Architecture Considerations for Migrating Load Balancers to AWS

September 2020

Notices

Contents

Abstract

Migrating applications from on-premises data centers to Amazon Web Services (AWS) can represent a fundamental shift in the way your company approaches architecture, developing, and operating applications. This whitepaper outlines the best practices and considerations for migrating load balancers to AWS, and designing new solution architectures to meet the needs of both your application and your company.

Introduction

The term *load balancing* refers to the practice of distributing requests across multiple application nodes with the objective of maximizing resource usage and increasing overall application performance. A *load balancer* is a piece of software, usually running on a dedicated node, that receives incoming requests and distributes them to application nodes. For cloud-native architectures where application nodes change often, load balancers represent a critical infrastructure component that should be carefully architected in order to ensure the performance, security, and availability of your application.

Load balancers are often presented as the entry point to an application, and they typically offer more functionality than simple request routing. In both traditional and cloud-native architectures, load balancers contain features such as performance monitoring, response caching, secure sockets layer (SSL) or transport layer security (TLS) offloading, distributed denial-of-service (DDOS) attack protection, and many others. For companies that are migrating applications to AWS, it is important to consider how the application's current load balancer features will be delivered after the application has been migrated. Additionally, it is important to consider additional features that may be needed as a result of the migration, for example, automatic registration of targets in an Auto Scaling group.

Most load balancing solutions on AWS leverage Elastic Load Balancing (ELB), which provides three types of fully managed load balancers designed to meet different requirements. In addition to managed load balancers, companies can opt to configure custom load balancers running on AWS compute resources, such as Amazon Elastic Compute Cloud (Amazon EC2) or Amazon Elastic Container Service (ECS) with AWS Fargate. Designing an effective and efficient load balancer solution involves evaluation of a number of technical, architectural, and organizational considerations. This whitepaper will discuss these considerations and highlight services offered by AWS that can be used to design load balancer architectures for any type of application.

Load Balancer Options

On AWS, most load balancer architectures use one of the three ELB services:

- Application Load Balancer (ALB): A Layer 7 load balancer that is best suited for load balancing of HTTP/HTTPS traffic and inspecting client requests.
- Network Load Balancer (NLB): A Layer 4 load balancer that is best suited for TCP, UDP, and TLS traffic where extreme performance is required.
- Classic Load Balancer (CLB): Provides basic load balancing across multiple Amazon EC2 instances. Intended for applications that were built within the EC2-Classic network rather than Amazon Virtual Private Cloud (Amazon VPC). If EC2-Classic is not required, it is recommended to use either an ALB or NLB to leverage the features not available on CLBs.

For a complete list of features and feature comparison between each type of ELB, see Elastic Load Balancing Product Comparisons.

ELBs are managed services that provide powerful, feature-rich load balancing solutions with simple configuration. ELBs are fully managed services, meaning that application owners do not have access to the underlying compute resources or operating systems powering the load balancing nodes. For companies migrating existing load balancer architectures to AWS, deprecating existing load balancing software and infrastructure in favor of ELBs can offer the following benefits:

- Simple, flexible configuration and support for multiple types of application targets. For more information on ELB basics, see Getting Started with Elastic Load Balancing.
- Common load balancer features such as health checks and sticky sessions, as well as a number of advanced options. For a full list of supported ELB features, see Elastic Load Balancing features.
- Built-in high availability and auto-scaling based on application traffic levels.
- Simplified integration with other core AWS services, such as Amazon CloudWatch for monitoring and metrics and Security Groups for network reachability.

Load Balancer Architecture Strategies

Layer 4 vs. Layer 7 Load Balancers

Layer 4 *transport layer* and Layer 7 *application layer* refer to abstractions outlined in the Open Systems Interconnection model (OSI model). Fundamentally, a load balancer can be classified as Layer 4 or Layer 7 based on the type of information the load balancer processes in order to make a decision on traffic routing. For example, HTTP headers are only available at Layer 7 of the OSI model, and therefore, a Layer 7 load balancer must be used if HTTP headers (URL, method, cookies, and etc.) are to be considered in routing policies. Information such as IP address source, IP address target, port, and protocol are available at Layer 4 of the OSI model, and can be used with a Layer 4 load balancer. On AWS, ALBs are Layer 7 load balancers, NLBs are Layer 4 load balancers, and custom load balancers can be either Layer 4 or Layer 7.

An NLB makes idempotent routing decisions for each received packet based on information available at Layer 4. Additionally, NLBs are pass-through load balancers, meaning the connection is bound between the client and load balancer target, not between the client and the load balancer node. Generally, NLBs provide lower latency than ALBs because they process less information when making routing decisions; however, overall application performance can vary depending on total information processing requirements. For example, if an application is using an NLB to reduce latency, but performing HTTP header inspection on the application nodes, overall application performance may be increased by using an ALB-based solution and performing HTTP header inspection directly on the load balancer nodes.

ALBs make routing decisions based on information available in Layer 4, Layer 7, or both. Different from NLBs, ALBs are proxy load balancers, meaning that ALBs maintain two separate connections: one between the client and the load balancer node, and one between the load balancer node and the load balancer target.

