IaC를 이용한 퍼블릭 클라우드 WordPress 자동화 배포

Playdata mini project 3

CONTENTS

01 역할 분담

02 AWS 아키텍처 구성 및 설명

03 AWS 실시간 데모

04 As-is, To-be

05 Azure 아키텍처 구성 및 설명

06 Azure 실시간 데모

07 Ansible-Playbook

08 As-is, To-be

CHAPTER 01

역할분담

01 역할 분담

박능력
- Terraform 코드 작성
- 보고서 작성

도효주
- Ansible Playbook 코드 작성
- 보고서 작성

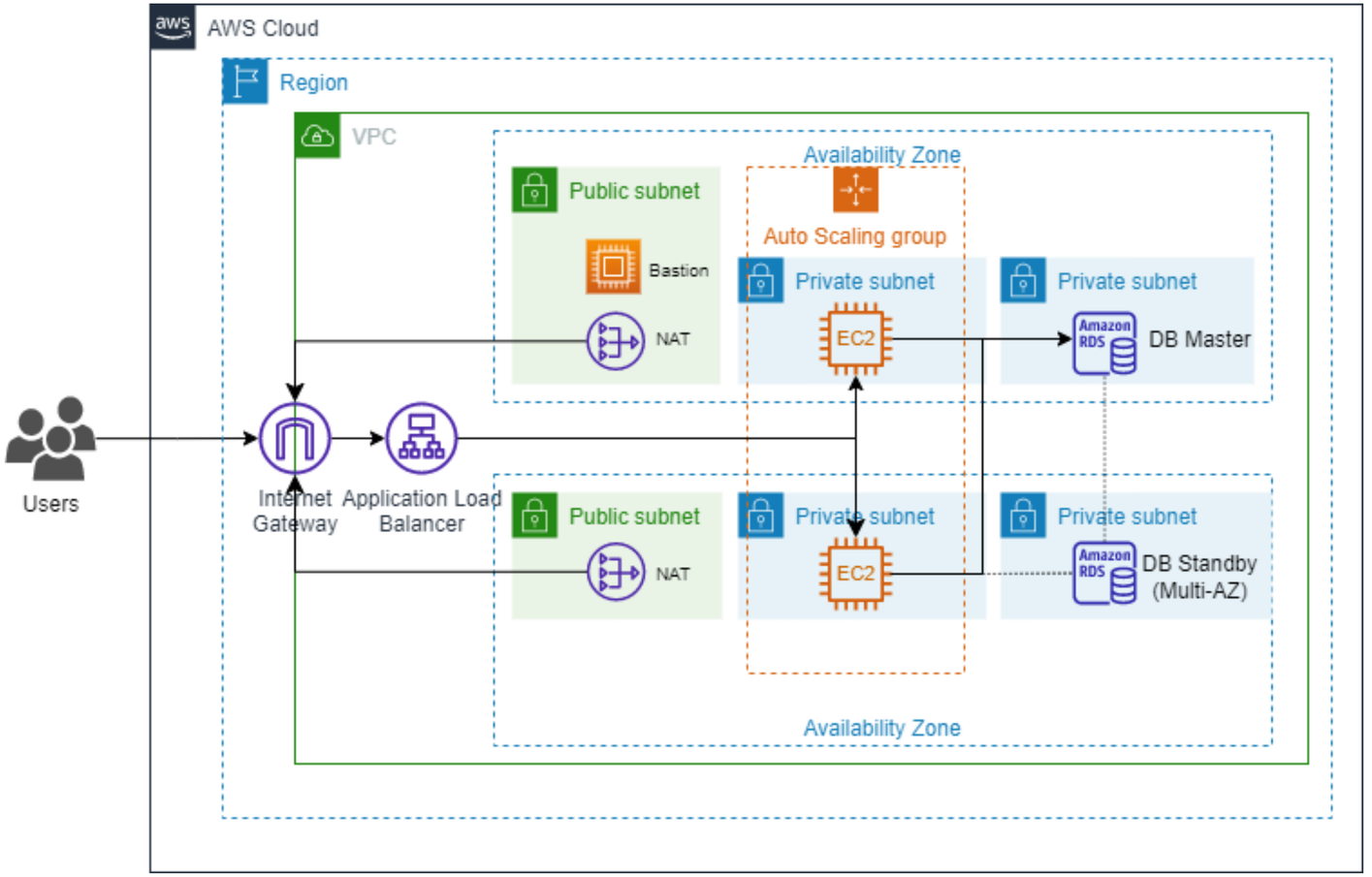
성나영
- Terraform 코드 작성
- PPT 제작

장현오
- 코드 테스트
- PPT 제작

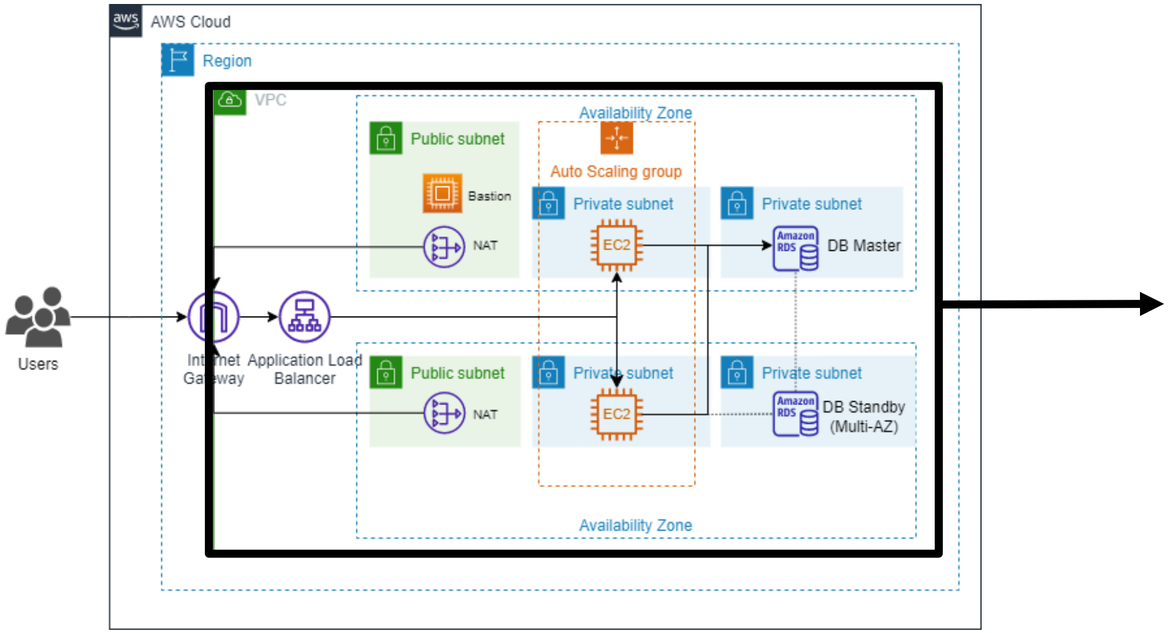
CHAPTER 02

AWS

02 AWS 아키텍처 구성 및 설명



02 AWS 아키텍처 구성 및 설명

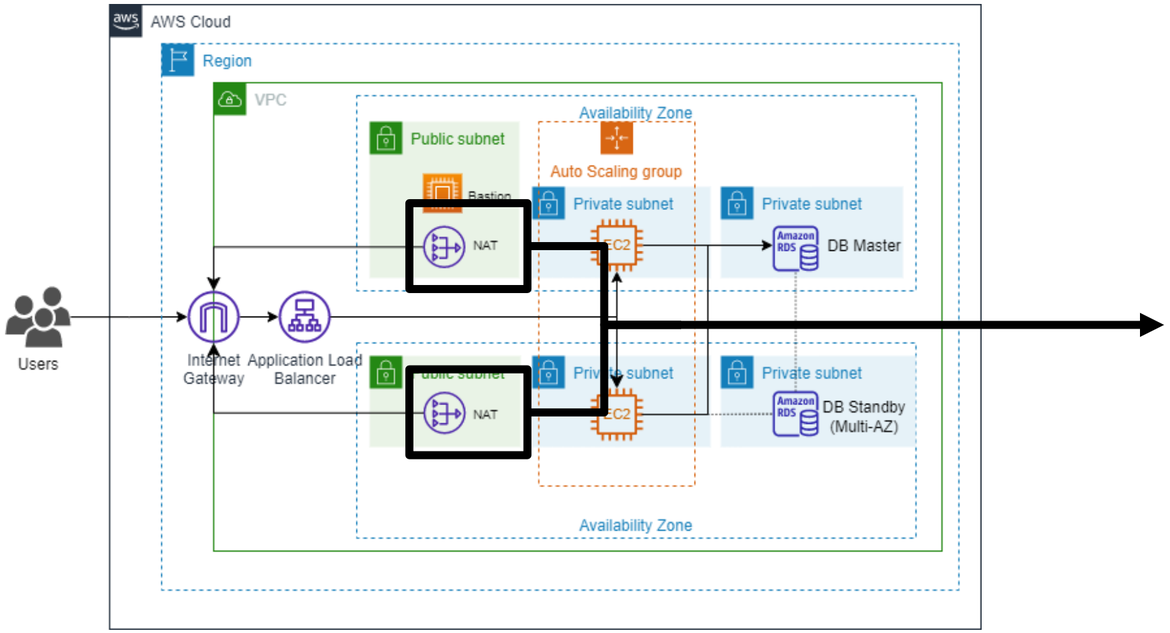


VPC

VPC를 사용하여 격리된 네트워크 구성

격리된 네트워크(VPC)를 통해 인터넷에 노출될 public subnet과 그렇지 않은 private subnet을 나누어 자유롭게 구성

02 AWS 아키텍처 구성 및 설명



NAT Gateway

Private Web Server가 VPC 외부의 서비스에 연결할 수 있지만 외부 서비스에서는 이러한 인스턴스와의 연결을 시작할 수 없도록 NAT 게이트웨이를 사용

가용 영역(AZ) 당 1개씩 배치하여, 속도를 향상

02 AWS 아키텍처 구성 및 설명

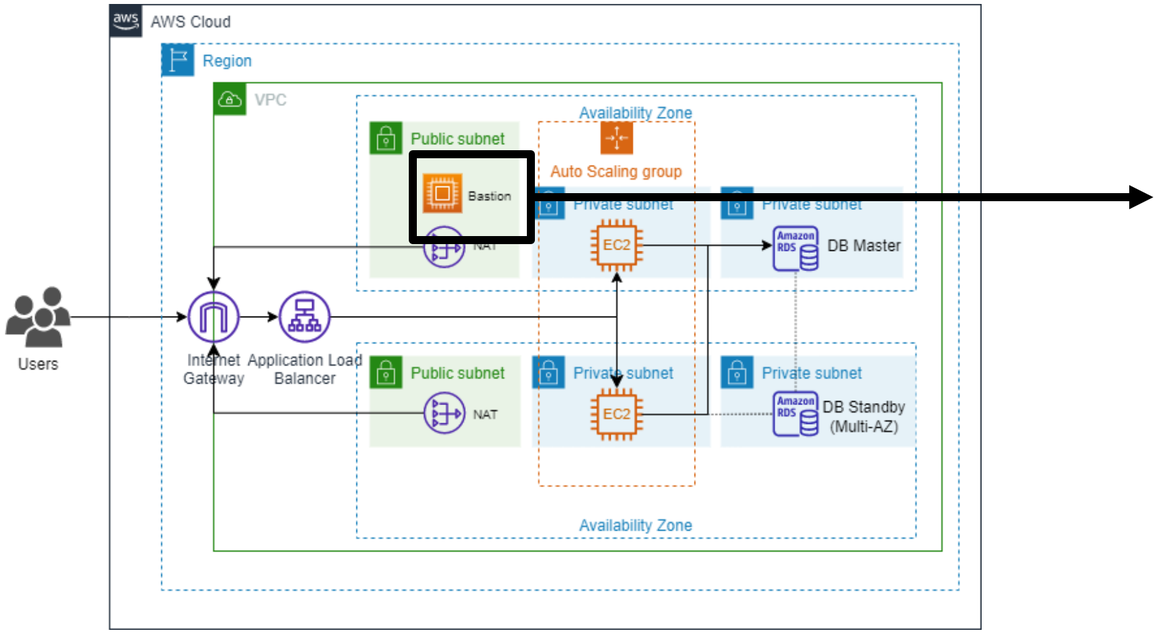
📌확인 Point

AWS의 app_vpc 모듈 사용

모듈을 사용하여 vpc 구성 후,
`one_nat_gateway_per_az` argument를 통해
AZ 각각에 NAT Gateway 구성

```
module "app_vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = "wordpress_vpc"
  cidr = "10.0.0.0/16"

  enable_nat_gateway     = true
  single_nat_gateway     = false
  one_nat_gateway_per_az = true #AZ 1개당 NAT Gateway 생성

  create_igw = true

  azs = [
    "ap-northeast-2a",
    "ap-northeast-2c"
