Terraform 2

- 실습가이드

최초 작성일 : 2024/04/17

최종 제출일 : 2024/04/18

김민경

내용

Ⅰ. 개념정리

1. vpc

1-1. 개념

- Virtual Private Cloud
- **논리적으로 완벽하게 격리된 네트워크 공간**

1-2. 특징

- 계정 당 1개의 default VPC 있음
- 리전 1개 당 5개의 vpc 생성 가능

1-3. vpc 생성에 필요한 구성 요소

1) 리전

2) az

3) CIDR

4) 서브넷

5) RT

6) IGW

7) Network ACL & SG

8) VPC

2. 서브넷

2-1. 개념 및 특징

- 거대한 네트워크 대역의 vpc 주소 쪼갠 것
- 서브네팅하는 이유는 IP 효율적으로 사용 가능하며, 네트워크를 분리해 보안성을 강화하여 외부에 노출되면 X되는 서비스 별도 관리 가능하기 때문에

2-2. 종류

1) public subnet

- IGW 통해 외부와 통신 가능

2) private subnet

- 기본적으로 외부와 차단되어 있음
- 퍼블릭 서브넷에 생성되어 있는 NAT 인스턴스 or NAT GW와 연결하면 프라이빗 서브넷 안에 생성된 인스턴스는 외부로 나갈 수 있음

3. IGW

3-1. 개념

- VPC에 생성된 인스턴스들은 격리된 환경이므로 기본적으로 인터넷 사용 불가

  so IGW에 연결해야 함

3-2. 특징

- VPC별로 생성되어야 함
- VPC는 IGW 1개에만 연결 가능
- 자체적으로 인터넷에 접근할 수 있는 것이 아니라, 라우팅 테이블을 수정해줘야 접근 가능

