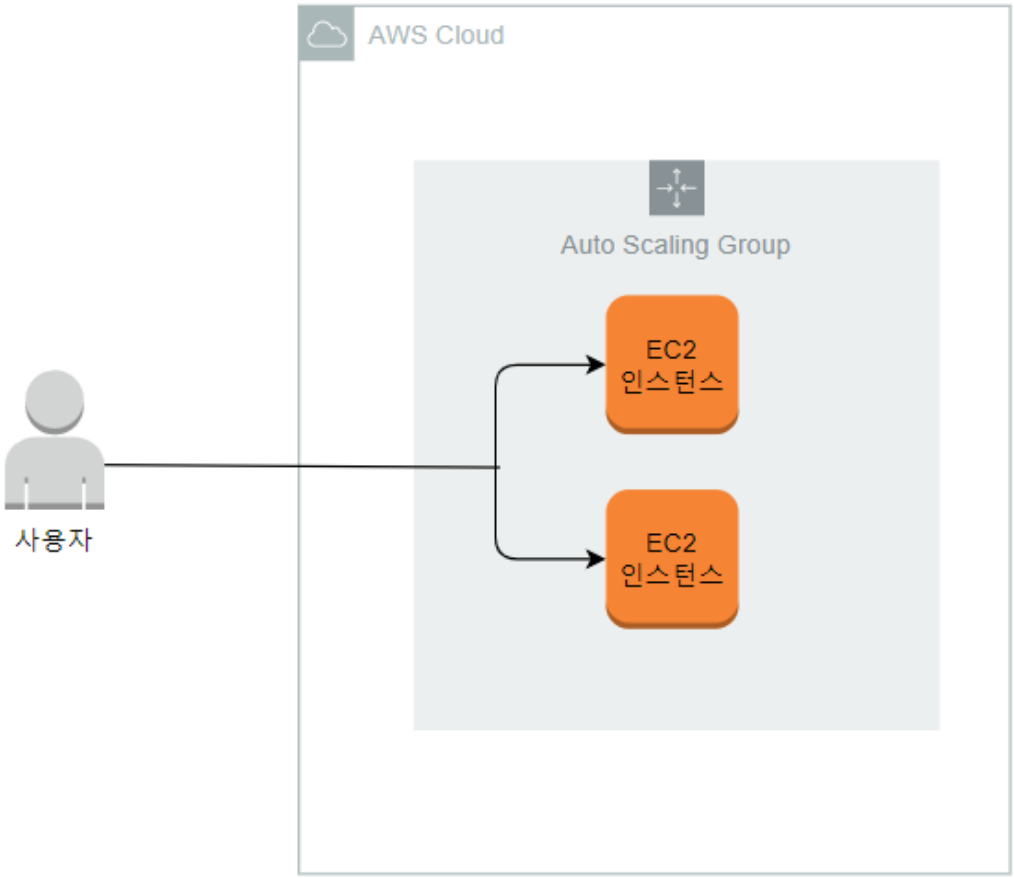
웹 서버 클러스터 배포

오토스케일링 그룹(ASG)를 만드는 첫 단계는 ASG에서 각 EC2인스턴스를 어떻게 구성할 것인지 설정하는 시작 구성을 만드는 것이다.



aws_launch_configuration 리소스는 aws_instance 리소스와 거의 동일한 매개 변수를 사용한다. (ami는 image_id로, vpc_security_group_ids는 security_groups로 두 리소스에서 사용하는 매개 변수의 이름이 다르다).

```
resource "aws_launch_configuration" "example" {
  image_id        = "ami-0ab04b3ccbadfae1f"
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instance.id]

  user_data = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p ${var.server_port} &
              EOF
}
```

이제 aws_autoscaling_group 리소스를 사용하여 ASG 자체를 생성할 수 있다.

```
resource "aws_autoscaling_group" "example" {
  launch_configuration = aws_launch_configuration.example.name

  min_size = 1
  max_size = 2

  tag {
    key                 = "Name"
    value               = "terraform-asg-example"
    propagate_at_launch = true
  }
}
```

이 ASG는 1~2개의 EC2 인스턴스(초기 시작의 경우 기본값은 1) 사이에서 실행되며 각각 terraform-asg-example 이라는 이름으로 태그가 지정된다.

tag의 propagate_at_launch = true 는 필수이고 이 ASG를 통해 시작된 Amazon EC2 인스턴스에 태그를 전파 할 수 있다.

