DolfinED

Transforming Lives Through Learning

AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE

SAA-C02 Exam Study Guide – Section 8

Version 1.0

www.dolfined.com

Table of Contents

Section 8 - Elastic Load Balancing and Auto Scaling on AWS 3

Section Outline 4

Elastic Load Balancing 6

Target Groups, Listeners, and Health Checks 7

Cross-Zone Load Balancing, Connection Draining & Subnet Design for HA 17

ELB Security Groups 24

ELB and SSL Certificates 29

Knowledge In Action – Project 3 42

ELB – Client IP Address, Monitoring, Stickiness, and PFS 47

Application Load Balancer 54

Network Load Balancer 59

ELB – Gateway Load Balancer (GWLB) 65

Amazon Auto Scaling 67

Auto Scaling 68

Amazon Auto Scaling – Launch Templates and Scaling Policies 75

Knowledge In Action – Project 4 83

www.dolfined.com

© DolfinED All rights reserved

The course is available on Udemy & DolfinED websites,
https://www.udemy.com/course/aws-certified-solutions-architect-associate-/

Visit DolfinED's website to browse the available courses and enroll at a discounted price.
www.dolfined.com/course-catalog

When you visit the above URL @ www.dolfined.com, Enroll in **the free course,** TCP/IP Introduction and Cloud Pre-Requisite Knowledge

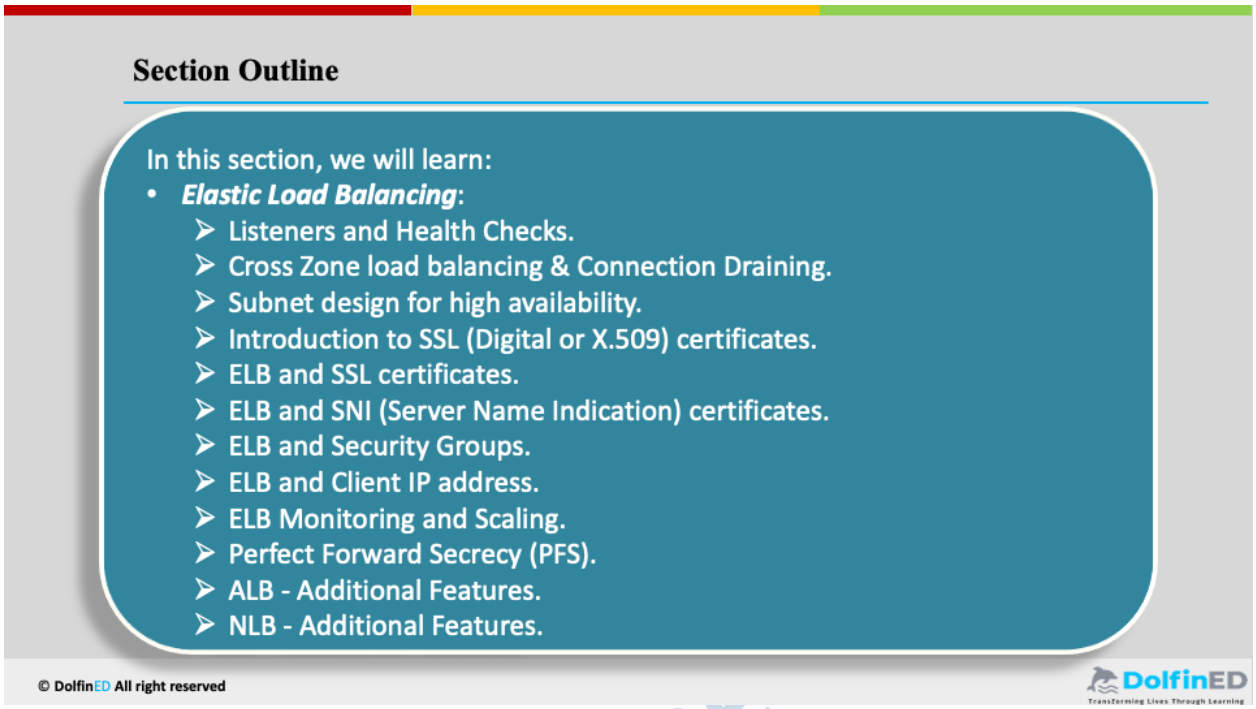
Which will teach you many topics required for Cloud and IT in general, including TCP/IP, IP Routing, IP Subnetting, Containers, Virtualization, SSL and Digital Certificates, Encryption, NAT, and PAT, among other topics.
The course is full of animation and graphics, high quality knowledge, audio, and video. Definitely worth having a look at.

© DolfinED All rights reserved

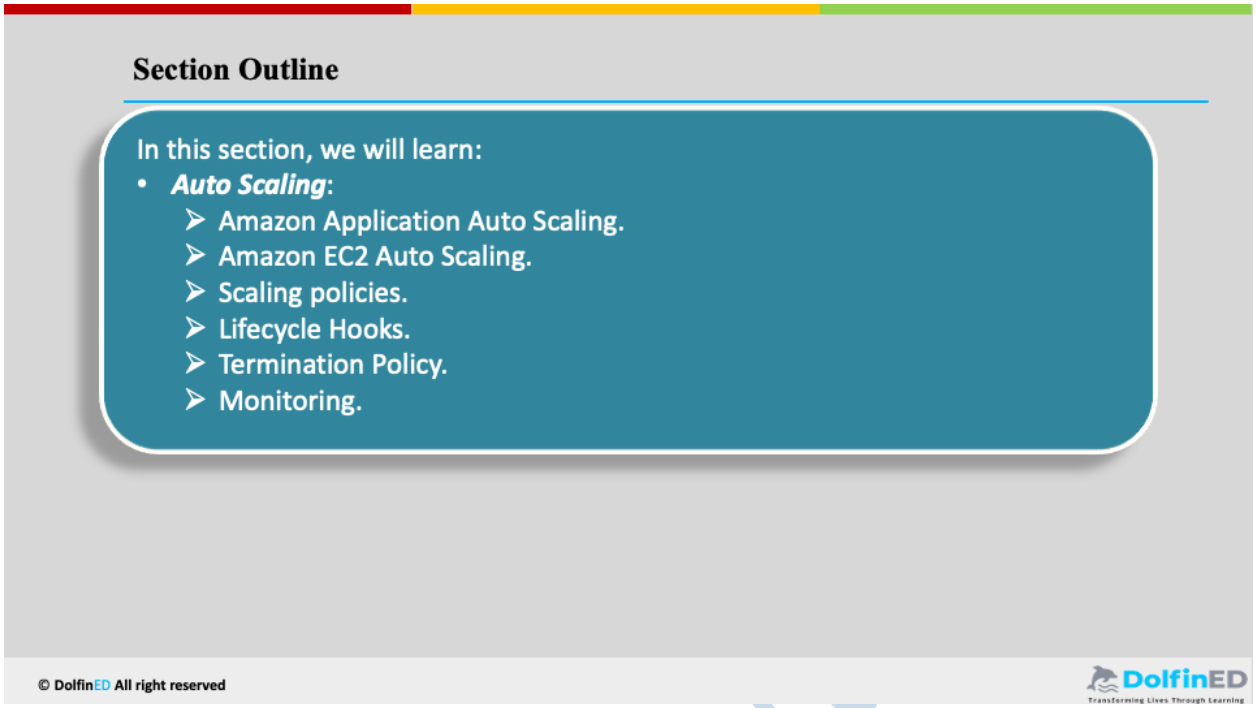
Section 8 - Elastic Load Balancing and Auto Scaling on AWS

