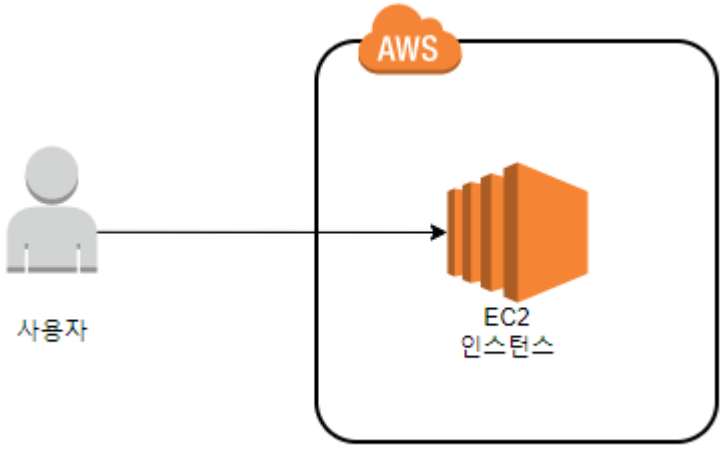
단일 웹서버 배포

단일 웹 서버 배포하기




```
#!/bin/bash
echo "Hello, World" > index.html
# busybox로 8080포트에서 웹서버를 백그라운드에서 실행한다.
nohup busybox httpd -f -p 8080 &
```

이 배시 스크립트는 index.html 파일에 ‘Hello, World’라는 텍스트를 저장한 다음, 비지박스라는 도구로 포트 8080에서 웹 서버를 실행하여 해당 파일을 제공한다. Busybox 명령을 nohub과 &로 래핑하여 배시 스크립트가 종료되더라도 웹 서버가 백그라운드에서 실행되도록 했다.

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_instance" "example" {
  ami           = "ami-06eea3cd85e2db8ce"
  instance_type = "t2.micro"
```

```
# user_data 인수를 설정하여 쉘스크립트를 사용자 데이터로 전달한다.
# 테라폼의 히어닥 구문을 이용해 줄바꿈 문자를 삽입하지 않고도 여러줄로 된 코드를 작성한다.
  user_data = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p 8080 &
               EOF
  # tags 추가
  tags = {
    Name = "example"
  }
}
```

- 사용자 데이터(user_data) 설정을 통해 쉘 스크립트를 인스턴스 기동시 수행 할 수 있게 했다.
- 쉘 스크립트는 index.html 파일에 ‘Hello, World’를 작성하고 busybox를 수행하여 8080 포트로 웹 서버를 시작하는 명령이다.
- busybox 명령어와 nohup과 &를 추가하여 bash 스크립트가 종료되더라도 백그라운드로 계속 수행할 수 있도록 설정했다.
- <<-EOF 와 EOF 표시는 새로운 줄에 문자를 매번 추가하는 것이 아니라 여러줄의 단락으로 처리하는 테라폼의 히어닥(heredoc) 문법이다.

EC2 인스턴스가 8080 포트에서 트래픽을 수신하도록 하려면 보안그룹을 추가해야 한다.

```
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_instance" "example" {
  ami           = "ami-06eea3cd85e2db8ce"
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p 8080 &
              EOF
  # tags 추가
  tags = {
    Name = "example"
  }
}

resource "aws_security_group" "instance" {
  name = "terrafrom-example-instance"

  ingress {
    from_port    = 8080
    to_port      = 8080
    protocol     = "tcp"
    cidr_blocks   = ["0.0.0.0/0"]
  }
}

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

이 코드는 aws_security_group 리소스를 생성하고 CIDR 블록 0.0.0.0/0으로부터 8080 포트에 대해 TCP 요청을 받을 수 있도록 설정하였다.

EC2 인스턴스가 실제로 이 보안그룹을 사용할 수 있게 하려면 보안 그룹의 ID를 aws_instane 리소스의 vpc_security_group_ids 인수에 저장되어야 한다.

보안 그룹의 ID를 채움 참조(interpolation) 구문으로 변수 처리하였다.

테라폼에서는 모든 리소스를 속성값으로 불러 변수를 사용할 수 있다.