ASG는 시작 구성 정보를 참고하여 인스턴스를 생성하는데 여기서 한 가지 문제가 발생한다. 시작 구성은 변경할 수 없으므로 시작 구성의 매개 변수를 변경하면 테라폼이 이를 대체하려고 한다. 일반적으로 리소스를 교체할 때 테라폼은 이전 리소스를 먼저 삭제한 다음 대체 리소스를 생성한다. 그러나 ASG에 이전 리소스에 대한 참조가 있기 때문에 테라폼이 해당 리소스를 삭제할 수 없다.

이 문제를 해결하기 위해 수명 주기 설정을 사용할 수 있다. 모든 테라폼 리소스는 리소스 생성, 업데이트 및 삭제 방법을 구성하는 몇 가지 수명 주기(lifecycle)설정을 지원한다.

특히 create_before_destroy는 유용하게 사용할 수 있는 수명 주기 설정이다.

create_before_destroy를 true로 설정하면 테라폼은 리소스를 교체하는 순서를 반대로 하여 교체 리소스를 먼저 생성하고(이전 리소스를 가리키고 있던 참조를 업데이트하여 교체한 리소스를 가리킴) 기존 리소스를 삭제한다.

```
resource "aws_launch_configuration" "example" {
  image_id        = "ami-0ab04b3ccbadfae1f"
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instance.id]

  user_data = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p ${var.server_port} &
              EOF

  # ASG에서 시작 구성을 사용할 때 필요하다.
  # https://www.terraform.io/docs/providers/aws/r/launch_configuration.html
  lifecycle {
    create_before_destroy = true
  }
}
```

ASG에 추가해야 작동하는 subnet_ids 매개 변수도 있다. subnet_ids는 ASG가 EC2를 어느 VPC의 서브넷에 배포할지 지정하는 매개 변수이다. 각 서브넷은 AWS AZ에 있으므로 인스턴스를 여러 서브넷에 배포하면 일부 데이터 센터가 중단된 경우에도 서비스를 계속 실행할 수 있다.

데이터 소스는 테라폼을 실행할 때마다 공급자(AWS)에서 가져온 읽기 전용 정보를 나타낸다. 테라폼 구성에 데이터 소스를 추가해도 새로운 것이 생성되지 않는다. 이는 단순히 데이터 공급자의 API만 물어보고 해당 데이터를 나머지 테라폼 코드에서 사용할 수 있도록 하는 방법이다.

각 테라폼 공급자는 다양한 데이터 소스를 노출한다. 예를 들어 AWS 공급자에는 VPC 데이터, 서브넷 데이터, AMI ID, IP 주소 범위, 현재 사용자의 자격 증명 등을 조회하는 데이터 소스가 포함되어 있다.

```
data "<PROVIDER>_<TYPE>" "<NAME>" {
  [CONFIG...]
}
```

PROVIDER : 공급자 이름

TYPE : vpc 같은 사용하려는 데이터 소스의 유형

NAME : 테라폼 코드에서 이 데이터 소스를 참조하는 데 사용할 수 있는 식별자

CONFIG : 해당 데이터 소스에 고유한 하나 이상의 인수

```
data "aws_vpc" "default" {
  default = true
}
```

데이터 소스에서 전달하는 인수는 일반적으로 원하는 정보를 데이터 소스에 표시하는 검색 필터이다. aws_vpc 데이터 소스에 필요한 유일한 필터는 default = true이며, 테라폼이 aws 계정에서 기본 VPC를 찾도록 지시한다.

데이터 소스에서 데이터를 가져오려면 다음과 같은 속성 참조 구문을 사용한다.