www.dolfined.com

Section Outline

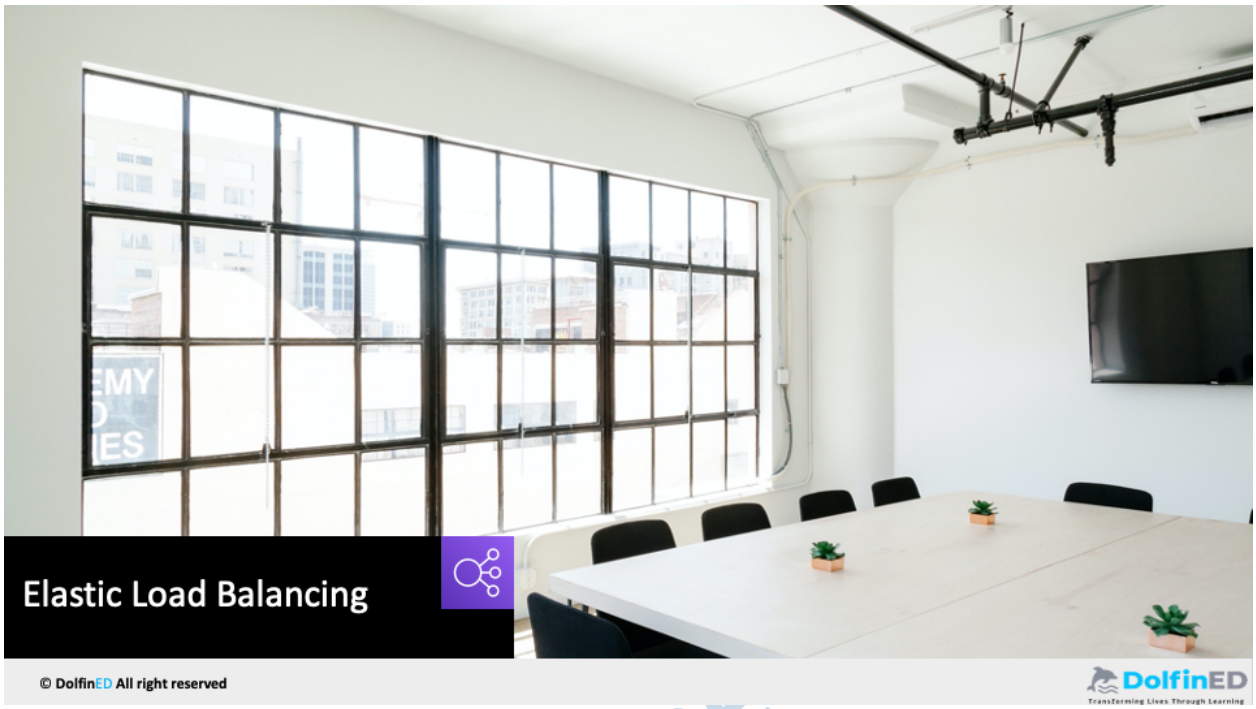


www.dolfined.com



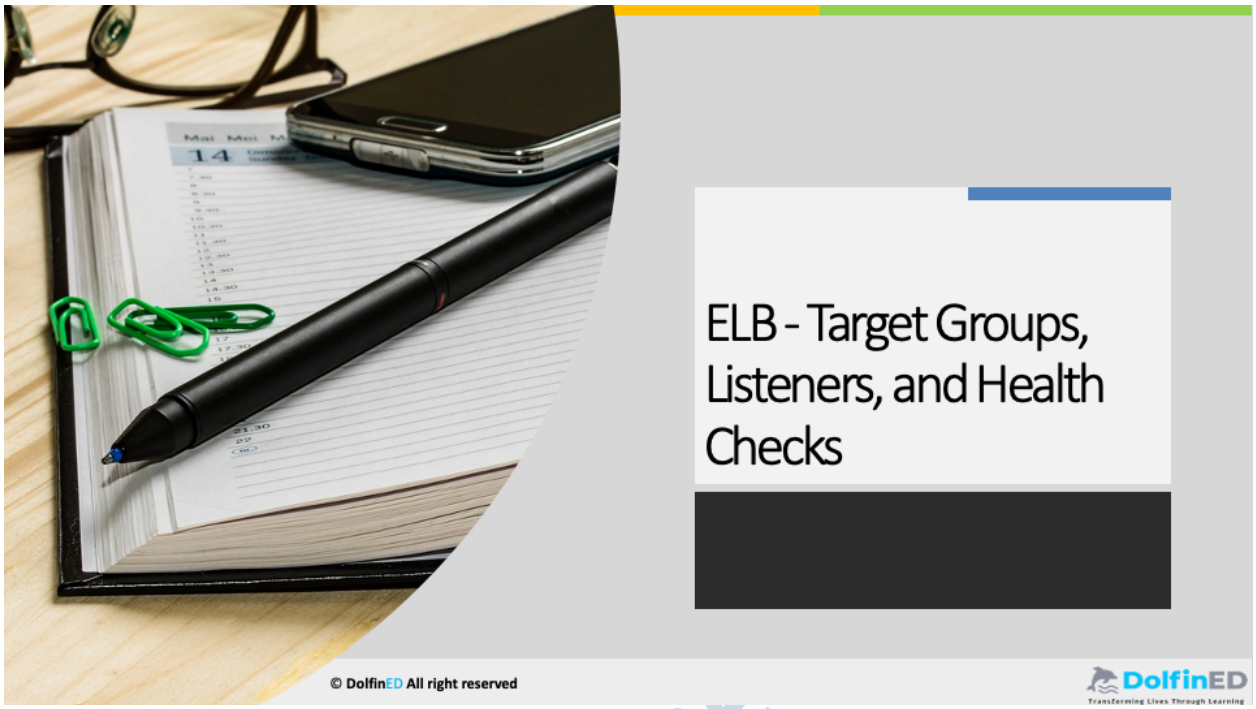
www.dolfined.com

Elastic Load Balancing

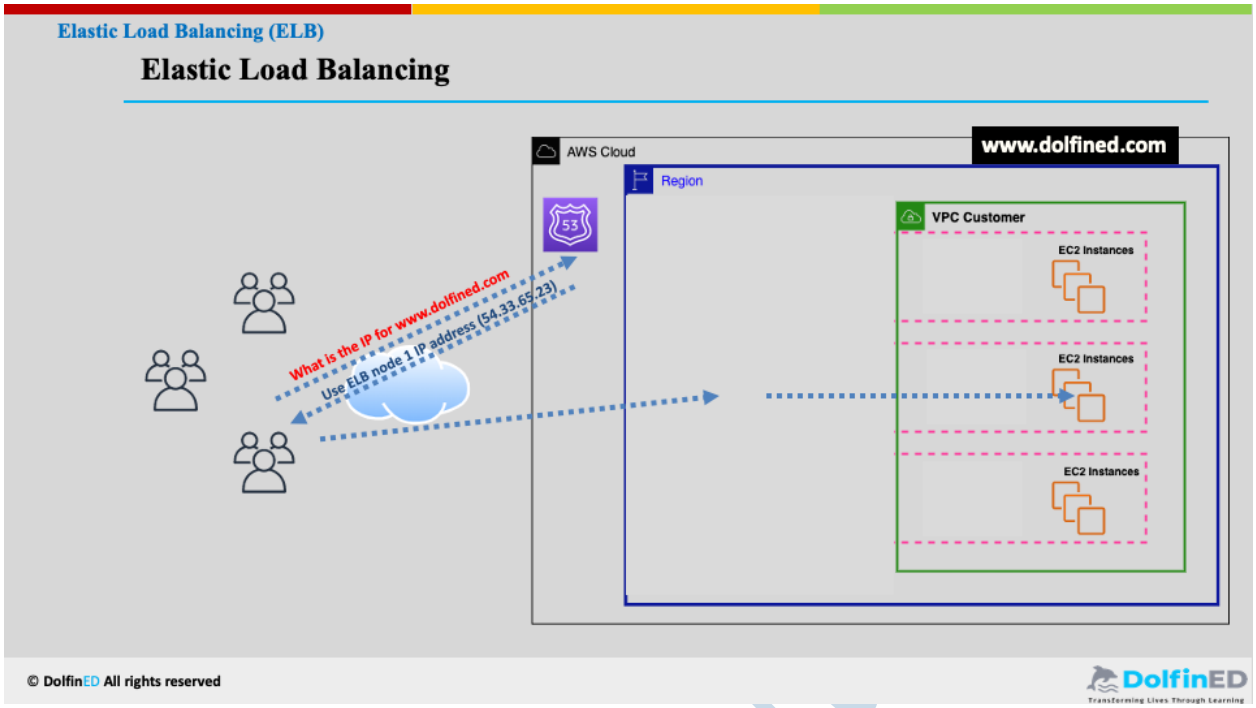


www.dolfined.com

Target Groups, Listeners, and Health Checks



www.dolfined.com



Elastic Load Balancing (ELB) service is region-specific, which means the ELB functions at the regional level across the AZs in that region. This also means an ELB cannot be used to load balance between two regions.

An ELB can be launched as *internet-facing* or *internal* in a VPC.

Use the **internal load balancer** when the source and workload or applications are internal to your VPCs.

On the other hand, launch an **internet-facing** ELB when your clients initiate traffic to your AWS resources from the internet. The internet-facing ELB will have an internet resolvable DNS name.

As shown, when you launch an ELB, it will be launched in the same availability zones in which you have enabled the ELB.

Under the hood, the ELB nodes are EC2 instances in a VPC controlled by AWS ELB in the customer region; these instances have load balancing software installed.

AWS recommends enabling the ELB in at least two availability zones. To do that, you have to define one subnet per AZ for the ELB. This is how you enable the ELB in that AZ

www.dolfined.com

Once you do that, an **Elastic Network Interface (ENI)** will be dropped into each subnet you define for the ELB. One ENI per enabled AZ.

For internet-facing ELBs, the subnets where ELB is defined must be public.

This is very important to keep in mind, as it will influence your subnet design.

Let's say we have www.dolfined.com hosted on these EC2 instances in three availability zones in one AWS region.

We need the load balancer to distribute the load to these instances and decouple the clients' instances.

Clients will resolve www.dolfined.com through Internet DNS servers, which will eventually point to the ELB.

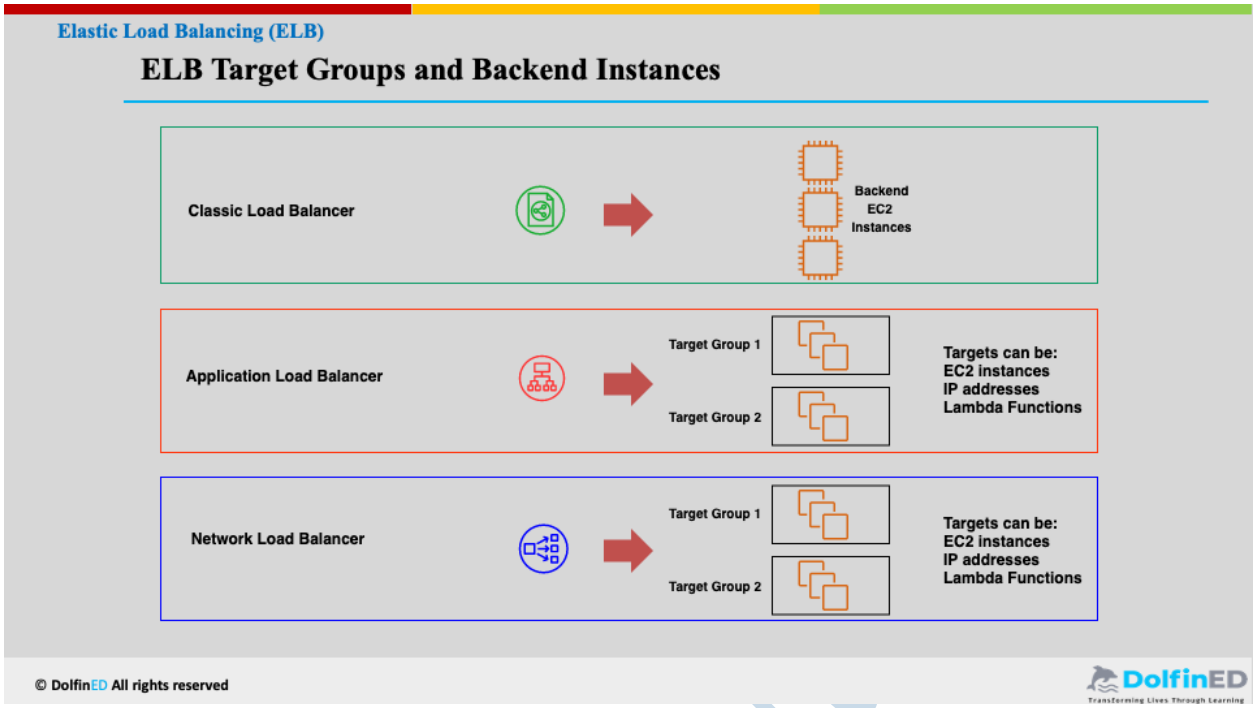
Each time the DNS will respond to DNS queries regarding www.dolfined.com with an IP address of one of the ELB nodes to distribute the load among them.

Now the client will direct the traffic to the received IP address of the ELB node (which is a public IP in this case since it is an internet-facing ELB).

Once the ELB receives the traffic, it will send it to one of the EC2 instances registered to serve www.dolfined.com. It will forward this traffic through the ENI in that AZ/Subnet where the chosen EC2 instance is located.

ELB nodes will distribute traffic among web tier (here are also called) backend instances.

www.dolfined.com



There are three types of Elastic Load Balancers in AWS:

1. First and oldest is the **Classic Load Balancer (CLB)**. This one is not recommended to use, and is more expensive than the other two, newer load balancers. It can do both Layer 7 HTTP/HTTPS and Layer 4 load balancing.

2. The second is the **Application Load Balancer (ALB,** a Layer 7 (HTTP HTTPS) load balancer. It has advanced features compared to CLB.

3. Last but not least, the **Network Load Balancer (NLB),** a Layer 4 (TCP/UDP) Load balancer, has the highest performance among the three.
Classic Load Balancers deal with workloads it is balancing to as individual EC2 instances which are registered with the load balancer as such.
It is not recommended to use Classic Load balancers anymore.
In ALB and NLB, the backend can be more than just EC2 instances; therefore, they are referred to as Targets.

Targets are joined in a virtual grouping called a **Target Group**.

Then we register target groups with the ALB or NLB.

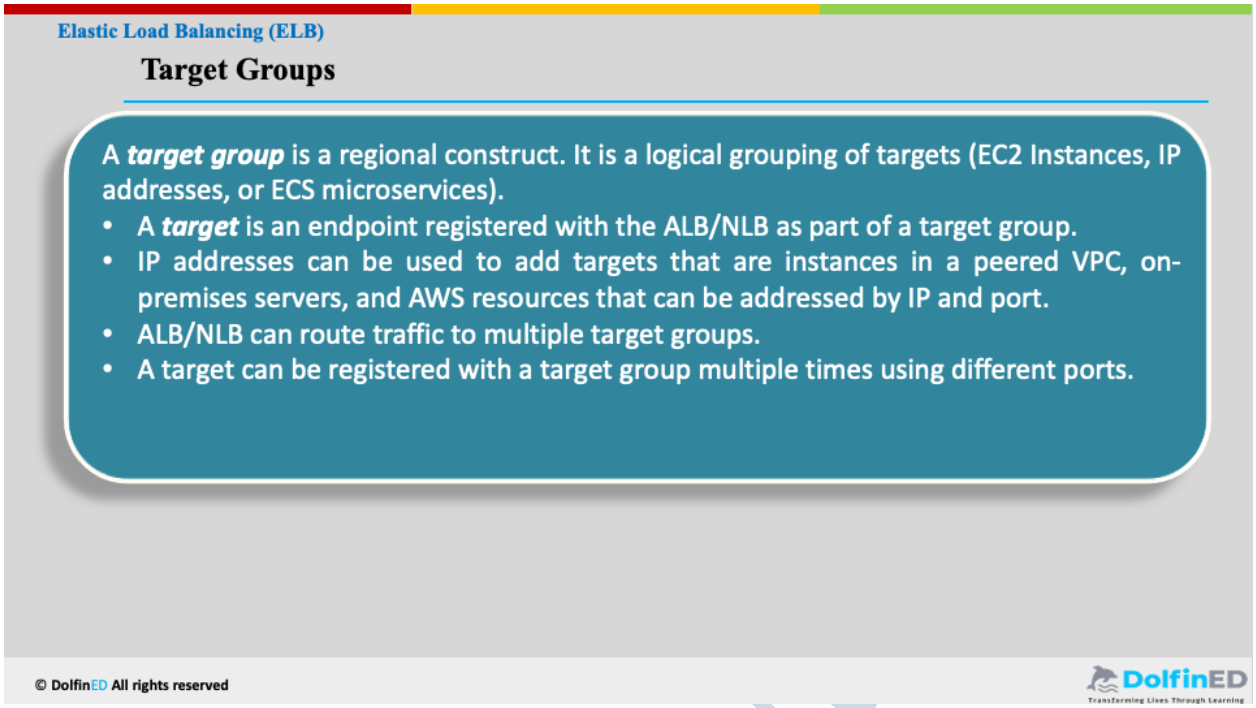
These targets can be:

www.dolfined.com

- EC2 instances.
- IP Addresses (non-internet routable) and these IPs can be in AWS or On-Premises, but not on the public Internet.
- Lambda functions – we will cover lambda later on when we get to serverless computing.

www.dolfined.com

© DolfinED All rights reserved



Target Groups:

Are regional constructs (confined to a region).

The **target group** is used to route requests to registered **targets** as part of a rule's action.

Targets specify the endpoints and are registered with the ALB or NLB as part of a target group.

We can add and remove targets from the load balancer as the needs change without disrupting the application's overall flow of requests.

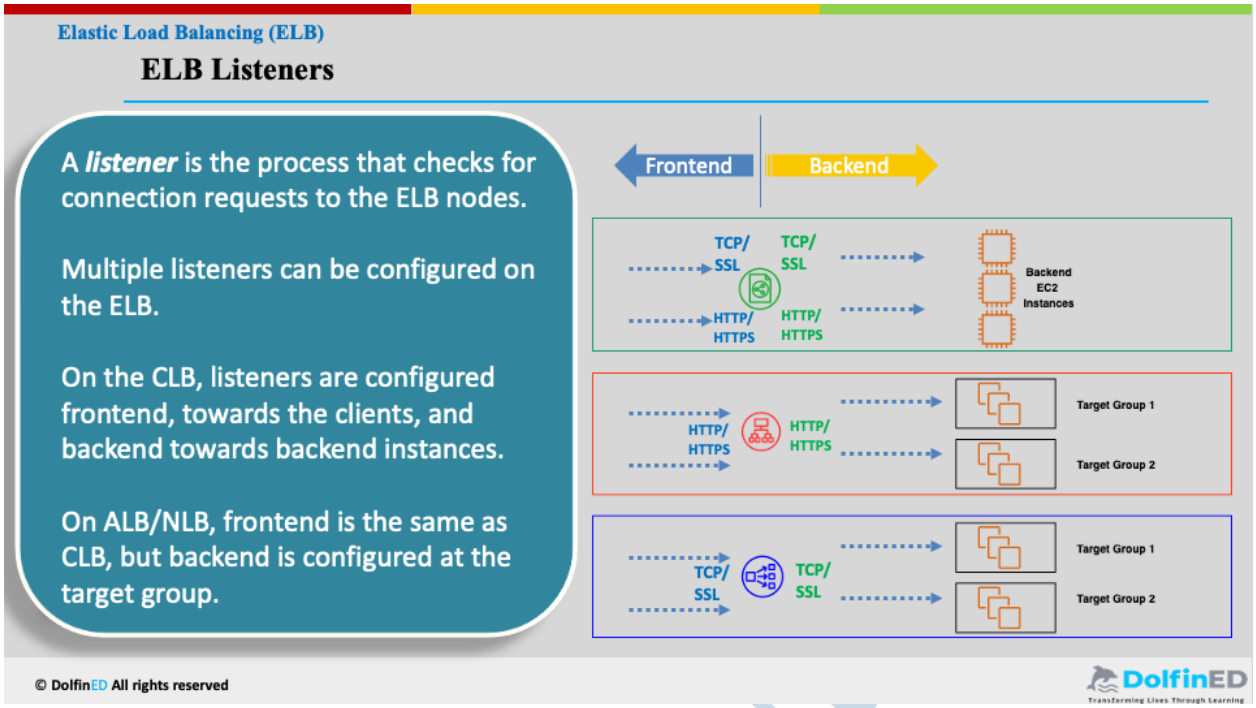
We can use IP addresses as targets to register:

- Instances in a peered VPC.
- AWS resources that are addressable by IP address and port (for example, databases).
- On-premises resources linked to AWS through AWS Direct Connect or a VPN connection.

