로드 밸런서에 대한 고정 전략 선택

AWS 권장 가이드

AWS 권장 가이드: 로드 밸런서에 대한 고정 전략 선택

Table of Contents

로드 밸런서에 대한 고정 전략 선택

Ryan Griffin, Amazon Web Services(AWS)

2024년 7월(문서 기록)

고정성은 클라이언트에서 단일 대상으로 트래픽을 반복적으로 라우팅하는 로드 밸런서의 기능을 설명하는 데 사용되는 용어로, 여러 대상에서 트래픽을 밸런싱하는 대신 사용됩니다. 예를 들어 클라이언트 A의 트래픽은 특정 서버로 지속적으로 라우팅되어 서버가 세션 상태 데이터를 유지할 수 있습니다. 클라이언트 A의 트래픽이 두 개의 개별 서버로 라우팅되는 경우 각 서버에 다른 서버에서 사용할 수 있는 중요한 정보가 누락되었을 수 있습니다.

따라서 로드 밸런서를 통해 일관된 클라이언트 연결을 유지해야 하는 경우가 많습니다. 고정성에는 고정 세션과 대상 그룹 고정성이라는 두 가지 유형이 있습니다.

- 고정 세션 - Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에서 로컬 세션 데이터를 유지 관리하여 애플리케이션 아키텍처를 단순화하거나 애플리케이션 성능을 개선합니다. 인스턴스는 세션 상태 정보를 로컬로 유지 또는 캐싱할 수 있기 때문입니다. AWS 현재는 애플리케이션 쿠키와 로드 밸런서 쿠키라는 두 가지 유형의 고정 세션을 제공합니다.
- 대상 그룹 고정성 - 블루/그린 배포에서는 여러 버전의 애플리케이션이 배포되어 클라이언트가 세션 중에 동일한 버전의 애플리케이션을 계속 사용하게 할 수 있습니다. 이 경우 대상 그룹 고정을 사용하여 클라이언트의 모든 통신을 동일한 EC2 인스턴스 대신 동일한 대상 그룹으로 라우팅할 수 있습니다.

이 두 가지 고정 전략을 개별적으로 또는 함께 사용할 수 있습니다.

이 가이드에서는 전략을 선택하는 데 도움이 되는 다양한 유형의 로드 밸런서 고정성과 해당 사용 사례를 설명합니다. 이 가이드에는 각 전략을 설명하는 AWS CloudFormation 템플릿이 포함되어 있습니다.

샘플 코드

이 가이드는 기본 아키텍처를 구축하고 각 고정 전략을 시도하기 위해 배포할 수 있는 4개의 AWS CloudFormation 템플릿이 포함된 첨부된 .zip 파일을 제공합니다. 이러한 템플릿을 랩 환경에 배포하여 각 접근 방식을 테스트하는 것이 좋습니다.

샘플 코드 다운로드

다운로드에는 다음 템플릿이 포함됩니다.

- `basic.yml` - 고정 없이 Application Load Balancer를 설정합니다.
- `targetgroupstickiness.yml` - 대상 그룹을 기반으로 고정성을 보여줍니다.
- `stickysessionslb.yml` - 로드 밸런서에서 생성한 쿠키를 사용하여 고정 세션을 보여줍니다.
- `stickysessionsapp.yml` - 애플리케이션 기반 쿠키로 고정 세션을 보여줍니다.

