AWS Prescriptive Guidance

Choosing a stickiness strategy for your load balancer

AWS Prescriptive Guidance: Choosing a stickiness strategy for your load balancer

Table of Contents

Choosing a stickiness strategy for your load balancer

Ryan Griffin, Amazon Web Services (AWS)

May 2023 (document history (p. 17))

Stickiness is a term that is used to describe the functionality of a load balancer to repeatedly route traffic from a client to a single destination, instead of balancing the traffic across multiple destinations. For example, traffic from client A can be continually routed to a specific server, so that server can maintain session state data. If traffic from client A is routed to two distinct servers, each server might be missing important information that's available to the other server.

Therefore, it's often necessary to maintain a consistent client connection through a load balancer. There are two types of stickiness: sticky sessions and target group stickiness.

- **Sticky sessions** – Maintaining local session data in an Amazon Elastic Compute Cloud (Amazon EC2) instance to simplify application architecture or to improve application performance, because the instance can maintain or cache the session state information locally. Amazon Web Services (AWS) currently offers two types of sticky sessions, which this guide discusses in detail: application cookies and load balancer cookies.
- **Target group stickiness** – In blue/green deployments you might have multiple versions of an application deployed, and you might want the client to continue to use the same version of the application during their session. In this case, you can use target group stickiness to route all communications from the client to the same target group instead of the same EC2 instance.

You can use these two stickiness strategies separately or together.

This guide describes different types of load balancer stickiness and applicable use cases, to help you choose a strategy. The guide includes AWS CloudFormation templates that illustrate each strategy.

Sample code

This guide provides an attached .zip file that includes four AWS CloudFormation templates that you can deploy to build a basic architecture and try out each stickiness strategy. We recommend that you deploy these templates in a lab environment to test each approach.

Download sample code

The download includes these templates:

- `basic.yml` – Sets up an Application Load Balancer without stickiness.
- `targetgroupstickiness.yml` – Demonstrates stickiness based on target groups.
- `stickysessionslb.yml` – Demonstrates sticky sessions with load balancer generated cookies.
- `stickysessionsapp.yml` – Demonstrates sticky sessions with application-based cookies.

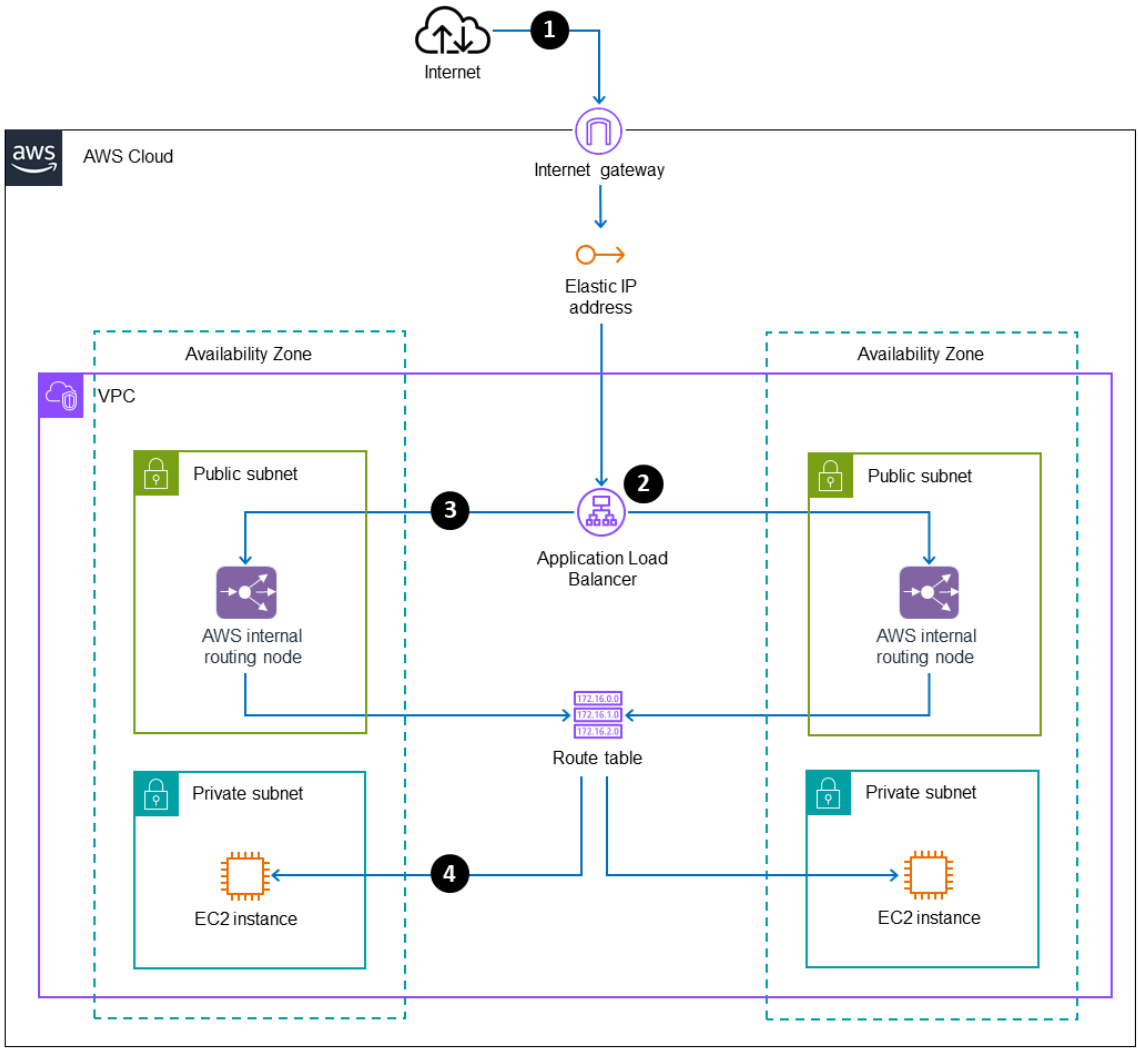
To deploy these templates, you will need an active AWS account and access to the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/. For step-by-steps instructions for deploying a CloudFormation template, see Creating a stack in the AWS CloudFormation documentation.

