aws

ロードバランサーの維持戦略の選択

AWS 規範ガイダンス

AWS 規範ガイダンス: ロードバランサーの維持戦略の選択

Table of Contents

ロードバランサーの維持戦略の選択

Ryan Griffin、Amazon Web Services (AWS)

2024 年 7 月 (ドキュメント履歴)

スティッキーとは、ロードバランサーの機能を記述して、複数の送信先間でトラフィックのバランスをとるのではなく、クライアントから単一の送信先にトラフィックを繰り返しルーティングするために使用される用語です。例えば、クライアント A からのトラフィックは特定のサーバーに継続的にルーティングできるため、サーバーはセッション状態データを維持できます。クライアント A からのトラフィックが 2 つの異なるサーバーにルーティングされる場合、各サーバーには他のサーバーで利用できる重要な情報が欠落している可能性があります。

したがって、ロードバランサーを介して一貫したクライアント接続を維持する必要があることがよくあります。維持には、スティッキーセッションとターゲットグループの維持の 2 種類があります。

- スティッキーセッション – インスタンスがセッション状態情報をローカルで維持またはキャッシュできるため、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスでローカルセッションデータを維持し、アプリケーションアーキテクチャを簡素化したり、アプリケーションのパフォーマンスを向上させたりします。 AWS は現在、アプリケーション Cookie とロードバランサー Cookie の 2 種類のスティッキーセッションを提供しています。
- ターゲットグループの維持 – ブルー/グリーンデプロイでは、アプリケーションの複数のバージョンがデプロイされている場合があり、セッション中にクライアントが同じバージョンのアプリケーションを引き続き使用したい場合があります。この場合、ターゲットグループの維持を使用して、クライアントからのすべての通信を、同じ EC2 インスタンスではなく同じターゲットグループにルーティングできます。

これら 2 つの維持戦略は、個別に使用することも、一緒に使用することもできます。

このガイドでは、戦略の選択に役立つ、さまざまなタイプのロードバランサーの維持と適用可能なユースケースについて説明します。このガイドには、各戦略を説明する AWS CloudFormation テンプレートが含まれています。

「サンプルコード」

このガイドでは、4 つの AWS CloudFormation テンプレートを含む .zip ファイルが添付されています。これらのテンプレートをデプロイして、基本的なアーキテクチャを構築し、各維持戦略を試すこ

とができます。これらのテンプレートをラボ環境にデプロイして、各アプローチをテストすることをお勧めします。

サンプルコードをダウンロードする

ダウンロードには、次のテンプレートが含まれています。

- `basic.yml` – 維持せずに Application Load Balancer を設定します。
- `targetgroupstickiness.yml` – ターゲットグループに基づいて維持を示します。
- `stickysessionslb.yml` – ロードバランサーが生成した Cookie を使用したスティッキーセッションを示します。
- `stickysessionsapp.yml` – アプリケーションベースの Cookie を使用したスティッキーセッションを示します。

