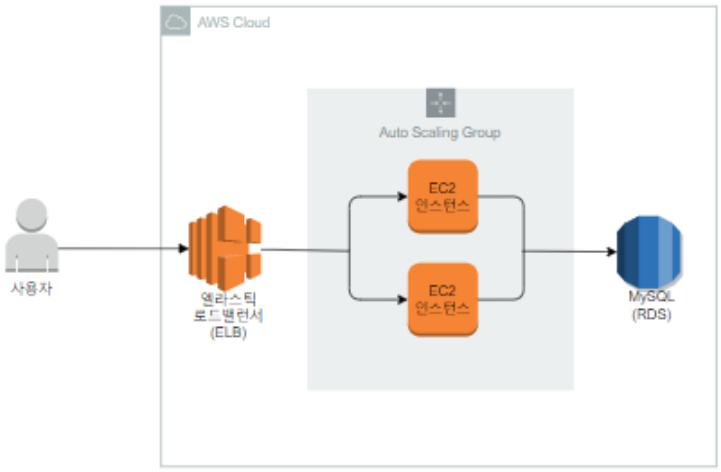
테라폼 모듈

앞에서 구축한 아키텍처는 아래 그림과 같다.



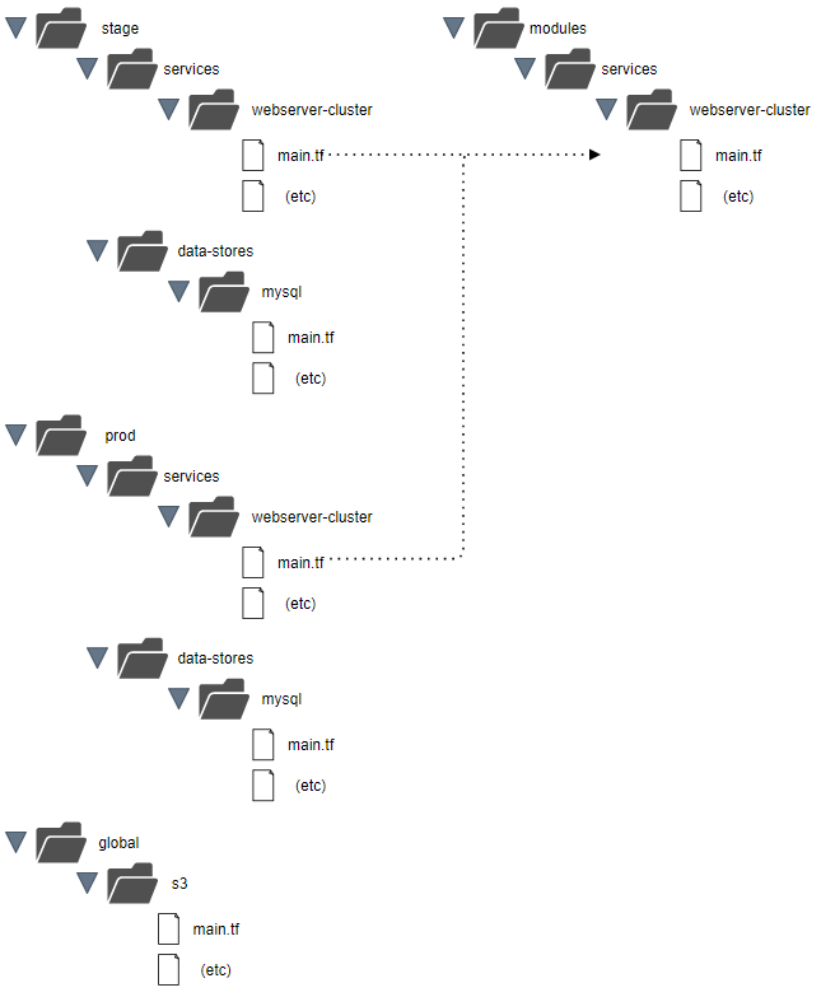
이 아키텍처는 하나의 환경에서는 잘 작동하지만 우리는 일반적으로 둘 이상의 환경이 필요하다. 하나는 팀의 내부 테스트(스테이징)을 위한 환경이고 다른 하나는 실제 사용자가 액세스(프로덕션)하기 위한 환경이다. 이론상 두 환경은 거의 동일하지만 비용을 절약하기 위해 스테이징 환경에서 좀 더 적은 수의 서버 또는 더 작은 서버로 테스트 할 수 있다.