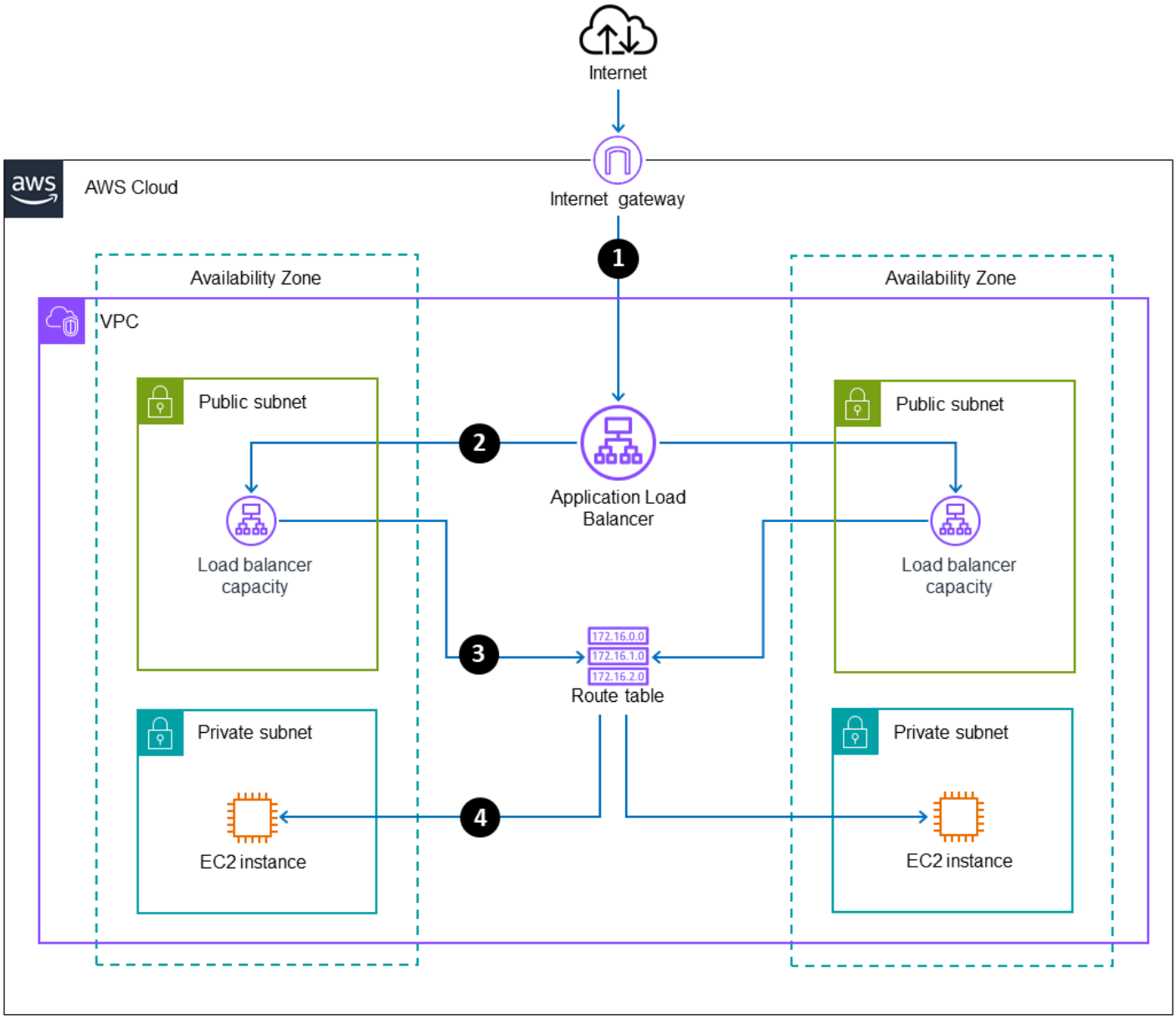
이러한 템플릿을 배포하려면 활성 :AWS account 및 CloudFormation 콘솔에 대한 액세스 권한이 필요합니다. CloudFormation 템플릿 배포에 대한 step-by-steps 지침은 CloudFormation 설명서의 스택 생성을 참조하세요.

로드 밸런서 서브넷 및 라우팅

인터넷을 향하는 Application Load Balancer를 프라이빗 서브넷의 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스로 구성할 때 트래픽이 어떻게 흐르는지 보여주는 것부터 시작하겠습니다. 이 아키텍처는 인터넷에 열려 있는 Application Load Balancer를 배포할 때의 모범 사례를 반영합니다.

인바운드 트래픽 경로

다음 다이어그램은 들어오는 트래픽 흐름과 연결된 Virtual Private Cloud(VPC) 서브넷 및 라우팅을 보여 주며, 명확성을 위해 다이어그램에서 반환 트래픽이 제거됩니다.