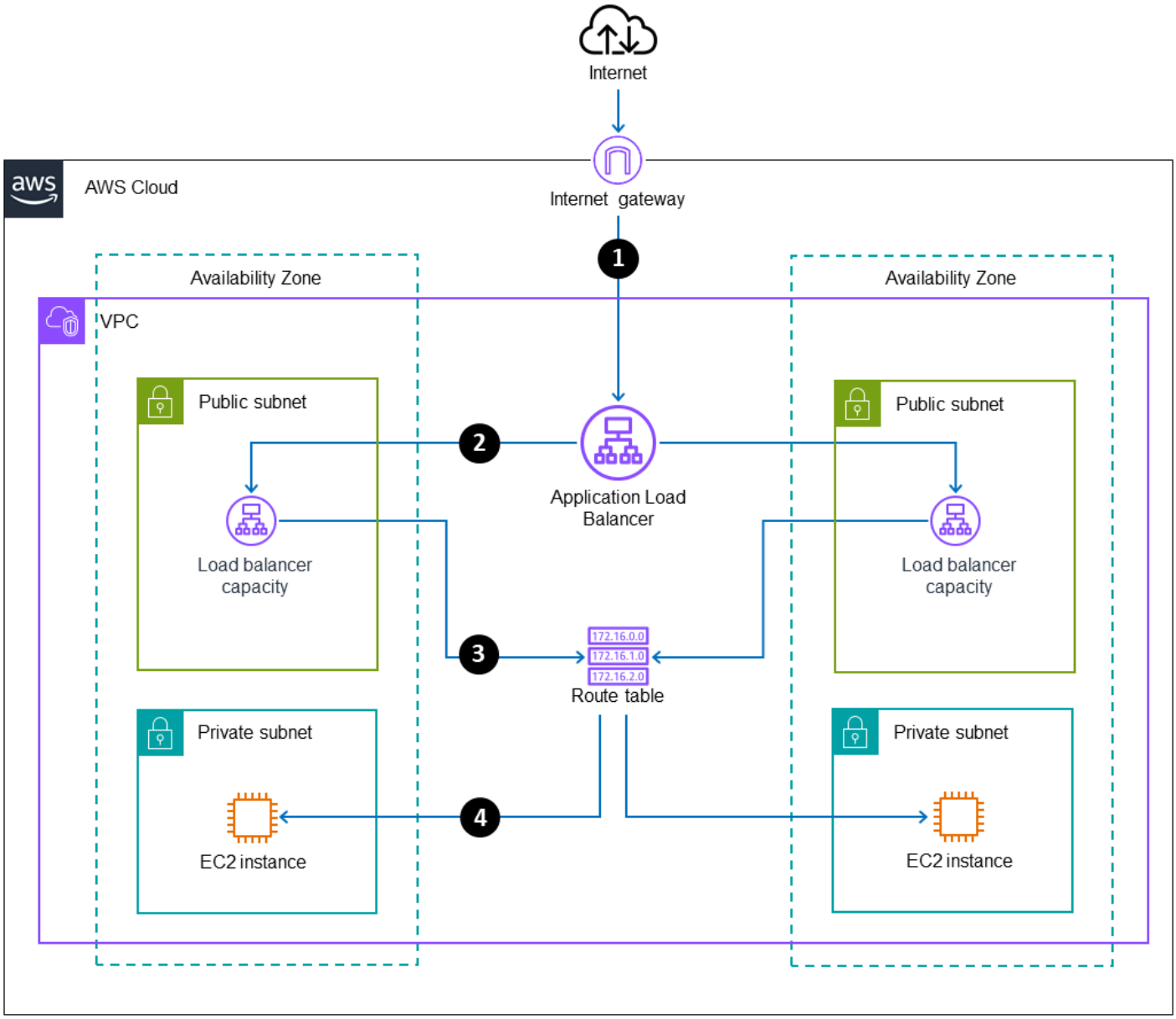
これらのテンプレートをデプロイするには、アクティブな :AWS account と CloudFormation コンソールへのアクセスが必要です。CloudFormation テンプレートをデプロイするstep-by-stepsについては、 CloudFormation ドキュメントの「スタックの作成」を参照してください。

ロードバランサーのサブネットとルーティング

まず、インターネットに面する Application Load Balancer をプライベートサブネット内の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスに設定するときに、トラフィックがどのように流れるかを説明します。このアーキテクチャは、インターネットに公開されている Application Load Balancer をデプロイする際のベストプラクティスを反映しています。

インバウンドトラフィックパス

次の図は、着信トラフィックフローに関連付けられた Virtual Private Cloud (VPC) サブネットとルーティングを示しています。リターントラフィックはわかりやすくするために図から削除されています。