  ]

  public_subnets = [
    "10.0.0.0/24",
    "10.0.1.0/24"
  ]

  private_subnets = [
    "10.0.2.0/24",
    "10.0.3.0/24",
    "10.0.4.0/24",
    "10.0.5.0/24"
  ]
}
```

02 AWS 아키텍처 구성 및 설명



Bastion Host

보안을 위해 고안된 Host로,
네트워크와 내부 네트워크 사이에서 일종의 게이트웨이 역할을 수행하는 호스트로 사용

Web 서버를 외부에서 접속할 때, Bastion Host의 경유를 통해서만 Private IP 서버에 접근하도록 설계하여 보안성 강화

02 AWS 아키텍처 구성 및 설명

📌확인 Point

Provisioner를 통해 Bastion Host로 Key 파일 복사

‘ec2-user’로 connection하기 때문에
파일권한 상의 문제로 /tmp/key_file로 복사 후
~/.ssh/id_rsa로 복사 후 권한 변경

```
resource "aws_instance" "bastion_host" {

  ami                    = var.amazon-linux-2-ami
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.bastion_sg.id]
  key_name               = aws_key_pair.wordpress_server_key.key_name
  subnet_id              = module.app_vpc.public_subnets[0]

  connection {
    user        = "ec2-user"
    host        = self.public_ip
    private_key = file("/home/vagrant/.ssh/id_rsa")
    timeout     = "1m"
  }

  provisioner "file" {
    source      = "/home/vagrant/.ssh/id_rsa"
    destination = "/tmp/key_file"
  }

  provisioner "remote-exec" {
    inline = [
      "sudo cp /tmp/key_file /home/ec2-user/.ssh/id_rsa",
      "sudo chmod 400 /home/ec2-user/.ssh/id_rsa"
    ]
  }

  tags = {
    Name = "Bastion Host"
}
```