When choosing between an NLB or an ALB, or a Layer 4 or Layer 7 custom load balancer, there are many architectural considerations that will be discussed in more detail in the next sections. However, in general, choosing between a Layer 4 or Layer 7 load balancer is ultimately based on the location of the information used to route requests to downstream targets. If an application requires inspection of Layer 7 information before reaching the load balancer target, an ALB or other Layer 7 load balancer is required. If an application relies solely on inspection of information present in Layer 4, then an NLB or another Layer 4 load balancer can be used.

Custom Load Balancers

In addition to using the managed services offered by ELB, companies can optionally configure a custom load balancing solution. Custom load balancing solutions can use any load balancing software, and many popular third-party load balancing solutions such as F5 BIG-IP and NGINX Plus are available as Amazon Machine Images (AMI) on the AWS Marketplace. Custom load balancing solutions can be run on Amazon EC2 instances, or can be containerized and run on ECS or another container orchestration solution, if desired. Running an EC2-based or containerized load balancer is a good option for organizations looking to *lift and shift* existing, on-premises load balancers without deprecating these load balancers in favor of ELBs. This can be useful if existing load balancers have complex or opaque configurations, or an application has a requirement to work with a specific, third-party load balancing software.

With custom load balancing solutions, companies can gain the ability to exert fine-grained control over their load balancer nodes, but take on the additional operational burden of architecting for high availability, scaling, monitoring, provisioning, and other features which are otherwise handled by ELBs. Throughout this whitepaper, custom load balancers are suggested as potential solutions for applications with requirements that cannot be met by ELBs. For example, an application that requires a load balancer that makes use of operating system customizations cannot use ELB because ELB does not expose the ability to customize the underlying compute resource powering a load balancer node. In this example, a company migrating this application to AWS could use a third-party load balancing software, such as NGINX, deployed on a fleet of customized EC2 instances.

Deploying a custom load balancing solution gives companies the flexibility to move any type of load balancer architecture to AWS, regardless of the degree of customization. While a custom solution can be flexible, it is usually more complex to replicate the *boilerplate* load balancing features offered by ELB. Companies evaluating a custom load balancing solution should consider the level of effort required to adopt an ELB-based solution against the level of effort required to operate and maintain a custom load balancing solution.

API Gateway vs. Load Balancer

An API Gateway refers to API management software that is deployed in front of a collection of backend services. Requests to an application are routed through the API Gateway, and the API Gateway provides common features such as authentication/authorization, request throttling, caching, and more. API Gateways can be implemented as a custom solution or a managed service. Using an API Gateway introduces the ability to add features to your application without requiring code to be present in the service to support these features. For example, with an API Gateway, you can validate request schemas before passing the request to a backend service, thereby removing the need for the backend service to validate the incoming request schema.

Amazon API Gateway is a fully managed AWS service that simplifies the process of creating and managing REST APIs at any scale. For a complete list of features, see Amazon API Gateway Features. For companies migrating applications to AWS, the question of whether or not to use an API Gateway or a load balancer is twofold:

- Should I use an API Gateway *in place* of a load balancer?
- Should I use an API Gateway *in front* of a load balancer?

API Gateways and load balancers can both be used to proxy requests to downstream targets; however, these solutions differ in terms of their supported features and intended use cases. API Gateway solutions typically offer different feature sets than load balancing solutions. For Amazon API Gateway, the ability to introduce request validation, rate limiting, API keys, SDK generation, or proxying requests directly to AWS services (e.g. Amazon Kinesis) are all features that are not supported by ELB. Amazon API Gateway does not support the ability to balance requests between a number of downstream targets, meaning, it does not support traditional *load balancing* between multiple application nodes belonging to the same service. Offloading API authentication and authorization is another common use case for API Gateway; however, ALB also supports this feature.

If using AWS Lambda as your main compute provider, you may want to consider using Lambda with Amazon API Gateway in place of an ELB-based solution. With API Gateway and AWS Lambda, you can easily add important features such as authentication/authorization and request validation, making API Gateway a good option for companies building around Lambda as a principal compute provider. If you want to add Lambda to an existing EC2-based web application, using an ELB-based solution is most likely a better fit due to the simplicity of Lambda integration with ALBs.

Using an API Gateway *in place* of a load balancer can be a good fit for applications using Lambda as the main compute provider, or applications that need to proxy requests directly to AWS services or other existing HTTP endpoints. Using API Gateway *in front* of a load balancer can be a good fit for applications that want to leverage API management features offered by API Gateway that are not supported by ELB, but still need to distribute requests across a number of application nodes.

Downstream Targets

One of the most important considerations for designing a load balancer architecture is identifying the compute engine used by downstream targets, as well as determining how these targets will be registered with the load balancer. When a target is registered with a load balancer, the load balancer will start to route client requests to the target. In a cloud-native architecture, the number and location of application nodes changes (e.g. in response to scaling events) more often than in a traditional on-premises architecture. While dynamic application nodes allow applications to be elastic and highly available, organizations migrating applications to AWS will have to consider how their new, dynamic application nodes will be registered and deregistered from load balancers.