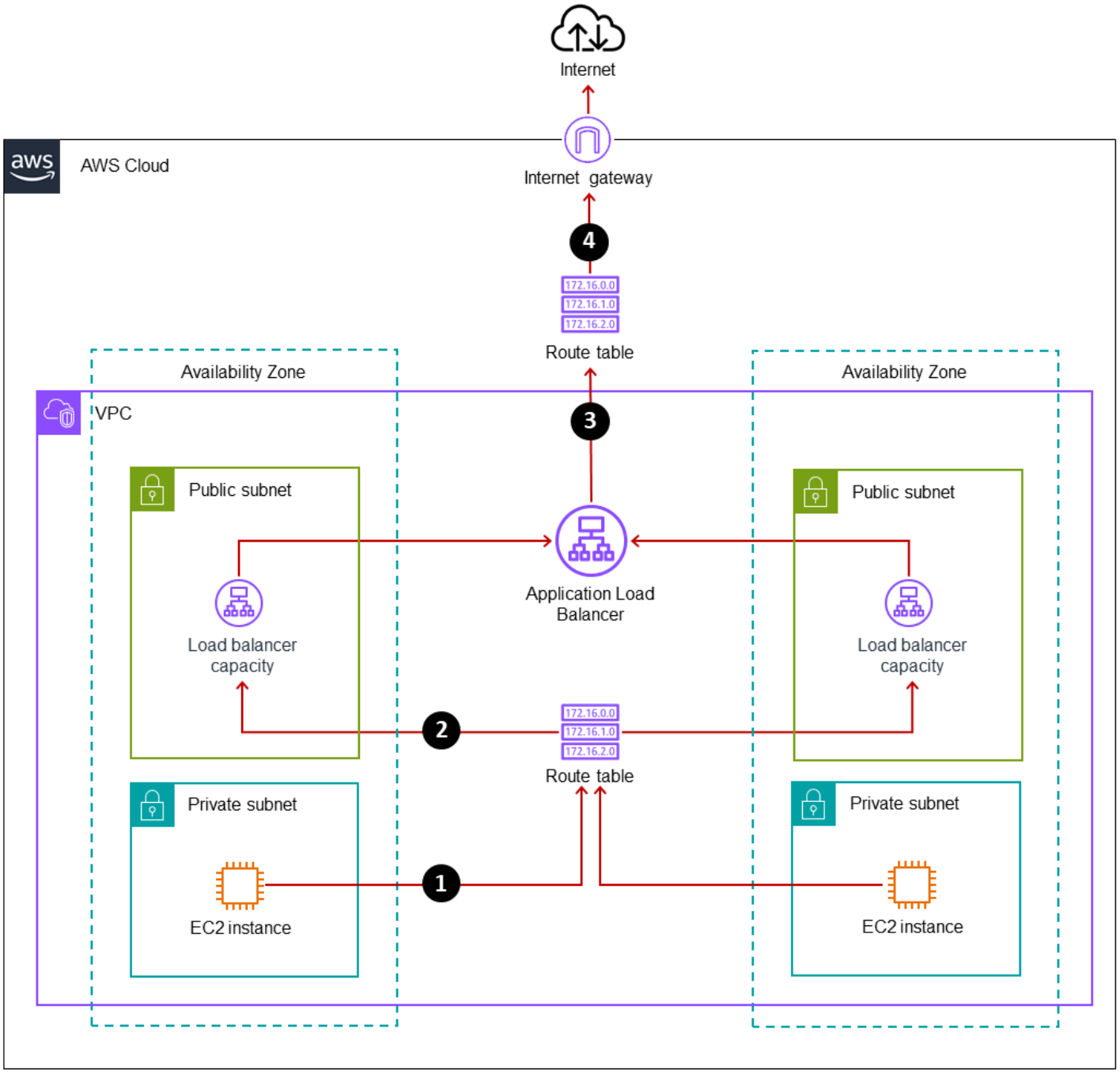
1. インターネットからのトラフィックは、Application Load Balancer DNS 名に流れます。
2. Application Load Balancer は、図に示すシナリオで 2 つのパブリックサブネットに関連付けられています。Elastic Load Balancing サービスは、有効な各アベイラビリティーゾーンにロードバランサー容量を作成し、アベイラビリティーゾーンを介してトラフィックを送信して適切なルーティングロジックを決定します。Application Load Balancer は内部ロジックを使用して、トラフィックをルーティングするターゲットグループとインスタンスを決定します。
3. Application Load Balancer は、同じアベイラビリティーゾーン内のパブリックサブネットに関連付けられているノードを介して EC2 インスタンスにリクエストをルーティングします。(これら