We need to define one protocol and one port per target group, which will route/forward traffic to the registered targets.
By default, the load balancer sends requests to registered targets using the port and protocol specified for the target group.

- An ALB/NLB can route to multiple target groups.
- An EC2 instance can be registered with the same target group multiple times using multiple ports.

www.dolfined.com



An **ELB listener** is the process that checks for connection requests

You can configure the protocol/port on which your ELB listener listens for connection requests.

- *Frontend* listeners check for traffic from clients to the ELB.
- CLB supports both Layer 4 TCP/SSL listeners and Layer 7 or HTTP/HTTPS listeners.
- ALB supports HTTP/HTTPS or layer 7 only.
- NLB supports TCP/SSL listeners or layer 4 only.

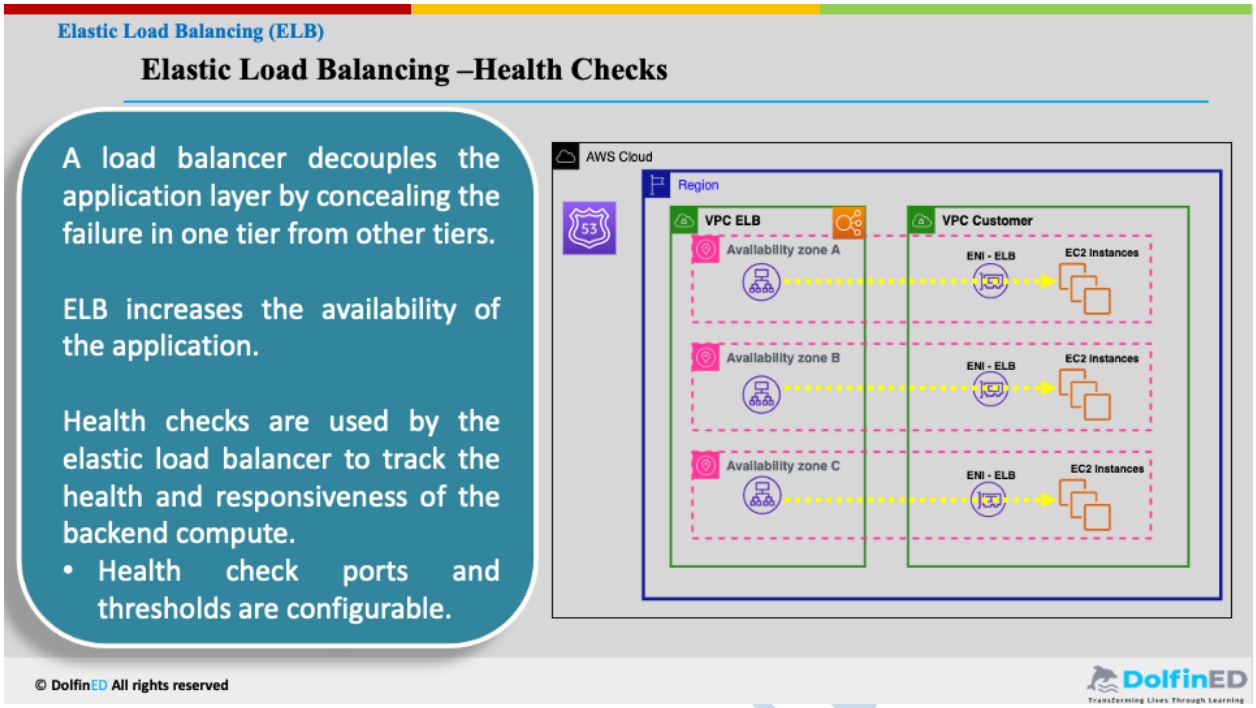
For *backend,* the story is a bit different

For CLB, backend listeners are configured with protocol/port to check for traffic from the ELB to the EC2 instances.

For ALB/NLB, we configure the forwarding port or the listeners at the target group level, not the individual instance level.

Similarly, CLB supports L4 and L7, whereas ALB supports L7 only, and NLB support L4 only.

www.dolfined.com



The load balancer helps decouple the application layer by concealing the failure in one tier's compute instances (or applications) from other tiers.

Therefore, the ELB increases the availability of the application.

The ELB continuously probes the health of the workloads. These probes are called **Health Checks**.

Here is what happens:
ELBs will check your back-end compute or targets' health using configurable health check parameters on a TCP/IP port.

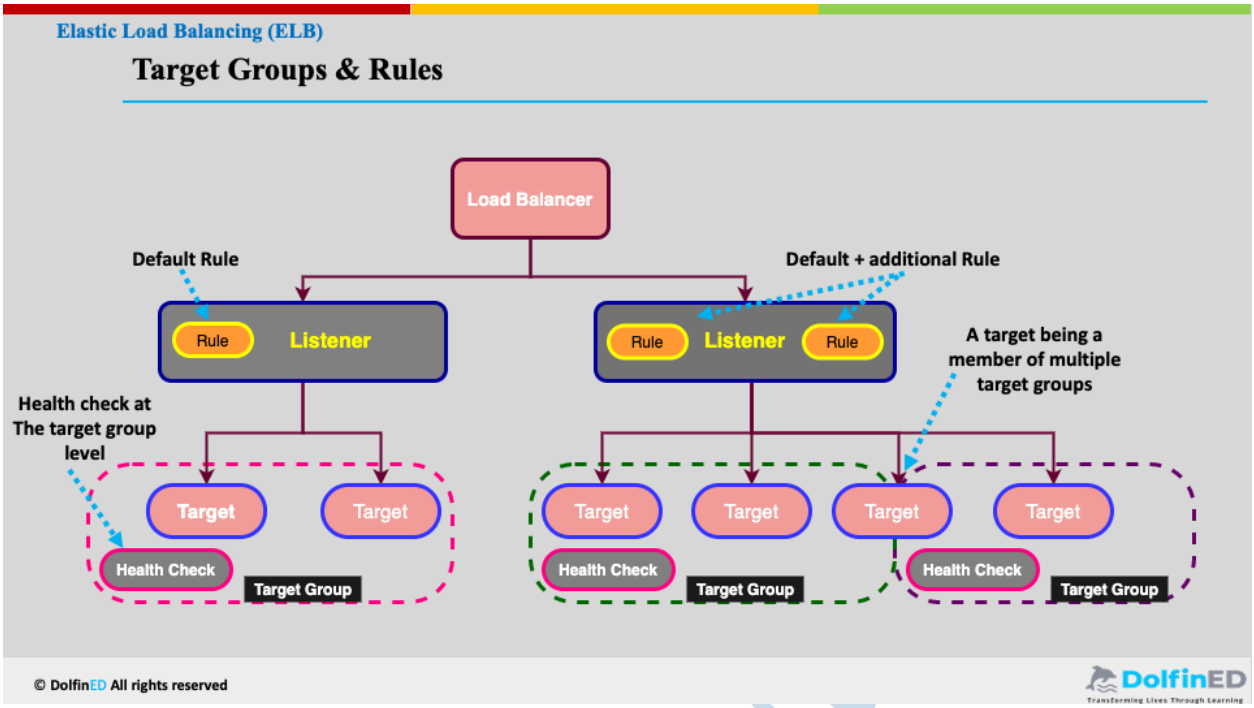
Health checks are configured at the backend instance level for the classic load balancer and the target group level for the ALB and NLB.

Health checks get performed on all targets registered to a target group as specified in a listener rule for your load balancer.

If the instance is healthy, it shows in the load balancer dashboard as **in-service. If** it is unhealthy, it shows as **out-of-service**.

An out-of-service instance does not get traffic from the ELB until it passes the health checks first and becomes in-service again.

www.dolfined.com



It is crucial to understand target groups, targets, and the rules that govern the relationship between front-end listeners, target groups, and forwarding ports or backend listeners.

Think of the **rule** as the glue between **front-end listeners** and **target groups**.

A target group, in turn, has a protocol and a forwarding port, targets, and a configured health check.

You must define a default rule for each listener and add rules that specify different target groups based on the request's content **(also known as *content-based routing*).**

If no specific rules are found, the request will follow the default rule, which forwards the request to the default target group.

Each rule consists of a target group, priority, action, optional host condition, and optional path condition.

When the condition is met, the traffic is forwarded to the target group.

Rule Conditions

A **condition** (optional) is the path pattern you want the ALB to evaluate to make request forwarding decisions.

- There are two types of rule conditions: host and path.
- Each rule can have up to 2 conditions, 1 path condition, and 1 host condition.
- When the conditions for a rule are met, then its action is taken.

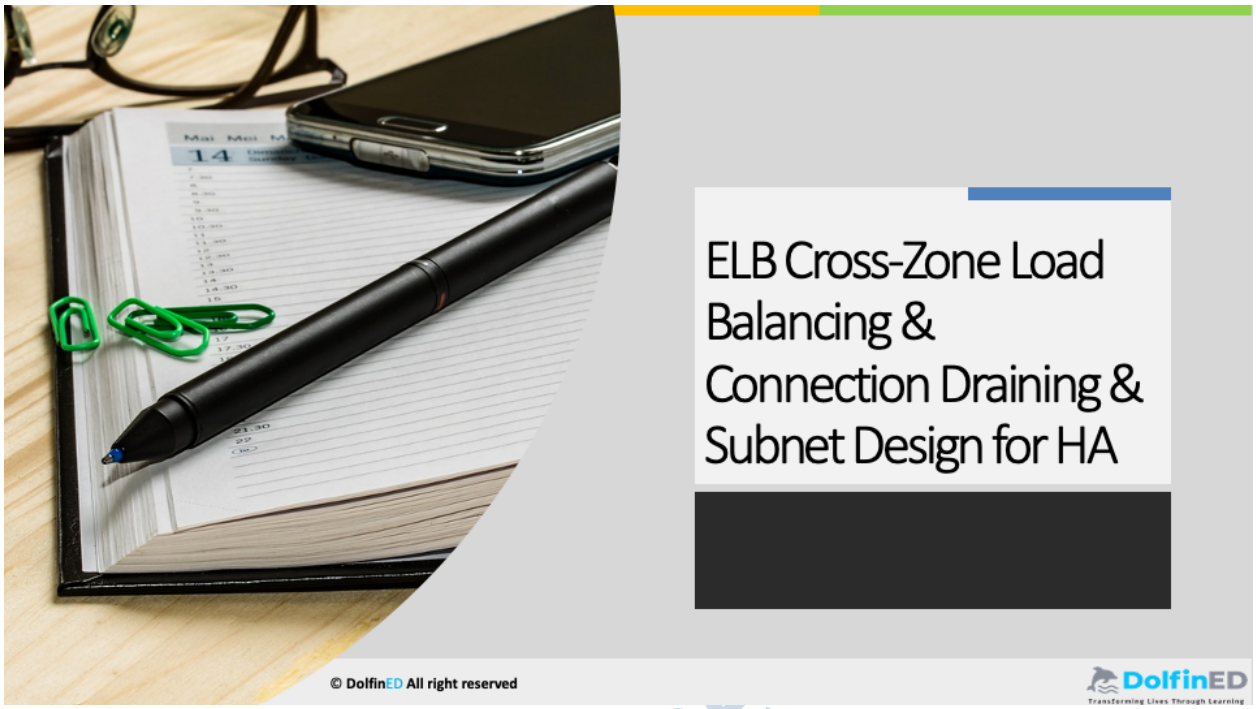
© DolfinED All rights reserved

A target can be registered with multiple target groups. Health checks are configured at the target group level.