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

예를 들어, aws_vpc 데이터 소스에서 VPC의 ID를 얻으려면 속성 참조 구문을 다음과 같이 사용한다.

```
data.aws_vpc.default.id
```

이 데이터를 다른 데이터 소스인 aws_subnet_ids와 결합하여 해당 VPC 내 서브넷을 조회할 수 있다.

```
data "aws_subnet_ids" "default" {
  vpc_id = data.aws_vpc.default.id
}
```

마지막으로 vpc_zone_identifier 인수를 이용해 aws_subnet_ids 데이터 소스에서 서브넷 ID를 가져와서 ASG가 이 서브넷을 사용하도록 지시할 수 있다.

```
resource "aws_autoscaling_group" "example" {
  launch_configuration = aws_launch_configuration.example.name
  vpc_zone_identifier  = data.aws_subnets.default.ids

  min_size = 1
  max_size = 2

  tag {
    key                 = "Name"
    value               = "terraform-asg-example"
    propagate_at_launch = true
  }
}
```

완성된 main.tf

```
terraform {
  # 테라폼 버전 지정
  required_version = ">= 1.0.0, < 2.0.0"
  # 공급자 버전 지정
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}
provider "aws" {
  region = "ap-northeast-2"
}
# 시작 구성 설정
resource "aws_launch_configuration" "example" {
  image_id = "ami-06eea3cd85e2db8ce" # Ubuntu 20.04 version
  instance_type = "t2.micro"
  security_groups = [aws_security_group.instance.id]
  user_data = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p ${var.server_port} &
              EOF
  lifecycle {
    create_before_destroy = true
  }
}
# 오토스케일링 생성
resource "aws_autoscaling_group" "example" {
```

```
  launch_configuration = aws_launch_configuration.example.name
  vpc_zone_identifier = data.aws_subnets.default.ids
  min_size = 1
  max_size = 2
  tag {
    key = "Name"
    value = "aws00-terraform-asg-example"
    propagate_at_launch = true
  }
}
# 보안그룹 생성
resource "aws_security_group" "instance" {
  name = "aws00-terrafrom-example-instance"
  # 인바운드 규칙 설정
  ingress {
    from_port = var.server_port # 출발 포트
    to_port = var.server_port # 도착 포트
   i protocol = "tcp" # 프로토콜
    cidr_blocks = ["0.0.0.0/0"] # 송신지
  }
}
# Default VPC 정보 가지고 오기
data "aws_vpc" "default" {
  default = true
}
# Subnet 정보 가지고 오기
data "aws_subnets" "default" {
  filter {
    name = "vpc-id"
    values = [data.aws_vpc.default.id]
  }
}
```

variables.tf

```
variable "server_port" {
  description = "The port the will use for HTTP requests"
  type        = number
  default     = 8080
}
```

시작템플릿을 이용한 Auto Scaling Group 생성

시작템플릿 구성

```
resource "aws_launch_template" "example" {
  name                   = "example"
  image_id               = "ami-0ab04b3ccbadfae1f"
  instance_type          = "t2.micro"
  key_name               = "aws00-key"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = filebase64("${path.module}/web.sh")

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

오토스케일링

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

  desired_capacity   = 1
  min_size           = 1
  max_size           = 2

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

  tag {
    key                 = "Name"
    value               = "terraform-asg-example"
    propagate_at_launch = true
  }
}
```

완성된 main.tf

```
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_launch_template" "example" {
  name                   = "example"
  image_id               = "ami-0ab04b3ccbadfae1f"
  instance_type          = "t2.micro"
  key_name               = "aws00-key"
  vpc_security_group_ids = [aws_security_group.instance.id]

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

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

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

  name               = "terraform-asg-example"
  desired_capacity   = 1
  min_size           = 1
  max_size           = 2

  launch_template {
    id      = aws_launch_template.example.id
    version = "$Latest"
  }
  tag {
    key                 = "Name"
    value               = "terraform-asg-example"
    propagate_at_launch = true
  }
}

# 보안 그룹
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"]
  }
}

data "aws_vpc" "default" {
  default = true
```

```
}

data "aws_subnets" "default" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default.id]
  }
}

data "template_file" "web_output" {
  template = file("${path.module}/web.sh")
  vars = {
    server_port = "${var.server_port}"
  }
}
```