のノードは Elastic Load Balancing サービスによって設定、管理、スケーリングされ、ユーザーには表示されません）。

4. ルートテーブルは、VPC 内、パブリックサブネットとプライベートサブネット間、および EC2 インスタンスにトラフィックをローカルにルーティングします。

トラフィックパスを返す

次の図は、トラフィックパスに関連付けられた VPC サブネットとルーティングをインターネットに戻し、受信トラフィックを図から削除してわかりやすくしています。

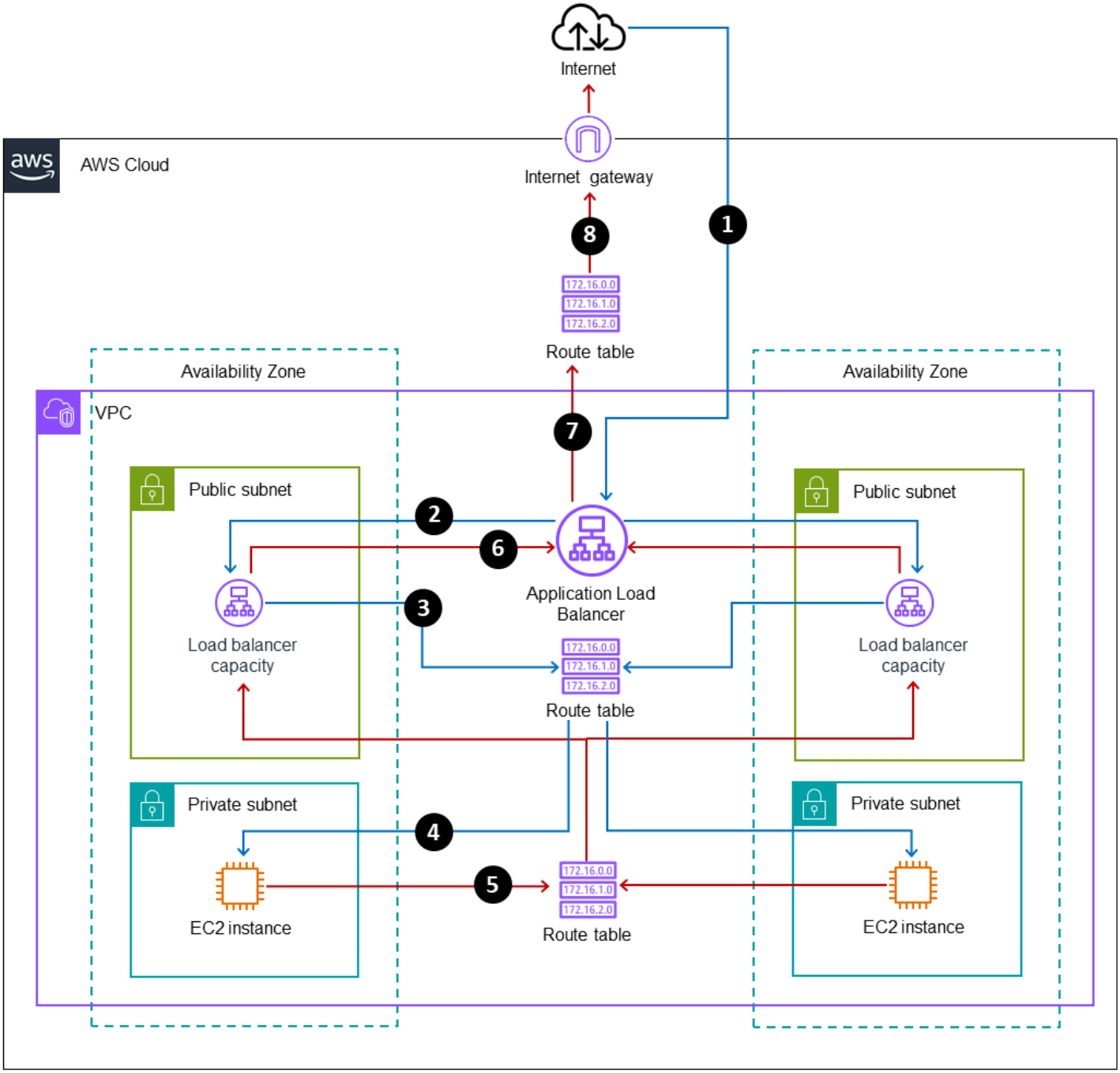


1. プライベートサブネットの EC2 インスタンスは、ルートテーブルを介してアウトバウンドトラフィックをルーティングします。
2. ルートテーブルには、パブリックサブネットへのローカルルートがあります。トラフィックが入り込んだパスを、対応するパブリックサブネット内の Application Load Balancer の容量に合わせることができます。

3. Application Load Balancer は、パブリックインターフェイスを介してトラフィックをルーティングします。
4. パブリックサブネットのルートテーブルには、トラフィックをインターネットにルーティングするインターネットゲートウェイを指すデフォルトルートがあります。