루비와 같은 범용 프로그래밍 언어에서 동일한 코드를 여러 곳에 복사하여 붙여 넣으면 그 코드를 함수 안에 넣어 어디에서나 재사용할 수 있다.

```
def example_function()
  puts "Hello, world"
end

# 다른 코드에서 example_function 함수 이용
example_function()
```

테라폼을 사용하면 코드를 테라폼 모듈에 넣고 전체 코드의 여러 위치에서 해당 모듈을 재사용할 수 있다. 스테이징(테스트) 및 프로덕션(사용자 액세스) 환경에서 동일한 코드를 복사하여 사용하는 대신 아래 그림처럼 두 환경에서 동일한 모듈의 코드를 재사용할 수 있다.



모듈은 재사용할 수 있고 유지 관리할 수 있으며 테스트할 수 있는 테라폼 코드를 작성하는 데 있어 핵심 요소이다.

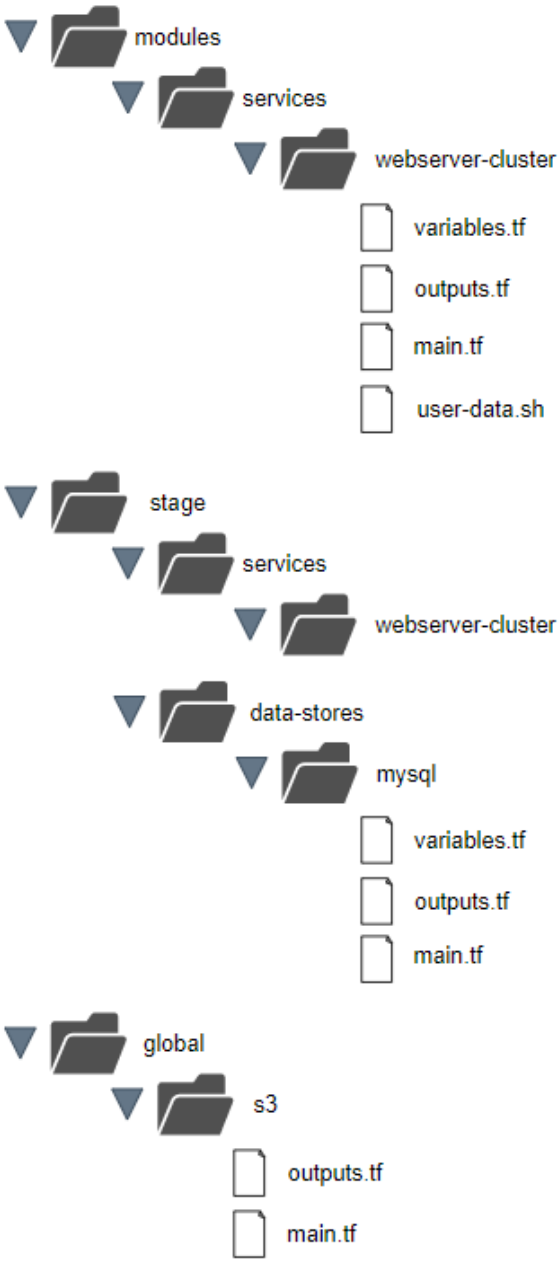
1. 모듈의 기본

폴더에 있는 모든 테라폼 구성 파일은 모듈이다.

