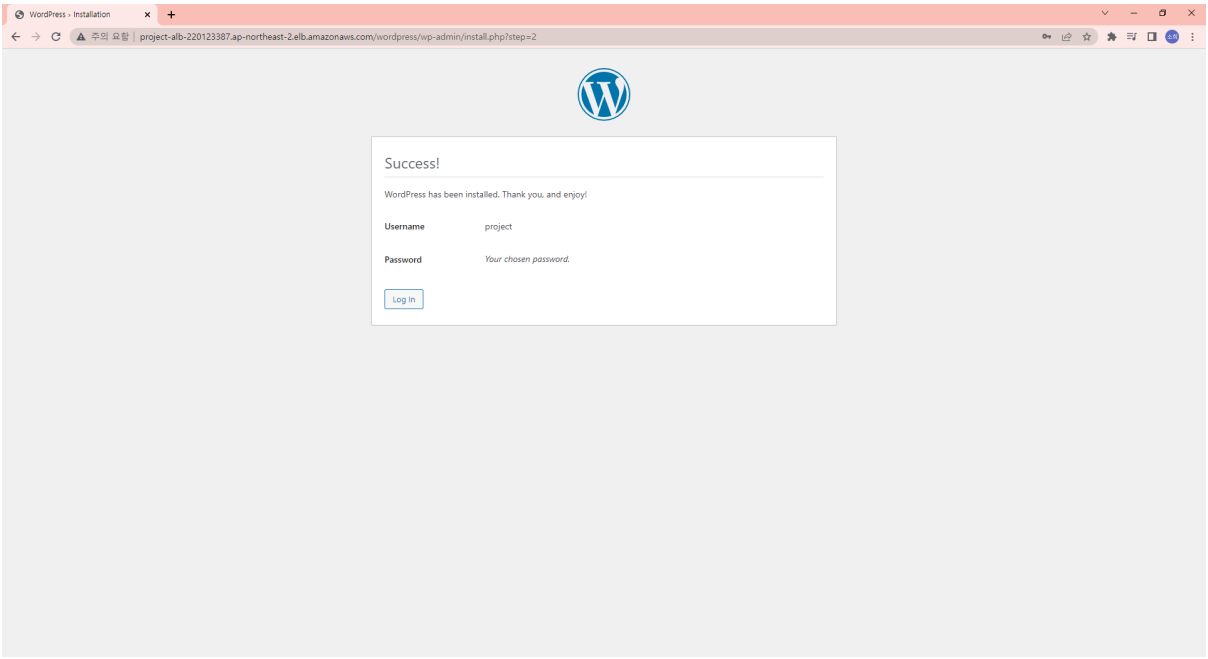
```
Apply complete! Resources: 35 added, 0 changed, 0 destroyed.

Outputs:

alb_domain = "project-alb-220123387.ap-northeast-2.elb.amazonaws.com"
bastion-instance-private = "10.0.10.222"
bastion-instance-public = "13.125.49.19"
pakcer-image = "ami-03a46dd884bd20a55"
wordpress_db_endpoint = "project-db.cicxdwwu5hr0.ap-northeast-2.rds.amazonaws.com:3306"
```



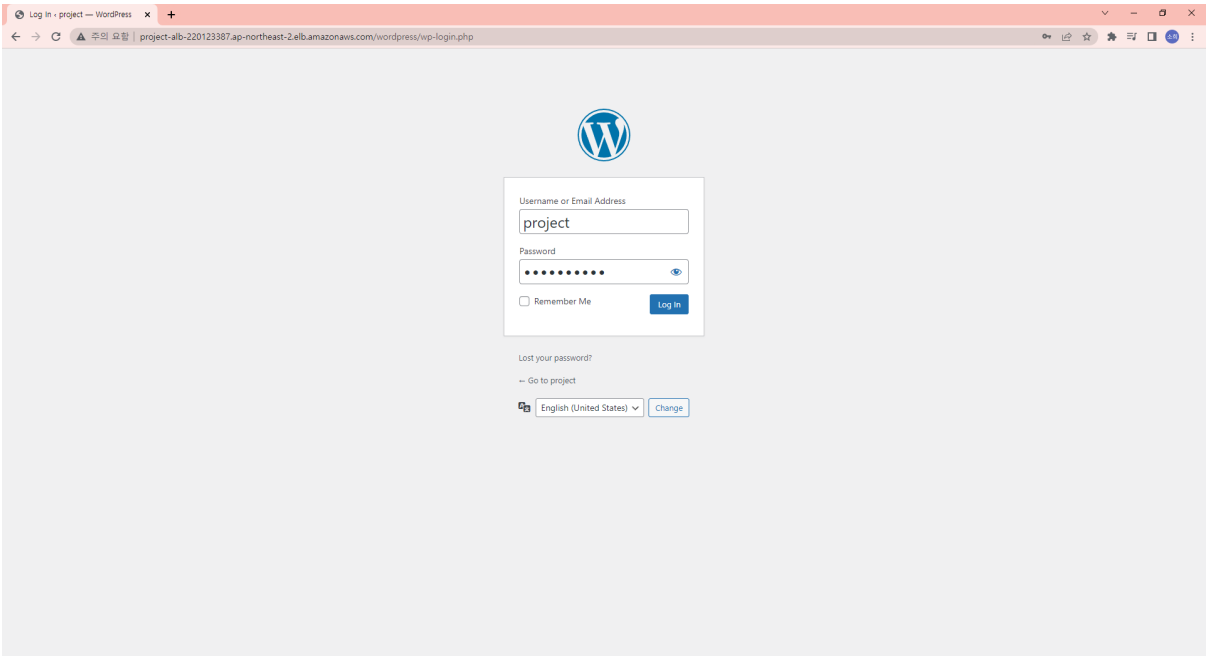
Terraform으로 리소스를 생성하고 출력한 로드 밸런서 주소를 통해 접속하면 wordpress 화면이 뜨는 것을 볼 수 있다.