完全なトラフィックフロー図

次の図は、インバウンドトラフィックフローとリターントラフィックフローを組み合わせて、ロードバランサーのルーティングの完全な図を示しています。



1. インターネットからのトラフィックは、Application Load Balancer DNS 名に流れます。
2. Application Load Balancer は、図に示すシナリオで 2 つのパブリックサブネットに関連付けられています。Elastic Load Balancing サービスは、有効な各アベイラビリティーゾーンにロードバランサー容量を作成し、アベイラビリティーゾーンを介してトラフィックを送信して適切なルーティングロジックを決定します。Application Load Balancer は内部ロジックを使用して、トラフィックをルーティングするターゲットグループとインスタンスを決定します。

3. Application Load Balancer は、同じアベイラビリティーゾーン内のパブリックサブネットに関連付けられているノードを介して EC2 インスタンスにリクエストをルーティングします。(これらのノードは Elastic Load Balancing サービスによって設定、管理、スケーリングされ、ユーザーには表示されません)。
4. ルートテーブルは、VPC 内、パブリックサブネットとプライベートサブネット間、および EC2 インスタンスにトラフィックをローカルにルーティングします。
5. プライベートサブネットの EC2 インスタンスは、ルートテーブルを介してアウトバウンドトラフィックをルーティングします。
6. ルートテーブルには、パブリックサブネットへのローカルルートがあります。トラフィックが入り込んだパスを、対応するパブリックサブネット内の Application Load Balancer の容量に合わせることができます。
7. Application Load Balancer は、パブリックインターフェイスを介してトラフィックをルーティングします。
8. パブリックサブネットのルートテーブルには、トラフィックをインターネットにルーティングするインターネットゲートウェイを指すデフォルトルートがあります。

どの維持戦略が適切ですか？

次の質問に答えると、ロードバランサーの維持戦略を決定するのに役立ちます。

WebSockets を使用していますか？

- はい: WebSocketsは本質的にスティッキーであるため、戦略を使用する必要はありません。
- いいえ: 次の質問に進みます。

ブルー/グリーンデプロイを計画していますか？

- はい: 少なくとも、ターゲットグループの維持を使用しますが、次の質問に進みます。
- いいえ: 次の質問に進みます。

クライアントからのトラフィックの一部またはすべてを同じ EC2 インスタンスに繰り返しルーティングする必要がありますか？

- はい: 次の質問に進みます。
- いいえ: スティッキーセッションを使用する必要はありません。

アプリケーションコードまたはクライアントで Cookie を使用して状態属性と期間をコントロールしますか？

- はい: アプリケーションベースのスティッキーセッションを有効にするには、アプリケーションベースの Cookie を使用します。
- いいえ: ロードバランサーが生成した Cookie を使用して、期間ベースのスティッキーセッションを有効にします。

Application Load Balancer の維持なし

Application Load Balancer をどのような形の維持もせずに使用する場合、ロードバランサーはデフォルトでラウンドロビンメソッドを使用して、トラフィックをルーティングする EC2 インスタンスを決定します。

テンプレート： CloudFormation テンプレート `basic.yml` (サンプルコードの .zip ファイルに含まれる) を使用して、この機能を試します。

> **Note**
>
> このガイドに含まれているすべての CloudFormation テンプレートは、カスタム VPC、ルートテーブル、ルート、インターネットゲートウェイ、Application Load Balancer、ターゲットグループ、リスナー、EC2 インスタンスをデプロイして、特定のロードバランサーの維持戦略を示します。

一般的なユースケース

これらのシナリオでは、維持せずに Application Load Balancer を使用します。

- トラフィックをルーティングするターゲットのリストはありますが、ターゲットはセッション状態を維持しません。
- セッション状態を維持しないウェブサーバーを使用している。
- セッション状態を維持しないアプリケーションサーバーを使用している。

ステップ

> **メモ**
>
> - NAT ゲートウェイには少額のコストがかかります。
> - 複数の EC2 インスタンスは、1 つの EC2 インスタンスよりも速く無料利用枠の時間を消費します。

1. CloudFormation テンプレート`basic.yml`をラボ環境にデプロイします。

2. ターゲットグループインスタンスのヘルスステータスが初期から正常に変わるまで待ちます。
3. HTTP (TCP/80) を使用して、ウェブブラウザの Application Load Balancer URL に移動します。