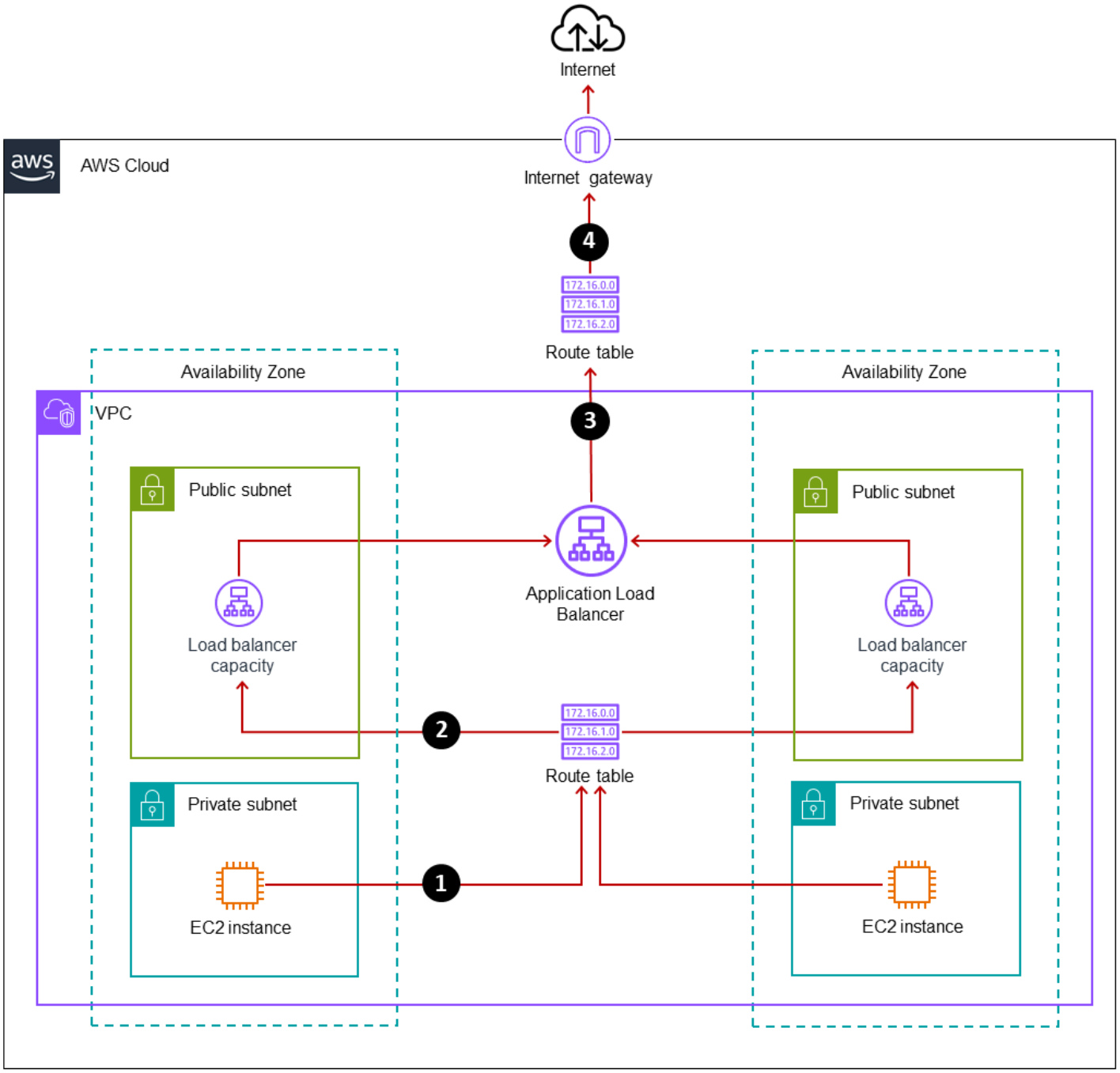


1. 인터넷의 트래픽은에서 Application Load Balancer DNS 이름으로 흐릅니다.
2. Application Load Balancer는 표시된 시나리오에서 두 개의 퍼블릭 서브넷과 연결됩니다. Elastic Load Balancing 서비스는 활성화된 각 가용 영역에 로드 밸런서 용량을 생성하고 가용 영역을 통해 트래픽을 전송하여 적절한 라우팅 로직을 결정합니다. Application Load Balancer는 내부 로직을 사용하여 트래픽을 라우팅할 대상 그룹과 인스턴스를 결정합니다.
3. Application Load Balancer는 동일한 가용 영역의 퍼블릭 서브넷과 연결된 노드를 통해 요청을 EC2 인스턴스로 라우팅합니다. (이러한 노드는 Elastic Load Balancing 서비스에 의해 구성, 관리 및 확장되며 사용자에게 표시되지 않습니다.)

4. 라우팅 테이블은 VPC 내 로컬, 퍼블릭 서브넷과 프라이빗 서브넷 간 및 EC2 인스턴스로 트래픽을 라우팅합니다.

트래픽 경로 반환

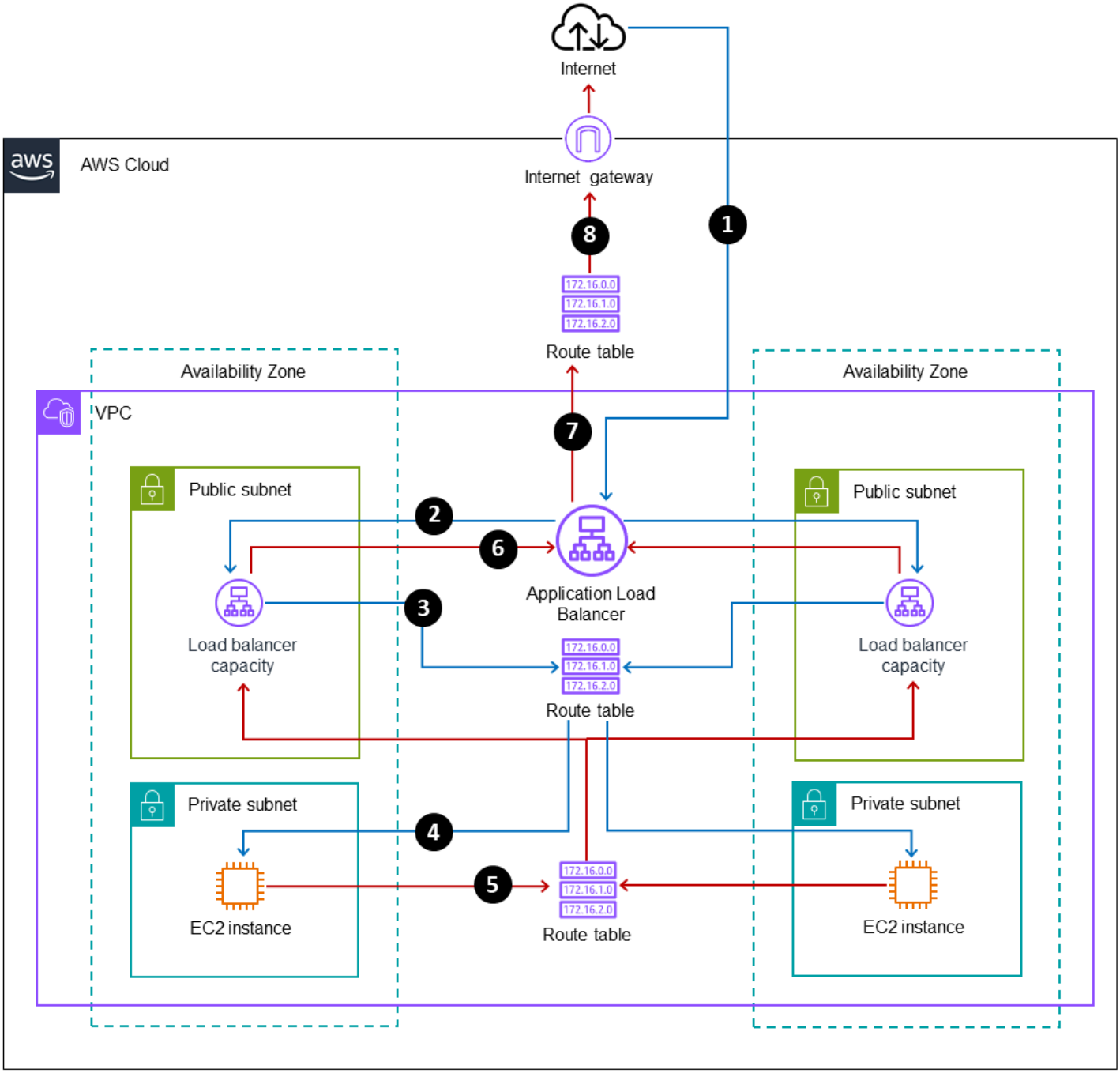
다음 다이어그램은 트래픽 경로와 연결된 VPC 서브넷 및 라우팅을 다시 인터넷으로 보여 주며, 들어오는 트래픽은 명확성을 위해 다이어그램에서 제거됩니다.