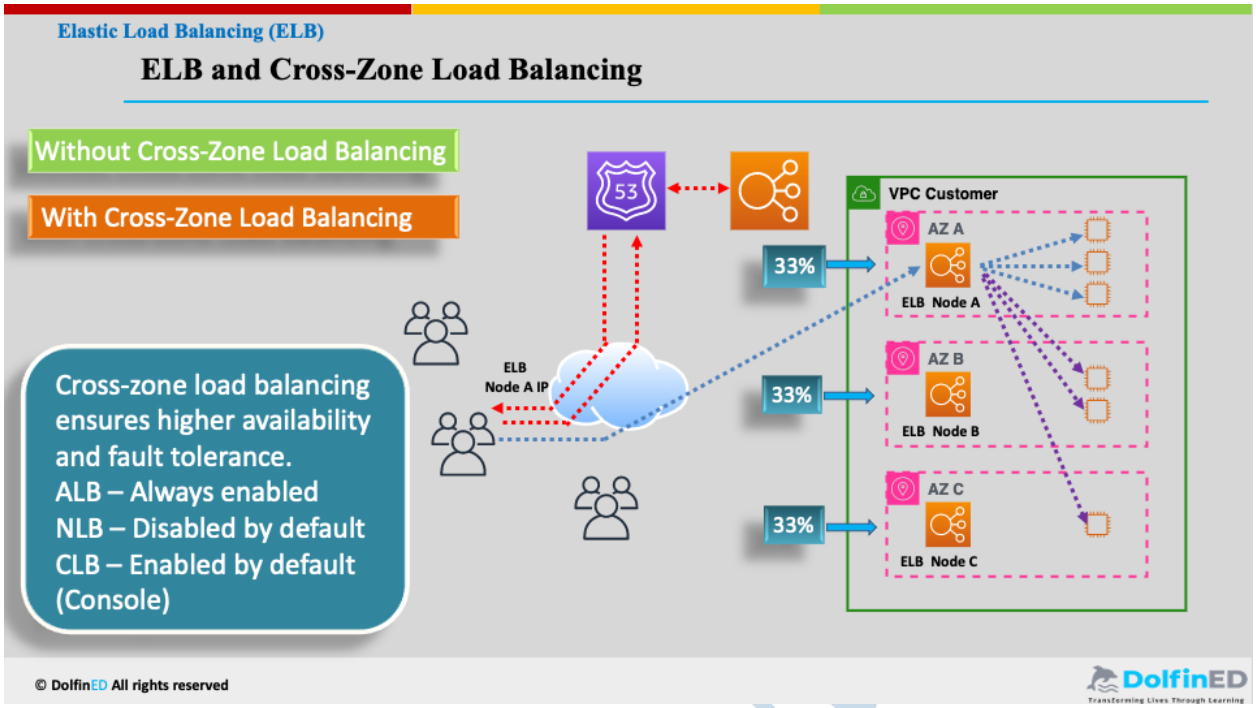
www.dolfined.com

© DolfinED All rights reserved

Cross-Zone Load Balancing, Connection Draining & Subnet Design for HA



www.dolfined.com



An ELB enabled in an AZ will cause the ELB service to create a load balancer node in that Availability Zone. ELB will work with the AWS DNS service (Route 53) to ensure that traffic is distributed evenly across these ELB nodes. In other words, to ensure a fair distribution of traffic across enabled Availability Zones.

Now, to better understand cross-zone load balancing, let's look at the visual on the screen.

We have a web application deployed across 3 Availability Zones:

- AZ A with 3 EC2 Application instances.
- AZ B with 2 Instances.
- And AZ C with 1 instance.

And we have an internet-facing ELB deployed and enabled in the three AZs.

So we have one ELB node in each AZ, namely ELB nodes A, B, and C.

Each load balancer node distributes the received traffic to backend instances or targets in its AZ only.

In our scenario, assuming all nodes and targets are healthy, each AZ where an ELB node exists will get roughly one-third or 33% of the traffic since we have the ELB enabled in 3 AZs.

First, let us investigate the ELB nodes' behavior without cross-zone load balancing.

In this case, each load balancer node will distribute the load to its own AZ's EC2 instances only and will not send traffic to the EC2 instances in the other AZs.

From the EC2 instance, target, or back-end instances, this is not a fair distribution of traffic.

If we look closely,

- In AZ A, we have three targets, which means the 33% load for this AZ will be split among these three targets, or each gets roughly 11% of the overall load.
- Whereas in AZ B, we have two targets, so each will get half of the 33% or 16.5%.
- In AZ C, one target will get the full AZ load, which is 33% of the overall load.

That means some targets will be overloaded than others, and this is not an even distribution based on all available healthy targets and compute power.

The cross-zone load balancing feature takes care of this to make the traffic distribution even across all targets.

With cross-zone load balancing, ELB nodes will not distribute the traffic to targets in the local AZ only; instead, they will distribute it evenly among all healthy targets in all AZs where the ELB is enabled.

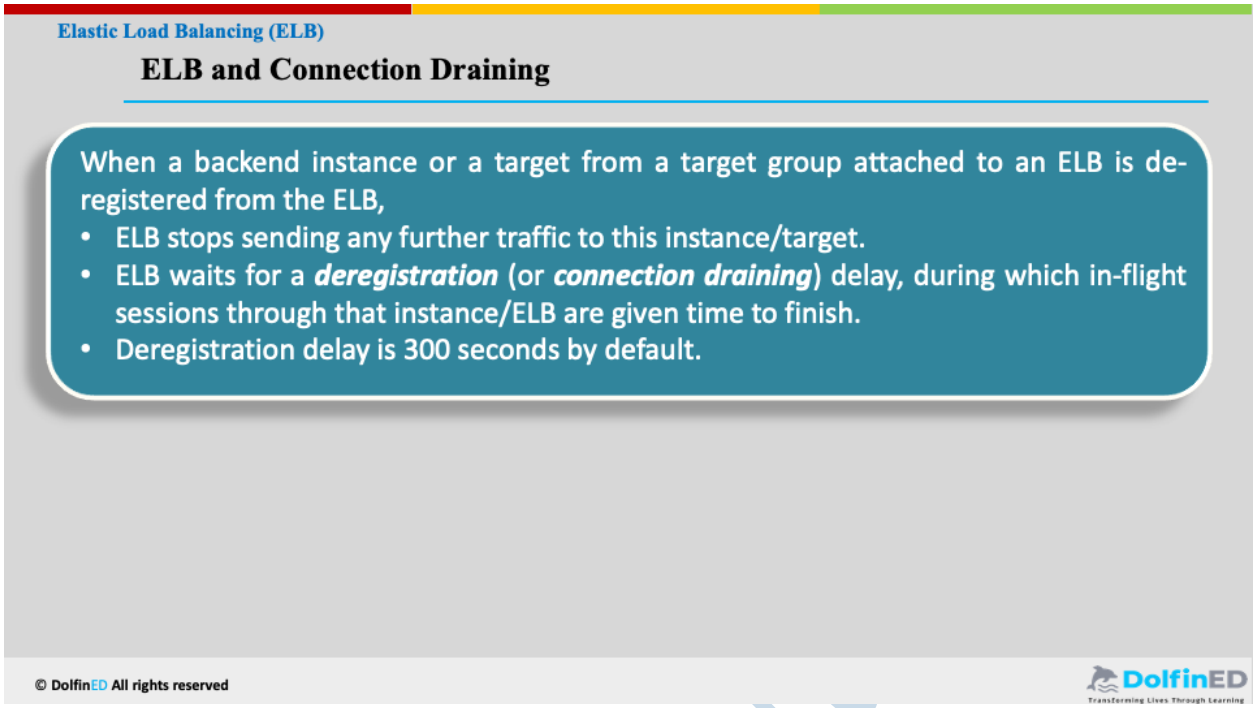
Distributing the end targets (compute instances) across Availability Zones increases the architecture's fault tolerance.

Now is Cross Zone load balancing enabled or supported by the different types of ELBs?

- ALB has it always enabled.
- For the NLB, it has it disabled.

www.dolfined.com

© DolfinED All rights reserved



When **de-registering** backend instances or targets from the ELB, the ELB will wait for 300 seconds (by default) for the in-flight sessions to this instance or target to complete.

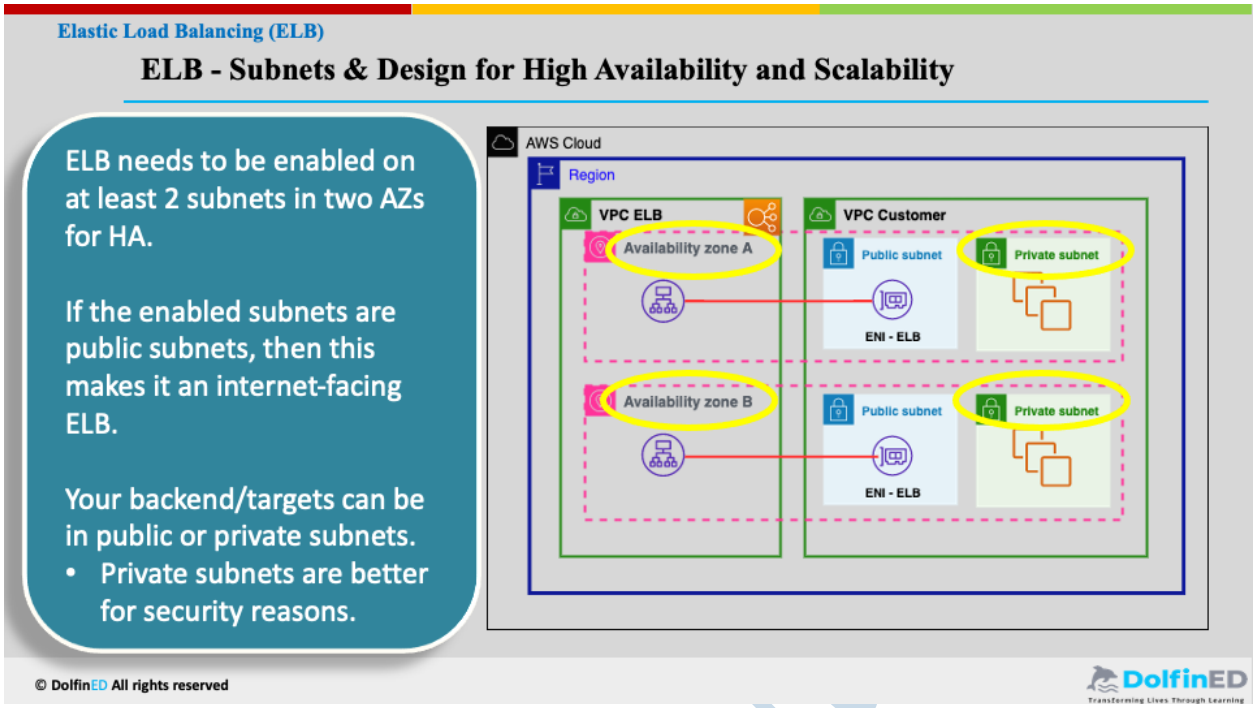
This is called the ***deregistration delay*** (configurable between 1 and 3600 seconds).

If the in-flight sessions are not completed before the maximum time (300 seconds), the ELB will force termination of these sessions

AWS Auto-Scaling would also honor the **connection draining** setting for unhealthy instances.

During the connection draining period, ELB will not send new requests to the unhealthy instance.

www.dolfined.com



We need to enable the ELB in at least two AZ's in the AWS region, where they are launched for high availability.

We need to make sure cross-zone load balancing is enabled (if on CLB or NLB) for fault tolerance and fair distribution of load.

In our example on the screen :

The ELB has been enabled on 2 AZs, which are AZ A and AZ B.

Internet-facing ELB MUST be enabled on public subnets. It requires one public subnet per AZ to distribute the load to all subnets in that AZ.

The ELB here has been enabled in a public subnet in each AZ. This makes it an internet-facing ELB. The ELB needs to be enabled on a subnet that has enough available IPv4 addresses for scaling.

Communication between the ELB nodes and your instances (targets) in a VPC will happen via the ENIs that the ELB drops in the subnets where it is enabled.

www.dolfined.com

These ENIs will have private IPv4 addresses allocated from these subnets, which are part of your CIDR block for the VPC.

Therefore, the load balancer nodes and backend will be communicating using their private IPv4 addresses. Thus, the backend (targets) do not need public IP addresses to receive traffic from the load balancer.

This also means it is possible to place the targets in public or private subnets.

However, it is better to conceal the web/app instances in private subnets from a security standpoint unless there is a clear requirement to have them in public subnets.