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

   ウェブページには、インスタンス 1 - TG1 またはインスタンス 2 - TG1 と表示されます。
4. ページを複数回更新します。

期待される結果

ウェブページをロードするインスタンス (インスタンス 1 またはインスタンス 2) は、ページテキストに反映されているように、毎回変更する必要があります。ロードバランサーロジックは、複数の内部ノード間で最後のターゲットを管理します。これにより、同期の遅延が発生する可能性があるため、同じターゲットにルーティングされる可能性があります。

仕組み

- この例では、2 つの EC2 インスタンスが 1 つのターゲットグループに割り当てられています。EC2 インスタンスには Apache ウェブサーバー (`httpd`) がインストールされており、各 EC2 インスタンスの`index.html`ページテキストは、そのインスタンスを識別するためにハードコードされています。
- Application Load Balancer は内部ラウンドロビンロジックを実行して、トラフィックを受信する EC2 インスタンスを決定します。
- ウェブページを再ロードするたびに、Application Load Balancer はルーティングロジックを実行し、ページにはインスタンス 1 - TG1 またはインスタンス 2 - TG1 が表示されます。

ターゲットグループの維持

ターゲットグループの維持で Application Load Balancer を使用する場合：

- Application Load Balancer は、ターゲットグループの重みを使用して、ターゲットグループ間で受信トラフィックのバランスをとる方法を決定します。
- デフォルトでは、Application Load Balancer はラウンドロビンメソッドを使用して、送信先ターゲットグループの EC2 インスタンスにリクエストをルーティングします。

テンプレート： CloudFormation テンプレート `targetgroupstickiness.yml` (サンプルコードの .zip ファイルに含まれる) を使用して、ターゲットグループの維持を試します。

一般的なユースケース

これらのシナリオでは、ターゲットグループの維持を使用します。

- ロードバランサーには複数のターゲットグループが割り当てられており、クライアントからのトラフィックは、そのターゲットグループ内のインスタンスに一貫してルーティングする必要があります。
- ブルー/グリーンデプロイ。

basic.yml からのコード変更

リスナーに 1 つの変更が加えられました。Application Load Balancer のデフォルトアクションを変更し、維持設定で同じ重みの 2 つのターゲットグループ (`TG1` と `TG2`) を指定しました。

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
```

ステップ

> **メモ**
>
> - NAT ゲートウェイには少額のコストがかかります。
> - 複数の EC2 インスタンスは、1 つの EC2 インスタンスよりも速く無料利用枠の時間を消費します。

1. CloudFormation テンプレート`targetgroupstickiness.yml`をラボ環境にデプロイします。
2. ターゲットグループインスタンスのヘルスステータスが初期から正常に変わるまで待ちます。
3. HTTP (TCP/80) を使用して、ウェブブラウザの Application Load Balancer URL に移動します。

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

   ウェブページには、インスタンス 1 - TG1、インスタンス 2 - TG1、インスタンス 3 - TG2、またはインスタンス 4 - TG2 のいずれかが表示されます。
4. ページを複数回更新します。

期待される結果

> **Note**
>
> この例の CloudFormation テンプレートは、維持期間を 10 秒間に設定します。

ウェブページをロードするインスタンスは、ページテキストに反映されているように、10 秒以内にターゲットグループ (TG1 または TG2) 内に留まる必要があります。

約 10 秒後、維持が解放され、ターゲットグループインスタンスセットが変更される可能性があります。

仕組み

- この例では、4 つの EC2 インスタンスが 2 つのターゲットグループに分割され、ターゲットグループごとに 2 つのインスタンスがあります。EC2 インスタンスには Apache ウェブサーバー (`httpd`) がインストールされており、各 EC2 インスタンスの`index.html`ページテキストは区別できるようにハードコーディングされています。
- Application Load Balancer は、有効期限を指定して、送信先ターゲットグループに対するユーザーのセッションのバインディングを作成します。
- ページを再ロードすると、Application Load Balancer はバインディングが存在し、有効期限が切れていないかどうかを確認します。
  - バインディングの有効期限が切れているか、存在しない場合、Application Load Balancer はルーティングロジックを実行し、送信先ターゲットグループを決定します。
  - バインドの有効期限が切れていない場合、Application Load Balancer はトラフィックを同じターゲットグループにルーティングしますが、必ずしも同じ EC2 インスタンスにルーティングするわけではありません。

ロードバランサーが生成した Cookie を使用したスティッキーセッション

ロードバランサーが生成した Cookie で Application Load Balancer を使用する場合：

- Application Load Balancer は、ターゲットグループの重みを使用して、ターゲットグループ間で受信トラフィックのバランスをとる方法を決定します。
- デフォルトでは、Application Load Balancer はラウンドロビンメソッドを使用して、送信先ターゲットグループの EC2 インスタンスにリクエストをルーティングします。

  トラフィックが最初にインスタンスにルーティングされた後、後続のトラフィックは指定された期間、その EC2 インスタンスに維持されます。

テンプレート： CloudFormation テンプレート `stickysessionslb.yml` (サンプルコードの .zip ファイルに含まれる) を使用して、ロードバランサーが生成した Cookie でスティッキーセッションを試します。

一般的なユースケース

以下のシナリオでは、ロードバランサーが生成した Cookie でスティッキーセッションを使用します。

- PHP ウェブサーバー
- ログ、ショッピングカート、チャット会話などの一時的なセッションデータを保持するサーバー

basic.yml からのコード変更

関連するコードの変更は、維持タイプを `lb_cookie` に設定し、期間を 10 秒に設定するターゲットグループ設定にあります。

basic.yml	**stickysessionslb.yml**
`TG1:` `  Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'`	`TG1:` `  Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'`