ELB uses the concept of a *target group* to route requests to a group of resources. A target group is a pool of similar resources that serve similar client requests. Like most load balancing solutions, ELB performs *health checks* against resources in a target group, and only sends traffic to healthy resources in the target group. A health check is a periodic request sent to each resource in the target group to test the resource's capability to serve client requests. For more information on the type of health checks that can be configured with ELB, see ALB, NLB, or CLB. The table below shows the supported target group types for each category of load balancing solution:

Load Balancing Solution	**Architecture Considerations**
ALB	ALB supports load balancing to IP addresses, EC2 instance IDs, or Lambda functions. For more information, see Target Groups for your ALB.
NLB	NLB supports load balancing to IP addresses, or EC2 instance IDs. For more information, see Target Groups for your NLB.

Load Balancing Solution	Architecture Considerations
CLB	CLB supports load balancing to EC2 Instance IDs. For more information, see Registered Instances for your CLB.
Custom	Custom load balancing solutions can support load balancing to any network reachable node. For example, a NGINX-based load balancer running on EC2 instances is capable of proxying requests to any type of endpoint. The type(s) of supported target group depends on the particular third-party load balancing software that is being used.

Table 1 - Supported target group types

Certain compute resources, such as AWS Lambda, can only be used with ALBs (or an Amazon API Gateway). Lambda is a serverless compute platform and automatically provides scaling and high-availability. Because of this, once a Lambda function is initially registered with an ALB, no additional configuration is required to ensure the Lambda function remains registered with the ALB. For more information on using Lambda with ALB, see Lambda Functions as Targets.

If using a target group relying on EC2 instance IDs, you can attach an Auto Scaling group to your target group. An Auto Scaling group contains a collection of EC2 instances that are treated as a logical group for the purposes of scaling and management. For more information on creating Auto Scaling groups to power your application nodes, see Getting Started with Amazon EC2 Auto Scaling. After you attach a target group to an Auto Scaling group, the instances created or destroyed by the Auto Scaling group are automatically registered or deregistered from the load balancer. Before instances are registered, they must pass the load balancer's configured health checks. If an instance fails the load balancer health check, it will be removed from the target group and replaced. Before instances are deregistered, ELB can optionally perform connection draining to ensure that requests are completed before the instance is deregistered.

ELB can be used with Amazon Elastic Container Service (ECS) to evenly distribute traffic across container instances in your application. When you deploy a container instance to Fargate or EC2, ECS can automatically register the instance with the appropriate load balancer. Once connected to the load balancer, the container is treated similarly to instances in Auto Scaling groups, that is, the container must pass health checks and will be deregistered if health checks fail. For more information on working with load balancers and containerized workloads, see Creating a Load Balancer in ECS. When working with ECS applications, ALB offers several features that are not offered by NLBs or CLBs, making ALB the recommended choice for ECS applications. These features include the ability for ECS and ALB to perform dynamic host port mapping (enabling multiple tasks from the same service to be hosted on a single container instance), and the ability to support path-based routing and rule prioritization (enabling multiple services to use the same listener port on an ALB).

Target groups relying on IP addresses usually require more configuration to ensure they are appropriately registered and deregistered with load balancers. When you specify targets with an IP address, the IP address must not be publicly routable. IP address targets can refer to any AWS resource that is addressable by IP address and port (e.g. databases or EC2 instances), and any on-premises resource linked to AWS through AWS Direct Connect or other software VPN connection. Although ELB will only send traffic to resources that pass health checks, ELB will not automatically register or deregister targets. To manually register a target, applications can use the console, CLI, SDK, or AWS CloudFormation. A common pattern for applications using IP address targets is to use Bootstrapping Container Instances with Amazon EC2 User Data to automatically register EC2 instances with a load balancer when the EC2 instance is created. In addition to user data, companies can use custom scripts to dynamically register or deregister an IP address from an ELB for any reason. Although target groups relying on IP addresses typically require more operational overhead than those relying on EC2 instance IDs and Auto Scaling groups, IP addresses offer more flexibility in terms of supported target types (e.g., on-premises nodes or database instances) and logic for registration and deregistration of targets.

If proxying to EC2, Fargate, Lambda, or IP address targets; both ALBs and NLBs provide full-featured load balancer solutions with simple configuration. Applications that require load balancing to other resources will usually require a custom load balancing solution. For example, if an application requires inspection and proxy of requests directly to another HTTP endpoint without a static IP address, a third-party load balancing software would be required.

Finally, companies still using EC2-Classic can use CLBs for a fully managed load balancing solution; however, CLBs offer significantly less features than ALBs or NLBs, and are generally not recommended for organizations using VPC.

Request Inspection

In order to route a request to its destination, a load balancer needs to inspect a request and compare the request to rules defined on the load balancer. In ELB, *listener rules* are defined for each port and protocol combination used with an ELB and consist of a priority, one or more actions, and one or more conditions. For more information about the types of rules that can be configured on an ELB, see Listener Rules for ALB, Listener Rules for NLB, and Listener Rules for CLB. In addition to basic request inspection and routing, some load balancer architectures have additional requirements for resource pool switching or payload scanning. Payload scanning refers to the practice of searching for patterns in an incoming request and acting on the result(s) of pattern matches. Resource pool switching refers to the practice of changing the destination of a request based on a dynamic value.