If you choose to deploy the web/app tier in private subnets, that means the subnet design will be:

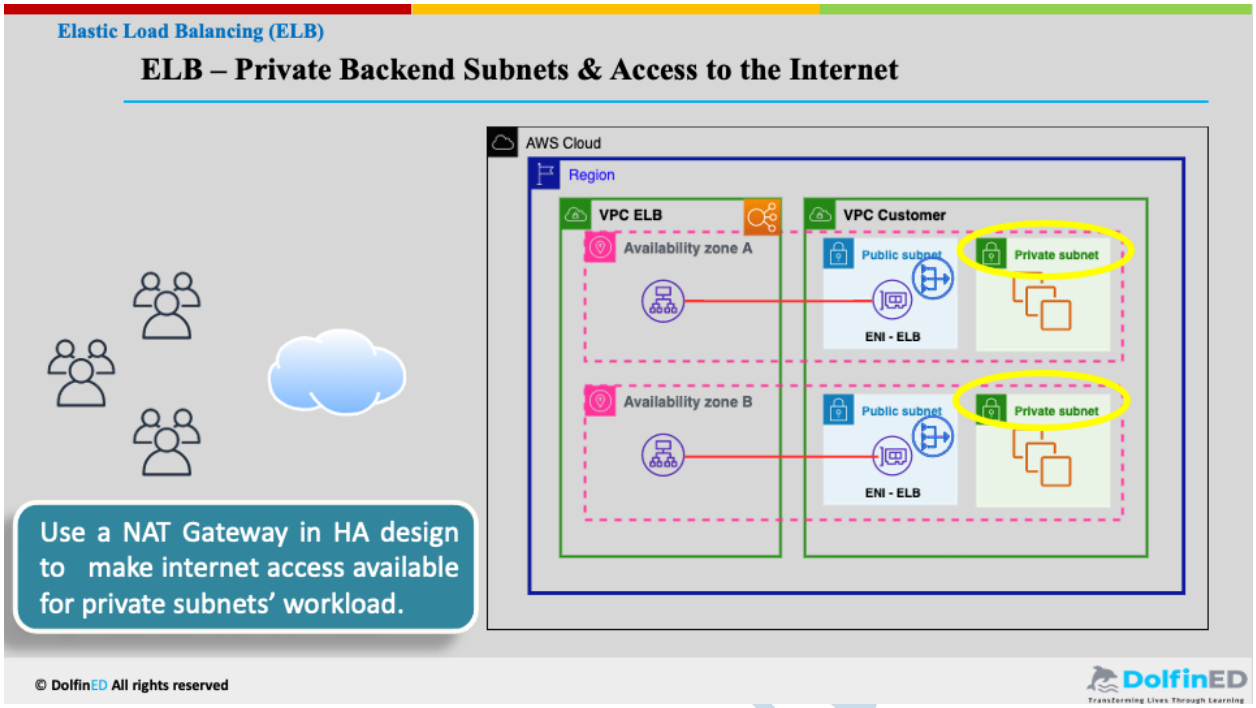
- Two public subnets to enable the ELB, one in each availability zone.
- Two private subnets (or more) for the web/application tier.
- Two private subnets for the database service.

This will give us a highly available subnet design for this case.

- We can live with 4 subnets if the client wants to have the web/app in public subnets for whatever reason by placing the web/app in the same subnets at the ELB ENIs.

www.dolfined.com

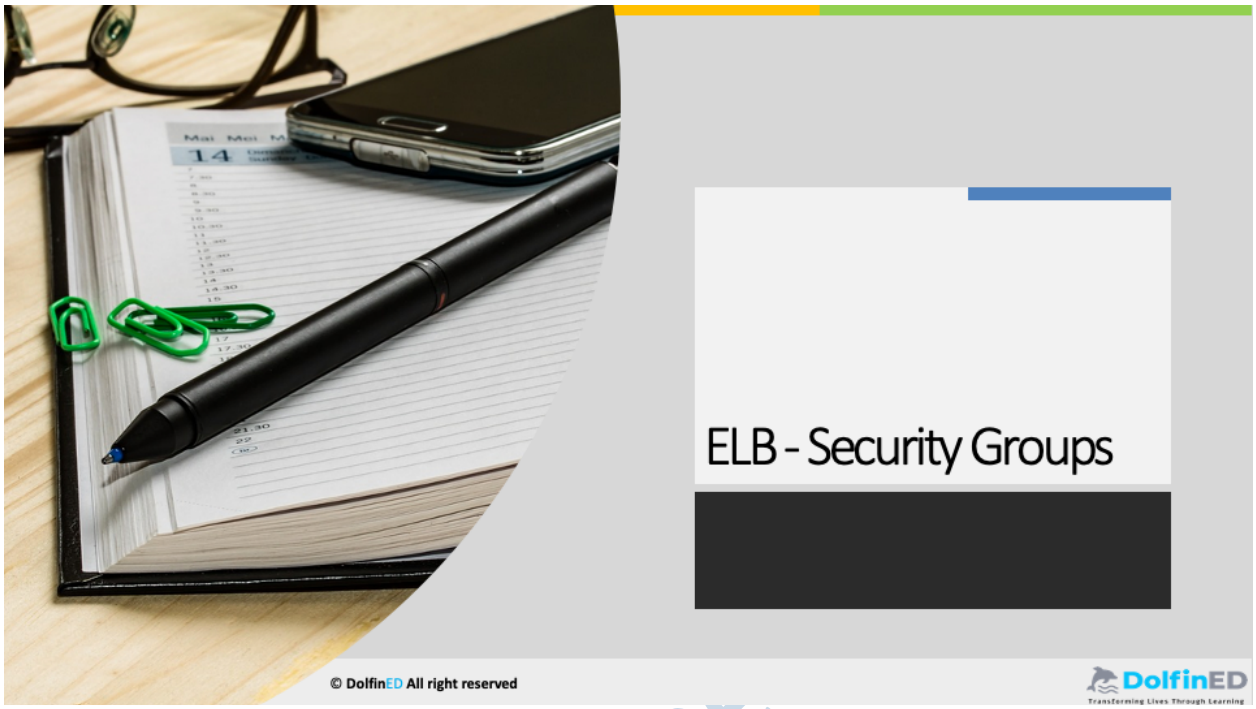
© DolfinED All rights reserved



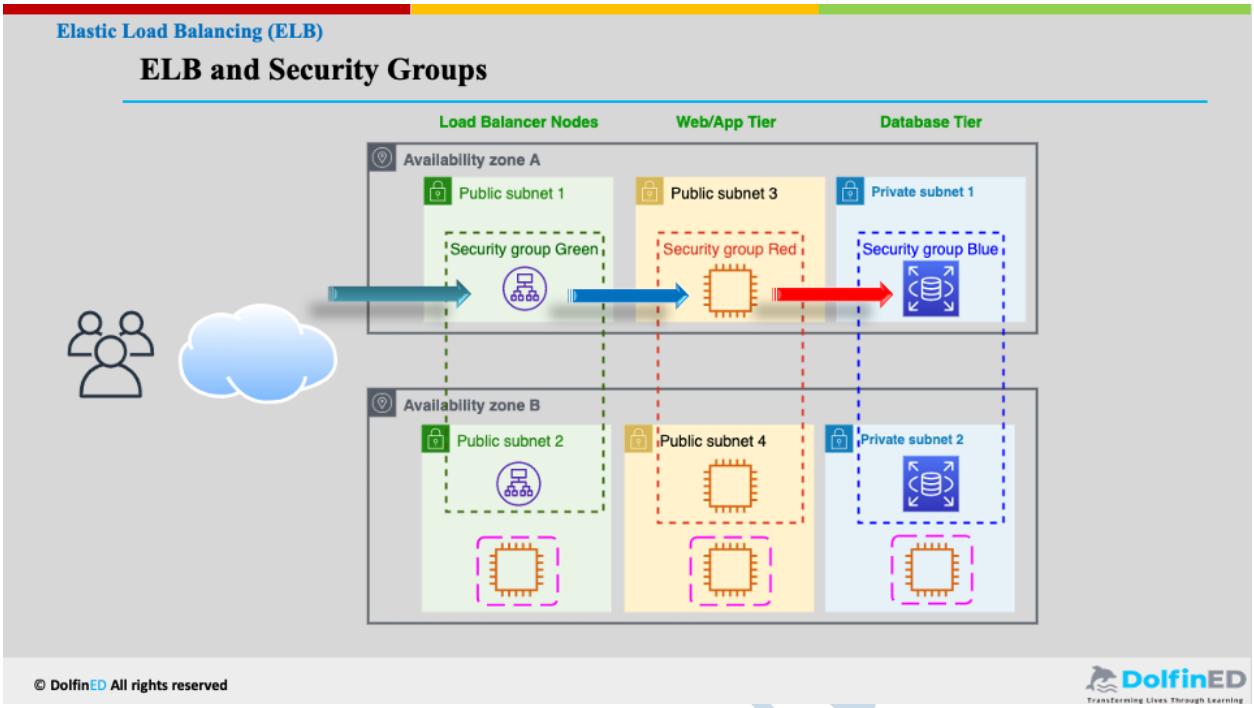
What if the application and database tiers need access to the internet for patching or software updates? What do we do then if we already placed them in private subnets?
The targets or backend instances are in private subnets and require access to the internet
For High Availability, we can deploy 2 NAT Gateways, one per AZ, to allow the private subnets' instances secure access to the internet.
This allows for HA access to the internet, where each AZ can route to the local NAT Gateway.

www.dolfined.com

ELB Security Groups



www.dolfined.com

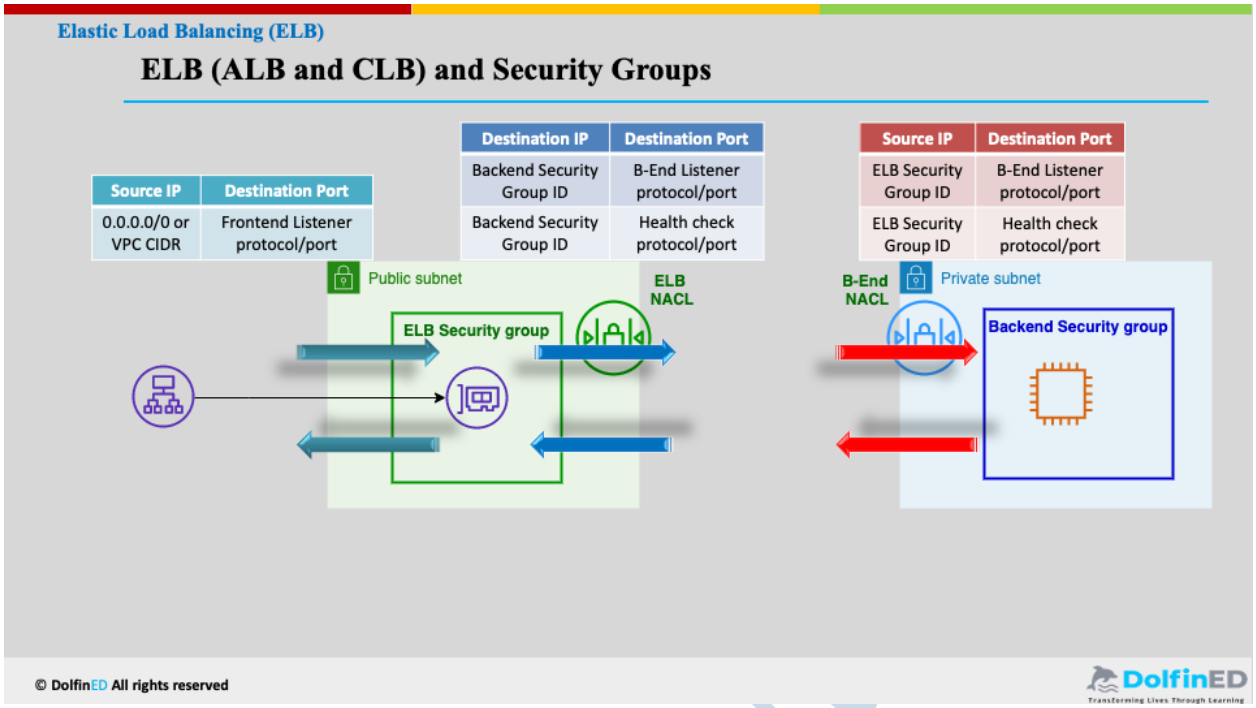


What if we have a two-tier application deployed as shown on the visual? We need to make sure that:

1) The web Application tier is only accessible by the ALB.
2) Database tier only accepts traffic from the Web/Application tier.

And we need to achieve this without blocking any other traffic.
How can we do this?
A hint, we can attach security groups to ALB and CLB

www.dolfined.com



CLB and ALB can both be associated with security groups.

NLBs do *not* have a security group associated. This is important to remember.

Let's have a look at the cases of CLBs and ALBs first.

In the architecture, we have on the visual,

- Traffic coming into the ELB will need to pass INBOUND through the CLB or ALB security group; It needs to be allowed.
- The source is 0.0.0.0 if this is an internet-facing ELB. It should be VPC CIDR if this was internal-facing ELB.
- The destination port will be the ELB frontend listener protocol and port.
- This traffic gets processed by the ELB, and now it is time to send it to the back end.

As can be seen now, the destination IP address is the backend range. If we have the backend in an auto-scaling group, it will be better to define the destination as the security group ID of the backend instances or targets.

The destination protocol and port are those of the backend (or target group) listener and health check protocol and port.

www.dolfined.com

This needs to be allowed in the ELB security group in the outbound direction.

Now this arrives at the backend or target's security group.

- It will now be in the inbound direction to this security group.
- The allowed source can be configured wide open as 0.0.0.0/0.
- This is very important to understand; if you want to limit who can communicate with the backend to the ELB nodes only, we need to configure the source as ELB security group ID.

Keep that in mind if we want to lock down who can communicate with the backend.

Also, keep in mind that we need to allow traffic destined to the listener protocol and port and those for the health checks.

The response to these packets or traffic that will eventually be sent back does not need to be explicitly configured since the security groups are stateful.