4. NAT GW

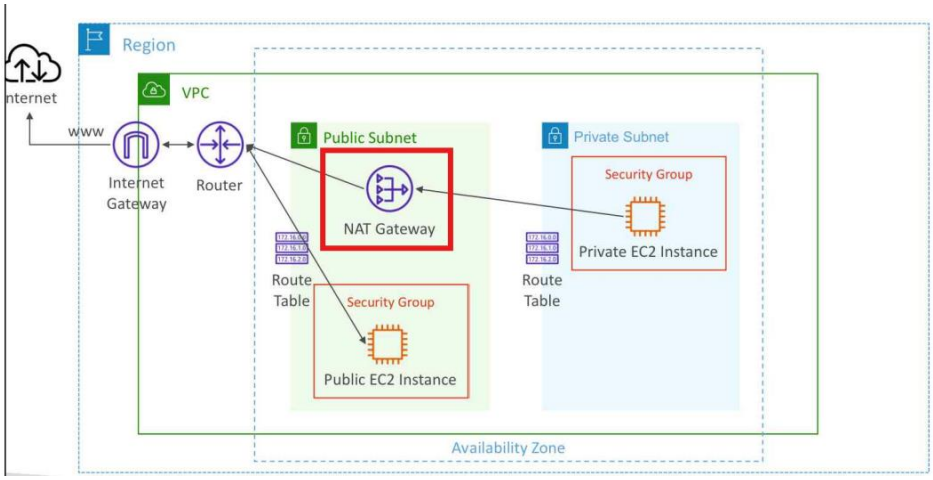
4-1. 개념

- AWS에서 제공하는 NAT 서비스

4-2. 특징

- 프라이빗 서브넷의 인스턴스가 VPC 외부에 연결 가능하게 하고,

  외부에서는 프라이빗 서브넷 내의 인스턴스와의 연결할 수 없도록 하기 위해 사용



5. SG

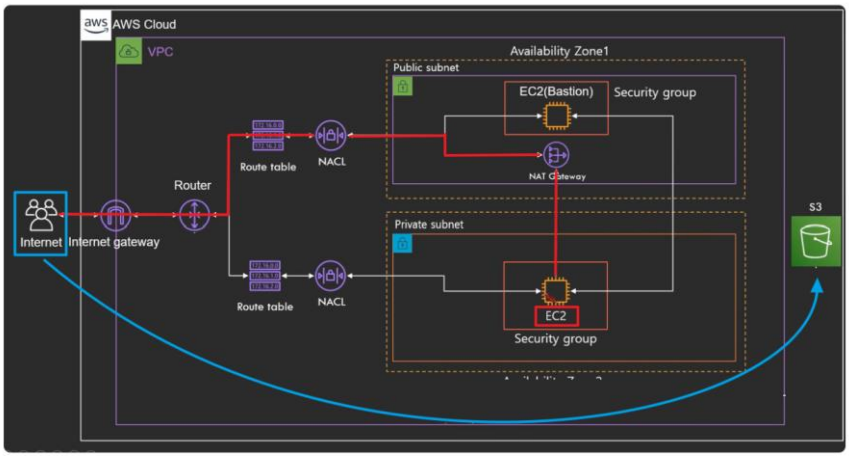
- 인스턴스에 대한 인바운드 및 아웃바운드 트래픽을 제어하는 가상 방화벽
- 서브넷 수준이 X닌 인스턴스 수준에서 작동함

6. vpc 엔드포인트

6-1. 개념

- cloudwatch, s3 등에 access 시 **퍼블릭 인터넷 거치지 x고 연결** (내부 네트워크 사용)
- private subnet의 ec2에서 s3에 대한 api를 부르게 된다면,

  아래 그림과 같이 <EC2(10.0.3.20, Private Subnet) → NAT Gateway → Router → Internet Gateway → 외부 인터넷 → S3> 순으로 이동함

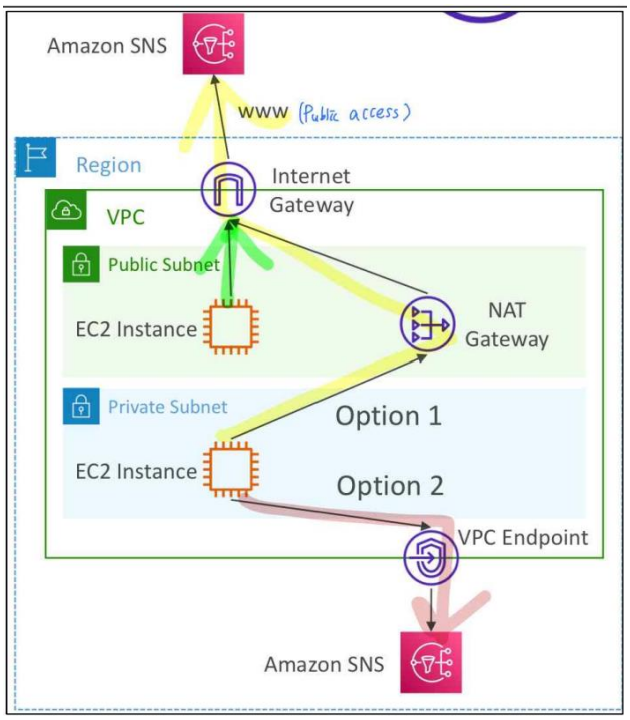


⇒ but 이는 통신 시 외부 인터넷에 공개적으로 연결되며 트래픽이 노출됨,

VPC 밖에서 들어오는 트래픽에는 과금이 되기 때문에 비용이 늘어남

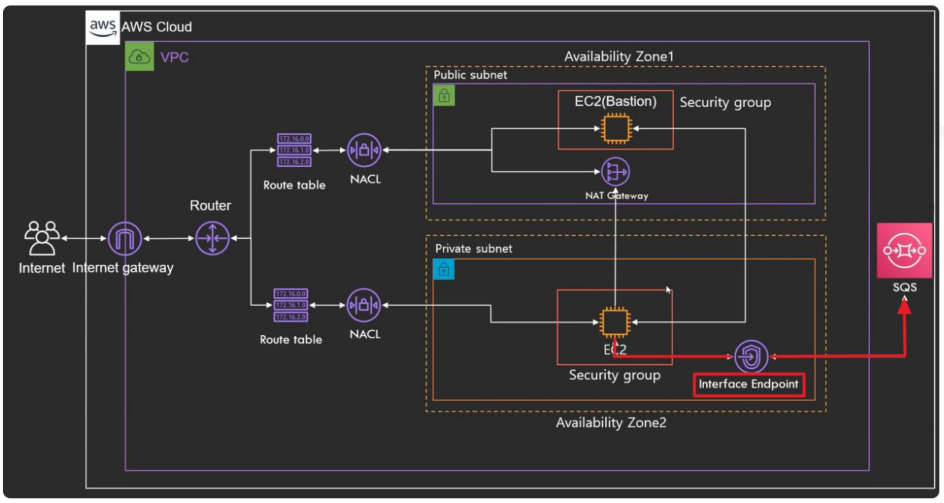
⇒ so vpc endpoint가 나오게 됨 (내부 네트워크로 접근)