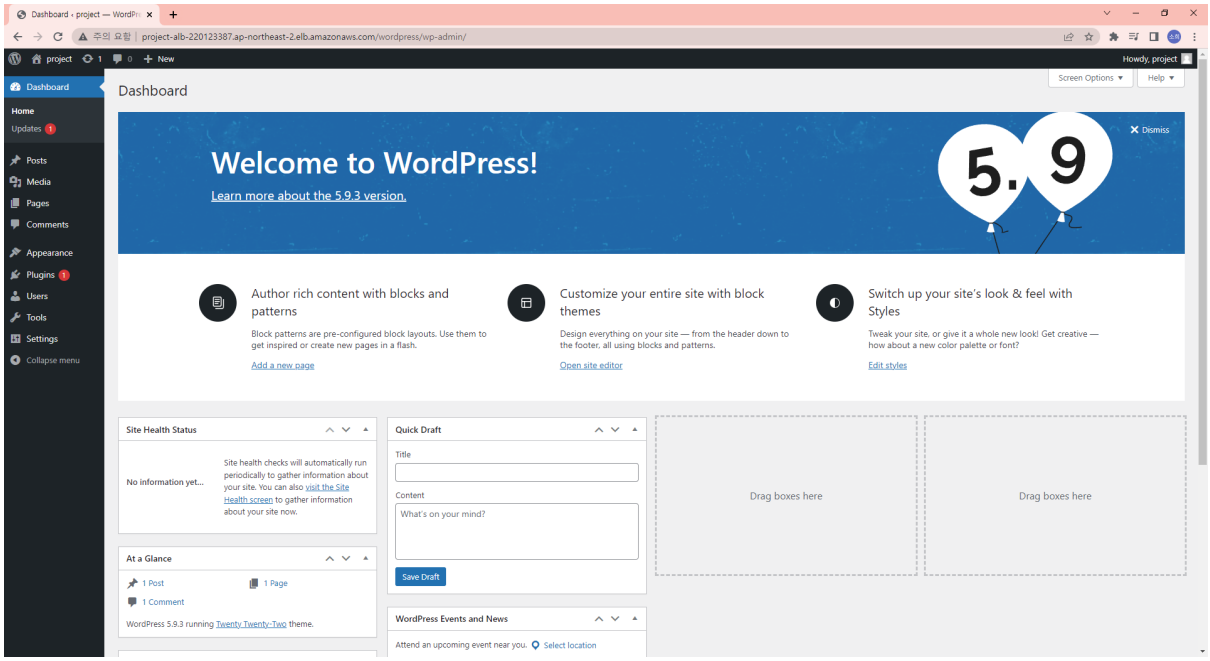
회원가입을 위한 정보를 입력하고 워드프레스 설치 버튼을 클릭한다.



회원가입과 워드프레스 설치가 완료됐다는 화면이 뜬다.



가입한 회원 정보를 입력하고 로그인 버튼을 클릭한다.



워드프레스 서비스 화면이 정상적으로 뜨는 것을 확인할 수 있다.

b. 느낀점

AWS 서비스는 수업 시간에도 길게 다루었고 AWS 특강도 들어서 Azure 보다는 훨씬 수월했다.

VPC 같은 경우도 처음에는 모든 기능을 직접 만들어봤는데 확실히 모듈을 사용하는 것이 정말 편하다는 것과 그럼에도 기본적인 동작 구조를 확실히 알고 모듈을 사용해야 한다는 것을 느꼈다.

AWS console 을 통해 서비스를 사용할 때는 AWS에서 사용자를 위해 UI를 제공하기 때문에 클릭 몇번을 통해 금방 필요한 서비스를 이용할 수 있었다.

Terraform 코드를 이용해 리소스를 생성할 때는 서비스 간의 의존 관계에 대한 고민도 해야하고 Terraform 공식 문서를 통해 해당 리소스 생성에 반드시 필요한 정보가 무엇인지를 찾아가는 과정이 쉽지는 않았던 것 같다.

그럼에도 모든 수단, 방법을 가리지 않고 결국 결과물을 낼 수 있어서 다행이고 뿌듯하다.

팀 프로젝트도 오랜만에 경험했는데 물론 짧은 시간이었지만 팀 프로젝트 진행에 있어 중요한 점이 무엇인지도 다시 한번 절감하는 좋은 경험이었다.