Day10 of learning AWS

15 February 2025 04:34 PM

Understanding Load Balancer Attributes

In the realm of load balancing, it's essential to grasp the various attributes that govern its behavior. Let's explore key attributes and their use cases, shedding light on how these settings can optimize the performance and security of your load balancer.

Key Attributes and their Meanings:

1. **deletion_protection.enabled:**
   - Determines if deletion protection is enabled.
   - Default: false.
2. **load_balancing.cross_zone.enabled:**
   - Manages cross-zone load balancing.
   - Default: Network and Gateway Load Balancers - false, Application Load Balancers - true (unchangeable).
3. **access_logs.s3.enabled:**
   - Activates access logs for tracking requests.
   - Default: false.
4. **access_logs.s3.bucket:**
   - Specifies the S3 bucket for access logs (required if enabled).
5. **access_logs.s3.prefix:**
   - Defines the prefix for access log location.
6. **ipv6.deny_all_igw_traffic:**
   - Blocks internet gateway access for security.
   - Default: Internet-facing - false, Internal - true.

Attributes for Application Load Balancers:

7. **idle_timeout.timeout_seconds:**
   - Sets the idle timeout value in seconds.
   - Default: 60 seconds.
8. **connection_logs.s3.enabled:**
   - Enables connection logs.
   - Default: false.
9. **connection_logs.s3.bucket:**
   - Specifies S3 bucket for connection logs (required if enabled).
10. **connection_logs.s3.prefix:**
    - Defines the prefix for connection log location.
11. **routing.http.desync_mitigation_mode:**
    - Manages handling of potentially risky requests.
    - Default: defensive.
12. **routing.http.drop_invalid_header_fields.enabled:**
    - Removes or routes headers with invalid fields.
    - Default: false.
13. **routing.http.preserve_host_header.enabled:**
    - Preserves the Host header in HTTP requests.
    - Default: false.
14. **routing.http.x_amzn_tls_version_and_cipher_suite.enabled:**
    - Adds headers with TLS version and cipher suite info.
    - Default: false.
15. **routing.http.xff_client_port.enabled:**
    - Determines if X-Forwarded-For should preserve client source port.
    - Default: false.
16. **routing.http.xff_header_processing.mode:**
    - Modifies, preserves, or removes X-Forwarded-For header.
    - Default: append.
17. **routing.http2.enabled:**
    - Enables or disables HTTP/2.
    - Default: true.
18. **waf.fail_open.enabled:**
    - Decides whether a WAF-enabled load balancer should route requests if unable to forward to AWS WAF.
    - Default: false.

Attributes for Network Load Balancers:

19. **dns_record.client_routing_policy:**

- Dictates traffic distribution among Availability Zones.
- Possible values: availability_zone_affinity, partial_availability_zone_affinity, any_availability_zone.

20. **Type (String):**
    - Specifies the type with length constraints (max 256 characters).
    - Pattern: ^[a-zA-Z0-9._]+$

These attributes empower you to tailor your load balancer configuration, ensuring it aligns with the specific requirements of your applications. Whether it's enhancing security, optimizing performance, or distributing traffic strategically, understanding and utilizing these attributes is key to harnessing the full potential of your load balancer.

Nginx Configuration to get client IP address in logs

```
log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                '$status $body_bytes_sent "$http_referer" '
                '"$http_user_agent" "$http_x_forwarded_for"';
```

Stickiness in Load Balancers

Introduction:

Load balancers play a crucial role in distributing incoming network traffic across multiple servers to ensure optimal resource utilization and prevent server overload. One key feature in load balancing is "stickiness." In this blog post, we'll unravel the concept of stickiness and provide a straightforward guide on configuring it in an Application Load Balancer (ALB).

Understanding Stickiness:

Stickiness, also known as session affinity, is the ability of a load balancer to bind a user's session to a specific server. This ensures that subsequent requests from the same user are directed to the same server that initially handled their request. Stickiness is particularly useful for applications that store session data locally on a server.

Configuration in Application Load Balancer (ALB): Configuring stickiness in an ALB involves a few steps. Let's break it down:

1. **Navigate to the AWS Management Console:**
   - Log in to your AWS account and go to the EC2 dashboard.
2. **Select the Load Balancer:**
   - Locate your Application Load Balancer in the Load Balancers section.
3. **Configure Stickiness:**
   - In the ALB details, find the "Listeners" tab.
   - Edit the listener for which you want to enable stickiness.
4. **Enable Stickiness:**
   - Check the box for "Enable Load Balancer Generated Cookie Stickiness."
   - Adjust the "Stickiness Duration" if needed. This determines how long the association between a user and a specific server lasts.
5. **Save Changes:**
   - Save your changes, and the ALB will now use stickiness based on the configured settings.

Example Use Case: Let's consider a scenario where an e-commerce application uses stickiness. When a user logs in and adds items to their cart, their session data is stored on the server handling the request. With stickiness enabled, subsequent requests from the same user will be directed to the same server, ensuring a seamless and consistent shopping experience.