1. 프라이빗 서브넷의 EC2 인스턴스는 라우팅 테이블을 통해 아웃바운드 트래픽을 라우팅합니다.
2. 라우팅 테이블에는 퍼블릭 서브넷에 대한 로컬 경로가 있습니다. 트래픽이 입력된 경로에 따라 해당 퍼블릭 서브넷에서 트래픽이 입력한 Application Load Balancer 용량에 도달합니다.
3. Application Load Balancer는 퍼블릭 인터페이스를 통해 트래픽을 라우팅합니다.
4. 퍼블릭 서브넷의 라우팅 테이블에는 트래픽을 다시 인터넷으로 라우팅하는 인터넷 게이트웨이를 가리키는 기본 라우팅이 있습니다.

전체 트래픽 흐름 다이어그램

다음 다이어그램은 인바운드 및 반환 트래픽 흐름을 결합하여 로드 밸런서 라우팅의 전체 그림을 제공합니다.



1. 인터넷의 트래픽은에서 Application Load Balancer DNS 이름으로 흐릅니다.
2. Application Load Balancer는 표시된 시나리오에서 두 개의 퍼블릭 서브넷과 연결됩니다. Elastic Load Balancing 서비스는 활성화된 각 가용 영역에 로드 밸런서 용량을 생성하고 가용 영역을 통해 트래픽을 전송하여 적절한 라우팅 로직을 결정합니다. Application Load Balancer는 내부 로직을 사용하여 트래픽을 라우팅할 대상 그룹과 인스턴스를 결정합니다.

3. Application Load Balancer는 동일한 가용 영역의 퍼블릭 서브넷과 연결된 노드를 통해 요청을 EC2 인스턴스로 라우팅합니다. (이러한 노드는 Elastic Load Balancing 서비스에 의해 구성, 관리 및 확장되며 사용자에게 표시되지 않습니다.)
4. 라우팅 테이블은 VPC 내 로컬, 퍼블릭 서브넷과 프라이빗 서브넷 간 및 EC2 인스턴스로 트래픽을 라우팅합니다.
5. 프라이빗 서브넷의 EC2 인스턴스는 라우팅 테이블을 통해 아웃바운드 트래픽을 라우팅합니다.
6. 라우팅 테이블에는 퍼블릭 서브넷에 대한 로컬 경로가 있습니다. 트래픽이 입력된 경로에 따라 해당 퍼블릭 서브넷에서 트래픽이 입력한 Application Load Balancer 용량에 도달합니다.
7. Application Load Balancer는 퍼블릭 인터페이스를 통해 트래픽을 라우팅합니다.
8. 퍼블릭 서브넷의 라우팅 테이블에는 트래픽을 다시 인터넷으로 라우팅하는 인터넷 게이트웨이를 가리키는 기본 라우팅이 있습니다.

어떤 고정 전략이 적합합니까?

다음 질문에 답하면 로드 밸런서 고정 전략을 결정하는 데 도움이 됩니다.

WebSockets를 사용하고 있습니까?

- 예: WebSockets는 기본적으로 고정되어 있으므로 전략을 사용할 필요가 없습니다.
- 아니요: 다음 질문으로 계속 진행합니다.