- **IGW, NATGW 없이도 AWS 서비스에 접근 가능**함 (네트워크 인프라 단순화)



6-2. 종류

1) 인터페이스 엔드포인트

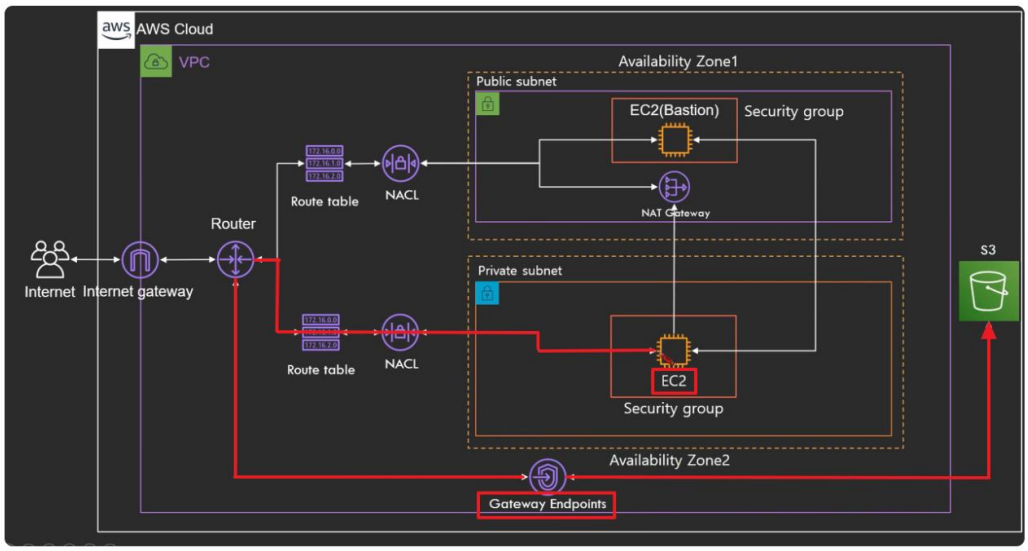


- 각 서브넷마다 Elastic network interface(ENI)를 생성하여 외부 서비스에 접근하기 위한 VPC 엔드포인트

> **참고) ENI**
>
> - 가상 네트워크 인터페이스
>
> - AWS 서비스, 다른 인스턴스, 온프레미스 서버, 인터넷 등 다른 네트워크 리소스와 통신할 수 있도록 하며, Secure Shell(SSH) 또는 Remote Desktop Protocol(RDP) 등을 이용해서 인스턴스에서 실행중인 OS 와도 통신할 수 있게 해줌

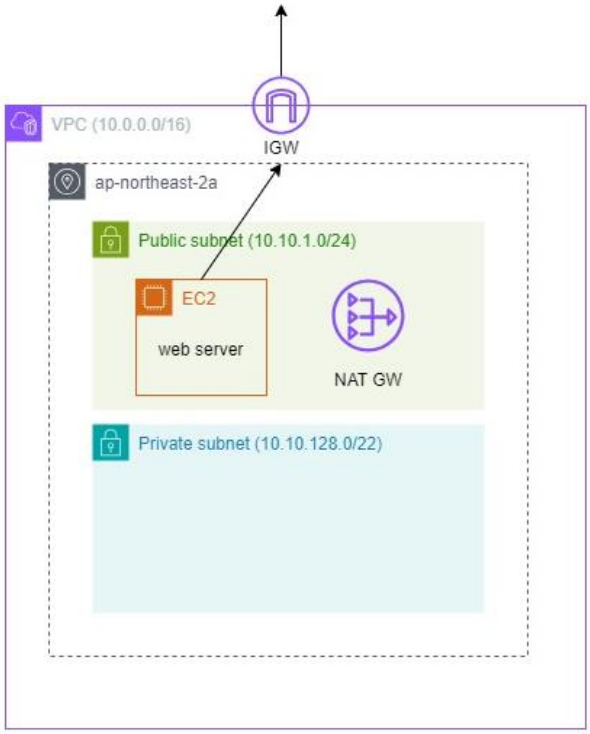
- private ip 만들어 서비스를 연결함
- private subnet 내부에 위치
- SQS, SNS, Kinesis 등 다양한 서비스 지원