02 AWS 아키텍처 구성 및 설명

📌확인 Point

Provisioner “remote-exec” 를 통해 생성된 instance 내부에서 명령어를 실행

`sudo ~ install -y ansible2`

andible 설치

`sed -i ‘s/ ~ /g’ /home/~/main.yaml`

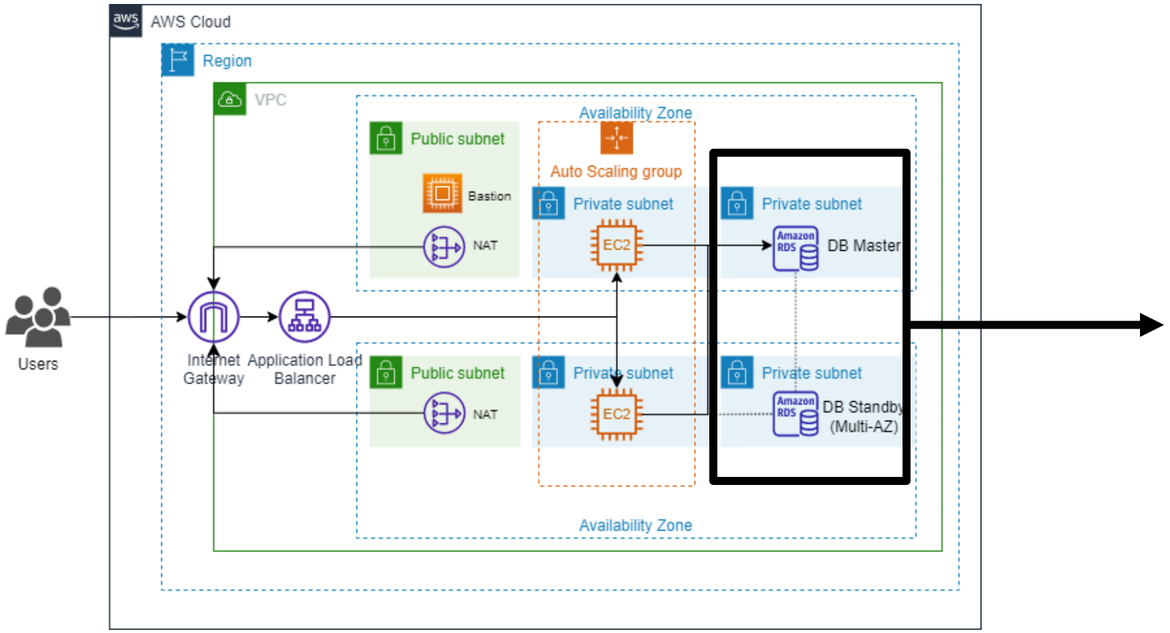
sed 명령어로 rdsendpoint 를 aws_db_instance로 생성된 rds endpoint로 변경

`ansible-playbook --connection=local ~ -b`

로컬로 /home/~ 경로에 있는 wordpress 플레이북 파일을 실행

```
resource "aws_instance" "instance_for_ami" {

  #RDS 엔드포인트가 필요하기 때문에, rds 생성 후 EC2 생성
  depends_on = [
    aws_db_instance.db-wordpress
  ]

  ami                    = var.amazon-linux-2-ami
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.wp_security.id]
  key_name               = aws_key_pair.wordpress_server_key.key_name
  subnet_id              = module.app_vpc.private_subnets[0]

  connection {
    ~~~~~~~~~~~~~~~~~~~~~~
  }

  #yaml 파일 복사
  provisioner "file" {
    source      = "wp"
    destination = "wp"
  }

  #ansible 설치 및 playbook 실행
  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras install -y ansible2",
      "sed -i 's/rdsendpoint/${aws_db_instance.db-wordpress.endpoint}/g' /home/ec2-
user/wp/roles/wordpress/vars/main.yaml",
      "ansible-playbook --connection=local /home/ec2-user/wp/wordpress.yaml -b"
    ]
  }
}
```

02 AWS 아키텍처 구성 및 설명



Multi-AZ DB

WordPress의 DB 인스턴스를 다중 AZ로 프로비저닝하여 기본 DB 인스턴스를 자동 생성하고 다른 가용 영역(AZ)에 있는 예비 인스턴스에 데이터를 동기적으로 복제하도록 구성

이와 같이 장애를 감지하면 Amazon RDS가 수동 개입 없이 자동으로 대기 인스턴스로 장애 조치하도록 하여 고가용성 확보

02 AWS 아키텍처 구성 및 설명

📌확인 Point

AWS RDS

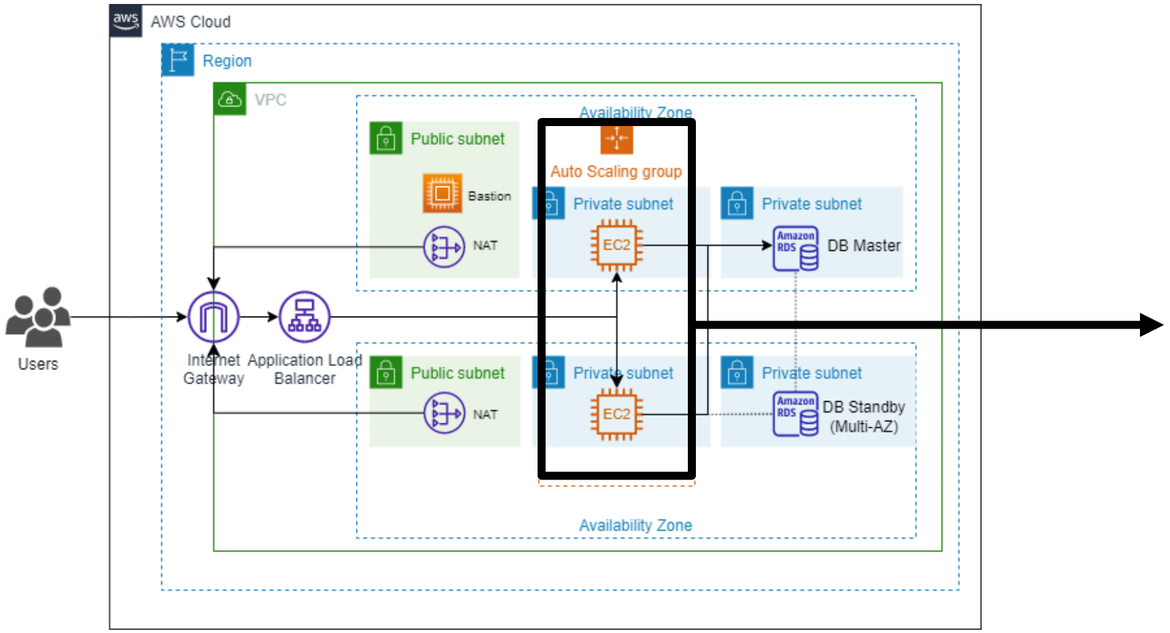
DB 서브넷 그룹을 생성하고, DB 인스턴스를 생성

`multi_az` argument를 true로 정의함으로써

DB를 Multi-AZ로 구성

```
#RDS Subnet group
resource "aws_db_subnet_group" "rds_subnet_group" {
  name = "db-wordpress-group"
  subnet_ids = [
    "${module.app_vpc.private_subnets[2]}",
    "${module.app_vpc.private_subnets[3]}"
  ]
}

#RDS Instance
resource "aws_db_instance" "db-wordpress" {
  allocated_storage    = 10
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t3.micro"
  name                 = "wordpress"
  identifier           = "wordpress"
  db_subnet_group_name = aws_db_subnet_group.rds_subnet_group.name
  username             = var.db-user
  password             = var.db-password
  parameter_group_name   = "default.mysql5.7"
  skip_final_snapshot    = true
  vpc_security_group_ids = [aws_security_group.allow_to_rds.id]
  multi_az = true
}
```