예를 들어 오토스케일링 그룹(ASG), 애플리케이션 로드 밸런서(ALB), 보안 그룹 및 기타 여러 리소스가 포함된 stage/services/webserver-cluster의 코드를 재사용 가능한 모듈로 바꾸어 보자.

첫 번째로 modules 라는 최상위 폴더를 생성하고 모든 파일을 stage/services/webserver-cluster에서 modules/stage/services/webserver-cluster로 이동한다. 이 과정은 스테이징 환경과 프로덕션 환경이 참조하기 위한 modules 디렉토리를 생성하는 것이다.

두 번째로 modules/stage/services/webserver-cluster에서 main.tf파일을 생성하고 provider 정의를 제거한다. 공급자는 기존의 모듈 자체가 아닌 사용자가 정의한 모듈로 정의해야 한다.



이제 스테이징 환경에서 이 모듈을 사용할 수 있다. 모듈을 사용하기 위한 구문은 다음과 같다.

```
module "<NAME>" {
  source = "<SOURCE>"

  [CONFIG ...]
}
```

NAME : 테라폼 코드 전체에서 web-service와 같은 모듈을 참조하기 위해 사용할 수 있는 식별자

SOURCE : modules/services/webserver-cluster 같은 모듈 코드를 찾을 수 있는 경로

CONFIG : 모듈과 관련된 특정한 하나 이상의 인수

예를 들어 stage/services/webserver-cluster/main.tf에 새 파일을 만들고 다음과 같이 webserver-cluster 모듈을 사용할 수 있다.

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

module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"
}
```

그리고 다음과 같이 prod/services/webserver-cluster/main.tf에 새 파일을 만들어 동일한 모듈을 프로덕션 환경에서 재사용할 수 있다.

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

module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"
}
```

코드 복사 및 붙여넣기를 최소화하여 필요한 여러 환경에서 코드를 재사용할 수 있다. 그러나 테라폼 구성에 모듈을 추가하거나 모듈의 source 매개 변수를 수정할 때마다 apply나 plan 명령어를 사용하기 전에 init 명령을 실행해야 한다.

이 코드에서 apply 명령을 실행하기 전에 webserver-cluster 모듈에 문제가 있음을 알아야 한다. 모든 이름은 하드 코딩되어 있다. 즉, 보안 그룹 이름, ALB 및 기타 리소스가 모드 하드 코딩되어 있으므로 이 모듈을 두 번 이상 사용하면 이름 충돌 오류가 발생한다. module/services/webserver-cluster에 복사한 main.tf파일이 terraform_remote_state 데이터 소스를 사용하여 데이터베이스 주소와 포트를 파악하기 때문에 데이터베이스 세부 사항도 하드 코딩되어 있다. terraform_remote_state가 스테이징 환경을 살펴보기 위해 하드 코딩되었기 때문에 데이터베이스 세부 정보 역시 하드 코딩되어 있다.

이 문제를 해결하려면 webserver-cluster 모듈에 구성 가능한 입력을 추가하여 다른 환경에서는 다르게 작동할 수 있도록 해야한다.

2. 모듈 입력

module/services/webserver-cluster/variables.tf를 열고 새로운 입력 변수 3개를 추가한다.

```
variable "cluster_name" {
  description = "The name to use to namespace all the resources in the cluster"
  type        = string
  default     = "webservers-stage"
}

variable "db_remote_state_bucket" {
  description = "The name of the S3 bucket used for the database's remote state storage"
  type        = string
}

variable "db_remote_state_key" {
  description = "The name of the key in the S3 bucket used for the database's remote state storage"
  type        = string
}
```

두 번째로 module/services/webserver-cluster/main.tf를 통해 하드 코딩된 이름 대신 var.cluster_name을 사용한다. 다음은 ALB 보안 그룹에서 var.cluster_name을 사용하는 방법이다.

```
resource "aws_security_group" "alb" {
  name = "${var.cluster_name}-alb"

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

다른 aws_security_group 리소스, aws_alb 리소스 및 aws_autoscaling_group 리소스의 tag 섹션도 비슷하게 변경한다.

반드시 올바른 환경의 상태 파일을 읽게 하기 위해 bucket과 key 매개 변수에 각각 db_remote_state_bucket과 db_remote_state_key를 사용하도록 terraform_remote_state 데이터 소스를 업데이트해야 한다.