Load balancer subnets and routing

Let's start with an illustration of how traffic flows when you configure an Application Load Balancer that faces the internet, to Amazon Elastic Compute Cloud (Amazon EC2) instances in a private subnet. This architecture reflects best practices when deploying an Application Load Balancer that's open to the internet.

Inbound traffic path

The following diagram illustrates the virtual private cloud (VPC) subnets and routing associated with the incoming traffic flow, with the return traffic removed from the diagram for clarity.

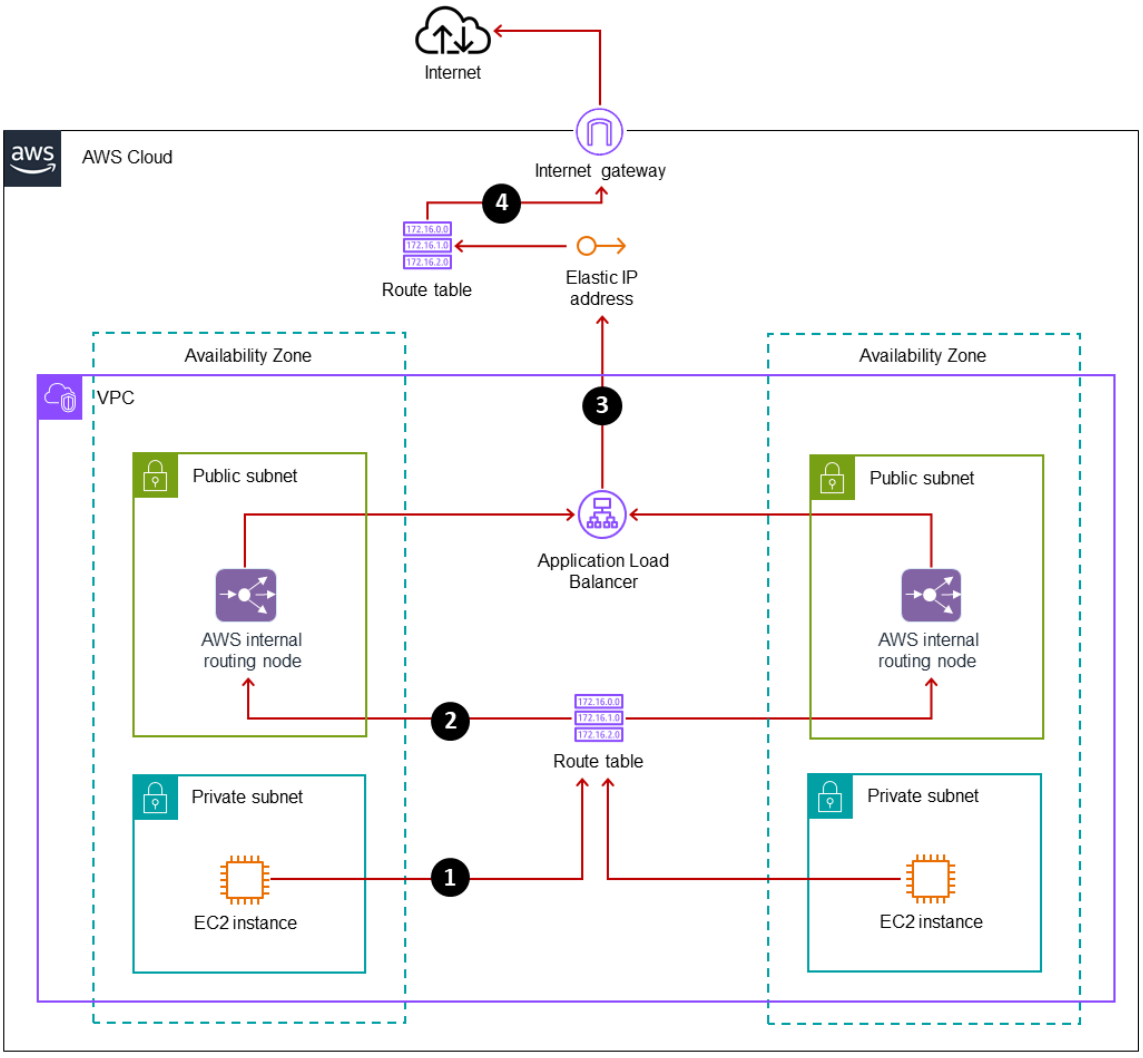


1. Traffic from the internet flows in to the Elastic IP address, which is dynamically created when you deploy an internet-facing Application Load Balancer.

2. The Application Load Balancer is associated with two public subnets in the scenario that's illustrated. Elastic Load Balancing sends traffic through internal nodes in the Availability Zone to determine the appropriate routing logic. The Application Load Balancer uses its internal logic to determine which target group and instance to route the traffic to.
3. The Application Load Balancer routes the request to the EC2 instance through a node that's associated with the public subnet in the same Availability Zone.
4. The route table routes the traffic locally within the VPC, between the public subnet and the private subnet, and to the EC2 instance.

Return traffic path

The following diagram illustrates the VPC subnets and routing associated with the traffic path back out to the internet, with the incoming traffic removed from the diagram for clarity.