02 AWS 아키텍처 구성 및 설명



Auto Scaling

WordPress 구성이 완료된 AMI를 활용하여
Auto Scaling Group을 구성

돌발 상황으로 인한 다운 타임을 방지하고 단일 장애로
인해 전체 시스템이 중단되지 않도록
내결함성과 고가용성 확보를 위해 사용

02 AWS 아키텍처 구성 및 설명

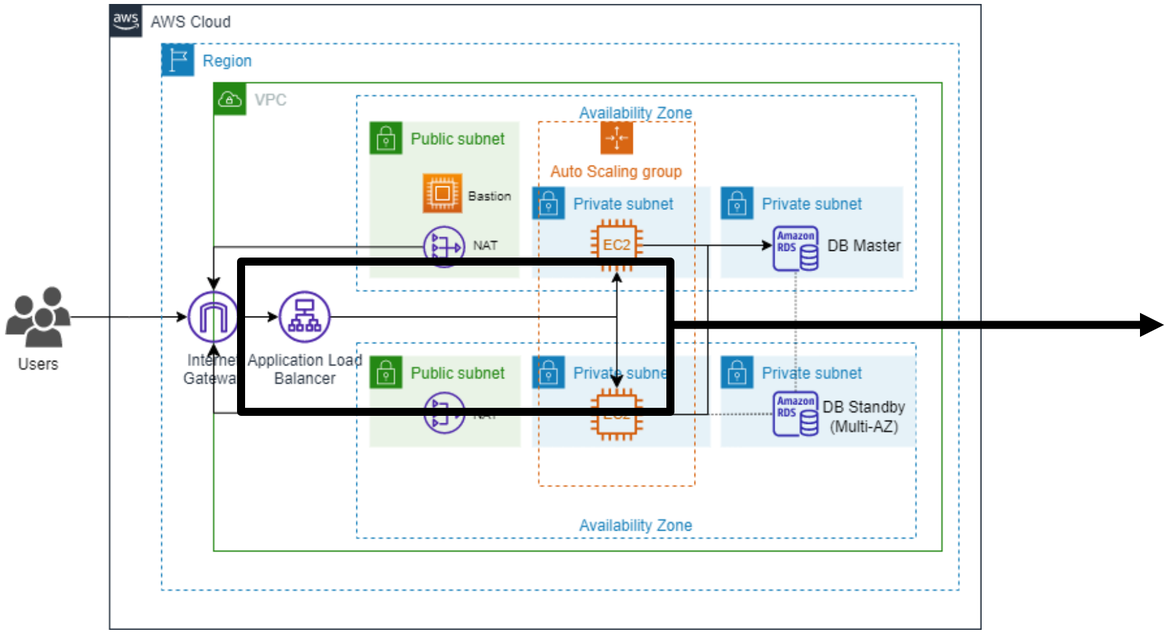
📌확인 Point

AWS Auto Scaling

WordPress 구성이 완료된 시작 템플릿을 사용하여
최소 용량 2개, 최대 용량 4개가 되도록
인스턴스의 크기를 오토 스케일링

```
#AutoScaling Group
resource "aws_autoscaling_group" "wp_atsg" {
  name                      = "wordpress-autoscaling"
  health_check_type         = "ELB"
  health_check_grace_period = 120
  force_delete              = true

  vpc_zone_identifier = [
    module.app_vpc.private_subnets[0],
    module.app_vpc.private_subnets[1]
  ]

  termination_policies = ["OldestInstance"]
  launch_template {
    id      = aws_launch_template.wordpress_template.id
    version = aws_launch_template.wordpress_template.latest_version
  }

  desired_capacity = 2
  min_size         = 2 #최소용량
  max_size         = 4 #최대용량

  lifecycle {
    create_before_destroy = true
  }

  target_group_arns = [aws_alb_target_group.wp-elb-tg.arn]
}
```

02 AWS 아키텍처 구성 및 설명



Application LB

Auto Scaling Group에 로드 밸런서 연결

Web 서버로의 트래픽을 분산하여
트래픽 병목 현상을 예측하고 막으며
향상된 속도 및 가용성을 확보하기 위해 로드 밸런서 사용

02 AWS 아키텍처 구성 및 설명

📌확인 Point

AWS Application Load Balancer

앞서 생성한 오토 스케일링 그룹에 로드 밸런서를 연결하여 외부에서 80 포트로 접속하는 사용자를 WordPress 서버로 연결하고 트래픽을 분산

```
#Applicatioin Load Balancer
resource "aws_alb" "wp-elb" {
  name                             = "wp-alb"
  internal                         = false # internet facing 설정
  load_balancer_type               = "application"
  security_groups                  = [aws_security_group.allow_http_sg.id]
  subnets                          = [module.app_vpc.public_subnets[0],
  module.app_vpc.public_subnets[1]]
  enable_cross_zone_load_balancing = true
}

#Target Group
resource "aws_alb_target_group" "wp-elb-tg" {
  name     = "wp-alb-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = module.app_vpc.vpc_id
}

#Load Balancer Listener
resource "aws_alb_listener" "wp-elb-listener" {
  load_balancer_arn = aws_alb.wp-elb.arn
  port              = 80
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_alb_target_group.wp-elb-tg.arn
  }
}

#AutoScaling attachment
resource "aws_autoscaling_attachment" "wp-atsg-attach" {
  autoscaling_group_name = aws_autoscaling_group.wp_atsg.name
  alb_target_group_arn   = aws_alb_target_group.wp-elb-tg.arn
}
```

CHAPTER 03

AWS Demo

03 AWS 실시간 Demo

output 확인해서 접속

CHAPTER 04

As-is, To-be

04 AWS As-is, To-be

- AWS Cloud 활용 가능 (포털 사용 및 리소스 코드화)
- Bastion Host 접속을 통한 Web Server 접속을 구현하여 보안 강화
- Auto Scaling을 이용하여 클라우드의 유연성을 강화
- Terraform을 이용하여 Wordpress 배포 자동화