블루/그린 배포를 계획하고 있습니까?

- 예: 최소한 대상 그룹 고정을 사용하지만 다음 질문으로 계속 진행합니다.
- 아니요: 다음 질문으로 계속 진행합니다.

클라이언트의 일부 또는 모든 트래픽을 동일한 EC2 인스턴스로 반복적으로 라우팅해야 합니까?

- 예: 다음 질문으로 계속 진행합니다.
- 아니요: 고정 세션을 사용할 필요가 없습니다.

애플리케이션 코드 또는 클라이언트가 쿠키를 사용하여 상태 속성 및 기간을 제어하도록 하시겠습니까?

- 예: 애플리케이션 기반 쿠키를 사용하여 애플리케이션 기반 고정 세션을 활성화합니다.
- 아니요: 로드 밸런서에서 생성한 쿠키를 사용하여 기간 기반 고정 세션을 활성화합니다.

고정성이 없는 Application Load Balancer

어떤 형태로든 고정 없이 Application Load Balancer를 사용하는 경우 로드 밸런서는 기본적으로 라운드 로빈 메서드를 사용하여 트래픽을 라우팅해야 하는 EC2 인스턴스를 결정합니다.

템플릿: CloudFormation 템플릿`basic.yml`(샘플 코드 .zip 파일에 포함됨)을 사용하여이 기능을 사용해 보세요.

> **Note**
>
> 이 가이드에 포함된 모든 CloudFormation 템플릿은 특정 로드 밸런서 고정 전략을 설명하기 위해 사용자 지정 VPC, 라우팅 테이블, 라우팅, 인터넷 게이트웨이, Application Load Balancer, 대상 그룹, 리스너 및 EC2 인스턴스를 배포합니다.

일반 사용 사례

다음 시나리오에서는 고정 없이 Application Load Balancer를 사용합니다.

- 트래픽을 라우팅할 대상 목록이 있지만 대상이 세션 상태를 유지하지 않습니다.
- 세션 상태를 유지하지 않는 웹 서버를 사용하고 있습니다.
- 세션 상태를 유지하지 않는 애플리케이션 서버를 사용하고 있습니다.

단계

> **Notes**
>
> - NAT 게이트웨이에는 적은 비용이 발생합니다.
> - 여러 EC2 인스턴스는 단일 EC2 인스턴스보다 프리 티어 시간을 더 빠르게 사용합니다.

1. CloudFormation 템플릿을 랩 환경에 배포`basic.yml`합니다.
2. 대상 그룹 인스턴스의 상태가 초기에서 정상으로 변경될 때까지 기다립니다.
3. HTTP(TCP/80)를 사용하여 웹 브라우저에서 Application Load Balancer URL로 이동합니다.

예: `http://alb-123456789.us-east-1.elb.amazonaws.com/`

웹 페이지에 인스턴스 1 - TG1 또는 인스턴스 2 - TG1이 표시됩니다.

4. 페이지를 여러 번 새로 고칩니다.

예상 결과

웹 페이지를 로드하는 인스턴스(인스턴스 1 또는 인스턴스 2)는 페이지 텍스트에 반영된 대로 매번 변경되어야 합니다. 로드 밸런서 로직은 여러 내부 노드에서 마지막 대상을 관리하므로 동기화 지연이 발생할 수 있으므로 동일한 대상으로 라우팅될 가능성이 있습니다.

작동 방법

- 이 예제에서는 두 개의 EC2 인스턴스가 단일 대상 그룹에 할당됩니다. EC2 인스턴스에는 Apache 웹 서버(`httpd`)가 설치되어 있으며 각 EC2 인스턴스의 `index.html` 페이지 텍스트는 하드코딩되어 해당 인스턴스를 식별합니다.
- Application Load Balancer는 내부 라운드 로빈 로직을 실행하여 트래픽을 수신할 EC2 인스턴스를 결정합니다.
- 웹 페이지를 다시 로드할 때마다 Application Load Balancer는 라우팅 로직을 실행하고 페이지에 인스턴스 1 - TG1 또는 인스턴스 2 - TG1이 표시됩니다.

대상 그룹 고정성

대상 그룹 고정성과 함께 Application Load Balancer를 사용하는 경우:

- Application Load Balancer는 대상 그룹 가중치를 사용하여 대상 그룹 간에 들어오는 트래픽의 균형을 맞추는 방법을 결정합니다.
- 기본적으로 Application Load Balancer는 라운드 로빈 메서드를 사용하여 대상 대상 그룹의 EC2 인스턴스로 요청을 라우팅합니다.

템플릿: CloudFormation 템플릿`targetgroupstickiness.yml`(샘플 코드 .zip 파일에 포함됨)을 사용하여 대상 그룹 고정성을 시도합니다.

일반 사용 사례

다음 시나리오에서는 대상 그룹 고정을 사용합니다.