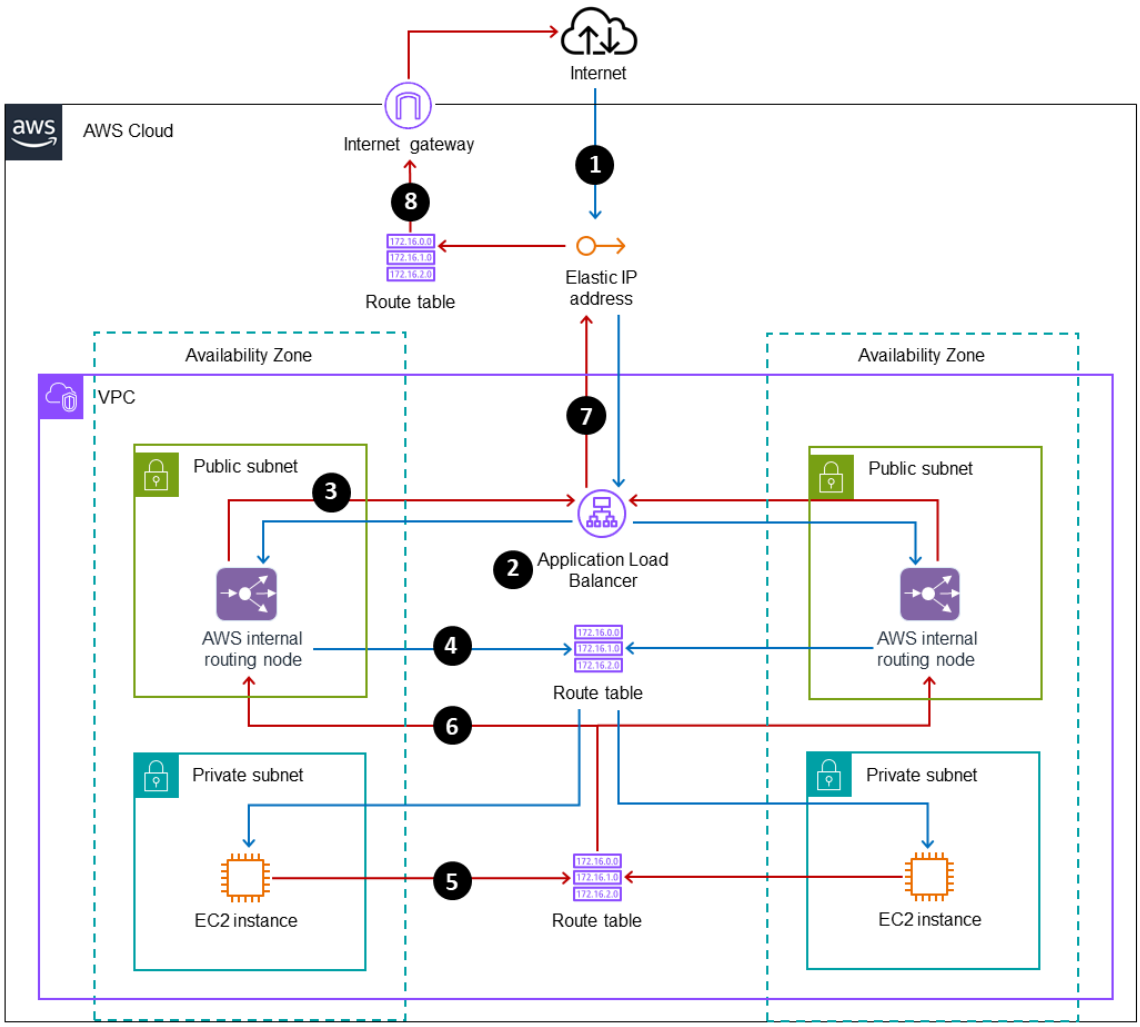


1. The EC2 instance in the private subnet routes the outbound traffic through the route table.
2. The route table has a local route to the public subnet. It reaches the Application Load Balancer on the node in the corresponding public subnet, by following the path back the way the traffic entered.

3. The Application Load Balancer routes traffic out through its public Elastic IP address.
4. The public subnet's route table has a default route pointing to an internet gateway, which routes the traffic back out to the internet.

Complete traffic flow diagram

The following diagram combines the inbound and return traffic flows to provide a complete illustration of load balancer routing.



1. Traffic from the internet flows in to the Elastic IP address, which is dynamically created when you deploy an internet-facing Application Load Balancer.
2. The Application Load Balancer is associated with two public subnets in the scenario that's illustrated. The Application Load Balancer uses its internal logic to determine which target group and instance to route the traffic to.
3. The Application Load Balancer routes the request to the EC2 instance through a node that's associated with the public subnet in the same Availability Zone.
4. The route table routes the traffic locally within the VPC, between the public subnet and the private subnet, and to the EC2 instance.

5. The EC2 instance in the private subnet routes the outbound traffic through the route table.
6. The route table has a local route to the public subnet. It reaches the Application Load Balancer on the node in the corresponding public subnet, by following the path back the way the traffic entered.
7. The Application Load Balancer routes traffic out through its public Elastic IP address.
8. The public subnet's route table has a default route pointing to an internet gateway, which routes the traffic back out to the internet.

Which stickiness strategy is right for you?

Answering the following questions will help you determine your load balancer stickiness strategy.

Are you using WebSockets?

- Yes: WebSockets are inherently sticky, so there is no need to use any strategy.
- No: Continue to the next question.

Are you planning a blue/green deployment?

- Yes: At the minimum, use target group stickiness, but continue to the next question.
- No: Continue to the next question.

Do you need some or all the traffic from a client to be routed to the same EC2 instance repeatedly?

- Yes: Continue to the next question.
- No: You do not need to use sticky sessions.

Do you want your application code or client to control the state attributes and duration by using cookies?

- Yes: Use application-based cookies to enable application-based sticky sessions.
- No: Use load balancer generated cookies to enable duration-based sticky sessions.