04

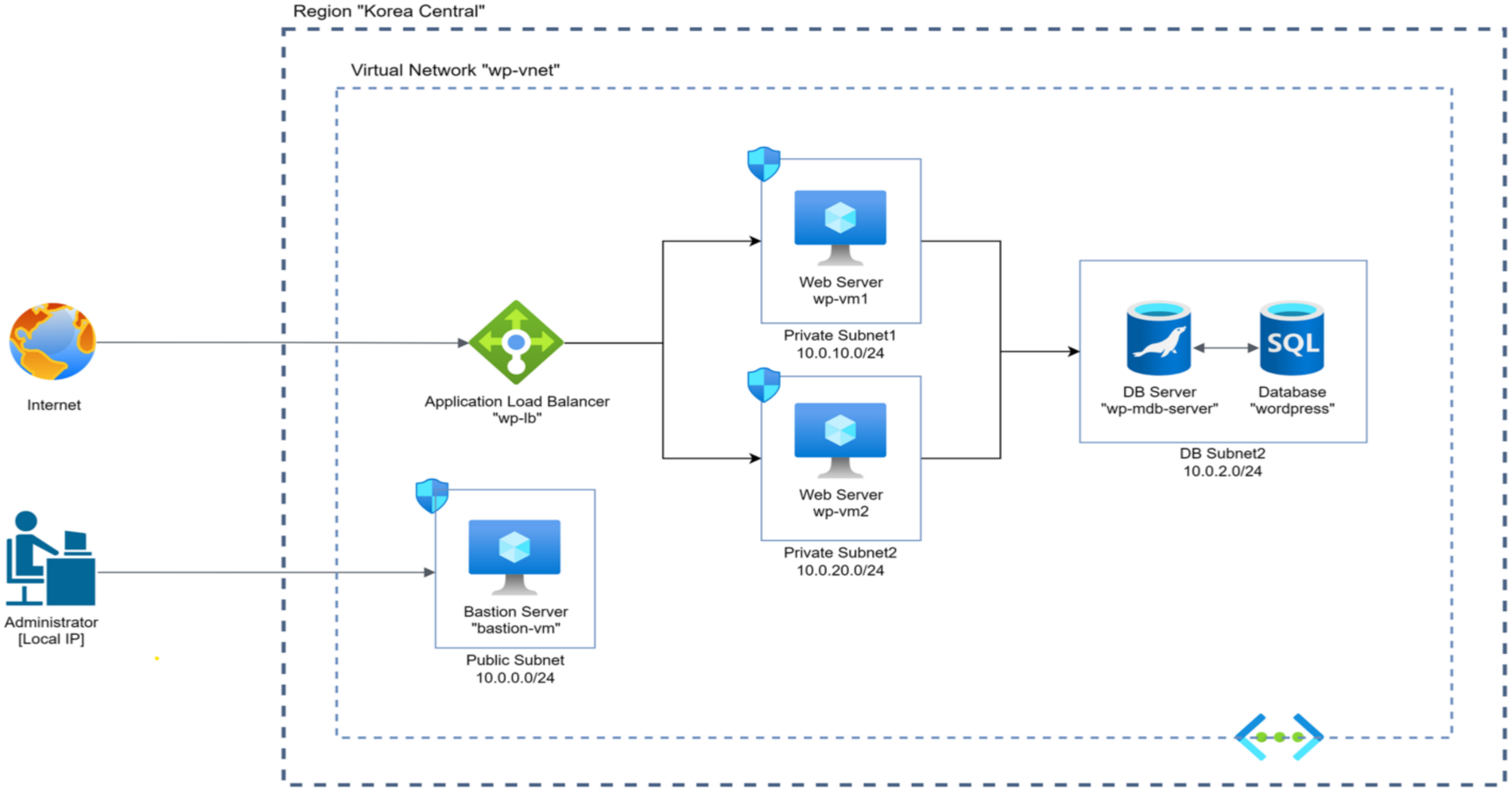
AWS As-is, To-be

- Bastion Host Server의 AutoScaling 구현으로 단일 장애점 해결
- Ansible Playbook 변수 파일 Vault 암호화를 통한 보안 강화
- 추가 변수 처리를 통한 실행 파일 재사용성 확보

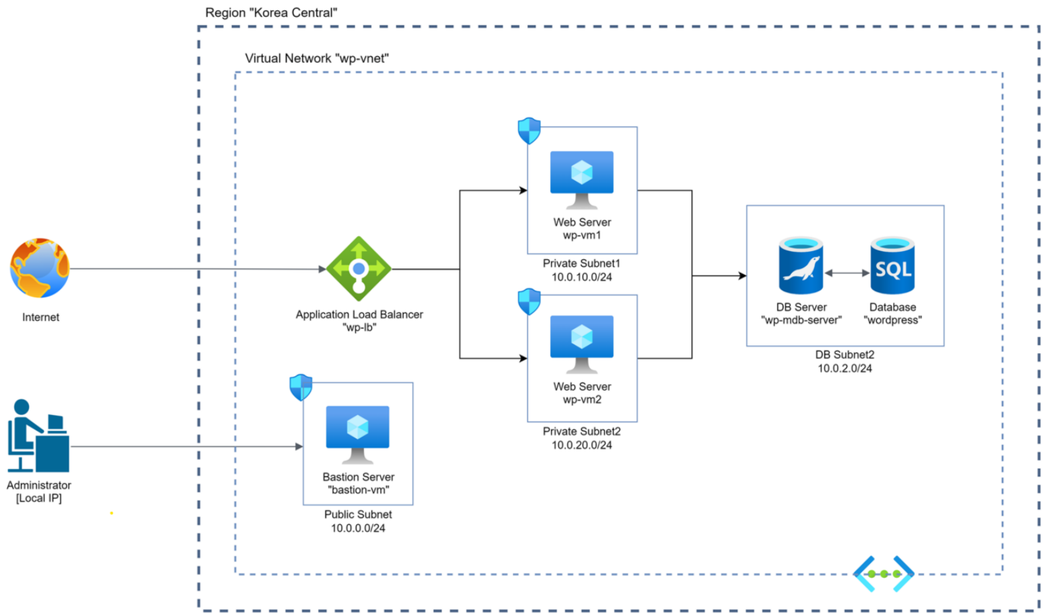
CHAPTER 05

Azure

05 Azure 아키텍처 구성 및 설명



05 Azure 아키텍처 구성 및 설명



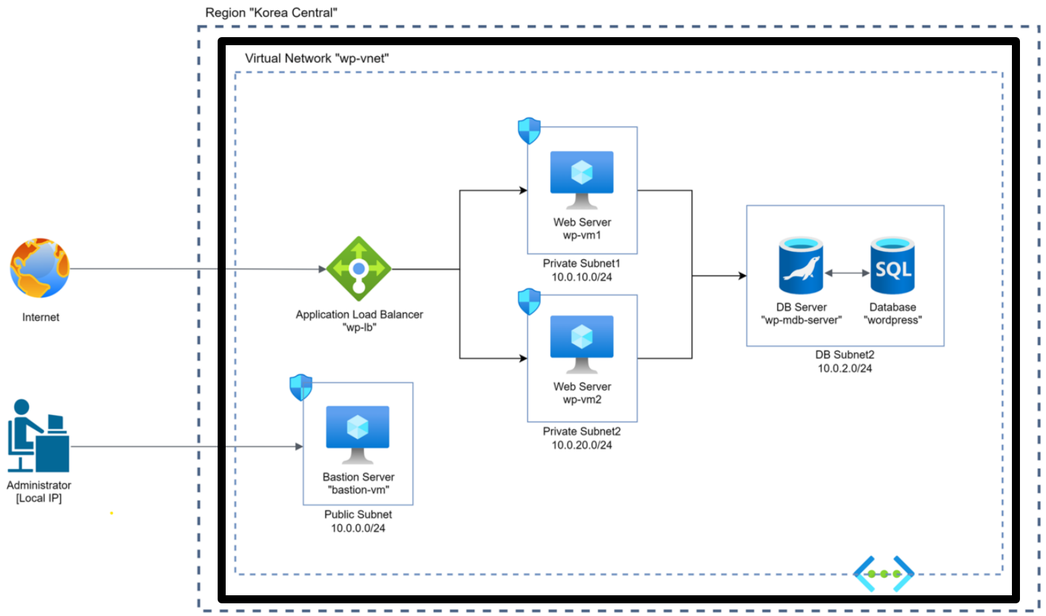
가용성 및 안정성 확보를 위한 아키텍쳐

Bastion Host를 Public Subnet에 구성

Web Server, DB 서버 Private Server에 구성하여 보안성 강화

Application Load Balancer를 통해 외부 트래픽을 분산하여 고가용성 확보

02 Azure 아키텍처 구성 및 설명



Resource Group

모든 리소스를 그룹화하여 관리

02 Azure 아키텍처 구성 및 설명

배포 전 확인 사항

테스트 환경

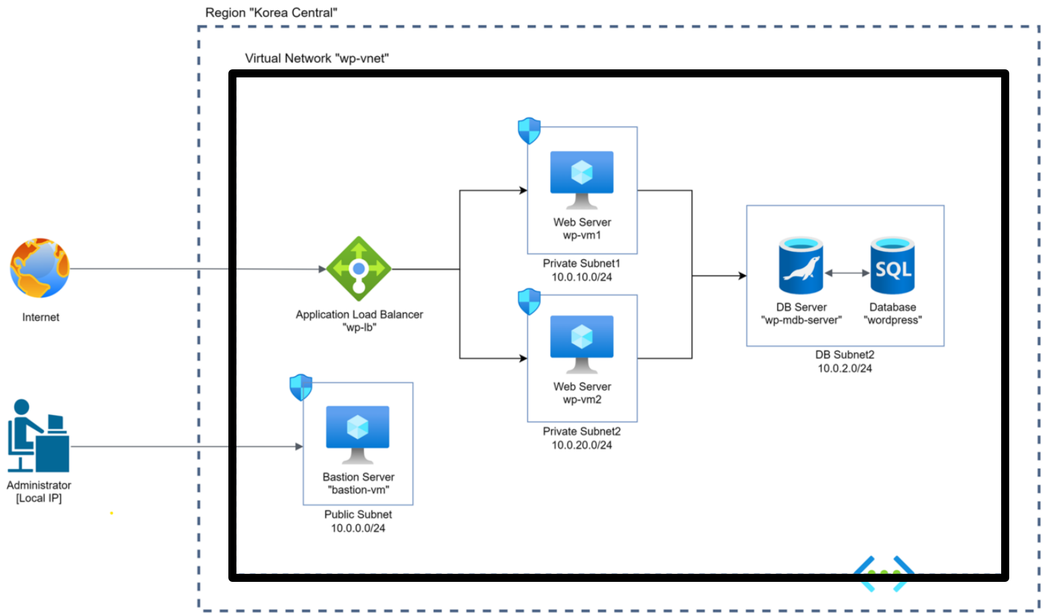
Resource	Configuration
CPU	2
Memory	4096 MB
Disk	40 GB
OS	CentOS 7