Two comments to keep in mind:

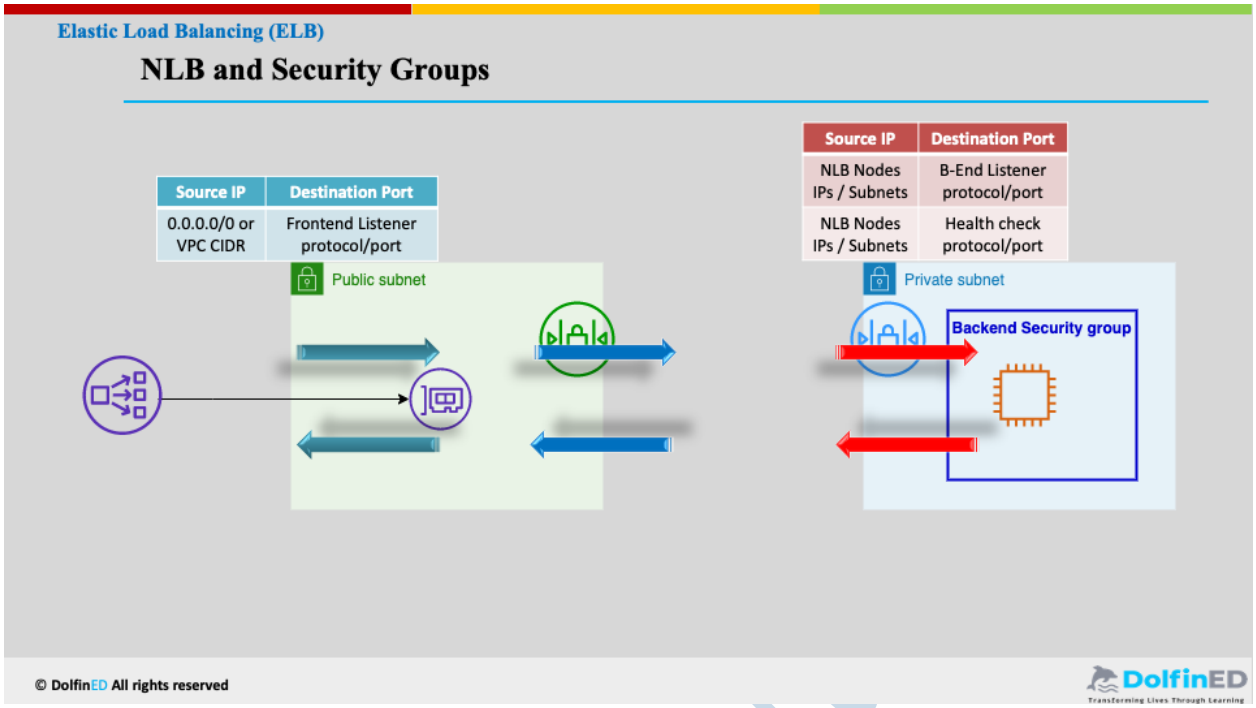
1. ELB has *nothing* to do with the outbound traffic that is initiated/generated from the registered EC2 instances and destined directly for the internet, nor to any other instances within the VPC.

Keep in mind that ELB has to do only with *inbound* traffic to the ELB and is intended for the EC2 registered instances (as the destination) and the respective return traffic.

2. Secondly, please note although we did not mention NACLs, you must also ensure that the subnets' NACLs allow traffic to/from the ELB both ways (on the front and backend side) as the NACLs are stateless.

www.dolfined.com

© DolfinED All rights reserved



NLBs do not have security groups associated.

In the setup, we have here,

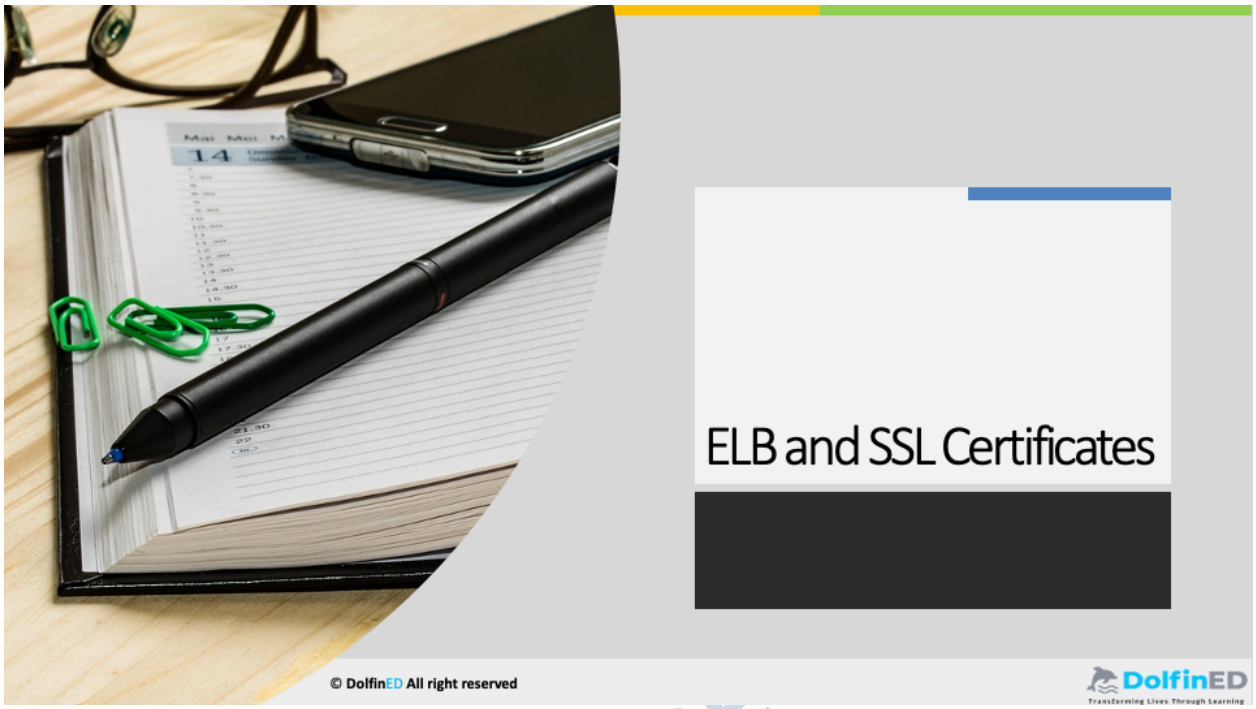
Traffic coming into the NLB will need to pass INBOUND through the NLB node. However, we cannot have a security group associated with the NLB. This traffic gets processed by the NLB and then send it out towards the backend. The destination IP address is the backend or target IP address. Moreover, the destination protocol and port are those of the backend (or target group) listener and health check protocol and port.

When the traffic arrives at the back end or target's security groups, it will be inbound to the back-end or target security groups. The backend security group can be configured where we have the source wide open as 0.0.0.0/0, which is not a good security practice. Instead, if we want to limit who can communicate with the backend to the ELB nodes, then we need to configure the source as the NLB CIDRs of the subnets where NLB nodes are defined for each AZ.

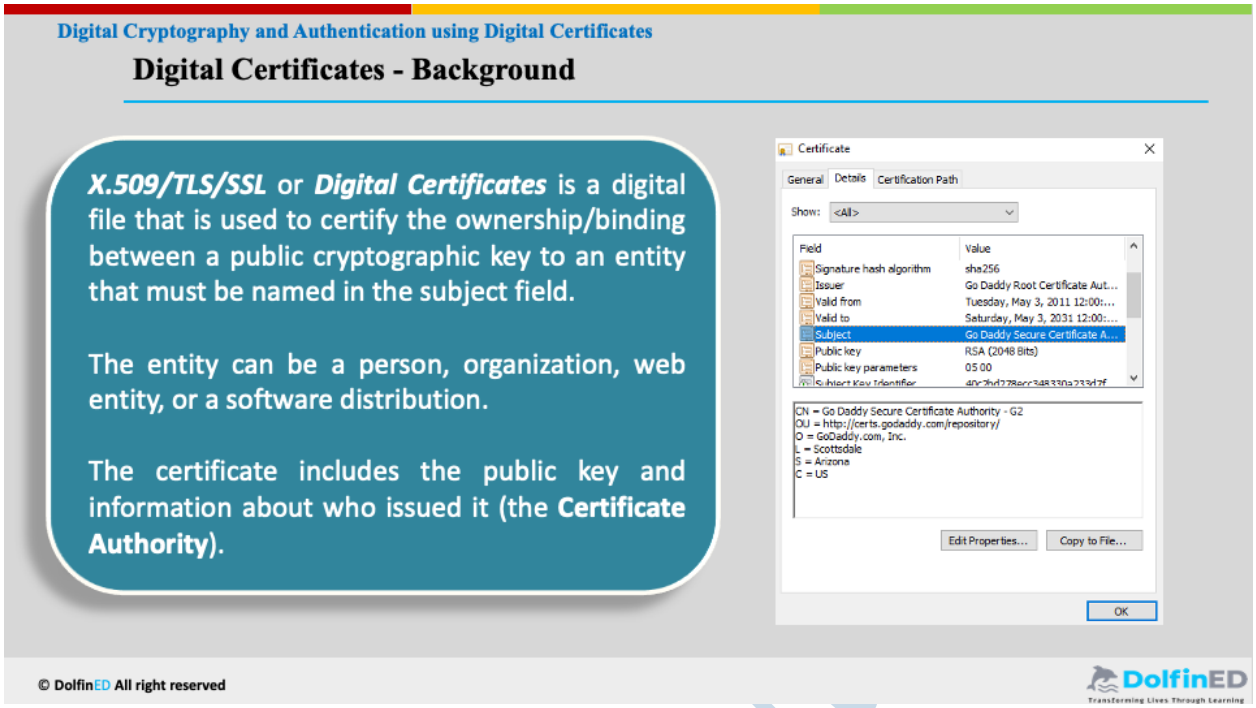
The response to these packets or traffic that will eventually be sent back does not need to be explicitly configured since the security groups are stateful.

www.dolfined.com

ELB and SSL Certificates



www.dolfined.com



A **Digital Certificate** is a digital file used to certify the ownership/binding between a cryptographic public key to an entity named in the subject field.

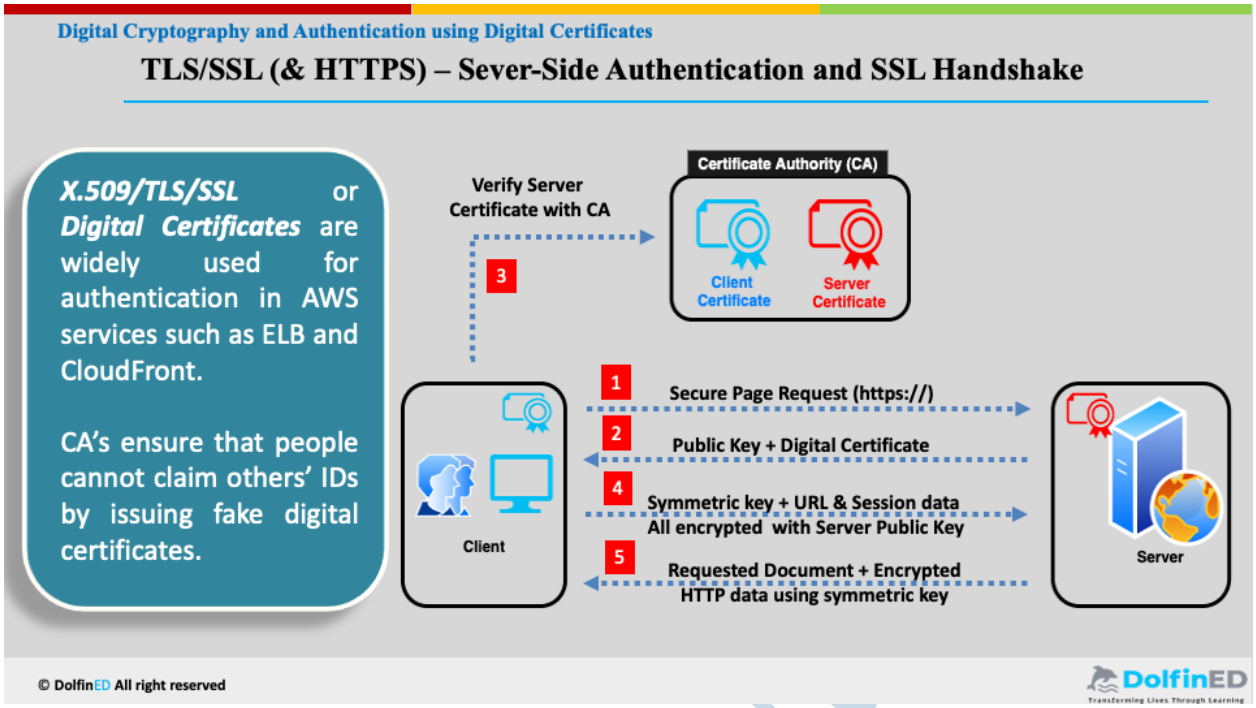
- It is issued following the **ITU-T X.509** format.

The entity can be a person, an organization, a web entity, or even a software distribution.

- Like a driver's license issued by an authentic entity.
- Or binding a public key's ownership to a webserver.

The digital certificate contains the public key and information about who issued the certificate (Certificate Authority, Self Signed, etc.), why it was issued, and other metadata information.

www.dolfined.com



Why do we need to learn about them in this course?

X.509/TLS/SSL or **Digital Certificates** are widely used today, and they are integral to authentication in any Cloud service provider services. For AWS services. This includes :

- Elastic load balancing.
- CloudFront.
- RDS database.

But how can we check the authenticity of a Certificate? This is where the Certificate Authority plays a role.

A certification authority (**CA**) can be a commercial or not-for-profit entity.

A CA performs the following actions:

- Accepts certificate request (issuing) applications from entities.
- Authenticates these applications to verify correctness.
- Issues the certificates.
- Maintains details and status of the issued certificates.

CA's ensure that people cannot claim others' IDs by issuing fake digital certificates.

Imagine that you are connecting to your bank website and want to do online banking. If you do not verify which server/domain you are connected to, your session might be redirected to a phishing website. In other words, a dangerous website that imitates the look of your bank's website but just wants to collect your username and password. And guess what happens after that?

www.dolfined.com

So how can your browser confirm it is connected to the right server? This is why we need Server-side authentication.