2) 게이트웨이 엔드포인트



- 서브넷에서 라우팅 추가해서 RT 경로 보고 VPC 엔드포인트 통해 외부 서비스 접근
- VPC 내부에 위치
- S3와 DynamoDB, 등의 일부 서비스만을 지원

Ⅱ. 실습 1 (웹 서비스 포트를 입력 변수 5000번을 통해 배포 후 접속 결과 확인)



1. variables.tf 설정하기

1-1. 아래 코드로 variables.tf 만들기

```
cat <<EOT > variables.tf
variable "server_port" {
  description = "The port the server will use for HTTP requests"
  type        = number
  default     = 50000
}
EOT
```

2. main.tf 설정하기

2-1. 아래 코드로 main.tf 만들기

```
cat <<EOT > main.tf
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_instance" "example" {
  ami                    = "ami-09a7535106fbd42d5"
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = <<-EOF
              #!/bin/bash
              echo "MinKyeong Server - var test" > index.html
              nohup busybox httpd -f -p \${var.server_port} &
              EOF

  user_data_replace_on_change = true

  tags = {
    Name = "Single-MyWebSrv"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name

  ingress {
    from_port   = var.server_port
    to_port     = var.server_port
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-my-instance"
}

output "public_ip" {
  value       = aws_instance.example.public_ip
  description = "The public IP of the Instance"
}
EOT
```

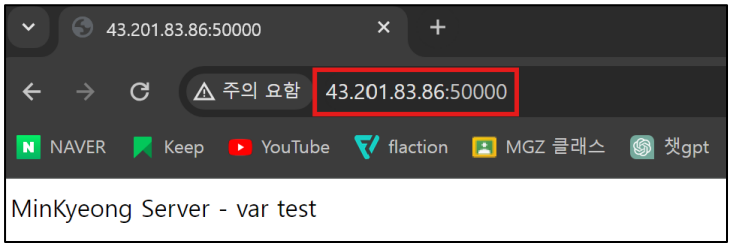
2-2. 적용하기

terraform apply

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_security_group.instance: Destroying... [id=sg-0eb1a7628a607b494]
aws_security_group.instance: Destruction complete after 1s
aws_security_group.instance: Creating...
aws_security_group.instance: Creation complete after 3s [id=sg-03e98febe60b91132]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Still creating... [30s elapsed]
aws_instance.example: Creation complete after 33s [id=i-097a855e7ba93a31b]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.

Outputs:

public_ip = "43.201.83.86"
vagrant@ubuntu-jammy:~/terraform$
```

2-3. 결과 확인하기

- 위 사진에서 얻은 public ip에 50000번 포트로 접속하기



Ⅲ. 실습 2 (Lab : AWS VPC 기본 구성 및 Terraform 명령 사용)

1. Provider 구성하기

1-1. 다음과 같이 코드 구성함

```
provider "aws" {                    # 사용하는 Vendor
  region = "ap-northeast-2"         # 서울 az
}
```

2. VPC & Public Subnet 생성하기

2-1. VPC & public 서브넷 1개 & private 서브넷 1개 생성하기

```
resource "aws_vpc" "mk_vpc" {                # VPC 생성
  cidr_block = "10.10.0.0/16"
  tags = {
    Name = "mk_vpc"
  }
}

resource "aws_subnet" "public_subnet1" {     # Public subnet 생성
  vpc_id = aws_vpc.mk_vpc.id
  cidr_block = "10.10.1.0/24"
  availability_zone = "ap-northeast-2a"
  tags = {
    Name = "public_subnet1"
  }
}
```

3. IGW 생성하기

3-1. 다음과 같이 코드 구성함

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "mk_vpc_IGW"
  }
}
```

