AWS-Differences

Siddesh Mandhare || 22/09/24

1) Difference between AWS DNS vs AWS Route 53

Feature/Type	AWS DNS (General)	AWS Route 53
Service Type	General DNS services provided by AWS	Managed DNS and domain registration service
Domain Registration	Not directly supported	Yes, allows domain registration and management
Traffic Management	Basic DNS functionality	Advanced traffic routing features, including geo-routing, latency-based routing, and weighted routing
Health Checks	Limited health checking options	Supports health checks to route traffic only to healthy endpoints
Integration with Other AWS Services	Basic integration	Deep integration with other AWS services (e.g., Elastic Load Balancing, CloudFront)
DNS Record Types	Supports standard DNS records (A, CNAME, MX, etc.)	Supports all standard DNS record types, plus alias records that point to AWS resources
DNS Query Logging	Not available	Yes, allows detailed logging of DNS queries for analysis
Latency	Dependent on the underlying DNS resolver	Uses a global network of DNS servers to reduce latency
DNS Failover	Limited to manual configurations	Automatic DNS failover to healthy resources
Pricing Model	Standard pricing based on usage	Pricing based on the number of hosted zones, DNS queries, and health checks
User Interface	Basic interface through AWS Management Console	Feature-rich management console with detailed configurations
API and SDK Access	Basic DNS APIs	Comprehensive API for DNS management, including batch changes and programmatic access

2) Difference between ALB vs NLB vs CLB vs GWLB

Feature/Type	Application Load Balancer (ALB)	Network Load Balancer (NLB)	Classic Load Balancer (CLB)	Gateway Load Balancer (GWLB)
Layer	Layer 7 (Application Layer)	Layer 4 (Transport Layer)	Layer 4 and Layer 7	Layer 3 (Network Layer)
Traffic Type	HTTP/HTTPS	TCP/UDP	HTTP/HTTPS and TCP	TCP
Routing	Advanced routing based on URL, headers, etc.	Basic round-robin routing	Basic round-robin routing	Transparent routing to virtual appliances
WebSocket Support	Yes	No	Limited support	No
SSL Termination	Yes	No	Yes	No
Health Checks	Application-level health checks	TCP-level health checks	Both application and TCP health checks	Health checks for virtual appliances
Use Cases	Microservices, web applications, APIs	High-performance applications, real-time streaming	Legacy applications, simple load balancing	Network appliance deployment and management
IP Addressing	Uses private IPs for targets	Can handle millions of requests using static IPs	Uses public and private IPs	Uses private IPs for appliances
Session Stickiness	Yes (using cookies)	No	Yes (source IP)	No
Scaling	Auto-scaling based on demand	Handles sudden spikes without performance loss	Limited auto-scaling capabilities	Scales based on the number of appliances
Cost Structure	Based on data processed and number of requests	Based on data processed and number of connections	Based on data processed and number of connections	Based on data processed and appliances deployed

3) Difference between NGinx web server vs Apache web server

Feature/Type	Nginx	Apache
Architecture	Nginx uses an event-driven, asynchronous architecture. This allows it to handle many connections simultaneously with low resource consumption. Each request is handled in a single thread, reducing overhead.	Apache uses a process-driven architecture with multi-threading options. Each request typically requires a separate process or thread, which can lead to higher memory usage, especially under heavy load.
Performance	Nginx excels in high-performance scenarios, particularly with static content. Its efficient handling of multiple connections makes it suitable for high-traffic websites.	Apache performs well but may struggle under heavy load due to its process/thread model, which can lead to higher CPU and memory usage.
Handling Static Content	Nginx is optimized for serving static files (HTML, CSS, JS) quickly and efficiently. It can serve these files directly from the filesystem with minimal overhead.	Apache can serve static content, but it generally has a higher overhead compared to Nginx. It's less efficient for large numbers of static file requests.
Handling Dynamic Content	Nginx typically acts as a reverse proxy for dynamic content. It forwards requests to backend servers (e.g., PHP-FPM, Node.js) for processing, which can improve performance.	Apache can handle dynamic content directly using various modules (e.g., mod_php for PHP), making it a good choice for applications tightly integrated with dynamic scripting.
Configuration	Nginx has a straightforward and concise configuration syntax, making it easier to read and maintain. It uses a single configuration file and includes directives that are intuitive.	Apache's configuration can be more complex, using a variety of configuration files (.htaccess, httpd.conf). It offers extensive options but may be harder to manage for large applications.
Load Balancing	Nginx has built-in load balancing capabilities, allowing it to distribute incoming traffic across multiple servers efficiently. It supports various algorithms like round-robin and least connections.	Apache can perform load balancing, but it often requires additional modules (e.g., mod_proxy_balancer) and configuration, making it less seamless than Nginx.
Module Support	Nginx has a smaller selection of third-party modules, but it includes built-in support for essential features like proxying and caching. Modules must be compiled with Nginx, limiting dynamic module loading.	Apache has a vast ecosystem of modules for a wide range of functionalities, including authentication, caching, and security. Modules can often be loaded dynamically without recompiling the server.