Application Load Balancer without stickiness

When you use an Application Load Balancer without any form of stickiness, by default, the load balancer uses the round robin method to determine the EC2 instance it should route traffic to.

Template: Use the AWS CloudFormation template `basic.yml` (included in the sample code .zip file) to try out this functionality.

> **Note**
> All CloudFormation templates included with this guide deploy a custom VPC, route tables, routes, an internet gateway, an Application Load Balancer, target groups, listeners, and EC2 instances, to illustrate a specific load balancer stickiness strategy.

Common use cases

Use an Application Load Balancer without stickiness in these scenarios:

- You have a list of targets to route traffic to, but the targets do not maintain session state.
- You're using web servers that do not maintain session state.
- You're using application servers that do not maintain session state.

Steps

Notes

- NAT gateways incur a small cost.
- Multiple EC2 instances use up your free tier hours faster than a single EC2 instance.

1. Deploy the CloudFormation template `basic.yml` in a lab environment.
2. Wait until the health status of your target group instances changes from **initial** to **healthy**.
3. Navigate to the Application Load Balancer URL in a web browser, using HTTP (TCP/80).

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

   The webpage displays **Instance 1 - TG1** or **Instance 2 - TG1**.
4. Refresh the page multiple times.

Expected results

The instance that loads the web page (**Instance 1** or **Instance 2**) should change every time, as reflected in the page text. The load balancer logic manages the last target across multiple internal nodes, which may introduce a synchronization delay, so there's a possibility that you will be routed to the same target.

How it works

- In this example, two EC2 instances are assigned to a single target group. The EC2 instances have an Apache web server (`httpd`) installed, and the `index.html` page text on each EC2 instance is hardcoded to identify that instance.
- The Application Load Balancer runs its internal round robin logic to determine which EC2 instance should receive the traffic.
- Each time you reload the web page, the Application Load Balancer runs its routing logic, and the page displays **Instance 1 - TG1** or **Instance 2 - TG1**.

Target group stickiness

When you use an Application Load Balancer with target group stickiness:

- The Application Load Balancer uses the target group weight to determine how to balance the incoming traffic between the target groups.
- By default, the Application Load Balancer uses the round robin method to route requests to the EC2 instances in the destination target group.

Template: Use the AWS CloudFormation template `targetgroupstickiness.yml` (included in the sample code .zip file) to try out target group stickiness.

Common use cases

Use target group stickiness in these scenarios:

- There are multiple target groups assigned to the load balancer, and the traffic from a client should be consistently routed to instances within that target group.
- Blue/green deployments.

Code changes from basic.yml

A single change has been made to the listener: We modified the Application Load Balancer default actions to specify two target groups (`TG1` and `TG2`) of equal weight, with a stickiness configuration.

basic.yml

```
Listener1:
  Type:
 'AWS::ElasticLoadBalancingV2::Listener'
  Properties:
    LoadBalancerArn: !Ref ALB
    Protocol: HTTP
    Port: 80
    DefaultActions:
      - TargetGroupArn: !Ref TG1
        Type: forward
```

targetgroupstickiness.yml

```
Listener1:
  Type:
 'AWS::ElasticLoadBalancingV2::Listener'
  Properties:
    LoadBalancerArn: !Ref ALB
    Protocol: HTTP
    Port: 80
    DefaultActions:
      - ForwardConfig:
          TargetGroups:
            - TargetGroupArn: !Ref TG1
              Weight: 1
            - TargetGroupArn: !Ref TG2
              Weight: 1
          TargetGroupStickinessConfig:
              DurationSeconds: 10
              Enabled: true
        Type: forward
```

Steps

Notes

- NAT gateways incur a small cost.
- Multiple EC2 instances use up your free tier hours faster than a single EC2 instance.

1. Deploy the CloudFormation template `targetgroupstickiness.yml` in a lab environment.
2. Wait until the health status of your target group instances changes from **initial** to **healthy**.
3. Navigate to the Application Load Balancer URL in a web browser, using HTTP (TCP/80).

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

   The webpage displays one of the following: **Instance 1 - TG1**, **Instance 2 - TG1**, **Instance 3 - TG2**, or **Instance 4 - TG2**.
4. Refresh the page multiple times.

Expected results

Note
The CloudFormation template in this example configures the stickiness to last 10 seconds.