구성 관리 / 배포 도구

배포 전 Master Server에 설치되어 있어야하는 도구

Tool	Version
Terraform	v 1.1.9 on linux_amd64
Ansible	v 2.9.27
	python v 2.7.5 필요
Azure CLI	v 2.36.0
	Python (Linux) 3.6.8 필요

02 Azure 아키텍처 구성 및 설명

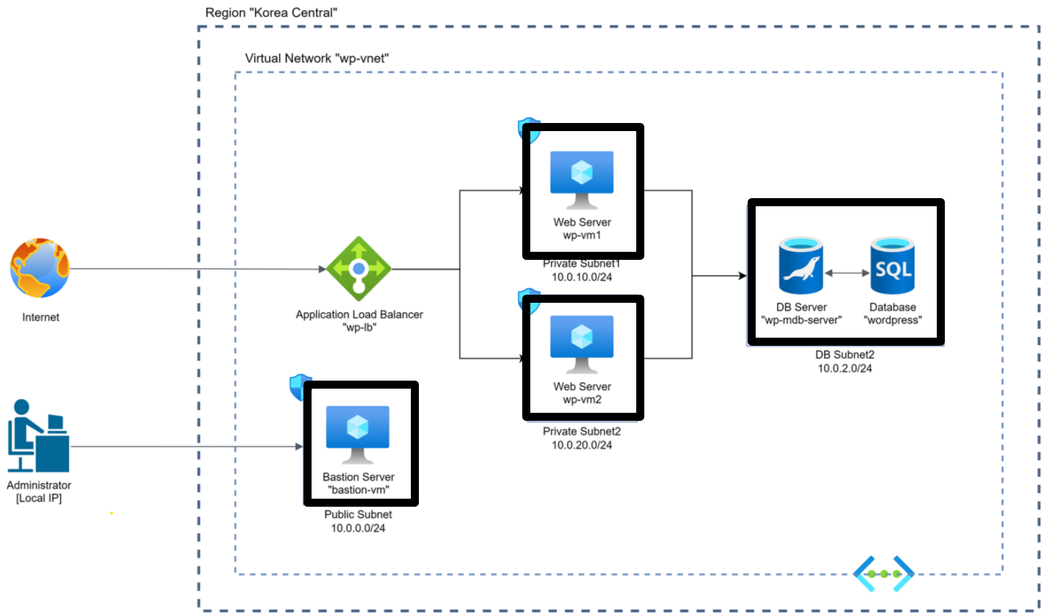


Vnet (Virtual Network)

할당 된 IP 주소 블록에 의해 정의

AWS와는 다르게 Public, Private Subnet을 따로 구분하지 않으나 임의로 Public, Private Subnet으로 구분하여 표기

02 Azure 아키텍처 구성 및 설명



Vnet - Subnet

Public subnet 1개, Private Subnet 2개,

DB Subnet 1개로 구성

Public Subnet - Bastion Host VM

Private Subnet - Web Server VM

DB Subnet - Maria DB 서버 + SQL 기반 데이터베이스

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

Meta Argument `count` 와 변수로 반복문 구성하여
Private Subnet 생성 코드 단순화

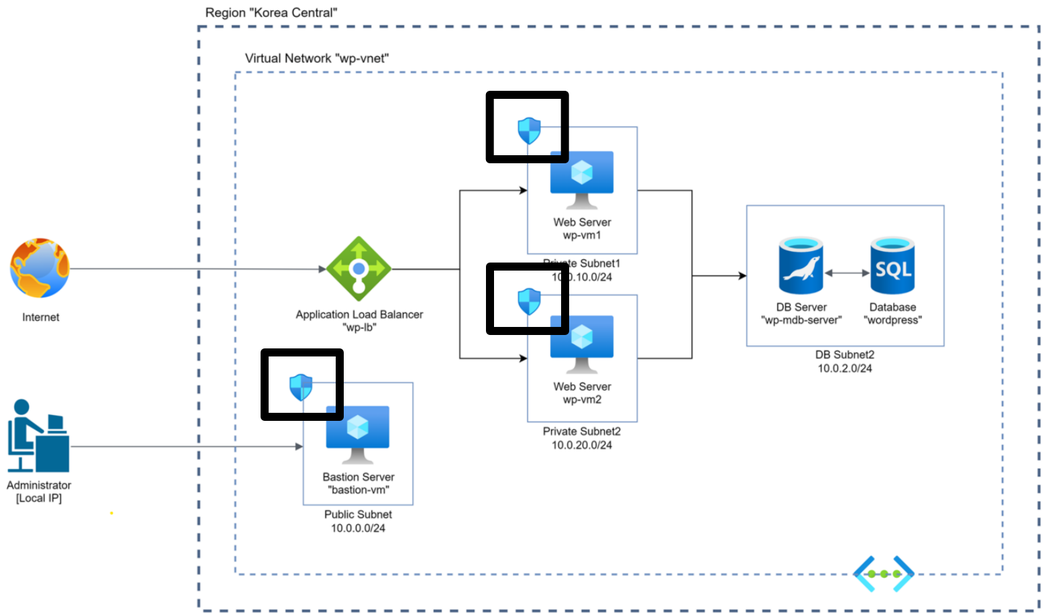
variable.tf

```
variable "vm-num" {

 description = "The Number Of VM"
 type = number
 default = 2

}
```

prv.tf

```
resource "azurerm_network_interface" "wp-prv-nip" {

 #반복문

 count = var.vm-num
 name                = "wp-prv-nic${count.index}"
 location            = azurerm_resource_group.wp-rg.location
 resource_group_name = azurerm_resource_group.wp-rg.name

 ip_configuration {

 name                          = "internal"
 subnet_id                     = azurerm_subnet.wp-private-subnet[count.index].id
 private_ip_address_allocation = "Dynamic"

 }
}
```

02 Azure 아키텍처 구성 및 설명



Network Security Group

인바운드 및 아웃바운드 트래픽 제어

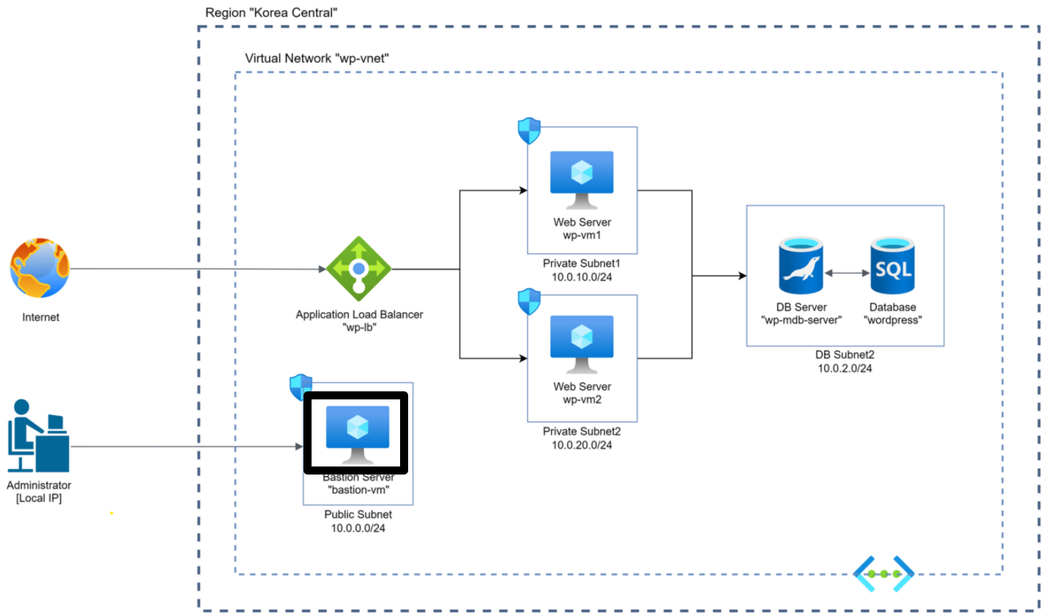
[Bastion Host NSG]