Resource pool switching

Some load balancer architectures may require the ability to dynamically change the downstream target of a request in response to request characteristics or application events. For example, consider an application that calls for directing requests larger than a certain size to a separate target group that is optimized to process large requests. When you build an ELB listener rule, you are restricted to the examination of the request attributes exposed by ELB. The available request attributes differ based on the type of ELB (ALB, NLB, or CLB) that you are using. ELB does not support adding new or custom attributes for inspection in listener rules, so if your application requires request attribute inspection beyond the attributes provided by ELB, then a third-party load balancing software that can be further customized, such as NGINX on EC2, is required. In the example of an application directing large requests to a separate target group, ELB does not support inspection of request size, so this application would require a third-party load balancing software that offers this capability.

ELB also does not offer the ability to change a listener rule or target group in response to application events; however, this result can be achieved by integrating with other AWS services depending on the type of event you want to trigger a change to your load balancer. Applications can manually change the priority of an existing load balancer rule, and therefore the location that requests will be routed, via the CLI or SDK. Additionally, an Amazon CloudWatch Event can be used to detect system events and trigger target service. With this method, a CloudWatch Event could be used to trigger a Lambda function that modifies ELB configurations.

Payload scanning

If an application is performing payload scanning, the load balancer scans the entire request (headers and body), for patterns. Once patterns are identified, the load balancer can perform an action on the request based on the type of pattern found. For example, a load balancer could use payload scanning to identify and block any requests that contain a forbidden keyword in the body of the request. Payload scanning is a compute-intensive process, and can be used for many purposes such as, protection against common attack patterns (SQL injection, XSS scripting, etc.), whitelisting and blacklisting requests, enforcing data quality standards, and more.

ELBs do not support payload scanning, and are restricted to inspection of request attributes that are available in the listener rules. Configuring inspection of requests directly on an ELB for the purposes of allowing or denying requests can be a simple, effective strategy for companies without numerous or complex payload scanning requirements; however, this strategy may not be scalable for applications with more sophisticated requirements. In addition to limiting the fields available for inspection, ELBs have limitations for the number of rules that can be directly placed on a load balancer. For more information on the service limitations of ELB, see Quotas for ALB, Quotas for NLB, and Quotas for CLB.

Although ELB does not support payload scanning, AWS WAF - Web Application Firewall can be used with an ALB to inspect incoming HTTP traffic and apply complex custom rules. WAF Rules can be configured to allow or deny a request based on a CIDR block range of IP addresses, the results of regular expression match on certain request attributes, the request size, and more. When used with an ALB, WAF intercepts and evaluates requests before the request is allowed to proceed to the ALB. For companies looking for a managed service solution for payload scanning for the purpose of allowing or denying requests, WAF can be a simple, scalable solution.

Advanced Monitoring

Monitoring client requests is a common feature in load balancers, and can take many forms depending on your application requirements. Load balancer monitoring allows application owners to analyze traffic patterns, troubleshoot issues, and improve application performance and security. Some common monitoring features that are found on load balancers include:

- **Access logs**: capture detailed information (e.g. source IP address, user agents, URLs) about each request made to a load balancer.

- **Performance and application metrics**: capture statistics (e.g. active connection counts, HTTP response code counts, throughput) about an application.
- **HTTP request tracing**: use custom HTTP headers to track requests as they move between clients to targets.
- **Management and governance**: capture detailed information (e.g. user IDs, timestamps) about changes made to load balancer configurations.

The table below highlights how these common use cases are addressed by ELB:

Feature	ALB	NLB	CLB
Access logs	Access Logs your ALB	Access Logs for NLB	Access Logs for CLB
Performance metrics	CloudWatch Metrics for ALB	CloudWatch Metrics for NLB	CloudWatch Metrics for CLB
Request tracing	Request Tracing for ALB	Not supported.	Not supported.
Management and governance	Monitor ALB Using AWS CloudTrail	Monitor NLB Using AWS CloudTrail	Monitor CLB Using AWS CloudTrail

Table 2 - ELB advanced monitoring support

ELB offers powerful, built-in monitoring features. These features enable companies migrating to ELB-based load balancing solutions to easily replicate existing monitoring features with minimal operational overhead, as well as add new monitoring features to address common use cases.

Access Logs

Access logs are easily enabled with ELB, and automatically dump batch log files to an Amazon Simple Storage Service (Amazon S3) bucket every 5 minutes. Depending on the type of ELB (ALB, NLB, or CLB) that you are using, different information will be available in the access log. Access logs can be helpful for troubleshooting network connections, providing advanced metrics about the consumers of an application, and maintaining compliance in security audits. While there is no additional cost for enabling access logs on an ELB, you are charged the storage costs associated with your log files in S3. For this reason, many companies use access logs with S3 lifecycle policies to control the costs associated with access log storage.

Performance and Application Metrics

ELB integrates with Amazon CloudWatch to automatically collect, publish, and view time-series data points for load balancers and targets. Depending on the type of load balancer (ALB, NLB, or CLB) that you are using, different metrics will be collected by CloudWatch. For example, CloudWatch automatically collects the metric `HTTPCode_ELB_5XX_Count` for ALBs, a metric that corresponds to the number of HTTP 5XX error codes returned to clients from the load balancer. For more information about the metrics collected for each type of ELB, see ALB, NLB, or CLB. Using CloudWatch, it is easy to verify that your system is performing as expected. You can also create a CloudWatch Alarm to initiate an action (such as sending an email or alert) if a metric is recorded outside a configured range.