참조(reference)는 코드의 다른 부분에서 값에 액세스할 수 있게 해주는 표현식이다. 보안 그룹의 ID에 액세스하려면 다음 구문을 사용하여 리소스 속성값을 참조한다.

<PROVIDER>_<TYPE>.<NAME>.<ATTRIBUTE>

- PROVIDER : ‘aws’와 같은 공급자의 이름
- TYPE : ‘security_group’과 같은 리소스의 유형
- NAME : ‘instance’와 같은 리소스의 이름
- ATTRIBUTE : ‘name’과 같은 리소스의 인수 중 하나이거나 리소스가 내보낸 속성 중 하나

표현식 예 : aws_security_group.instance.id

aws_instance 리소스에서 해당 보안 그룹 ID를 vpc_security_group_ids 변수로 사용할 수 있다.

하나의 리소스에서 다른 리소스로 참조를 추가하면 내재된 종속성이 작성된다. 테라폼은 이러한 종속성 구문을 분석하여 종속성 그래프를 작성하고, 이를 사용하여 리소스를 생성하는 순서를 자동으로 결정한다.

예를 들어 EC2 인스턴스가 보안 그룹의 ID를 참조하므로 코드를 처음 배포하는 경우 테라폼은 EC2 인스턴스를 생성하기 전에 먼저 보안 그룹을 만들어야 한다고 판단한다. graph 명령을 실행하여 테라폼이 종속성 그래프를 표시하도록 할 수 있다.

```
$ terraform graph                                                                    [1:36:02]
digraph {
        compound = "true"
        newrank = "true"
        subgraph "root" {
                "[root] aws_instance.example (expand)"
                  [label = "aws_instance.example", shape = "box"]
                "[root] aws_security_group.instance (expand)"
                  [label = "aws_security_group.instance", shape = "box"]
                "[root] provider[\"registry.terraform.io/hashicorp/aws\"]"
                  [label = "provider[\"registry.terraform.io/hashicorp/aws\"]",
                   shape = "diamond"]
                "[root] aws_instance.example (expand)" ->
                  "[root] aws_security_group.instance (expand)"
                "[root] aws_security_group.instance (expand)" ->
                  "[root] provider[\"registry.terraform.io/hashicorp/aws\"]"
                "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)" ->
```

```
                  "[root] aws_instance.example (expand)"
                "[root] root" ->
                 "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)"
        }
}
```

결괏값은 DOT라는 그래프 설명 언어로 되어있다.

테라폼은 종속성 트리를 따라갈 때 가능한 많은 리소스를 병렬로 생성하므로 변경 사항을 매우 효율적으로 적용할 수 있다.

apply 명령을 실행하면 테라폼이 보안 그룹을 생성하고 EC2 인스턴스를 새로운 사용자 데이터가 있는 새 인스턴스로 교체한다.

```
$ tf apply                                                                                                  [1:35:35]
...

Terraform will perform the following actions:

  # aws_instance.example will be updated in-place
  ~ resource "aws_instance" "example" {
        id                                   = "i-07db7da88fe26dea2"
      ~ tags                                 = {
          ~ "Name" = "example" -> "terraform-example"
        }
      ~ tags_all                             = {
          ~ "Name" = "example" -> "terraform-example"
        }
      ~ vpc_security_group_ids               = [
          - "sg-0812605686ede7b2b",
        ] -> (known after apply)
         (...)
    }

  # aws_security_group.instance will be created
  + resource "aws_security_group" "instance" {
      + arn                    = (known after apply)
      + description            = "Managed by Terraform"
      + egress                 = (known after apply)
      + id                     = (known after apply)
      + ingress                = [
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + description      = ""
              + from_port        = 8080
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "tcp"
              + security_groups  = []
              + self             = false
              + to_port          = 8080
            },
        ]
      + name                   = "terrafrom-example-instance"
      + name_prefix            = (known after apply)
      + owner_id               = (known after apply)
      + revoke_rules_on_delete = false
      + tags_all               = (known after apply)
      + vpc_id                 = (known after apply)
    }

Plan: 1 to add, 1 to change, 0 to destroy.

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

(...)
Apply complete! Resources: 1 added, 1 changed, 0 destroyed.
```

```
$ curl http://<EC2_INSTANCE_PUBLIC_IP>:8080
Hello, World
```