```
data "terraform_remote_state" "db" {
  backend = "s3"

  config = {
    bucket = var.db_remote_state_bucket
    key    = var.db_remote_state_key
    region = "ap-northeast-2"
  }
}
```

이제 스테이징 환경의 stage/services/webserver-cluster/main.tf에서 다음과 같이 새로운 입력 변수들을 설정할 수 있다.

```
module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"

  cluster_name           = "webservers-stage"
  db_remote_state_bucket = "<본인의 버킷 이름>"
  db_remote_state_key    = "stage/data-stores/mysql/terraform.tfstate"
}
```

프로덕션 환경의 prod/services/webserver-cluster/main.tf에도 동일한 작업을 수행한다.

```
module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"

  cluster_name           = "webservers-prod"
  db_remote_state_bucket = "<본인의 버킷 이름>"
  db_remote_state_key    = "prod/data-stores/mysql/terraform.tfstate"
}
```

리소스에 인수를 설정하는 것과 같은 구문을 사용해 모듈의 입력 변수를 설정한다. 입력 변수는 모듈의 API이며 다른 환경에서 작동하는 방식을 제어한다. 이 예제는 여러 환경에서 서로 다른 변수 이름을 사용하지만 각 환경에서 매개 변수를 구성 가능하도록 할 수 있다. 예를 들어 스테이징 환경에서는 비용을 절약하기 위해 적은 수의 웹 서버 클러스터를 실행하옥 프로덕션 환경에서는 많은 트래픽을 처리하기 위해 더 큰 클러스터를 실행할 수 잇다. 이를 위해 module/services/webserver-cluster/variables.tf에 입력 변수 3개를 추가해야 한다.

```
variable "instance_type" {
  description = "The type of EC2 Instances to run (e.g. t2.micro)"
  type        = string
}

variable "min_size" {
  description = "The minimum number of EC2 Instances in the ASG"
  type        = number
```

```
}

variable "max_size" {
  description = "The maximum number of EC2 Instances in the ASG"
  type        = number
}
```

다음으로 module/services/webserver-cluster/main.tf에서 시작 템플릿을 업데이트하여 instance_type 매개 변수를 새 var.instance_type 입력 변수로 설정한다.

```
resource "aws_launch_template" "example" {
  name                   = "aws00-example"
  image_id               = "ami-06eea3cd85e2db8ce"
  instance_type          = var.instance_type
  key_name               = "aws00-key"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = "${base64encode(data.template_file.db_output.rendered)}"

  # Required when using a launch configuration with an auto scaling group.
  lifecycle {
    create_before_destroy = true
  }
}
```

마찬가지로 동일한 파일에서 ASG 정의를 업데이트하여 min_size 및 max_size 매개 변수를 각각 새로운 var.min_size 및 var.max_size 입력 변수로 설정해야 한다.

```
resource "aws_autoscaling_group" "example" {
  availability_zones = ["ap-northeast-2a", "ap-northeast-2c"]

  name               = var.alb_name
  desired_capacity   = 1
  min_size           = var.min_size
  max_size           = var.max_size

  target_group_arns  = [aws_lb_target_group.asg.arn]
  health_check_type  = "ELB"

  launch_template {
    id      = aws_launch_template.example.id
    version = "$Latest"
  }

  tag {
    key                 = "Name"
    value               = var.cluster_name
    propagate_at_launch = true
  }
}
```

이제 스테이징 환경(stage/services/webserver-cluster/main.tf)에서 instance_type을 t2.micro로, min_size와 max_size를 2로 설정하여 클러스터를 작고 저렴하게 유지하도록 설정한다.