Some load balancer architectures require collecting metrics that are not captured by CloudWatch, for example, an application with a requirement to monitor the number of requests containing a certain HTTP header. This can be useful for companies seeking to monitor authenticated versus unauthenticated requests by examining HTTP `Authorization` or `Cookie` headers, or companies seeking to monitor requests containing a custom, application-specific HTTP header.

Although CloudWatch supports Publishing Custom Metrics, this requires sending the metric to CloudWatch through the AWS CLI or CloudWatch API. Since ELB does not offer the ability to customize the underlying compute resources of an ELB node, you cannot send custom metrics to CloudWatch directly from an ELB node. If your application has a requirement to monitor custom request metrics, there are several ways this can be accomplished:

- **Monitor requests with AWS WAF:** use WAF with an ALB and create a custom WAF rule for each metric that is to be tracked. For example, create a custom rule to search for requests that include a certain value of HTTP header `Custom-App-Header` or a certain regular expression match in the HTTP `Authorization` header. CloudWatch can monitor the number of requests that match a given custom rule. For more information on configuring advanced monitoring with WAF and ALB, see Monitoring WAF with Amazon CloudWatch. This method can be a useful for companies seeking to use ALB for its managed service capability, but also seeking to customize request monitoring capabilities.
- **Custom load balancing solution:** use a third-party load balancer deployed on a fleet of EC2 instances, and configure the load balancer to inspect requests and publish a custom metric to CloudWatch via the CloudWatch API. This method requires adoption of a complex custom load balancing solution, and is not suitable for companies seeking to take advantage of the managed service capabilities of ELB.
- **Send metrics directly from application:** configure application nodes to inspect requests and publish a custom metric to CloudWatch via the CloudWatch API. While this method permits the use of ELBs, this method requires modification of application code, and may not be suitable for companies undergoing *lift and shift* migrations.

HTTP Request Tracing

HTTP request tracing refers to the practice of using HTTP headers to track an HTTP request as it progresses through an application stack. In AWS, this is usually done with AWS X-Ray, a service that provides an end-to-end view of requests as they travel through an application. ALB integrates automatically with X-Ray, and modifies the custom `X-Amzn-Trace-Id` header used by X-Ray to track requests, if present.

Management and Governance

AWS CloudTrail integrates with ELB to monitor configuration changes to deployed ELBs. CloudTrail provides an event history of all ELB configuration activity, including actions taken from the console, CLI, SDK, or CloudFormation. This can be useful for companies with strict compliance requirements, and is a valuable feature that can be challenging to replicate with custom load balancing solutions.

SSL/TLS Offloading and Certificate Management

SSL/TLS offloading is the process of terminating incoming SSL/TLS connections at the load balancer and proxying requests, unencrypted, to downstream targets. This is generally done to improve application performance by offloading cryptographic operations associated with SSL/TLS to the load balancer nodes instead of the application nodes. SSL/TLS offloading should only be done in situations where the application nodes are deployed in a secure, private subnet. Because SSL/TLS offloading involves sending unencrypted traffic to an application node, it may not be feasible for applications with end-to-end encryption requirements or applications with nodes in public subnets.

Both ALB and NLB support SSL/TLS termination and Server Name Indication (SNI), allowing multiple SSL/TLS certificates to be presented on the same hostname, (i.e., on the same load balancer).

Load Balancing Solution	**Architecture Considerations**
ALB	ALB supports SSL/TLS termination of one or multiple certificates. Check out Create an HTTPS Listener for ALB for more information.
NLB	ALB supports SSL/TLS termination of one or multiple certificates. Check out TLS Listeners for NLB for more information.
CLB	CLB does not support SSL offloading

Table 3 - ELB SSL/TLS offloading support

With a custom load balancing solution, such as NGINX on EC2, management of SSL/TLS certificates can be complicated, as application certificate(s) must be maintained on each load balancer node. ALB and NLB integrate with AWS Certificate Manager to easily provide a centralized, managed service for storing, deploying, and updating public and private SSL/TLS certificates.

Multi-Protocol Support

Many cloud-native applications rely solely on HTTP and HTTPS for public and private service communication; however, it is possible that an application requires multi-protocol support through a single load balancer. For example, if your application requires support from File Transfer Protocol (FTP) and HTTP from the same hostname, then your load balancer will need to support both protocol and port combinations.

ALBs only support HTTP and HTTPS protocols. For more information, see Listeners for ALB. NLBs support any TCP, TLS, UDP, or TCP_UDP protocol, making them far more flexible than ALBs in terms of supported network protocols. For more information, see Listeners for NLB. Although CLBs Can Support Multiple Protocols, the lack of features compared to NLBs make NLBs a better managed service option for supporting multiple protocols.

Static IP Addresses

Attaching static IP addresses to load balancers is a common requirement for organizations migrating to AWS. Consumers, especially external consumers, of an application can have the expectation that the application is exposed with one or several static IP addresses for reasons such as simplified network configuration or network security compliance (e.g. IP whitelisting in firewalls).