4. RT 생성 및 연결하기

4-1. 다음과 같이 코드 구성함

- 하나의 RT로 여러 서브넷을 연결 가능
- 서브넷은 연결된 RT을 참조해 트래픽 전달함

```
resource "aws_route_table" "public_rt" {   # aws_route_table로 라우팅 테이블 생성
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "public_routing_table"
```

```
  }
}

# aws_route_table_association으로 서브넷을 라우팅 테이블에 연결
resource "aws_route_table_association" "public_rt_association1" {
  subnet_id = aws_subnet.public_subnet1.id
  route_table_id = aws_route_table.public_rt.id
}

resource "aws_route" "public_rt_igw" {
  route_table_id = aws_route_table.public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.igw.id
}
```

- aws_route_table로 라우팅 테이블 생성

- aws_route_table_association으로 서브넷을 라우팅 테이블에 연결

5. Private Subnet 생성하기

5-1. 다음과 같이 코드 구성함

- Public 과 다르게 인터넷이 직접 접속하지 못하는 네트워크 대역
- 생성 방법은 Public 과 동일함

  but 이를 IGW or NAT Gateway 에 연결하냐에 따라 Public or Private 이 결정됨

```
resource "aws_subnet" "private_subnet1" {
  vpc_id = aws_vpc.mk_vpc.id
  cidr_block = "10.10.128.0/22"
  availability_zone = "ap-northeast-2a"
  tags = {
    Name = "private_subnet1"
  }
}
```

6. NAT GW 생성 및 Private 네트워크 환경 구축하기

6-1. 다음과 같이 코드 구성함

- private subnet 은 외부와 통신하려면 NAT GW 가 필요함

  (NAT 인스턴스도 가능하지만 이 실습에서는 NAT GW 를 사용함)

- 다수의 서버가 NAT GW 1 개 공유 가능

  ⇒ NAT GW IP (EIP)를 다수의 private 서버가 공유함

- so **NAT GW 생성 시 EIP 연결하기,**

  public subnet 에 생성하고, private subnet 과 연결하기

```
resource "aws_eip" "nat_ip" {     # EIP 생성
  vpc = true
  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = aws_eip.nat_ip.id
  subnet_id = aws_subnet.public_subnet1.id  # NAT GW는 public subnet에 위치함
  tags = {
    Name = "NAT_gateway"
  }
}
```

```
resource "aws_route_table" "private_rt" {  # private rt 생성
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "private_routing_table"
  }
}

# private subnet 1을 rt에 연결
resource "aws_route_table_association" "private_rt_association1" {
  subnet_id = aws_subnet.private_subnet1.id
  route_table_id = aws_route_table.private_rt.id
}

resource "aws_route" "private_rt_nat" {
  route_table_id = aws_route_table.private_rt.id  # 이 rt이
  destination_cidr_block = "0.0.0.0/0"   # 이 도착지에 갈 때
  nat_gateway_id = aws_nat_gateway.nat_gateway.id   # 이 NAT GW를 거쳐가겠다
}
```

7. 웹서버 생성하기

7-1. 다음과 같이 코드 구성함

• public subnet 에 ec2 인스턴스 생성하기

```
resource "aws_instance" "example" {
  ami = "ami-09a7535106fbd42d5"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.public_subnet1.id
  vpc_security_group_ids = [aws_security_group.instance.id]
  associate_public_ip_address = true                    # public ip 할당하기
  user_data = <<-EOF
    #!/bin/bash
    sudo apt update -y
    sudo apt install apache2 -y
    sudo systemctl start apache2
    sudo echo "<img
src='https://cdn.011st.com/11dims/resize/600x600/quality/75/11src/product/5736697188/B.jpg?424000000'></img>" > /var/www/html/index.html
    sudo systemctl restart apache2
    sudo systemctl enable apache2
  EOF
  user_data_replace_on_change = true
  tags = {
    Name = "Web_Server"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  vpc_id = aws_vpc.mk_vpc.id
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # aws는 outbound는 기본으로 모든 프로토콜 허용이지만, 테라폼은 아님
  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"                  # 모든 프로토콜에 대해 출발지 트래픽을 허용
    cidr_blocks = ["0.0.0.0/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type = string
  default = "terraform-my-instance"
}

output "public_ip" {
  value = aws_instance.example.public_ip
  description = "The public IP of the Instance"
}
```