In *server-side authentication*, the client needs to validate that it is connected to the right server, while the server does not need to check who the client actually is.

In this case, the server has to present its certificate and its public key to the client upon receiving the client's secure connection request.

So let us look at the dialogue or the handshake that happens between the two. This is the basis of SSL, TLS, or HTTPS sessions.

Client browser:
"Hello there. I want to establish secure communication between the two of us. Here are my cipher suites and compatible SSL/TLS version."

Now comes the Server:
"Hello, Client. I have checked your cipher suites and SSL/TLS version. I think we're good to go. Here are my certificate file and my public key. Check them out."

Client:
"Let me verify your certificate." ... Client checks that with the CA (if not already available in the client browser).
The client again: "Okay, it seems fine, but I need to verify your private key. I will generate and encrypt a pre-master (shared secret) key using the server's public key. This shared secret key will be used as the symmetric key of the session to be. "

Here we go.

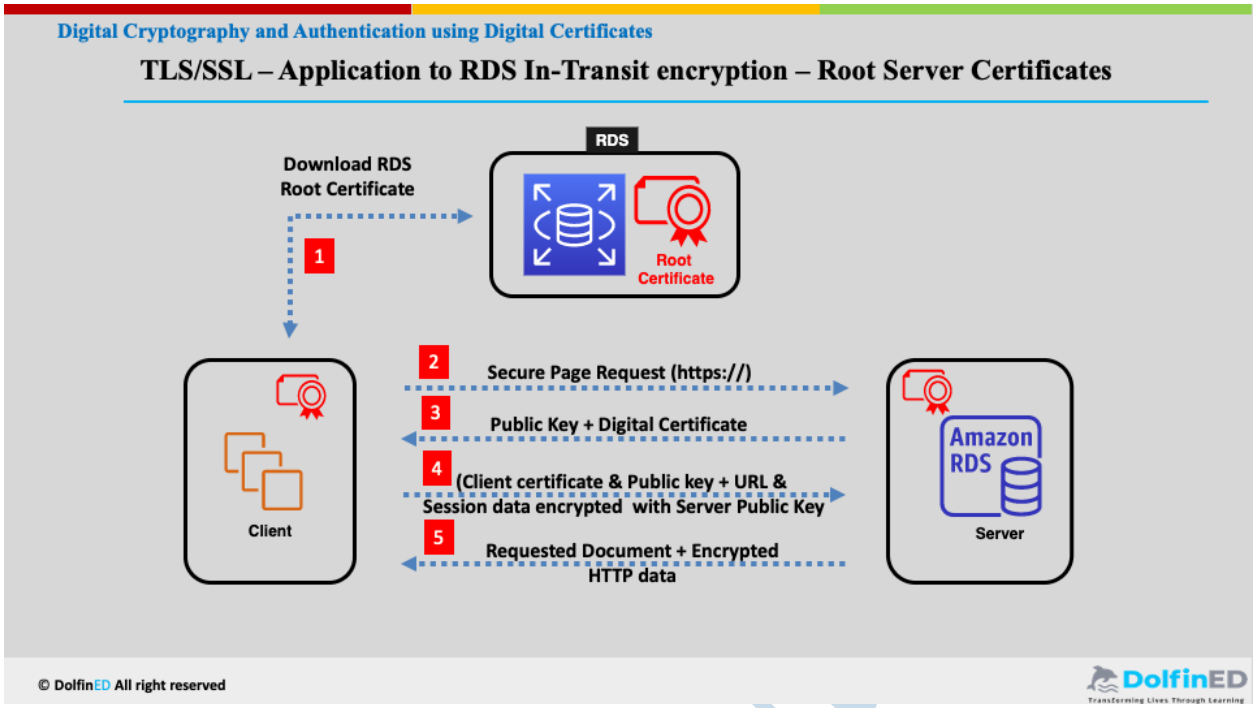
Server:
"Ok, let me go ahead and decrypt this using my private key. Alright, it was done. I got the symmetric master key. We will use this for the secure session. "

Now both can use the Master or symmetric key. This is also called symmetric encryption.

The client will send the server messages encrypted with the Master Symmetric key; the server will check whether it can decrypt them; now, life is good.

Server:
"Here we go, the data you requested, encrypted with the Symmetric Master key. "

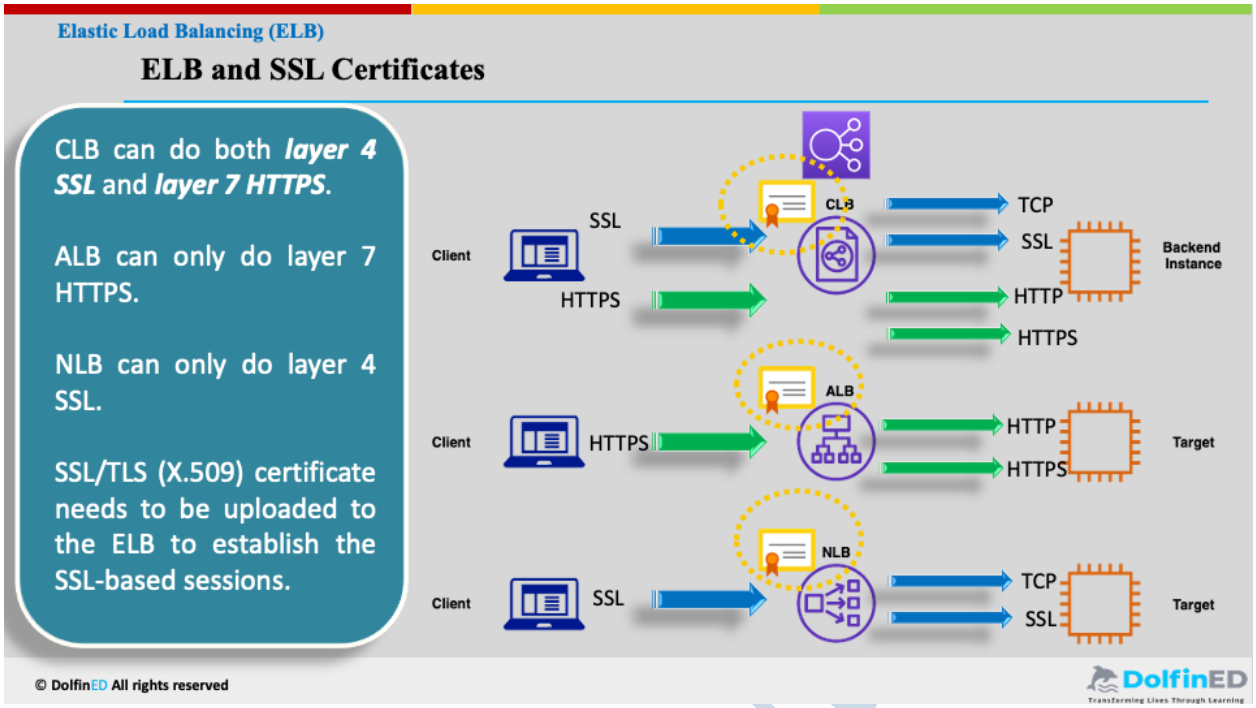
www.dolfined.com



If we need to establish encryption in-transit between an application and Amazon RDS, we need first to download the RDS root certificate to the DB instance while launching it.

In this case, the client is the Application that needs to establish an encrypted in-transit SSL or TLS session with the RDS database. At the beginning of the SSL handshake, the RDS instance will respond to the SSL session request by its public key and RDS root certificate.

www.dolfined.com



Let us now look at what we must do to ensure that the communication between clients and ELB is secure and encrypted.

This is also called **encrypted in-transit** or **in-flight.**

We need to load an SSL server certificate uploaded to the ELB to negotiate and establish secure sessions.

The certificate can be obtained from *Amazon Certificate Manager (ACM)* or the customer's certificate that can be uploaded into ACM/IAM.

1. Classic load balancer supports SSL at layer 4 and HTTPS at layer 7.

Front end SSL can be terminated on the ELB and then TCP listener configured on the backend.

Or it can be both SSL front and backend.

With CLB, you can do HTTPS front end, with HTTP or HTTPS on the backend.

2. For ALB:

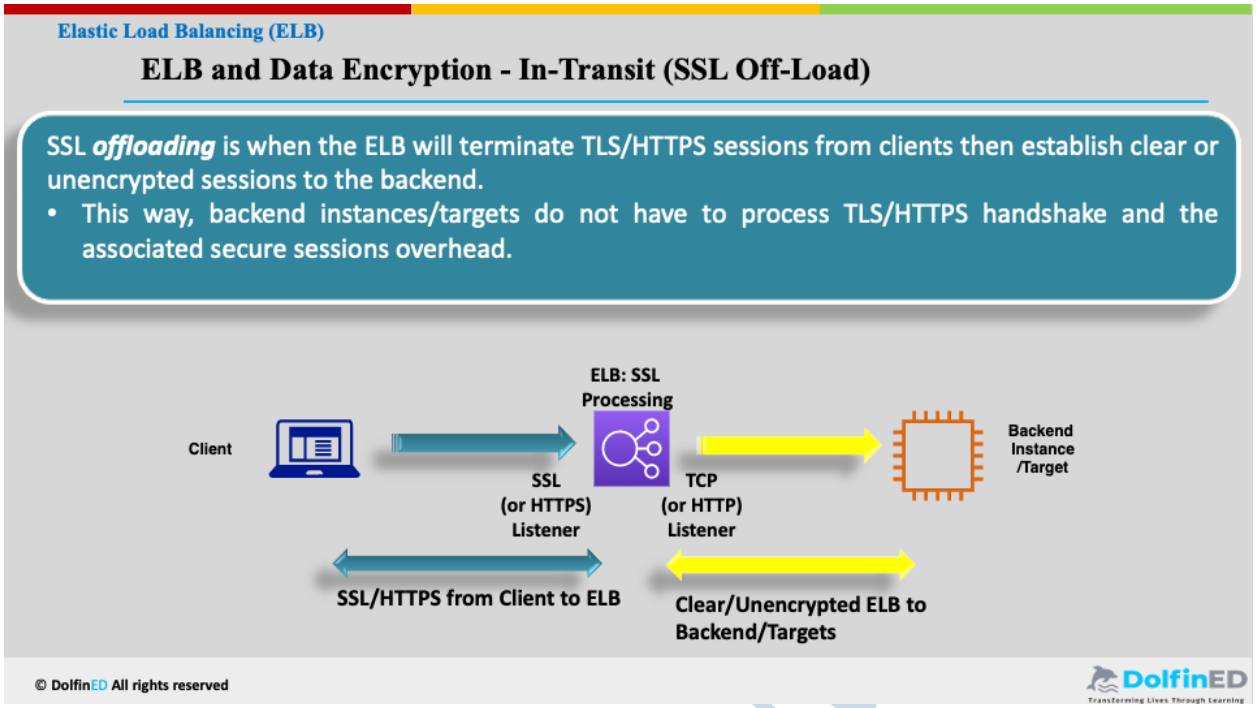
You can only do layer 7 HTTPS front-end listener and forwarding to the target group on HTTP or HTTPS.

3. For NLB:

You can only do SSL at layer 4 for the front-end and either SSL or TCP for target group forwarding.

www.dolfined.com

© DolfinED All rights reserved



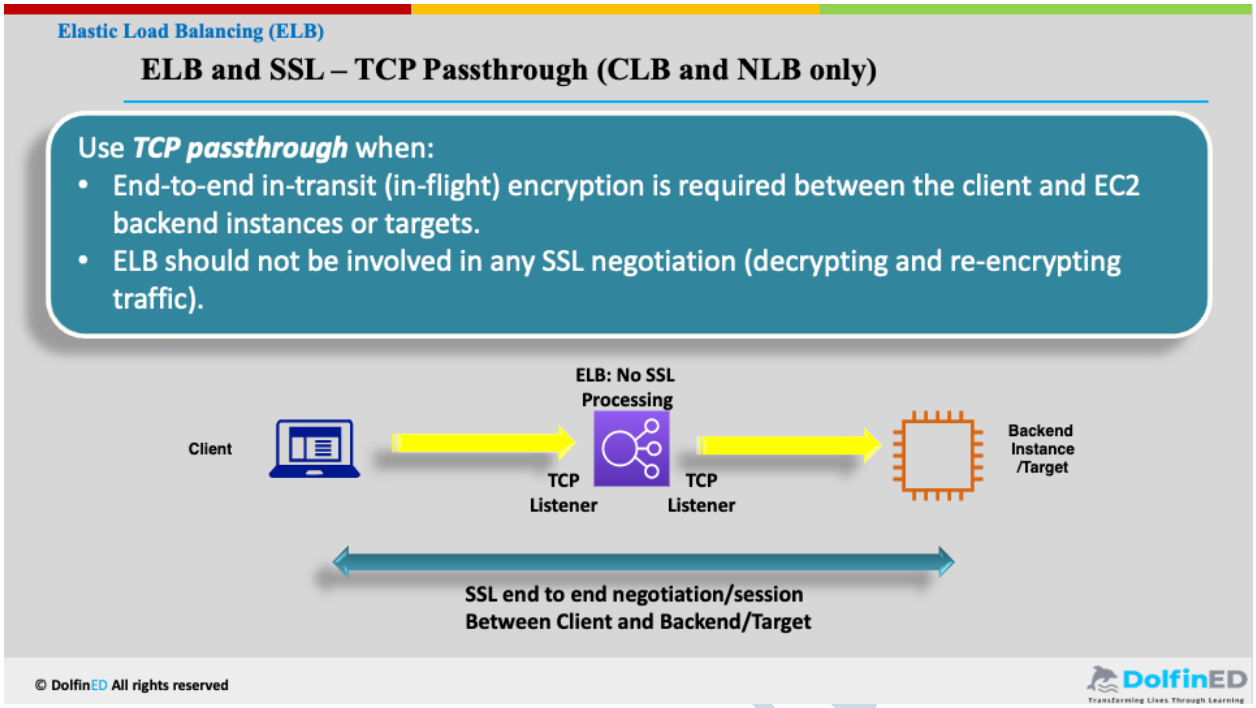
Encryption in Transit

ELB can terminate HTTPS and TLS from clients at the load balancer. This simplifies the process of building secure web applications. This way, the load balancer performs the work of encrypting and decrypting the traffic. This will offload the targets from doing TLS encryption/decryption tasks. Traffic from the load balancer to the backend instances or targets is clear or unencrypted.