The type of available static IP address support is determined by the underlying compute resources supporting the load balancing solution. When you use an NLB in an Availability Zone, a single load balancer node and single network interface is created in the Availability Zone. To provide static IP addresses for an NLB-based application, Elastic IP addresses can be used to associate static IP addresses to the network interface created to support each Availability Zone. EC2-based custom load balancing solutions can also use Elastic IP addresses to provide static IP addresses assigned to load balancer nodes. For more information, see Create a Network Load Balancer, and Elastic IP address basics.

Alternatively, when using ALBs, the underlying IP addresses of the compute nodes supporting the ALB are subject to change, for example, in response to scaling events. Because an ALB is not supported by a static compute node, or static group of compute nodes, it does not support direct integration with Elastic IP addresses. Static IP addresses can still be provided to consumers of ALB-based applications via AWS Global Accelerator, a service that provides fixed entry points to your application in single or multiple AWS regions. Global Accelerators can be used with NLBs and custom load balancing solutions as well, if desired.

HTTP Request Manipulation

Some load balancers perform manipulation of HTTP requests for incoming traffic before proxying these requests to application nodes. These manipulations can add, remove, or rewrite parts of the request. Request manipulation can be done for a number of reasons:

- Rewriting any combination of HTTP headers including host, path, or verb (e.g., GET, POST). This can be useful for companies seeking to maintain compatibility with previous versions of a service by exposing existing HTTP services via a different URL.
- Insert additional, non-standard HTTP headers into a request to provide additional information about a client request to application nodes.
- Remove HTTP headers present in a response. This can be useful for companies seeking to hide sensitive information that is present in HTTP response headers from their users without modifying the application.
- Redirect requests to new URLs. This can be useful for companies seeking to automatically redirect HTTP to HTTPS, or for companies seeking to route traffic from a previous version of an application to a new version.

Because HTTP request manipulation involves inspection of HTTP headers, a layer 7 load balancer must be used. ALBs and CLBs support adding the `X-Forwarded-For` (contains the original client IP address because the downstream target will see the request as originating from the load balancer node), `X-Forwarded-Proto` (identifies whether the original request was HTTP or HTTPS), and `X-Forwarded-Port` (identifies the port the original client used to connect to the load balancer) headers to requests. Additionally, ALBs support request redirection. For more information on redirecting client requests with an ALB, see ALB Redirect Actions.

For direct manipulation of HTTP headers, removal of HTTP headers, or addition of HTTP headers beyond those supported by ALB, a third-party load balancing solution that supports client request rewrites, such as NGINX on EC2, is required.

SSL/TLS Negotiation Configurations

For load balancers making use of SSL/TLS connections, the SSL/TLS negotiation configurations supported by the load balancer must be defined. For load balancers, these configurations are comprised of two principal components:

- **SSL/TLS protocol version:** highest supported version of the SSL/TLS protocol.
- **SSL/TLS cipher suite:** a set of cryptographic algorithms used to secure data transmitted through SSL/TLS.

When forming an SSL/TLS connection between a client and server, the client's highest supported SSL/TLS version is presented to the server at the beginning of the connection process. If the server supports this version, then the connection process proceeds. Later during the connection process, the client and the server present a list of cipher suites that they each support, in order of preference. By default, the first cipher suite on the server's list that matches any of the client's cipher suites is selected and used to form the secure connection. In order to form a successful connection, the client and server need to agree upon a mutually supported protocol version and cipher suite.

ELB defines a *security policy* to specify the supported SSL/TLS negotiation configurations required for SSL/TLS listeners. A security policy is a combination of supported protocol versions and cipher suites that can be automatically deployed onto an ALB, NLB, or CLB without any additional configurations. The default `ELBSecurityPolicy-2016-08` security policy supports the most up-to-date protocol versions and cipher suites, and is generally recommended for maximum compatibility with client user agents. While ELB does not support custom security policies, it does offer security policies with more specialized functionality for applications that require Forward Secrecy or disabling of certain protocol versions. For more information about available configurations and security policies, see ALB, NLB, and CLB.

When you change the security policy on an ELB, this change only applies to client connections. ELB always uses the `ELBSecurityPolicy-2016-08` security policy when initiating connections to downstream targets. Therefore, if your application nodes do not support any protocol version and cipher suite combinations listed in `ELBSecurityPolicy-2016-08`, then a custom load balancing solution is required if the application nodes cannot be modified. Additionally, if your security policy does not support the protocol version and cipher suite combination required by a client, then the client will be unable to connect to your application and a custom load balancing solution may be required if the client is unable to use other SSL/TLS negotiation configurations.

In these scenarios, a custom load balancing solution will allow you to customize the underlying compute node supporting the load balancer to offer additional SSL/TLS negotiation configurations not supported by ELB security policies.

Modifying an ELB security policy can enable companies to restrict the SSL/TLS protocol versions and cipher suites available to clients, if desired. Monitoring ELB access logs is a good way to determine which protocol versions are most often requested by clients. Finally, ELB does not support SSL/TLS renegotiation. If this requirement is critical to your application functionality, a third-party load balancing solution running on customized infrastructure is required.