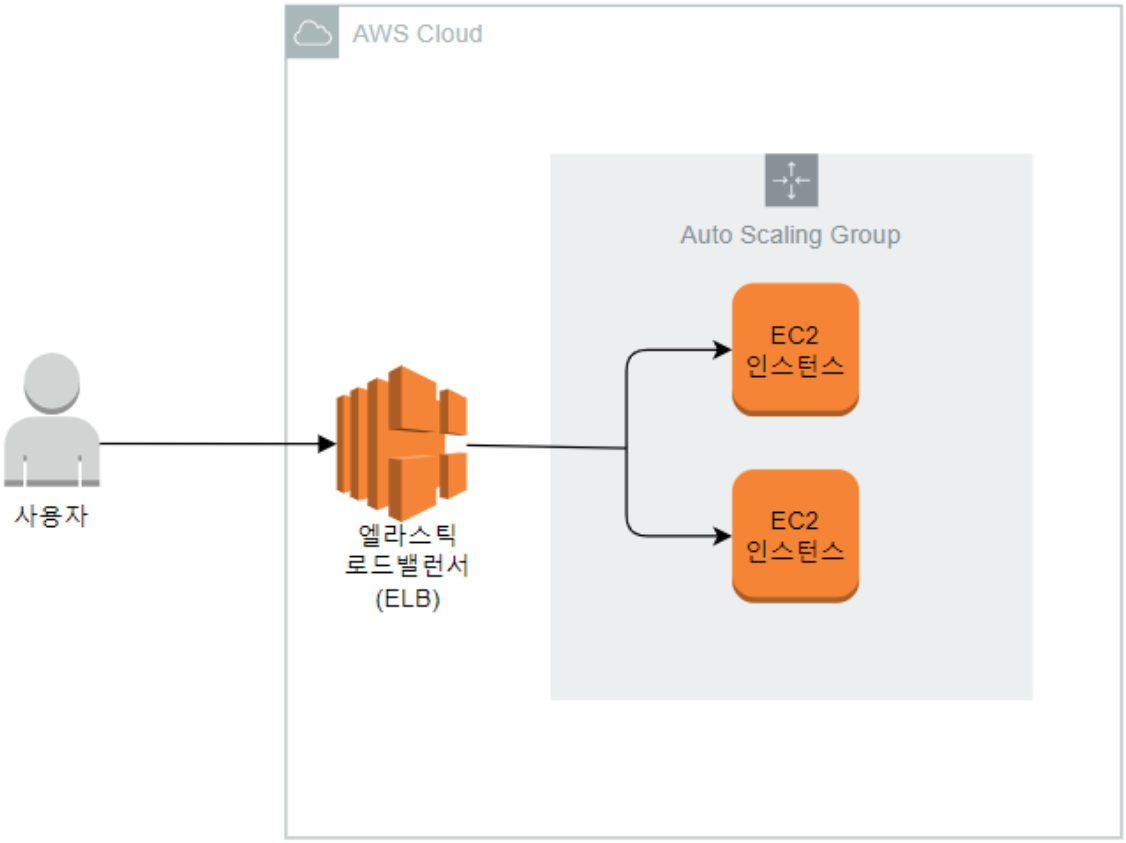
web.sh

```
#!/bin/bash
cat > index.html <<EOF
<H1>Hello, World</H1>
EOF

nohup busybox httpd -f -p ${server_port} &
```

variables.tf

```
variable "server_port" {
  description = "The port the will use for HTTP requests"
  type        = number
  default     = 8080
}

variable "security_group_name" {
  type    = string
  default = "aws00-terraform-example-instance"
}
```

로드 밸런서 배포



AWS는 3가지 유형의 로드밸런서를 제공한다.

- 애플리케이션 로드 밸런서(Application Load Balancer, ALB)

HTTP, HTTPS 트래픽 처리에 적합한 로드 밸런서이다. OSI 모델의 응용 계층에서 동작한다.

- 네트워크 로드 밸런서(Network Load Blancer, NLB)

TCP, UDP 및 TLS 트래픽 처리에 적합한 로드 밸런서이다. ALB보다 빠르게 로드에 응답하여 확장 및 축소할 수 있다. NLB는 초당 수천만 개의 요청을 처리할 수 있도록 설계되었다. OSI모델의 전송 계층에서 작동한다.

- 클래식 로드 밸런서(Classic Load Blancer, CLB)

클래식 로드 밸런서는 '레거시'로드 밸런서 이다. HTTP, HTTPS, TCP 및 TLS트래픽을 처리할 수 있지만 ALB 또는 NLB 보다 기능은 훨씬 적다. OSI 모델의 응용 계층 및 전송 계층에서 모두 작동한다.