```
    Properties:
      Name: TG1
      Protocol: HTTP
      Port: 80
      TargetType: instance
      Targets:
        - Id: !Ref Instance1
        - Id: !Ref Instance2
VpcId: !Ref CustomVPC
```

```
    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: lb_cookie
        - Key: stickiness.lb_cook
ie.duration_seconds
Value: 10
```

ステップ

> **メモ**
>
> - NAT ゲートウェイには少額のコストがかかります。
> - 複数の EC2 インスタンスは、1 つの EC2 インスタンスよりも速く無料利用枠の時間を使い果たします。

1. CloudFormation テンプレート`stickysessionslb.yml`をラボ環境にデプロイします。
2. ターゲットグループインスタンスのヘルスステータスが初期から正常に変わるまで待ちます。
3. HTTP (TCP/80) を使用して、ウェブブラウザの Application Load Balancer URL に移動します。

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

   ウェブページには、インスタンス 1 - TG1、インスタンス 2 - TG1、インスタンス 3 - TG2、またはインスタンス 4 - TG1 のいずれかが表示されます。
4. ページを複数回更新します。

期待される結果

> **Note**
>
> この例の CloudFormation テンプレートは、維持期間を 10 秒間に設定します。

ウェブページをロードするインスタンスは、ページテキストに反映されているように、10 秒以内は同じままにする必要があります。約 10 秒後、維持が解放され、送信先インスタンスが変更される可能性があります。

仕組み

- この例では、1 つのターゲットグループに 2 つの EC2 インスタンスがあります。EC2 インスタンスには Apache ウェブサーバー (`httpd`) がインストールされており、各 EC2 インスタンスの`index.html`ページテキストは区別できるようにハードコーディングされています。
- Application Load Balancer は、ユーザーのセッションのバインディングを作成します。バインディングは、有効期限とともに送信先に対してバインドされます。
- ページを再ロードすると、Application Load Balancer はバインディングが存在し、有効期限が切れていないかどうかを確認します。
  - バインドの有効期限が切れているか、存在しない場合、Application Load Balancer はルーティングロジックを実行し、送信先インスタンスを決定します。
  - バインディングの有効期限が切れていない場合、Application Load Balancer はトラフィックを同じ送信先インスタンスにルーティングします。

アプリケーションベースの Cookie を使用したスティッキーセッション

アプリケーションベースの Cookie で Application Load Balancer を使用する場合：

- Application Load Balancer は、ターゲットグループの重みを使用して、ターゲットグループ間で受信トラフィックのバランスをとる方法を決定します。
- デフォルトでは、Application Load Balancer はラウンドロビンメソッドを使用して、送信先ターゲットグループの EC2 インスタンスにリクエストをルーティングします。
- トラフィックが最初に EC2 インスタンスにルーティングされた後、EC2 インスタンスアプリケーションのレスポンスにはカスタムアプリケーション Cookie が含まれ、自動 Application Load Balancer Cookie とともにクライアントに返されます。
- クライアントがアプリケーション Cookie と Application Load Balancer Cookie を返送すると、それ以降のトラフィックは EC2 インスタンスに保持されます。
- アプリケーションベースの Cookie は、設定した期間使用しない場合に期限切れになります。

テンプレート： CloudFormation テンプレート `stickysessionsapp.yml` (サンプルコードの .zip ファイルに含まれる) を使用して、アプリケーションベースの Cookie でスティッキーセッションを試します。

一般的なユースケース

これらのシナリオで追加の制御が必要な場合は、アプリケーション生成 Cookie でスティッキーセッションを使用します。

- PHP ウェブサーバー
- ログ、ショッピングカート、チャット会話などの一時的なセッションデータを保持するサーバー

basic.yml からのコード変更

唯一のコード変更は、ターゲットグループ設定です。Application Load Balancer とターゲットグループ属性に維持設定を追加しました。アプリケーション Cookie の期間が指定され、ターゲットグループでアプリケーション Cookie の維持が有効になっています。

basic.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
```