SSH : Port 22 : Administrator IP

[Web NSG]

SSH : Port 22 : Bastion Host NSG

02 Azure 아키텍처 구성 및 설명

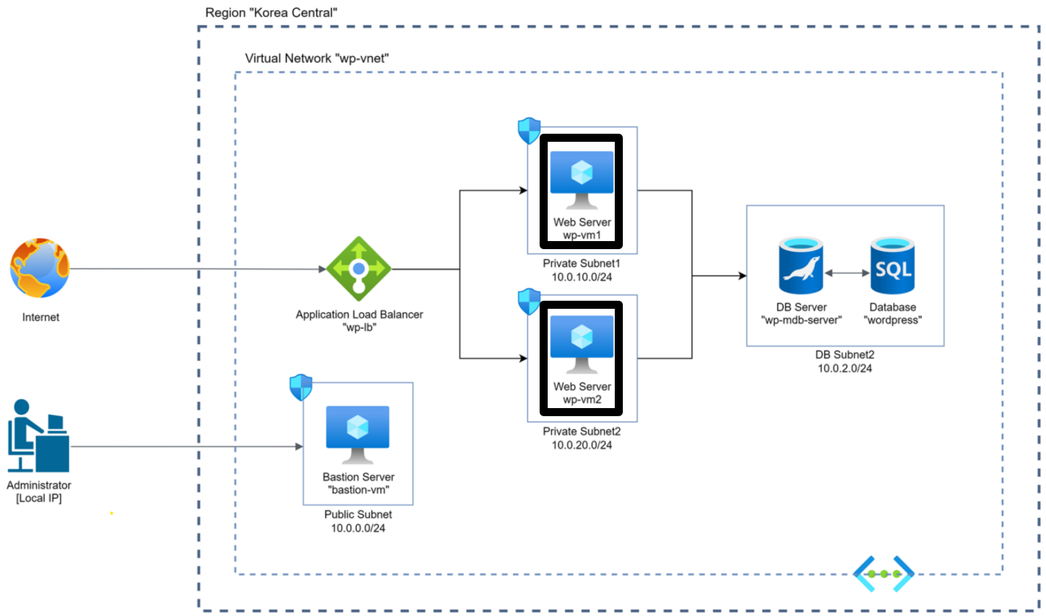


VM (Virtual Machine)

- **Bastion Host Server**

허용된 IP 주소로 접근하는 사용자만 Web Server에 접근 가능하게 하는 중간자 역할

02 Azure 아키텍처 구성 및 설명



VM (Virtual Machine)

- **Web Server**

Apache 웹 서버 프로그램과 PHP를 사용하여 구현

Wordpress 배포에 필요한 구성 패키지 설치, 설정 변경

Bastion Host 서버를 거쳐서 접속

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

VM 생성 시 네트워크 인터페이스 리소스를 함께 생성하여 서브넷에 연결

```
resource "azurerm_network_interface" "wp-pub-nip" {

 name                = "wp-pub-nic"
 location            = azurerm_resource_group.wp-rg.location
 resource_group_name = azurerm_resource_group.wp-rg.name

 ip_configuration {

 name                          = "internal"
 subnet_id                     = azurerm_subnet.wp-public-subnet.id
 private_ip_address_allocation = "Dynamic"
 public_ip_address_id          = azurerm_public_ip.wp-publicip.id

 }
}
```

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

Playbook 실행 시

원격 서버 계정의 절대 경로가

Web Server VM 생성 시 지정한 이름과 동일한지 확인

roles/wordpress/vars/main.tf

```
# Wordpress Destination
remote_dest: "/home/adminuser"
```

wordpress.yaml

```
---
- hosts : all
  become : true
  vars_files:
    - /home/adminuser/wp/roles/apache/vars/main.yaml
    - /home/adminuser/wp/roles/wordpress/vars/main.yaml

  roles :
    - apache
    - wordpress
```

02 Azure 아키텍처 구성 및 설명

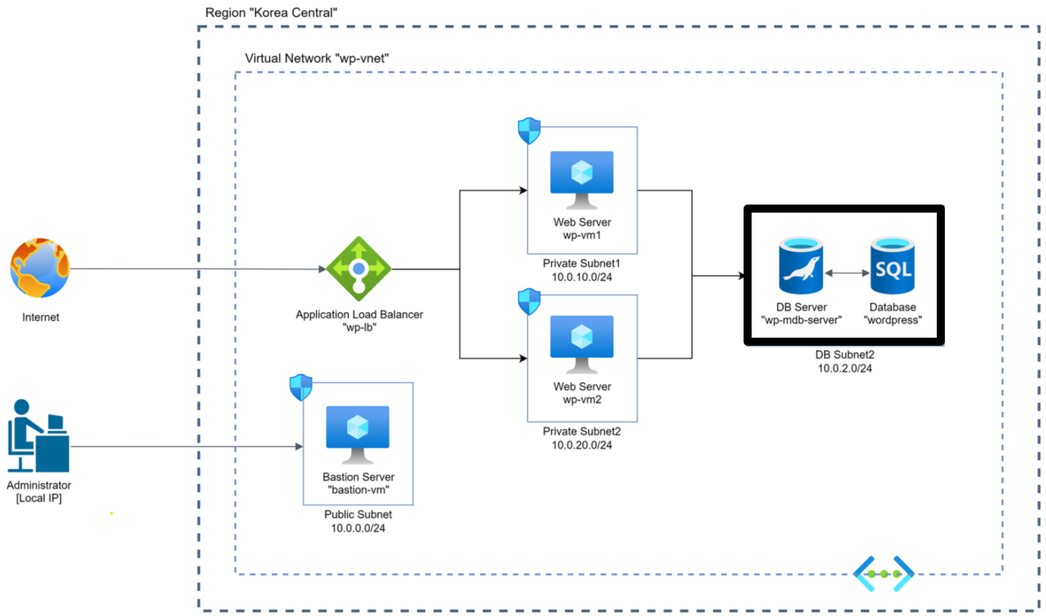
📌확인 포인트

Playbook 실행 시

Wordpress 설정 파일의 DB Charset을 “utf8”로 설정

```
// ** 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["host"] }}' );

/** 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["utf"] }}' );
```