```
module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"

  cluster_name           = var.cluster_name
  db_remote_state_bucket = var.db_remote_state_bucket
  db_remote_state_key    = var.db_remote_state_key

  instance_type = "t2.micro"
  min_size      = 2
  max_size      = 2
}
```

반면에 프로덕션 환경에서는 m4.large와 같이 더 많은 CPU, 메모리, 그리고 더 큰 instance_type을 사용할 수 있다. 이 인스턴스 유형은 AWS 프리티어에 포함되지 않으니 유의한다.

```
module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"

  cluster_name           = var.cluster_name
  db_remote_state_bucket = var.db_remote_state_bucket
  db_remote_state_key    = var.db_remote_state_key

  instance_type = "m4.large"
  min_size      = 2
  max_size      = 10
}
```

3. 모듈과 지역 변수

보안 그룹의 모든 IP를 입력 변수를 사용하는 대신 이러한 값을 modules/services/webserver-cluster/main.tf의 locals 블록에서 로컬 값으로 정의한다.

```
locals {
  http_port    = 80
  any_port     = 0
  any_protocol = "-1"
  tcp_protocol = "tcp"
  all_ips      = ["0.0.0.0/0"]
}
```

로컬 값을 사용하면 모든 테라폼 표현식에 이름을 할당하고 모듈 전체에서 해당 이름을 사용할 수 있다. 이러한 이름은 모듈 내에서만 표시되므로 다른 모듈에는 영향을 미치지 않으며, 모듈 외부에서 이 값을 재정의할 수 없다. 로컬 값을 읽으려면 다음 구문으로 된 로컬 참조를 사용해야 한다.

```
local.<NAME>
```

이 구문을 사용하여 로드 밸런서 리스너를 업데이트한다.

```
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.example.arn
  port              = local.http_port
  protocol          = "HTTP"

  # 기본값으로 단순한 404 페이지 에러를 보여준다.
  default_action {
    type = "fixed-response"

    fixed_response {
      content_type = "text/plain"
      message_body = "404: page not found"
      status_code  = 404
    }
  }
```

로드 밸런서의 보안 그룹을 포함해 모듈의 모든 보안 그룹에도 적용한다.

```
resource "aws_security_group" "alb" {
  name = "${var.cluster_name}-alb"

  ingress {
    from_port   = local.http_port
    to_port     = local.http_port
    protocol    = local.tcp_protocol
    cidr_blocks = local.all_ips
  }

  egress {
    from_port   = local.any_port
    to_port     = local.any_port
    protocol    = local.any_protocol
    cidr_blocks = local.all_ips
  }
}
```

4. 모듈 출력

webserver-cluster 모듈에 예약된 작업을 정의하면 스테이징 환경과 프로덕션 환경 모두에 적용된다. 스테이징 환경에서는 이러한 확장 작업이 필요 없기 때문에 당분간 프로덕션 구성에서 직접 자동 확장 일정을 정의할 수 있다.

예약된 작업을 정의하려면 2개의 aws_autoscaling_schedule 리소스를 prod/services/webserver-cluster/main.tf에 추가한다.

```
resource "aws_autoscaling_schedule" "scale_out_during_business_hours" {
  scheduled_action_name = "scale-out-during-business-hours"
  min_size              = 2
  max_size              = 10
  desired_capacity      = 10
  recurrence            = "0 9 * * *"
}