ALB의 구성 요소

- 리스너(Listener)

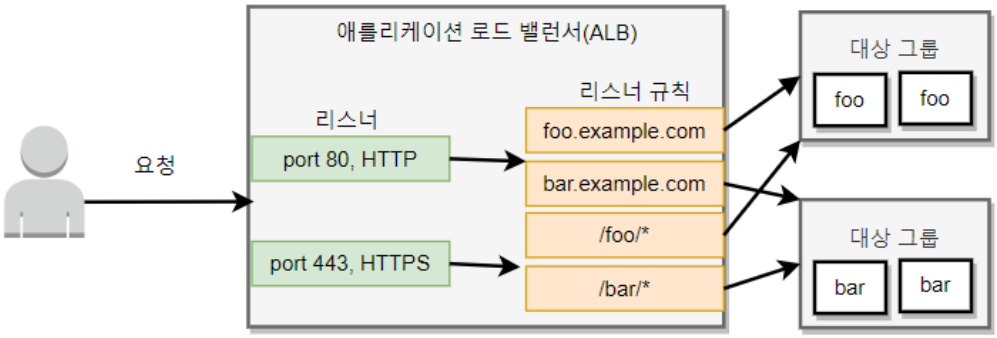
80 같은 특정 포트와 HTTP 같은 프로토콜을 수신한다.

- 리스너 규칙(Listener eule)

리스너에 들어오는 요청을 가져와 /foo 및 /bar 같은 특정 경로나 foo.example.com 및 bar.example.com 같은 호스트 이름과 일치하는 요청을 특정 대상 그룹으로 보낸다.

- 대상 그룹(Target groups)

로드 밸런서에서 요청을 받는 하나 이상의 서버이다. 대상 그룹은 서버의 상태를 확인하고 요청을 정상 노드로 보낸다.



ALB를 생성하는 첫 번째 단계는 aws_lb 리소스를 사용하여 ALB 자체를 작성하는 것이다.

```
resource "aws_lb" "example" {
  name               = "terraform-alb-example"
  load_balancer_type = "application"
  subnets            = data.aws_subnets.default.ids
}
```

subnets 매개 변수는 aws_subnets 데이터 소스를 사용하여 기본 VPC의 모든 서브넷을 사용하도록 로드 밸런서를 구성한다. AWS 로드 밸런서는 단일 서버가 아니라 별도의 서브넷에서 실행될 수 있는 여러 서버로 구성되어 있다. AWS는 트래픽에 따라 로드 밸런서 서버 수를 자동으로 확장 또는 축소하고 해당 서버 중 하나가 다운되면 장애 조치를 활성화하므로 확장성과 가용성을 즉시 얻을 수 있다.

ALB를 생성하는 두 번째 단계는 aws_lb_listener 리소스를 사용하여 ALB의 리스너를 정의하는 것이다.

```
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.example.arn
  port              = 80
  protocol          = "HTTP"

  # 기본값으로 단순한 404 페이지 오류를 반환한다.
  default_action {
    type = "fixed-response"

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

이 리스너는 기본 HTTP 포트인 80번 포트를 수신하고, HTTP를 프로토콜로 사용하고, 리스너 규칙과 일치하지 않는 요청에 대해 기본 응답으로 404 페이지를 보내도록 ALB를 구성한다.

기본적으로 ALB를 포함한 모든 AWS 리소스는 들어오는 트래픽 또는 나가는 트래픽을 허용하지 않으므로 ALB를 위해 특별히 새 보안 그룹을 생성해야 한다. 이 보안 그룹은 80번 포트에서 들어오는 요청을 허용하여 HTTP를 통해 로드 밸런서에 접속할 수 있게 한다. 그리고

바깥으로 나가는 요청은 포트와 상관없이 허용하여 로드 밸런서가 '상태 확인(Health check)'을 수행하도록 한다.