02 Azure 아키텍처 구성 및 설명



Database

MariaDB 서버

Wordpress 실행 시 10.2 이상 버전 설치

HTTP 프로토콜을 이용해 서버에 접속하기 위해 SSL 설정과 selinux를 비활성화

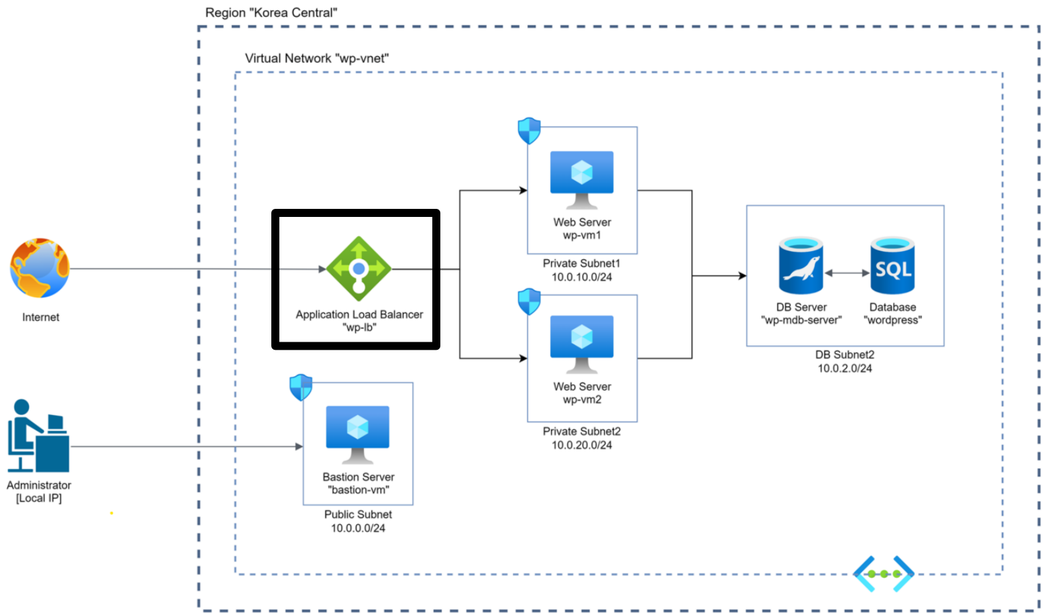
02 Azure 아키텍처 구성 및 설명

📌확인 포인트

Wordpress 데이터베이스의 DB collation을 `utf8_general_cli` 로 설정

```
resource "azurerm_mariadb_database" "wordpress" {

 name                = "wordpress"
 resource_group_name = azurerm_resource_group.wp-rg.name
 server_name         = azurerm_mariadb_server.wp-mdb-server.name
 charset             = "utf8"
 collation           = "utf8_general_ci"

}
```

02 Azure 아키텍처 구성 및 설명



Load Balancer

앞서 생성한 두 VM을 LB의 Backend Pool에 포함시켜 부하 분산

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

비어있는 Backend Pool을 생성한 후,

부하 분산을 할 리소스들을 생성된 Backend Pool에 추가

```
# 빈 Backend Pool 생성

resource "azurerm_lb_backend_address_pool" "lb-pool" {

 loadbalancer_id = azurerm_lb.wp-lb.id
 name            = "lb-pool"

 depends_on = [

 azurerm_lb.wp-lb

 ]
}

# Backend Pool에 제어할 리소스 추가

resource "azurerm_lb_backend_address_pool_address" "vm-address" {

 count = var.vm-num

 name                    = "vm${count.index}-address"
 backend_address_pool_id = azurerm_lb_backend_address_pool.lb-pool.id
 virtual_network_id      = azurerm_virtual_network.wp_vnet.id
 ip_address              = azurerm_network_interface.wp-prv-nip[count.index].private_ip_address

 depends_on = [
 azurerm_lb_backend_address_pool.lb-pool
 ]
}
```

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

LB Endpoint 부하 분산을 위해 상태 확인을 위해
Probe를 생성
HTTP 프로토콜을 사용하여 Probe를 진행하기 위해
80번 포트를 지정

```
resource "azurerm_lb_probe" "lb-probe" {

 loadbalancer_id = azurerm_lb.wp-lb.id
 name            = "lb-probe"
 port            = 80

 depends_on = [
 azurerm_lb.wp-lb
 ]
}
```

02 Azure 아키텍처 구성 및 설명

📌확인 포인트

Load Balancer는 따로 보안 규칙을 지정하여 트래픽 관리하고 Load Balancing 관련 리소스 연결

```
resource "azurerm_lb_rule" "lb-rule" {

 loadbalancer_id                = azurerm_lb.wp-lb.id
 name                           = "lb-rule"
 protocol                       = "Tcp"
 frontend_port                  = 80
 backend_port                   = 80

 frontend_ip_configuration_name = "lb-publicip"

 backend_address_pool_ids = [azurerm_lb_backend_address_pool.lb-pool.id]

 probe_id = azurerm_lb_probe.lb-probe.id

 depends_on = [

 azurerm_lb.wp-lb,
 azurerm_lb_probe.lb-probe

 ]
}
```

CHAPTER 06

Azure Demo

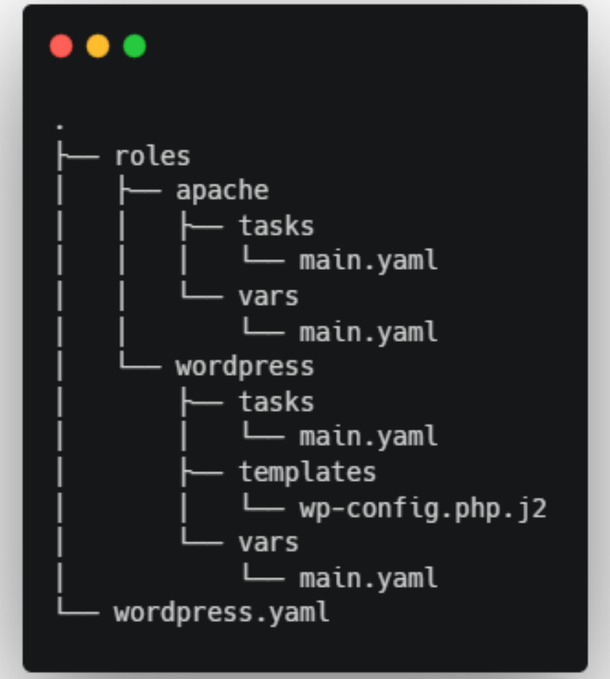
06 AWS 실시간 Demo

output 확인해서 접속

CHAPTER 07

Ansible-Playbook

07 Ansible Wordpress Playbook



📌 Roles

Role	Directory	Description
apache	tasks	Remi Repo + PHP74 Apache 관련 서비스 패키지 다운로드 및 실행
	vars	Rpm 설치 url Mirrorlist url 서비스 패키지명

Role	Directory	Description
wordpress	tasks	Wordpress 다운 데이터베이스 설정
	Templates	Wordpress 설정 파일 템플릿
	vars	Wordpress 다운로드 url 원격 접속 할 계정 절대 경로 데이터베이스 설정

07 Ansible Wordpress Playbook

```yaml
---
- hosts : all
  become : true
  vars_files:
    - /home/ec2-user/wp/roles/apache/vars/main.yaml
    - /home/ec2-user/wp/roles/wordpress/vars/main.yaml

  roles :
    - apache
    - wordpress
```

📋 Wordpress.yaml

모든 host에 대해 관리자 권한으로 해당 파일을 실행

WP 디렉토리의 변수 파일을 참조
이때, 원격 접속하는 사용자에 따라 절대 경로 확인

플레이북에 지정 된 역할에 할당된 역할 실행

Local에서 실행되기 때문에
Inventory 및 Ansible 설정 파일 생성 불필요

07 Ansible Wordpress Playbook

```yaml
# Amazon Linux 사용을 위한 기본 설정
- name: "amazon linux2 setting"
  block:
    - name: "install epel"
      command:
          cmd: sudo amazon-linux-extras install -y epel
    - name: "install dependency"
      command:
          cmd: sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
  when: ansible_facts['distribution'] == "Amazon"
```

📋 Apache/tasks/main.yaml

내부에 설정 된 조건문에 따라 OS를 구분하여 실행

Ansible facts를 사용하여 해당 OS가 Amazon Linux일 때는 Amazon Linux 관련 패키지를 설치하도록 설정

CHAPTER 08

As-Is, To-Be

08 Azure As-Is, To-Be

- Azure Cloud 활용 가능 (포털 사용 및 리소스 코드화)
- Bastion Host 접속을 통한 Web Server 접속을 구현하여 보안 강화
- Load Balancer를 이용하여 Web 부하 분산 - Ansible Playbook을 이용하여 Web Server 구성 관리 자동화
- Terraform을 이용하여 Wordpress 배포 자동화

08 Azure As-Is, To-Be

- Bastion Host Server의 VMSS 구현으로 단일 장애점 해결
- Web Server의 VMSS 구현으로 가용성 및 안정성 확보
- Ansible Playbook 변수 파일 Vault 암호화를 통한 보안 강화
- 추가 변수 처리를 통한 실행 파일 재사용성 확보

THANK

YOU EVERYONE

PLEASE ENTER YOUR TEXT