Reverse Proxy	Nginx is highly efficient as a reverse proxy server, allowing it to forward requests to various backend applications while handling SSL termination and caching.	Apache can also function as a reverse proxy, but it may not perform as efficiently as Nginx under high loads, especially when handling many simultaneous connections.
SSL Support	Nginx provides robust SSL/TLS support with straightforward configuration for HTTPS. It also includes features like OCSP stapling for better security.	Apache supports SSL/TLS, but configuration can be more complex. It offers good SSL features but may not be as straightforward as Nginx.
OS Compatibility	Nginx is cross-platform and can run on various operating systems, including Linux, Windows, and macOS.	Apache primarily runs on Unix-like systems (Linux, BSD) but also supports Windows, making it versatile for various environments.
Resource Consumption	Nginx is known for its low resource consumption, especially under high loads. It can efficiently manage thousands of concurrent connections without significant memory overhead.	Apache can consume more memory, particularly with many concurrent requests. Its process/thread model can lead to increased resource usage compared to Nginx.
Community and Support	Nginx has a growing community with good documentation and support resources, including forums and commercial support options.	Apache has a long-established community with extensive documentation and a wealth of online resources. It is widely used and supported.
Use Cases	Ideal for high-traffic websites, content delivery networks (CDNs), microservices architecture, and applications requiring efficient static content serving.	Versatile and suitable for various web applications, particularly those using content management systems (CMS) like WordPress, Drupal, or applications that require extensive module support.
Caching	Nginx has built-in caching mechanisms that can significantly improve performance for both static and dynamic content.	Apache supports caching through additional modules (e.g., mod_cache) but may not be as efficient or easy to configure as Nginx.
WebSocket Support	Nginx provides excellent support for WebSockets, making it suitable for real-time applications.	Apache has limited WebSocket support and may require additional configuration to work effectively with WebSocket connections.

4) Difference between NACL vs Security Group

Feature	AWS Network ACL (NACL)	AWS Security Group
Level of Operation	Operates at the subnet level	Operates at the instance level
Default Behaviour	Stateless; rules must be defined for both inbound and outbound traffic	Stateful; automatically allows return traffic for allowed inbound requests
Rule Evaluation	Evaluates rules in numerical order, and processes all rules	Evaluates all rules at once; if any rule allows the traffic, it is permitted
Allow/Deny Rules	Can have both allow and deny rules	Only allows allow rules; no explicit deny rules
Number of Rules	Can have up to 20 rules (can be increased to 40)	Can have up to 60 rules (can be increased to 120)
Traffic Type	Suitable for both inbound and outbound traffic	Primarily used for inbound traffic but manages outbound as well
Associated Resources	Can be associated with multiple subnets	Can be associated with multiple instances (or ENIs)
IP Address Range	Supports CIDR blocks for more granular control	Works primarily with individual IP addresses or CIDR blocks
Logging	Can be logged using AWS CloudTrail and VPC Flow Logs	Does not have built-in logging; relies on other AWS services for logging
Use Case	Ideal for broad traffic control at the subnet level, such as denying or allowing traffic from specific IP ranges	Ideal for fine-grained access control at the instance level, such as allowing access to specific ports for specific IPs
Complexity	Generally, more complex to manage due to stateless nature and rule order	Simpler to manage due to stateful nature and grouped rules
Support for IPv6	Yes	Yes