```
resource "aws_security_group" "alb" {
  name = "terraform-example-alb"

  # 인바운드 HTTP 트래픽 허용
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # 모든 아웃바운드 트래픽 허용
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

security_groups 인수를 통해 이 보안 그룹을 사용하려면 aws_lb 리소스에 지시해야 한다.

```
resource "aws_lb" "example" {
  name               = "terraform-asg-example"
  load_balancer_type = "application"
  subnets            = data.aws_subnets.default.ids
  security_groups    = [aws_security_group.alb.id]
}
```

다음으로 aws_lb_target_group 리소스를 사용하여 ASG의 대상 그룹을 생성해야 한다.

```
resource "aws_lb_target_group" "asg" {
  name     = "terraform-asg-example"
  port     = var.server_port
  protocol = "HTTP"
  vpc_id   = data.aws_vpc.default.id

  health_check {
    path                = "/"
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 15
    timeout             = 3
```

```
    healthy_threshold   = 2
    unhealthy_threshold = 2
  }
}
```

이 대상 그룹은 각 인스턴스에 주기적으로 HTTP 요청을 전송하여 인스턴스 상태를 점검하고, 구성된 matcher와 일치하는 응답을 반환하는 경우에만 인스턴스를 '정상(healthy)'으로 간주한다. 예를 들어 200 OK 응답을 찾도록 matcher를 구성할 수 있다. 인스턴스가 다운되었거나 오버로드되어 응답하지 않으면 '비정상(unhealthy)'으로 표시되고 대상 그룹은 사용자가 받는 지장을 최소화하기 위해 트래픽 전송을 자동으로 중지한다.

aws_lb_target_group_attacement 리소스를 사용하면 EC2 인스턴스의 정적 목록을 대상 그룹에 연결할 수 있다. 그러나 ASG를 사용하면 언제든지 인스턴스를 시작하거나 종료할 수 있으므로 정적 목록이 작동하지 않는다. 대신 ASG와 ALB를 통합하는 이점을 활용할 수 있다. aws_autoscaling_group 리소스로 돌아가서 target_group_arns 인수를 설정하여 새 대상 그룹을 지정한다.

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

  desired_capacity   = 1
  min_size           = 1
  max_size           = 2

  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               = "terraform-asg-example"
    propagate_at_launch = true
  }
}
```

health_check_type도 'ELB'로 설정한다. 기본 health_check_type은 'EC2"인데, 이는 AWS 하이퍼바이저가 아닌 VM이 완전히 다운되었거나 도달할 수 없다고 판단하는 경우에만 인스턴스가 비정상 상태라고 간주하는 최소한의 상태 확인이다. 반면에 'ELB' 상태 점검

은 ASG가 대상 그룹의 상태 확인을 하여 인스턴스가 정상인지 여부를 판별하고 대상 그룹이 상태 불량으로 보고되면 인스턴스를 자동으로 교체하도록 지시하기 때문에 더욱 강력하다. 이렇게 하면 인스턴스가 완전히 다운되었을 때뿐 아니라 메모리 부족으로 인해 요청 처리가 중단되거나 중요한 프로세스가 중단되는 경우에도 인스턴스가 교체된다.

마지막으로 aws_lb_listener_rule 리소스를 사용해 리스너 규칙을 생성하여 이 모든 부분을 연결해야 한다.

```
resource "aws_lb_listener_rule" "asg" {
  listener_arn = aws_lb_listener.http.arn
  priority     = 100

  condition {
    path_pattern {
      values = ["*"]
    }
  }

  action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.asg.arn
  }
}
```

위의 코드는 모든 경로와 일치하는 요청을 ASG가 포함된 대상 그룹으로 보내는 리스너 규칙을 추가한다.

이전의 단일 EC2 인스턴스의 기존 public_ip 출력을 ALB의 DNS 이름을 표시하는 출력으로 바꾼다.

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

완성된 main.tf

```
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_launch_template" "example" {
  name                   = "example"
  image_id               = "ami-0ab04b3ccbadfae1f"
  instance_type          = "t2.micro"
  key_name               = "aws00-key"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = base64encode(data.template_file.web_output.rendered)

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

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

  name             = "aws00-terraform-asg-example"
  desired_capacity = 1
  min_size         = 1
  max_size         = 2

  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               = "terraform-asg-example"
    propagate_at_launch = true
  }
}

# 로드밸런서
resource "aws_lb" "example" {
  name               = "aws00-terraform-asg-example"
```

```
  load_balancer_type = "application"
  subnets            = data.aws_subnets.default.ids
  security_groups    = [aws_security_group.alb.id]
}

# 로드밸런서 리스너
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.example.arn
  port              = 80
  protocol          = "HTTP"