resource "aws_autoscaling_schedule" "scale_in_at_night" {
  scheduled_action_name = "scale-in-at-night"
  min_size              = 2
  max_size              = 10
  desired_capacity      = 2
  recurrence            = "0 17 * * *"
}
```

이 코드는 우선 첫 번째 aws_autoscaling_schedule 리소스를 사용하여 오전 시간 동안 서버 수를 10으로 늘리고 - recurrence(반복) 매개 변수는 크론 구문을 사용하므로 0 9 * * * 는 매일 9시를 의미한다. -, 두번째 aws_autoscaling_schedule 리소스로는 밤에 서버 수를 2로 줄인다. 그러나 이 두번의 aws_autoscaling_schedule 리소스에는 ASG의 이름을 지정하는 autoscaling_group_name이라는 필수 매개 변수가 누락되어 있다.

테라폼에서는 모듈 역시 값을 반환할 수 있다. 다음과 같이 modules/services/webserver-cluster/output.tf에서 ASG 이름을 출력 변수로 추가할 수 있다.

```
output "asg_name" {
  value       = aws_autoscaling_group.example.name
  description = "The name of the Auto Scaling Group"
}
```

다음 구문을 사용하여 모듈 출력 변수에 액세스할 수 있다.

```
module.<MODULE_NAME>.<OUTPUT_NAME>
```

예를 들어 구문을 다음과 같이 사용한다.

```
module.frontend.asg_name
```

prod/services/webserver-cluster/main.tf에서 이 구문을 사용하여 각 aws_autoscaling_schedule 리소스에서 autoscaling_group_name 매개 변수를 설정할 수 있다.

```
resource "aws_autoscaling_schedule" "scale_out_during_business_hours" {
  scheduled_action_name = "scale-out-during-business-hours"
  min_size              = 2
  max_size              = 10
  desired_capacity      = 10
  recurrence            = "0 9 * * *"

  autoscaling_group_name = module.webserver_cluster.asg_name
}

resource "aws_autoscaling_schedule" "scale_in_at_night" {
  scheduled_action_name = "scale-in-at-night"
  min_size              = 2
  max_size              = 10
  desired_capacity      = 2
  recurrence            = "0 17 * * *"

  autoscaling_group_name = module.webserver_cluster.asg_name
}
```

클러스터가 배포될 때 테스트할 URL을 알 수 있도록 webserver-cluster 모듈에 다른 출력인 ALB의 DNS 이름을 노출할 수 있다. 이를 위해 module/services/webserver-cluster/outputs.tf에 출력 변수를 다시 추가한다.

```
output "alb_dns_name" {
  value       = aws_lb.example.dns_name
  description = "The domain name of the load balancer"
}
```

그러면 다음과 같이 stage/services/webserver-cluster/outputs.tf 및 prod/services/webserver-cluster/outputs.tf에서 이 출력값을 통과할 수 있다.

```
output "alb_dns_name" {
  value       = module.webserver_cluster.alb_dns_name
  description = "The domain name of the load balancer"
}
```

5. 모듈 주의 사항

모듈을 만들 때는 다음과 같은 사항을 주의해야 한다.

- 파일 경로
- 인라인 블록

5.1 파일 경로

앞에서 웹 서버 클러스터의 사용자 데이터 스크립트를 외부 파일인 user-data.sh로 이동하고, 내장 함수 file을 사용하여 디스크에서 파일을 읽을 수 있었다. file 함수를 사용할 때 파일 경로가 상대 경로여야 한다.

기본적으로 테라폼은 현재 작업 중인 디렉토리를 기준으로 경로를 해석한다. terraform apply를 실행하는 디렉토리와 동일한 디렉토리에 있는 테라폼 구성 파일에서 file 함수를 사용하는 것, 즉 루트 모듈에서 file 함수를 사용하는 것은 가능하지만, 별도의 폴더에 정의된 모듈에서 file함수를 사용할 수 는 없다.

이 문제를 해결하기 위해 path.<TYPE>형태의 경로 참조 표현식을 사용할 수 있다. 테라폼은 다음 유형의 경로 참조를 지원한다.

- **path.module** : 표현식이 정의된 모듈의 파일 시스템 경로를 반환한다.
- **path.root** : 루트 모듈의 파일 시스템 경로를 반환한다.
- **path.cwd** : 현재 작업 중인 디렉토리의 파일 시스템 경로를 반환한다. 테라폼을 일반적으로 사용할 때 이것은 path.root와 동일하지만 테라폼의 일부 기능은 루트 모듈 디렉토리 이외의 디렉토리에서 작동하므로 경로가 달라진다.

사용자 데이터 스크립트이 경우 모듈 자체에 상대 경로가 필요하므로 modules/services/webserver-cluster/main.tf의 terraform_file 데이터 소스에서 path.module을 사용해야 한다.