8. terraform으로 인프라 생성하기

8-1. main.tf 의 전체 코드는 다음과 같음

```
cat <<EOT > main.tf
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_vpc" "mk_vpc" {    # VPC 생성
  cidr_block = "10.10.0.0/16"
  tags = {
```

```
    Name = "mk_vpc"
  }
}

resource "aws_subnet" "public_subnet1" {  # Public subnet 생성
  vpc_id = aws_vpc.mk_vpc.id
  cidr_block = "10.10.1.0/24"
  availability_zone = "ap-northeast-2a"
  tags = {
    Name = "public_subnet1"
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "mk_vpc_IGW"
  }
}

resource "aws_route_table" "public_rt" {  # aws_route_table로 라우팅 테이블 생성
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "public_routing_table"
  }
}

# aws_route_table_association으로 서브넷을 라우팅 테이블에 연결
resource "aws_route_table_association" "public_rt_association1" {
  subnet_id = aws_subnet.public_subnet1.id
  route_table_id = aws_route_table.public_rt.id
}

resource "aws_route" "public_rt_igw" {
  route_table_id = aws_route_table.public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.igw.id
}

resource "aws_subnet" "private_subnet1" {
  vpc_id = aws_vpc.mk_vpc.id
  cidr_block = "10.10.128.0/22"
  availability_zone = "ap-northeast-2a"
  tags = {
    Name = "private_subnet1"
  }
}

resource "aws_eip" "nat_ip" {     # EIP 생성
  vpc = true
  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = aws_eip.nat_ip.id
  subnet_id = aws_subnet.public_subnet1.id  # NAT GW는 public subnet에 위치함
  tags = {
    Name = "NAT_gateway"
  }
}

resource "aws_route_table" "private_rt" {  # private rt 생성
  vpc_id = aws_vpc.mk_vpc.id
  tags = {
    Name = "private_routing_table"
  }
}

# private subnet 1을 rt에 연결
resource "aws_route_table_association" "private_rt_association1" {
```

```
  subnet_id = aws_subnet.private_subnet1.id
  route_table_id = aws_route_table.private_rt.id
}

resource "aws_route" "private_rt_nat" {
  route_table_id = aws_route_table.private_rt.id  # 이 rt이
  destination_cidr_block = "0.0.0.0/0"   # 이 도착지에 갈 때
  nat_gateway_id = aws_nat_gateway.nat_gateway.id   # 이 NAT GW를 거쳐가겠다
}

resource "aws_instance" "example" {
  ami = "ami-09a7535106fbd42d5"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.public_subnet1.id
  vpc_security_group_ids = [aws_security_group.instance.id]
  associate_public_ip_address = true  # public ip 할당하기
  user_data = <<-EOF
    #!/bin/bash
    sudo apt update -y
    sudo apt install apache2 -y
    sudo systemctl start apache2
    sudo echo "<img
src='https://cdn.011st.com/11dims/resize/600x600/quality/75/11src/product/5736697188/B.jpg?424000000'></img>" > /var/www/html/index.html
    sudo systemctl restart apache2
    sudo systemctl enable apache2
  EOF
  user_data_replace_on_change = true
  tags = {
    Name = "Web_Server"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  vpc_id = aws_vpc.mk_vpc.id
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {   # aws는 outbound는 기본으로 모든 프로토콜 허용이지만, 테라폼은 아님
    from_port = 0
    to_port = 0
    protocol = "-1"  # 모든 프로토콜에 대해 출발지 트래픽을 허용
    cidr_blocks = ["0.0.0.0/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type = string
  default = "terraform-my-instance"
}

output "public_ip" {
  value = aws_instance.example.public_ip
  description = "The public IP of the Instance"
}
EOT
```

8-2. terraform 으로 인프라 생성하기

terraform plan

```
vagrant@ubuntu-jammy:~/terraform$ terraform plan
aws_security_group.instance: Refreshing state... [id=sg-03e98febe60b91132]
aws_instance.example: Refreshing state... [id=i-097a855e7ba93a31b]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_eip.nat_ip will be created
  + resource "aws_eip" "nat_ip" {
      + allocation_id        = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = (known after apply)
      + id                   = (known after apply)
      + instance             = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
      + public_ipv4_pool     = (known after apply)
      + tags_all             = (known after apply)
      + vpc                  = true
    }
```