  # 기본값으로 단순한 404 페이지 오류를 반환한다.
  default_action {
    type = "fixed-response"

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

# 로드 밸런서 리스너 룰 구성
resource "aws_lb_listener_rule" "asg" {
  listener_arn = aws_lb_listener.http.arn
  priority     = 100

  condition {
    path_pattern {
      values = ["*"]
    }
  }

  action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.asg.arn
  }
}


# 로드밸런서 대상그룹
resource "aws_lb_target_group" "asg" {
  name     = "aws00-terraform-asg-example"
  port     = var.server_port
  protocol = "HTTP"
  vpc_id   = data.aws_vpc.default.id

  health_check {
    path                = "/"
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 15
    timeout             = 3
    healthy_threshold   = 2
    unhealthy_threshold = 2
  }
}
```

```
# 보안 그룹 - instance
resource "aws_security_group" "instance" {
  name = "aws00-terraform-example-instance"
  ingress {
    from_port   = var.server_port
    to_port     = var.server_port
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# 보안 그룹 - ALB
resource "aws_security_group" "alb" {
  name = "aws00-terraform-example-alb"

  # 인바운드 HTTP 트래픽 허용
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # 모든 아웃바운드 트래픽 허용
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

data "aws_vpc" "default" {
  default = true
}

data "aws_subnets" "default" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default.id]
  }
}

data "template_file" "web_output" {
  template = file("${path.module}/web.sh")
  vars = {
    server_port = "${var.server_port}"
  }
}
```

설정	설명
HealthCheckProtocol	대상에 대한 상태 확인을 수행할 때 로드 밸런서가 사용하는 프로

	토콜입니다. HTTP, HTTPS 등의 프로토콜이 여기에 해당됩니다. HTTP 프로토콜이 기본 설정값입니다.
HealthCheckPort	대상에 대한 상태 확인을 수행할 때 로드 밸런서가 사용하는 포트입니다. 각 대상이 로드 밸런서에서 트래픽을 수신하는 포트를 사용하도록 기본 설정되어 있습니다.
HealthCheckPath	대상에 대한 상태 확인을 위한 대상입니다. 프로토콜 버전이 HTTP/1.1 또는 HTTP/2인 경우 유효한 URI(/*path*?*query*)를 참조하세요. 기본값은 /입니다. 프로토콜 버전이 gRPC인 경우, 사용자 지정 상태 확인 방법의 경로를 `/package.service/method` 형식으로 지정합니다. 기본값은 `/AWS.ALB/healthcheck` 입니다.
HealthCheckTimeoutSeconds	상태 확인 실패를 의미하는 대상으로부터 응답이 없는 기간(초 단위)입니다. 범위는 2~120초입니다. 대상 유형이 `instance` 또는 `ip`이면 기본값은 5초이며, 대상 유형이 `lambda`이면 기본값은 30초입니다.
HealthCheckIntervalSeconds	개별 인스턴스의 상태 확인 간의 대략적인 간격(초 단위)입니다. 범위는 5~300초입니다. 대상 유형이 `instance` 또는 `ip`이면 기본값은 30초이며, 대상 유형이 `lambda`이면 기본값은 35초입니다.
HealthyThresholdCount	비정상 상태의 대상을 정상으로 간주하기까지 필요한 연속적인 상태 확인 성공 횟수입니다. 범위는 2~10회입니다. 기본값은 5입니다.
UnhealthyThresholdCount	대상을 비정상 상태로 간주하기까지 필요한 연속적인 상태 확인 실패 횟수입니다. 범위는 2~10회입니다. 기본값은 2입니다.
Matcher	대상으로부터 응답 성공을 확인할 때 사용하는 코드입니다. 이를 콘솔에서 **성공 코드**라고 합니다. 프로토콜 버전이 HTTP/1.1 또는 HTTP/2인 경우 가능한 값은 200~499입니다. 값 범위(예: "200-299")에서 여러 값(예: "200,202")을 지정할 수 있습니다. 기본값은 200입니다. 프로토콜 버전이 gRPC인 경우 가능한 값은 0~99입니다. 여러 값(예: "0,1") 또는 값 범위(예: "0-5")를 지정할 수 있습니다. 기본값은 12입니다.