- 로드 밸런서에 여러 대상 그룹이 할당되어 있으므로 클라이언트의 트래픽은 해당 대상 그룹 내의 인스턴스로 일관되게 라우팅되어야 합니다.
- 블루/그린 배포.

basic.yml에서 코드 변경

리스너가 한 번 변경되었습니다. Application Load Balancer 기본 작업을 수정하여 고정 구성으로 동일한 가중치의 두 대상 그룹(`TG1` 및 `TG2`)을 지정했습니다.

basic.yml

```
Listener1:
  Type: 'AWS::ElasticLoadBalancingV
2::Listener'
  Properties:
    LoadBalancerArn: !Ref ALB
    Protocol: HTTP
    Port: 80
    DefaultActions:
```

targetgroupstickiness.yml

```
Listener1:
  Type: 'AWS::ElasticLoadBalancingV
2::Listener'
  Properties:
    LoadBalancerArn: !Ref ALB
    Protocol: HTTP
    Port: 80
    DefaultActions:
```

```
      - TargetGroupArn: !Ref TG1
        Type: forward
```

```
      - ForwardConfig:
          TargetGroups:
            - TargetGroupArn: !Ref TG1
              Weight: 1
            - TargetGroupArn: !Ref TG2
              Weight: 1
          TargetGroupStickinessConfig
:
              DurationSeconds: 10
              Enabled: true
        Type: forward
```

단계

> **Notes**
>
> - NAT 게이트웨이에는 적은 비용이 발생합니다.
> - 여러 EC2 인스턴스는 단일 EC2 인스턴스보다 프리 티어 시간을 더 빠르게 사용합니다.

1. CloudFormation 템플릿을 랩 환경에 배포`targetgroupstickiness.yml`합니다.
2. 대상 그룹 인스턴스의 상태가 초기에서 정상으로 변경될 때까지 기다립니다.
3. HTTP(TCP/80)를 사용하여 웹 브라우저에서 Application Load Balancer URL로 이동합니다.

   예: `http://alb-123456789.us-east-1.elb.amazonaws.com/`

   웹 페이지에는 인스턴스 1 - TG1, 인스턴스 2 - TG1, 인스턴스 3 - TG2 또는 인스턴스 4 - TG2 중 하나가 표시됩니다.
4. 페이지를 여러 번 새로 고칩니다.

예상 결과

> **Note**
>
> 이 예제의 CloudFormation 템플릿은 고정을 10초 동안 지속되도록 구성합니다.

웹 페이지를 로드하는 인스턴스는 페이지 텍스트에 반영된 대로 10초 동안 대상 그룹(TG1 또는 TG2) 내에 있어야 합니다.

약 10초 후에 고정이 해제되고 대상 그룹 인스턴스 세트가 변경될 수 있습니다.

작동 방법

- 이 예제에서는 4개의 EC2 인스턴스가 대상 그룹당 2개의 인스턴스와 함께 2개의 대상 그룹으로 분할됩니다. EC2 인스턴스에는 Apache 웹 서버(`httpd`)가 설치되어 있으며, 각 EC2 인스턴스의 `index.html` 페이지 텍스트는 고유하도록 하드코딩됩니다.
- Application Load Balancer는 만료 시간을 두고 대상 대상 그룹에 대한 사용자 세션의 바인딩을 생성합니다.
- 페이지를 다시 로드하면 Application Load Balancer는 바인딩이 존재하는지 여부와 만료되지 않았는지 확인합니다.
  - 바인딩이 만료되었거나 존재하지 않는 경우 Application Load Balancer는 라우팅 로직을 실행하고 대상 대상 그룹을 결정합니다.
  - 바인딩이 만료되지 않은 경우 Application Load Balancer는 트래픽을 동일한 대상 그룹으로 라우팅하지만 반드시 동일한 EC2 인스턴스로 라우팅하지는 않습니다.

로드 밸런서에서 생성된 쿠키가 있는 고정 세션

로드 밸런서 생성 쿠키와 함께 Application Load Balancer를 사용하는 경우:

- Application Load Balancer는 대상 그룹 가중치를 사용하여 대상 그룹 간에 들어오는 트래픽의 균형을 맞추는 방법을 결정합니다.
- 기본적으로 Application Load Balancer는 라운드 로빈 메서드를 사용하여 대상 대상 그룹의 EC2 인스턴스로 요청을 라우팅합니다.

  트래픽이 처음에 인스턴스로 라우팅된 후 후속 트래픽은 지정된 기간 동안 해당 EC2 인스턴스에 유지됩니다.

템플릿: CloudFormation 템플릿`stickysessionslb.yml`(샘플 코드 .zip 파일에 포함됨)을 사용하여 로드 밸런서에서 생성된 쿠키로 고정 세션을 시도합니다.

일반 사용 사례

다음 시나리오에서는 로드 밸런서에서 생성한 쿠키와 함께 고정 세션을 사용합니다.