The instances that load the web page should stay within the target group (TG1 or TG2) within the 10-second duration, as reflected in the page text.

After approximately 10 seconds, the stickiness is released and the target group instance set might change.

How it works

- In this example, four EC2 instances are split across two target groups, with two instances per target group. The EC2 instances have an Apache web server (`httpd`) installed, and the `index.html` page text on each EC2 instance is hardcoded to be distinct.
- The Application Load Balancer creates a binding for the user's session toward the destination target group, with an expiration time.
- When you reload the page, the Application Load Balancer checks whether the binding exists and has not expired.
  - If the binding has expired or doesn't exist, the Application Load Balancer runs its routing logic and determines the destination target group.
  - If the binding has not expired, the Application Load Balancer routes traffic to the same target group, but not necessarily to the same EC2 instance.

Sticky sessions with load balancer generated cookies

When you use an Application Load Balancer with a load balancer generated cookie:

- The Application Load Balancer uses the target group weight to determine how to balance the incoming traffic between the target groups.
- By default, the Application Load Balancer uses the round robin method to route requests to the EC2 instances in the destination target group.

  After traffic has been initially routed to an instance, subsequent traffic will stick to that EC2 instance for a specified duration.

Template: Use the AWS CloudFormation template `stickysessionslb.yml` (included in the sample code .zip file) to try out sticky sessions with load balancer generated cookies.

Common use cases

Use sticky sessions with load balancer generated cookies in these scenarios:

- PHP web servers
- Servers that maintain temporary session data such as logs, shopping carts, or chat conversations

Code changes from basic.yml

The relevant code changes are in the target group configuration, to set the stickiness type to `lb_cookie` and the duration to 10 seconds.

basic.yml

```
TG1:
   Type:
 'AWS::ElasticLoadBalancingV2::TargetGroup'
   Properties:
     Name: TG1
     Protocol: HTTP
     Port: 80
     TargetType: instance
     Targets:
       - Id: !Ref Instance1
       - Id: !Ref Instance2
VpcId: !Ref CustomVPC
```

stickysessionslb.yml

```
TG1:
   Type:
 'AWS::ElasticLoadBalancingV2::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: lb_cookie
       - Key:
 stickiness.lb_cookie.duration_seconds
Value: 10
```

Steps

Notes

- NAT gateways incur a small cost.
- Multiple EC2 instances will use up your free tier hours faster than a single EC2 instance.

1. Deploy the CloudFormation template `stickysessionslb.yml` in a lab environment.
2. Wait until the health status of your target group instances changes from **initial** to **healthy**.
3. Navigate to the Application Load Balancer URL in a web browser, using HTTP (TCP/80).

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

   The webpage displays one of the following: **Instance 1 - TG1**, **Instance 2 - TG1**, **Instance 3 - TG2**, or **Instance 4 - TG1**.
4. Refresh the page multiple times.

Expected results

Note
The Cloudformation template in this example configures the stickiness to last 10 seconds.

The instance that loads the web page should stay the same within the 10-second duration, as reflected in the page text. After approximately 10 seconds, the stickiness is released and the destination instance might change.

How it works

- In this example, two EC2 instances are present in one target group. The EC2 instances have an Apache web server (`httpd`) installed, and the `index.html` page text on each EC2 instance is hardcoded to be distinct.
- The Application Load Balancer creates a binding for the user's session, which binds toward the destination, with an expiration time.
- When you reload the page, the Application Load Balancer checks whether the binding exists and has not expired.
  - If the binding has expired or doesn't exist, the Application Load Balancer runs its routing logic and determines the destination instance.
  - If the binding has not expired, the Application Load Balancer routes traffic to the same destination instance.

Sticky sessions with application-based cookies

When you use an Application Load Balancer with an application-based cookie:

- The Application Load Balancer uses the target group weight to determine how to balance the incoming traffic between the target groups.
- By default, the Application Load Balancer uses the round robin method to route requests to the EC2 instances in the destination target group.
- After traffic has been initially routed to an EC2 instance, the EC2 instance application response should contain a custom application cookie, which is sent back to the client along with an automated Application Load Balancer cookie.
- Subsequent traffic will stick to the EC2 instance if the client sends back the application cookie and the Application Load Balancer cookie.
- The application-based cookie expires after non-use for the configured duration.