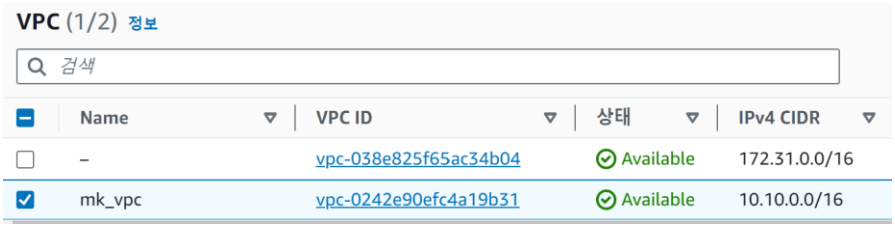
terraform apply

```
Apply complete! Resources: 14 added, 0 changed, 2 destroyed.

Outputs:

public_ip = "43.203.226.54"
vagrant@ubuntu-jammy:~/terraform$
```

8-3. 리소스 생성 확인하기

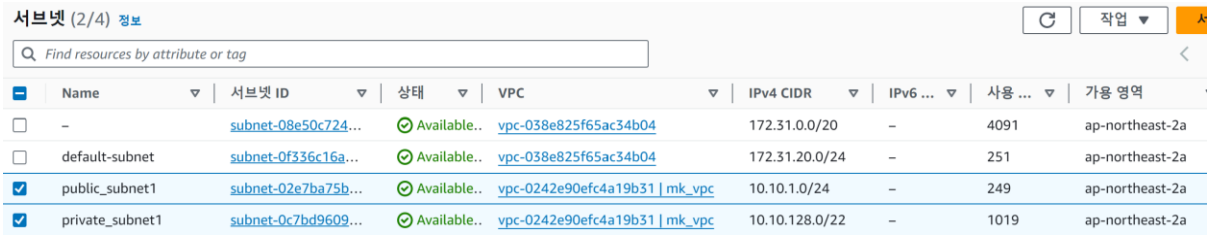
• aws 콘솔에서 리소스가 모두 잘 생성되었는지 확인하기

1) vpc 생성



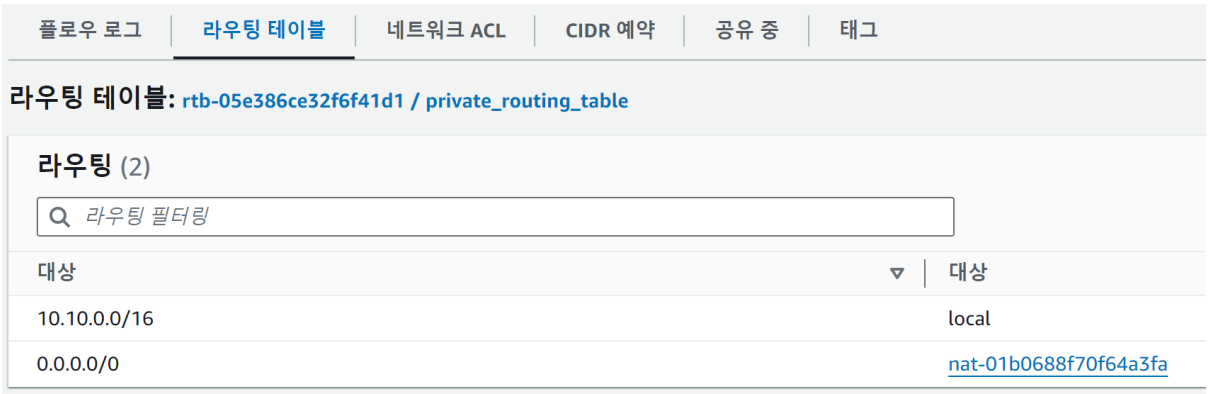
2) 서브넷 생성

- ap-northeast-2a 가용영역 안에 public, private 서브넷 1 개씩 생성

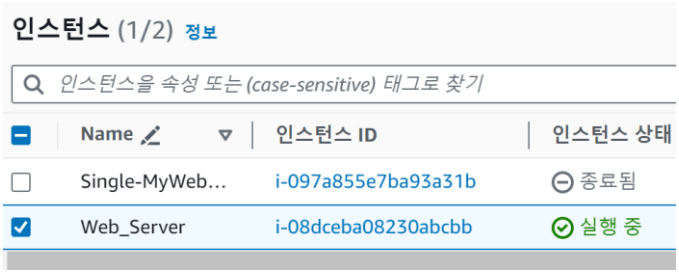