- PHP 웹 서버
- 로그, 장바구니 또는 채팅 대화와 같은 임시 세션 데이터를 유지하는 서버

basic.yml에서 코드 변경

관련 코드 변경은 고정 유형을 로 설정하고 `lb_cookie` 지속 시간을 10초로 설정하기 위해 대상 그룹 구성에 있습니다.

basic.yml

```
TG1:
   Type: 'AWS::ElasticLoadBalancingV
2::TargetGroup'
   Properties:
     Name: TG1
     Protocol: HTTP
     Port: 80
     TargetType: instance
```

stickysessionslb.yml

```
TG1:
   Type: 'AWS::ElasticLoadBalancingV
2::TargetGroup'
   Properties:
     Name: TG1
     Protocol: HTTP
     Port: 80
     TargetType: instance
```

```
      Targets:
        - Id: !Ref Instance1
        - Id: !Ref Instance2
VpcId: !Ref CustomVPC
```

```
      Targets:
        - Id: !Ref Instance1
        - Id: !Ref Instance2
      VpcId: !Ref CustomVPC
      TargetGroupAttributes:
        - Key: stickiness.enabled
          Value: true
        - Key: stickiness.type
          Value: lb_cookie
        - Key: stickiness.lb_cook
ie.duration_seconds
Value: 10
```

단계

> **Notes**
>
> - NAT 게이트웨이에는 적은 비용이 발생합니다.
> - 여러 EC2 인스턴스는 단일 EC2 인스턴스보다 프리 티어 시간을 더 빠르게 사용합니다.

1. 랩 환경에 CloudFormation 템플릿을 배포`stickysessionslb.yml`합니다.
2. 대상 그룹 인스턴스의 상태가 초기에서 정상으로 변경될 때까지 기다립니다.
3. HTTP(TCP/80)를 사용하여 웹 브라우저에서 Application Load Balancer URL로 이동합니다.

   예: `http://alb-123456789.us-east-1.elb.amazonaws.com/`

   웹 페이지에는 인스턴스 1 - TG1, 인스턴스 2 - TG1, 인스턴스 3 - TG2 또는 인스턴스 4 - TG1 중 하나가 표시됩니다.
4. 페이지를 여러 번 새로 고칩니다.

예상 결과

> **Note**
>
> 이 예제의 CloudFormation 템플릿은 고정을 10초 동안 지속되도록 구성합니다.

웹 페이지를 로드하는 인스턴스는 페이지 텍스트에 반영된 대로 10초 기간 내에 동일하게 유지되어야 합니다. 약 10초 후에 고정이 해제되고 대상 인스턴스가 변경될 수 있습니다.

작동 방법

- 이 예제에서는 하나의 대상 그룹에 두 개의 EC2 인스턴스가 있습니다. EC2 인스턴스에는 Apache 웹 서버(`httpd`)가 설치되어 있으며, 각 EC2 인스턴스의 `index.html` 페이지 텍스트는 고유하도록 하드코딩되어 있습니다.
- Application Load Balancer는 사용자 세션에 대한 바인딩을 생성하여 만료 시간과 함께 대상에 바인딩합니다.
- 페이지를 다시 로드하면 Application Load Balancer는 바인딩이 존재하는지 여부와 만료되지 않았는지 확인합니다.
  - 바인딩이 만료되었거나 존재하지 않는 경우 Application Load Balancer는 라우팅 로직을 실행하고 대상 인스턴스를 결정합니다.
  - 바인딩이 만료되지 않은 경우 Application Load Balancer는 트래픽을 동일한 대상 인스턴스로 라우팅합니다.

애플리케이션 기반 쿠키가 포함된 고정 세션

Application Load Balancer를 애플리케이션 기반 쿠키와 함께 사용하는 경우:

- Application Load Balancer는 대상 그룹 가중치를 사용하여 대상 그룹 간에 수신 트래픽의 균형을 맞추는 방법을 결정합니다.
- 기본적으로 Application Load Balancer는 라운드 로빈 메서드를 사용하여 대상 대상 그룹의 EC2 인스턴스로 요청을 라우팅합니다.
- 트래픽이 처음에 EC2 인스턴스로 라우팅된 후 EC2 인스턴스 애플리케이션 응답에는 사용자 지정 애플리케이션 쿠키가 포함되어야 합니다.이 쿠키는 자동 Application Load Balancer 쿠키와 함께 클라이언트로 다시 전송됩니다.
- 후속 트래픽은 클라이언트가 애플리케이션 쿠키와 Application Load Balancer 쿠키를 다시 보내면 EC2 인스턴스에 유지됩니다.
- 애플리케이션 기반 쿠키는 구성된 기간 동안 사용하지 않으면 만료됩니다.

템플릿: CloudFormation 템플릿`stickysessionsapp.yml`(샘플 코드 .zip 파일에 포함됨)을 사용하여 애플리케이션 기반 쿠키로 고정 세션을 시도합니다.

일반 사용 사례

다음 시나리오에서 추가 제어를 원하는 경우 애플리케이션 생성 쿠키와 함께 고정 세션을 사용합니다.