Template: Use the AWS CloudFormation template `stickysessionsapp.yml` (included in the sample code .zip file) to try out sticky sessions with application-based cookies.

Common use cases

Use sticky sessions with application-generated cookies when you want additional control in these scenarios:

- PHP web servers
- Servers that maintain temporary session data such as logs, shopping carts, or chat conversations

Code changes from basic.yml

The only code change is in the target group configuration. We added a stickiness configuration to the Application Load Balancer and the target group attributes. The application cookie duration is specified, and the target group has application cookie stickiness enabled.

basic.yml	**stickysessionsapp.yml**

`TG1:`
`  Type:`
`'AWS::ElasticLoadBalancingV2::TargetGroup'`
`  Properties:`
`  Name: TG1`
`  Protocol: HTTP`
`  Port: 80`
`  TargetType: instance`
`  Targets:`
`  - Id: !Ref Instance1`
`  - Id: !Ref Instance2`
	`TG1:`
	`  Type:`
	`'AWS::ElasticLoadBalancingV2::TargetGroup'`
	`  Properties:`
	`  Name: TG1`
	`  Protocol: HTTP`
	`  Port: 80`
	`  TargetType: instance`
	`  Targets:`
	`  - Id: !Ref Instance1`
	`  - Id: !Ref Instance2`

```
VpcId: !Ref CustomVPC
```

```
   VpcId: !Ref CustomVPC
   TargetGroupAttributes:
     - Key: stickiness.enabled
       Value: true
     - Key: stickiness.type
       Value: app_cookie
     - Key:
stickiness.app_cookie.duration_seconds
       Value: 10
     - Key:
stickiness.app_cookie.cookie_name
       Value: TESTCOOKIE
```

Steps

Notes

- NAT gateways incur a small cost.
- Multiple EC2 instances will use up your free tier hours faster than a single EC2 instance.

1. Deploy the CloudFormation template `stickysessionslb.yml` in a lab environment.
2. Wait until the health status of your target group instances changes from **initial** to **healthy**.
3. Navigate to the Application Load Balancer URL in a web browser, using HTTP (TCP/80).

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

   The webpage displays one of the following: **Instance 1 - TG1**, **Instance 2 - TG1**, **Instance 3 - TG2**, or **Instance 4 - TG2**.
4. Refresh the page multiple times.

Expected results

Note
The Cloudformation template in this example has configured the stickiness to last 10 seconds. Valid stickiness duration configuration is between 1 second and 1 week.

The instance that loads the web page should stay the same, as indicated by the page text.

The stickiness duration doesn't refresh, but is based on the expiration configured in the Application Load Balancer for the application cookie that is generated by the EC2 instance.

Example 1: Wait 5 seconds to refresh the page. The same instance will load and the stickiness will be refreshed for another 10 seconds.

Example 2: Wait longer than 10 seconds to reload the page. The application cookie expires, and you are routed to a different EC2 instance. This new instance generates another application cookie with a 10-second duration.

How it works

- In this example the EC2 instances have an Apache web server (`httpd`) installed. The `httpd.conf` file is configured to return a static `Set-Cookie` value back to the client (your web browser). The `Set-Cookie` value is hardcoded to be `TESTCOOKIE=<somevalue>`.

- Open your browser's **Inspect Element** option, choose the **Network** tab, and then choose the **Get** method, which loads the page. You will see a **Cookies** tab.
- The browser is a client application that is automatically configured to return any subsequent updates to the server with the cookies it receives in the server's `Set-Cookie` response.
- When you reload the page, the cookies received in the initial page load are automatically sent back to the Application Load Balancer.
  - If the cookie has expired (that is, 10 seconds have elapsed since you placed the last call), the Application Load Balancer uses new logic to determine which EC2 instance to route traffic to.
  - If the cookie has not expired, the Application Load Balancer routes traffic to the same EC2 instance.

Resources

- Sticky sessions for your Application Load Balancer (Elastic Load Balancing documentation)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Updates and corrections (p. 17)	Updated the code, diagrams, and examples.	May 31, 2023
Updated sections (p. 17)	Updated the *How it works* sections in Target group stickiness and Sticky sessions with load balancer generated cookies to provide more accurate and detailed information.	July 18, 2022
— (p. 17)	Initial publication	August 5, 2021

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Networking terms

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

endpoint

See service endpoint (p. 18).

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Managing AWS Regions in *AWS General Reference*.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.