- public subnet 의 RT 는 IGW 와 연결됨을 확인

- 반면 private subnet 의 RT 는 IGW 에 연결되어 있지 X 고, NAT GW 와 연결됨을 확인

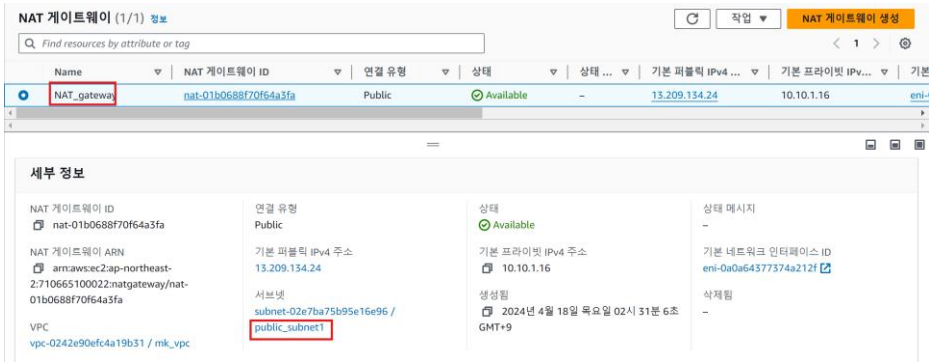


3) 웹서버 생성

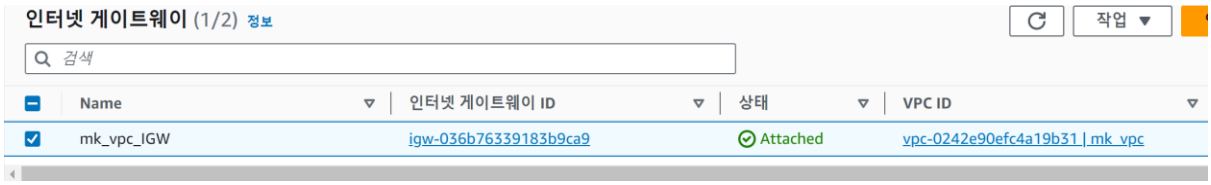


4) NAT GW 생성

- public subnet 안에 생성됨

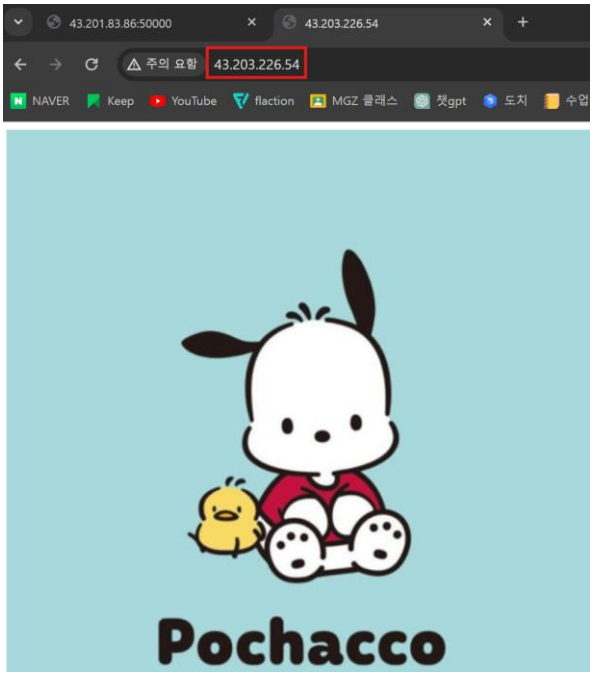


5) 인터넷 게이트웨이 생성



9. 웹 접근하기

- public ip 로 웹 접근 성공함



[출처]

동진님, 대수님 감사합니다

https://inpa.tistory.com/entry/AWS-%F0%9F%93%9A-VPC-End-Point-%EA%B0%9C%EB%85%90-%EC%9B%90%EB%A6%AC-%EA%B5%AC%EC%B6%95-%EC%84%B8%ED%8C%85