stickysessionsapp.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
    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
```

ステップ

> **メモ**
>
> - NAT ゲートウェイには少額のコストがかかります。
> - 複数の EC2 インスタンスは、1 つの EC2 インスタンスよりも速く無料利用枠の時間を使い果たします。

1. CloudFormation テンプレート`stickysessionslb.yml`をラボ環境にデプロイします。
2. ターゲットグループインスタンスのヘルスステータスが初期から正常に変わるまで待ちます。

3. HTTP (TCP/80) を使用して、ウェブブラウザの Application Load Balancer URL に移動します。

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

   ウェブページには、インスタンス 1 - TG1、インスタンス 2 - TG1、インスタンス 3 - TG2、またはインスタンス 4 - TG2 のいずれかが表示されます。
4. ページを複数回更新します。

期待される結果

> **Note**
>
> この例の CloudFormation テンプレートは、維持期間を 10 秒に設定しています。有効な維持期間の設定は 1 秒から 1 週間です。

ウェブページをロードするインスタンスは、ページテキストで示されているように、同じままにする必要があります。

維持期間は更新されませんが、EC2 インスタンスによって生成されるアプリケーション Cookie の Application Load Balancer で設定された有効期限に基づいています。

例 1: ページを更新するまで 5 秒待ちます。同じインスタンスがロードされ、維持状態がさらに 10 秒間更新されます。

例 2: ページを再ロードするまで 10 秒以上待ちます。アプリケーション Cookie の有効期限が切れ、別の EC2 インスタンスにルーティングされます。この新しいインスタンスは、10 秒の期間で別のアプリケーション Cookie を生成します。

仕組み

- この例では、EC2 インスタンスに Apache ウェブサーバー (`httpd`) がインストールされています。`httpd.conf` ファイルは、静的な`Set-Cookie`値をクライアント (ウェブブラウザ) に返すように設定されています。`Set-Cookie` 値は にハードコードされています`TESTCOOKIE=<somevalue>`。
- ブラウザの Inspect Element オプションを開き、Network タブを選択し、Get メソッドを選択してページをロードします。Cookie タブが表示されます。

- ブラウザは、サーバーSet-Cookieレスポンスで受信した Cookie を使用してサーバーに後続の更新を返すように自動的に設定されるクライアントアプリケーションです。
- ページを再ロードすると、最初のページのロードで受信した Cookie が自動的に Application Load Balancer に返されます。
  - Cookie の有効期限が切れている場合 (つまり、前回の呼び出しから 10 秒が経過している場合) 、Application Load Balancer は新しいロジックを使用してトラフィックをルーティングする EC2 インスタンスを決定します。
  - Cookie の有効期限が切れていない場合、Application Load Balancer はトラフィックを同じ EC2 インスタンスにルーティングします。

リソース

- Application Load Balancer のスティッキーセッション (Elastic Load Balancing ドキュメント)

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
更新済みのセクション	「ロードバランサーのサブネットとルーティング」セクションを更新し、新しい図と説明を追加しました。	2024 年 7 月 5 日
更新と修正	コード、図、例を更新しました。	2023 年 5 月 31 日
更新されたセクション	「ターゲットグループの維持」および「スティッキーセッション」の「仕組み」セクションを更新し、ロードバランサーが生成した Cookie を使用して、より正確で詳細な情報を追加しました。	2022 年 7 月 18 日
—	初版発行	2021 年 8 月 5 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。