```
data "template_file" "db_output" {
  template = file("${path.module}/user-data.sh")
  vars = {
    server_port = var.server_port
    db_address  = data.terraform_remote_state.db.outputs.address
    db_port     = data.terraform_remote_state.db.outputs.port
  }
}
```

5.2 인라인 블록

일부 테라폼 리소스의 구성은 인라인 블록 또는 별도의 리소스로 정의 할 수 있다. 모듈을 만들 때는 항상 별도의 리소스를 사용하는 것이 좋다.

예를 들어, aws_security_group 리소스를 사용하면 webserver-cluster 모듈 (modules/services/webserver-cluster/main.tf)에서 볼 수 있듯 인라인 블록을 통해 수신(ingress) 및

송신(egress) 규칙을 정의할 수 있다.

별도의 aws_security_group_rule 리소스를 사용하여 정확히 동일한 수신 및 송신 규칙을 정의하도록 이 모듈을 변경해야 한다. 모듈의 두 보안 그룹 모두에 이 작업을 수행해야 한다.

```
resource "aws_security_group" "alb" {
  name = "${var.cluster_name}-alb"
}

resource "aws_security_group_rule" "allow_http_inbound" {
  type              = "ingress"
  security_group_id = aws_security_group.alb.id

  from_port   = local.http_port
  to_port     = local.http_port
  protocol    = local.tcp_protocol
  cidr_blocks = local.all_ips
}

resource "aws_security_group_rule" "allow_all_outbound" {
  type              = "egress"
  security_group_id = aws_security_group.alb.id

  from_port   = local.any_port
  to_port     = local.any_port
  protocol    = local.any_protocol
  cidr_blocks = local.all_ips
}
```

인라인 블록과 별도의 리소스를 혼합하여 사용하려고 하면 라우팅 규칙이 충돌하여 서로 덮어쓰는 오류가 발생한다. 따라서 둘 중 하나만 사용해야 한다. 이 제한 사항으로 인해 모듈을 작성할 때 항상 인라인 블록 대신 별도의 리소스를 사용해야 한다. 그렇지 않으면 모듈의 유연성이 떨어진다.

예를 들어 wevserver-cluster 모듈 내의 모든 수신 및 송신 규칙이 별도의 aws_security_group_rule 리소스로 정의된 경우 사용자가 모듈 외부에서 사용자 정의 규칙을 추가할 수 있도록 모듈을 유연하게 만들 수 있다. 이를 위해 aws_security_group의 ID를 modules/services/webserver-cluster/outpurs.tf에서 출력 변수로 내보낸다.

```
output "alb_security_group_id" {
  value       = aws_security_group.alb.id
  description = "The ID of the Security Group attached to the load balancer"
}
```

6. 모듈 버전 관리

스테이징 환경과 프로덕션 환경이 동일한 모듈 폴더를 가리키는 경우 해당 폴더를 변경하면 바로 다음 배포 시 두 환경 모두에 영향을 미친다. 이러한 종류의 결합은 프로덕션에 영향을 미치지 낳고 스테이징 변화를 테스트하기 어렵게 만든다.

버전 0.0.2 버전을 스테이징 환경에서 사용하고 0.0.1 버전은 프로덕션 환경에서 사용할 수 있도록 버전이 지정된 모듈을 만드는 것이 더 적절한 접근 방식이다.