Idle Connections

If an application has long-lived requests, that is, requests that take longer than ~15 seconds to complete, then the idle timeout of connections should be considered. The idle timeout refers to the amount of time that a load balancer will keep a connection open without sending or receiving any data. With ALBs, the idle connection timeout is configurable between 1 and 4000 seconds, with a default of 60 seconds. ALBs maintain two connections: one between the client and the load balancer node, and one between the load balancer node and the load balancer target. High idle connection timeouts can place unnecessary strain on load balancer nodes and application nodes by causing unused connections to remain open, possibly leading to increased costs and decreased application performance. For more information, see ALB Connection idle timeout. With NLB-based solutions, the idle connection timeout is 350 seconds and is not configurable. Since NLBs are pass through load balancers, the connection is bound to the client and load balancer target, making TCP Keepalive packets on either end of the connection a viable option for maintaining active connections. For more information, see NLB Connection Idle Timeout. NAT gateways also have a non-configurable 350 second timeout, which should be considered if working with load balancers in private subnets.

Load Balancer Architecture Example

The following diagram illustrates an example of an on-premises load balancer architecture that is to be migrated to AWS. This architecture has the following requirements:

- Provide a static IP address to consumers of the application.
- Perform inspection of HTTP headers to route requests to appropriate downstream targets.

- Implement complex, whitelisting/blacklisting rules based on client IP addresses.
- Collect and store all access logs for 5 years.
- Install multiple SSL/TLS certificates on a single load balancer node.
- During the migration, parts of the application are refactored and deployed as Lambda functions.

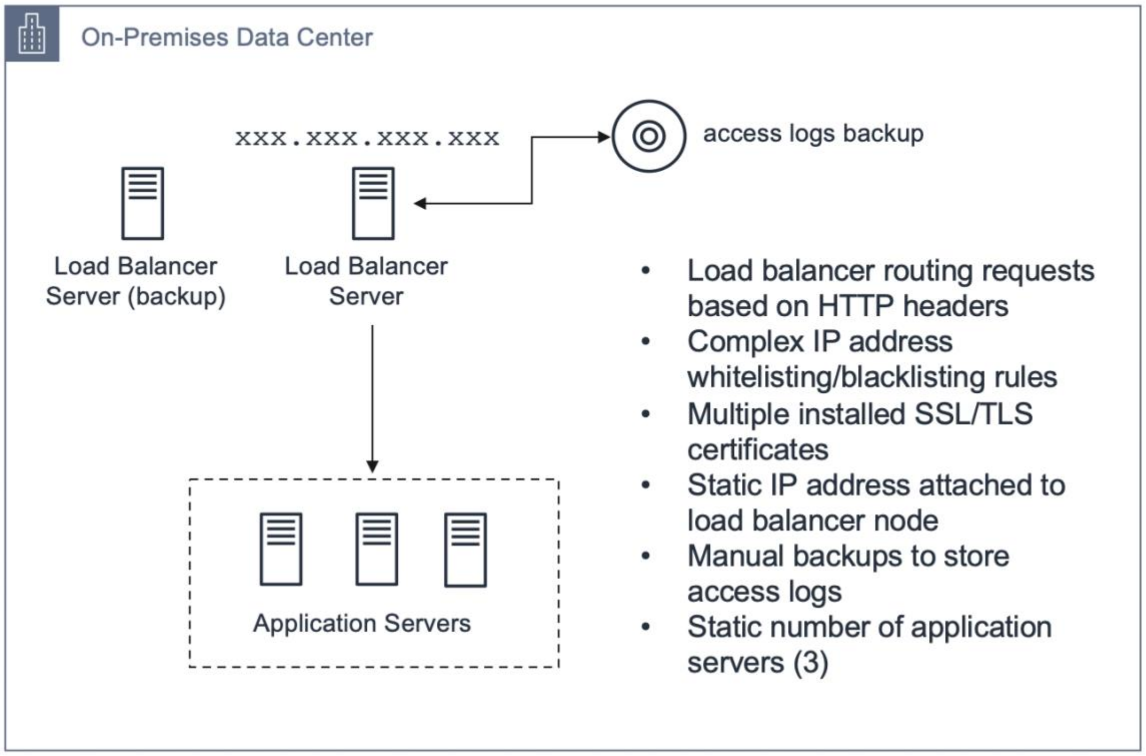
On-Premises Data Center
xxx.xxx.xxx.xxx
access logs backup
Load Balancer Server (backup)
Load Balancer Server
Application Servers
• Load balancer routing requests based on HTTP headers
• Complex IP address whitelisting/blacklisting rules
• Multiple installed SSL/TLS certificates
• Static IP address attached to load balancer node
• Manual backups to store access logs
• Static number of application servers (3)

Figure 1 - On-Premises Load Balancer Architecture

In an on-premises architecture, each of these requirements necessitates manual configuration and operation (or a configuration management software such as Chef). When migrating this solution to AWS, the following architecture can be used:

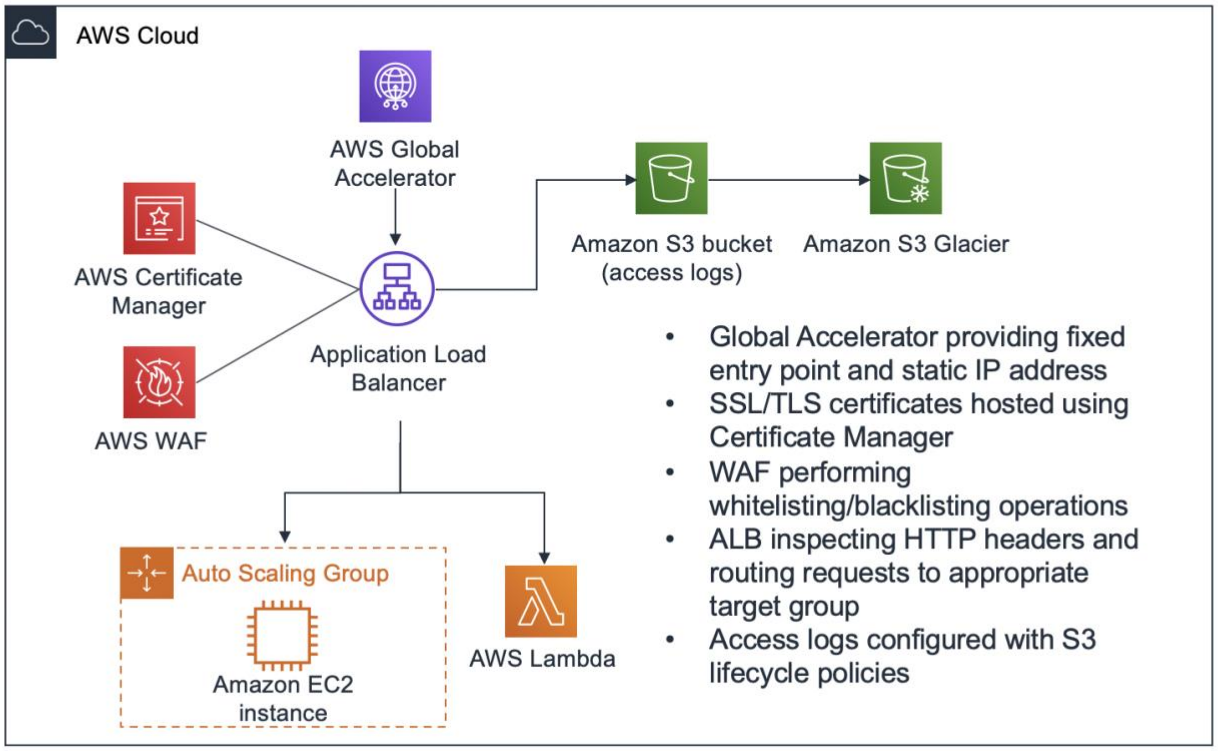
AWS Cloud
AWS Global Accelerator
AWS Certificate Manager
AWS WAF
Application Load Balancer
Amazon S3 bucket (access logs)
Amazon S3 Glacier
Auto Scaling Group
Amazon EC2 instance
AWS Lambda
• Global Accelerator providing fixed entry point and static IP address
• SSL/TLS certificates hosted using Certificate Manager
• WAF performing whitelisting/blacklisting operations
• ALB inspecting HTTP headers and routing requests to appropriate target group
• Access logs configured with S3 lifecycle policies

Figure 2 - AWS Load Balancer Architecture

This architecture takes advantage of managed services offered by AWS to meet the application requirements outlined above:

- AWS Global Accelerators are used to improve the availability and performance of your application, as well as provide a set of static IP addresses for use by application consumers.
- AWS WAF is used to implement complex whitelisting and blacklisting configurations. WAF also offers built-in support for common exploits such as SQL injection and cross-site scripting.
- AWS Certificate Manager is used to provision, update and deploy your existing SSL/TLS certificates to load balancer nodes. With Certificate Manager, you can centrally manage multiple SSL/TLS certificates, and deploy one or multiple certificates to ELB nodes.
- An ALB is configured to inspect HTTP headers and route requests to an appropriate downstream target. The ALB is configured with two target groups: EC2 instances in an Auto Scaling group, and a Lambda function. Both of these target groups are elastic, and will scale to meet the current application traffic.

- Access logs are configured on the ALB. These access logs are dumped to S3, which is configured with lifecycle policies to move access logs to Amazon S3 Glacier after 30 days.

In addition to meeting the requirements present in the on-premises architecture, the ALB-based architecture offers the benefit of relying on AWS managed services, simplifying the operation of this application by automating common activities and adding the ability for the application to take advantage of the elasticity offered by cloud-based workloads. The ALB-based architecture also integrates closely with other core AWS services for configuration and infrastructure management (CloudFormation). permissions management (AWS Identity and Access Management), and monitoring and metrics (CloudWatch).

Conclusion

For companies migrating to AWS, deprecating existing load balancing architectures in favor of ELB-based architectures can reduce the operational effort required to support load balancing infrastructure without sacrificing any existing features. In addition to ELB, AWS offers a number of other services that integrate closely with ELBs to deliver common features and improve overall application performance.

Contributors

Contributors to this document include:

- Bryant Bost, AWS ProServe consultant

Document Revisions

Date	**Description**
September 2020	First publication