- PHP 웹 서버
- 로그, 장바구니 또는 채팅 대화와 같은 임시 세션 데이터를 유지하는 서버

basic.yml에서 코드 변경

유일한 코드 변경은 대상 그룹 구성에 있습니다. Application Load Balancer와 대상 그룹 속성에 고정 구성을 추가했습니다. 애플리케이션 쿠키 기간이 지정되고 대상 그룹에 애플리케이션 쿠키 고정이 활성화되어 있습니다.

basic.yml	**stickysessionsapp.yml**
`TG1:`	`TG1:`

```
    Type: 'AWS::ElasticLoadBalancingV
2::TargetGroup'
    Properties:
      Name: TG1
      Protocol: HTTP
      Port: 80
      TargetType: instance
      Targets:
        - Id: !Ref Instance1
        - Id: !Ref Instance2
VpcId: !Ref CustomVPC
```

```
  Type: 'AWS::ElasticLoadBalancingV
2::TargetGroup'
  Properties:
    Name: TG1
    Protocol: HTTP
    Port: 80
    TargetType: instance
    Targets:
      - Id: !Ref Instance1
      - Id: !Ref Instance2
    VpcId: !Ref CustomVPC
    TargetGroupAttributes:
      - Key: stickiness.enabled
        Value: true
      - Key: stickiness.type
        Value: app_cookie
      - Key: stickiness.app_coo
kie.duration_seconds
        Value: 10
      - Key: stickiness.app_coo
kie.cookie_name
        Value: TESTCOOKIE
```

단계

> **Notes**
>
> - NAT 게이트웨이에는 적은 비용이 발생합니다.
> - 여러 EC2 인스턴스는 단일 EC2 인스턴스보다 프리 티어 시간을 더 빠르게 사용합니다.

1. CloudFormation 템플릿을 랩 환경에 배포`stickysessionslb.yml`합니다.
2. 대상 그룹 인스턴스의 상태가 초기에서 정상으로 변경될 때까지 기다립니다.
3. HTTP(TCP/80)를 사용하여 웹 브라우저에서 Application Load Balancer URL로 이동합니다.

   예: `http://alb-123456789.us-east-1.elb.amazonaws.com/`

   웹 페이지에는 인스턴스 1 - TG1, 인스턴스 2 - TG1, 인스턴스 3 - TG2 또는 인스턴스 4 - TG2 중 하나가 표시됩니다.

4. 페이지를 여러 번 새로 고칩니다.

예상 결과

> **Note**
>
> 이 예제의 CloudFormation 템플릿은 고정을 10초 동안 지속되도록 구성했습니다. 유효한 고정 기간 구성은 1초에서 1주 사이입니다.

웹 페이지를 로드하는 인스턴스는 페이지 텍스트에 표시된 대로 동일하게 유지되어야 합니다.

고정 기간은 새로 고쳐지지 않지만 EC2 인스턴스에서 생성되는 애플리케이션 쿠키에 대해 Application Load Balancer에 구성된 만료를 기반으로 합니다.

예제 1: 페이지를 새로 고치려면 5초 동안 기다립니다. 동일한 인스턴스가 로드되고 10초 동안 고정성이 새로 고쳐집니다.

예제 2: 페이지를 다시 로드하려면 10초 이상 기다립니다. 애플리케이션 쿠키가 만료되고 다른 EC2 인스턴스로 라우팅됩니다. 이 새 인스턴스는 10초 동안 다른 애플리케이션 쿠키를 생성합니다.

작동 방법

- 이 예제에서는 EC2 인스턴스에 Apache 웹 서버(`httpd`)가 설치되어 있습니다. `httpd.conf` 파일은 정적 `Set-Cookie` 값을 클라이언트(웹 브라우저)에 반환하도록 구성됩니다. `Set-Cookie` 값은로 하드코딩됩니다`TESTCOOKIE=<somevalue>`.
- 브라우저의 요소 검사 옵션을 열고 네트워크 탭을 선택한 다음 페이지를 로드하는 메서드 가져오기를 선택합니다. 쿠키 탭이 표시됩니다.
- 브라우저는 서버 `Set-Cookie` 응답에서 수신하는 쿠키를 사용하여 후속 업데이트를 서버에 반환하도록 자동으로 구성된 클라이언트 애플리케이션입니다.
- 페이지를 다시 로드하면 초기 페이지 로드 시 수신된 쿠키가 Application Load Balancer로 자동으로 다시 전송됩니다.
  - 쿠키가 만료된 경우(즉, 마지막 호출 이후 10초가 경과한 경우) Application Load Balancer는 새 로직을 사용하여 트래픽을 라우팅할 EC2 인스턴스를 결정합니다.
  - 쿠키가 만료되지 않은 경우 Application Load Balancer는 트래픽을 동일한 EC2 인스턴스로 라우팅합니다.

리소스

- Application Load Balancer에 대한 고정 세션(Elastic Load Balancing 설명서)

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
업데이트된 섹션	로드 밸런서 서브넷 및 라우팅 섹션을 새 다이어그램 및 설명으로 업데이트했습니다.	2024년 7월 5일
업데이트 및 수정	코드, 다이어그램 및 예제를 업데이트했습니다.	2023년 5월 31일
업데이트된 섹션	보다 정확하고 자세한 정보를 제공하기 위해 로드 밸런서에서 생성한 쿠키를 사용하여 대상 그룹 고정 및 고정 세션의 작동 방식 섹션을 업데이트했습니다. https://docs.aws.amazon.com/prescriptive-guidance/latest/load-balancer-stickiness/alb-cookies-stickiness.html	2022년 7월 18일
—	최초 게시	2021년 8월 5일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.