This is also called an **SSL offloading configuration**. The client's traffic will need to be decrypted first on the ELB, then re-encrypted before being sent to the backend.

Please note that this is encryption in-transit for in-flight sessions between clients and the ELB, but not end-to-end encryption in-transit. If end-to-end encryption is required, ELB also supports TLS and HTTPS between ELBs and targets.

www.dolfined.com



Some corporate security architects and Chief Security Information Officers (CISO) may not like that the data is decrypted on the ELB first before being re-encrypted to be sent to the targets. Or when the encrypted sessions need to be between the targets and clients without the ELB getting involved in SSL or TLS.

One solution to this would be to deploy end-to-end encryption, where the client negotiates SSL or HTTPS directly with the targets instead of with the ELB. The ELB will not participate in SSL or HTTPS processing at all.
This configuration is supported by CLB and NLB only since this is a configuration at Layer 4.
ALB **cannot** do this since it functions at layer 7.

To do so, you simply disable SSL processing on the ELB and have it process TCP traffic, including that on port 443, as normal TCP.

Doing that, we have ...

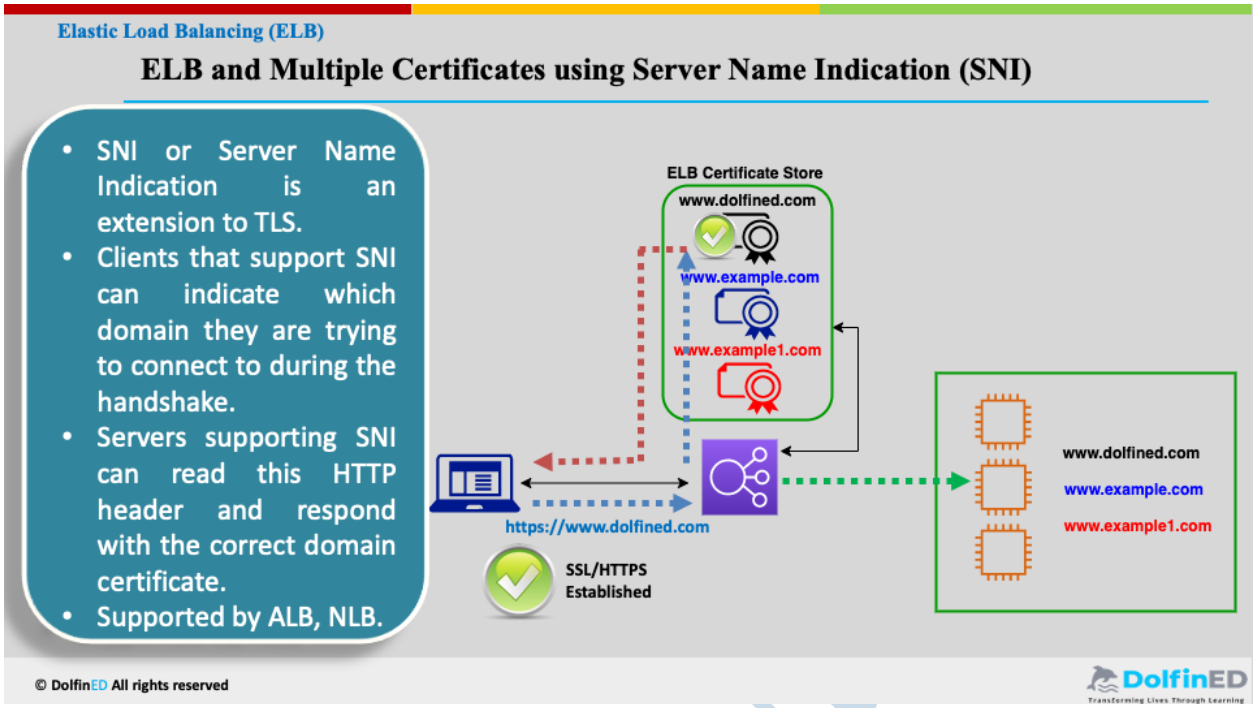
TCP listeners configured for the front end and backend listeners on the ELB

www.dolfined.com

SSL traffic from the client will be forwarded to the EC2 backend or target as TCP traffic. Negotiation happens, an end-to-end session gets established.

www.dolfined.com

© DolfinED All rights reserved



To use an ELB to terminate SSL for multiple domains, it needs to be able to have multiple certificates uploaded to it, right?

This way, we can save costs and have one ELB load balance to multiple applications or domains.

The problem now, or the question is, how would the ELB know which request from a client matches which certificate?

This is where Server Name Indication or SNI comes in handy.

SNI or Server Name Indication is an extension of the TLS protocol that allows multiple certificates to be hosted on a single IP address/Port. Therefore, SNI is not a certificate type itself.

The browser supporting SNI can indicate in the HTTPS requests (HTTPS) which hostname or domain it is trying to connect to at the beginning of the handshake process. If the ELB supports SNI, it will read the hostname or domain from the Client HTTPS request and fetch the correct certificate to complete the negotiations.

SNI is supported by ALB and NLB only, not the CLB.

This means we can host multiple apps or domains on the same backend targets and tie multiple **SSL Certificates** to them using a single ELB with a single IP address and same listener port.

Let us look at this quick animation to understand how SNI works on ELB.

www.dolfined.com

Let us consider a company on AWS that hosts three different websites: www.dolfined.com, www.example.com, and www.example1.com, all on port 443 listeners.

Each of these three domains has its own TLS/X.509 domain certificate.

The company wants to use ELB services and use the ELB to do SSL offloading. This means the ELB will terminate SSL on-behalf of the backend targets.

To satisfy the requirements, we need to load these certificates to the ELB and have the ELB terminate and negotiate SSL/HTTPS connections with the clients.

The flow will be as follows:

1. The client sends a request for an HTTPS session to the ELB, indicating in the SNI HTTP header that this is intended for www.dolfined.com.

2. The ELB checks the request, reads the header, and finds out it is for www.dolfined.com.
It checks its certificate list and finds a certificate for that domain in its certificate store.

3. The ELB responds with www.dolfined.com's certificate and its public key to the client and continues to negotiate the HTTPS/SSL session.

Everything is cool, and the session gets established.

4. Now, the ELB establishes the connection with the backend or target and forwards the traffic. This connection to the target can be HTTP or HTTPS.

The same method applies if we were establishing an SSL connection since HTTPS is HTTP over SSL.

5. If another request comes in destined to another domain, then the ELB will recheck the certificates list to see if it has a certificate for that domain; if yes, it responds with that certificate, then the same steps follow.

Please remember that SNI is supported in ALB and NLB only.

Certificate List

Using a certificate list enables the load balancer to support multiple domains on the same port and provide a different certificate for each domain.

© DolfinED All rights reserved

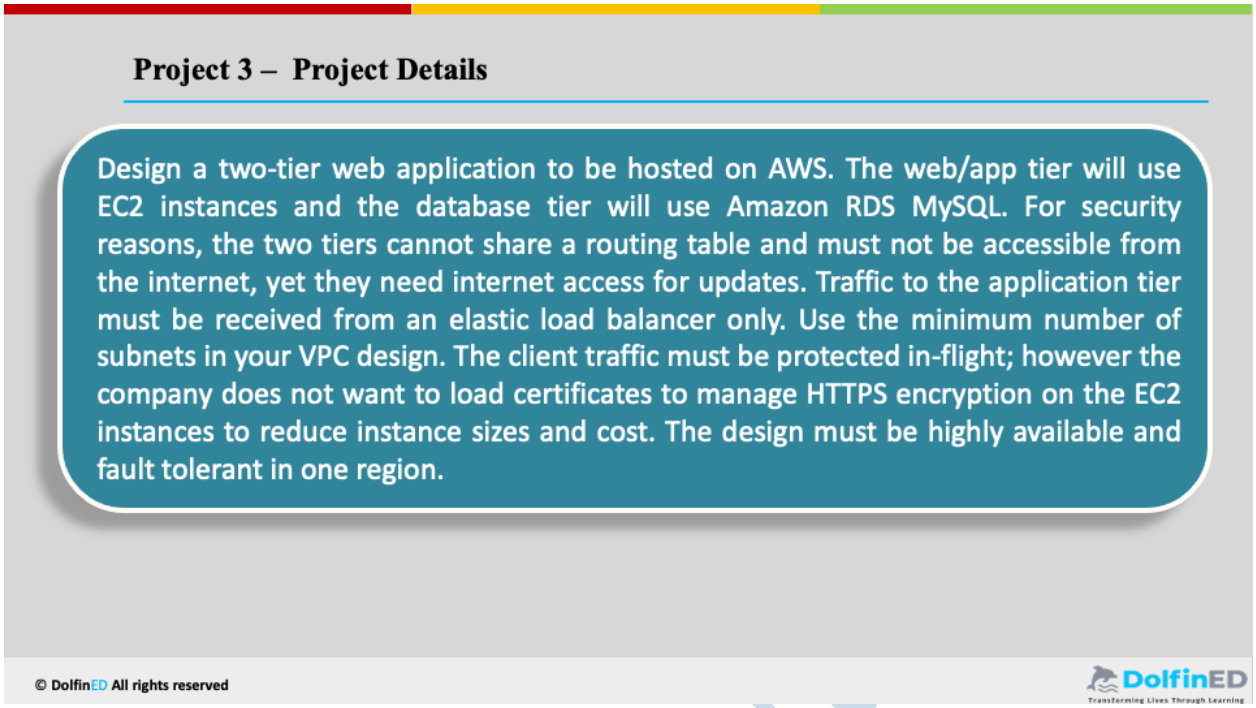
The certificates can be created by Amazon Certificate Manager (ACM), which can issue certificates. They can be customer-managed or created, in which case the customer uploads the certificate to IAM himself.

www.dolfined.com

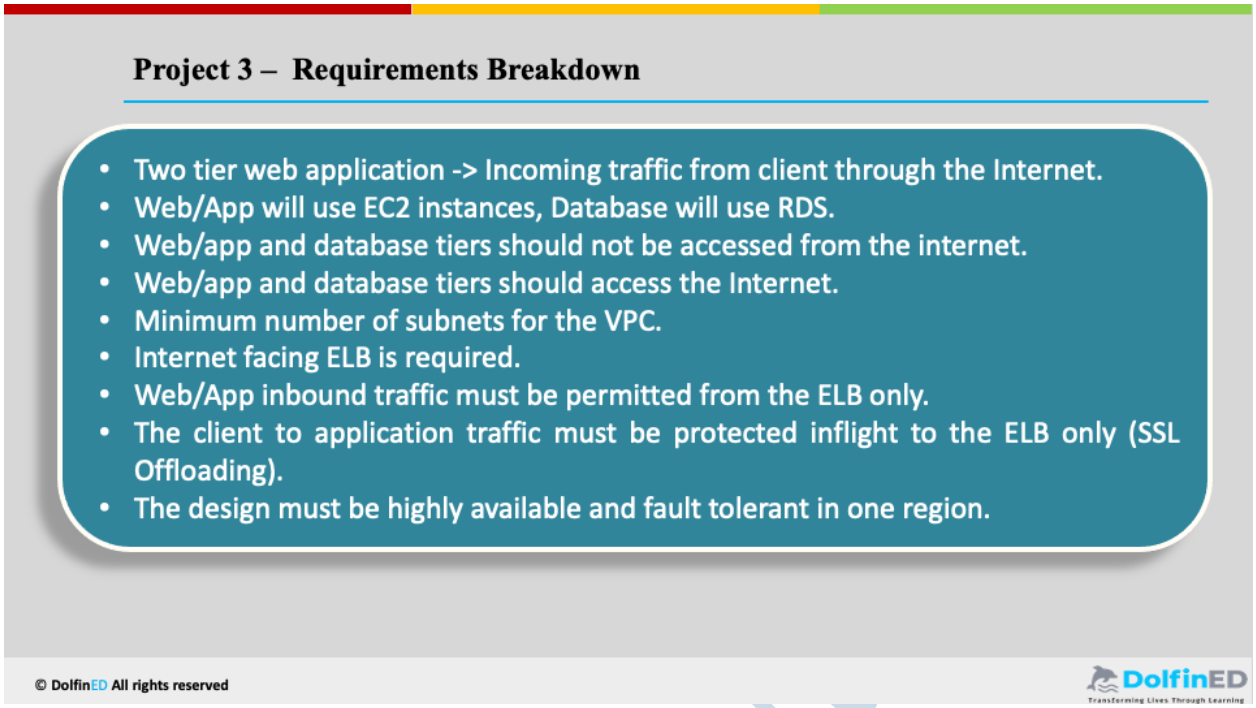
© DolfinED All rights reserved

Knowledge In Action – Project 3

www.dolfine

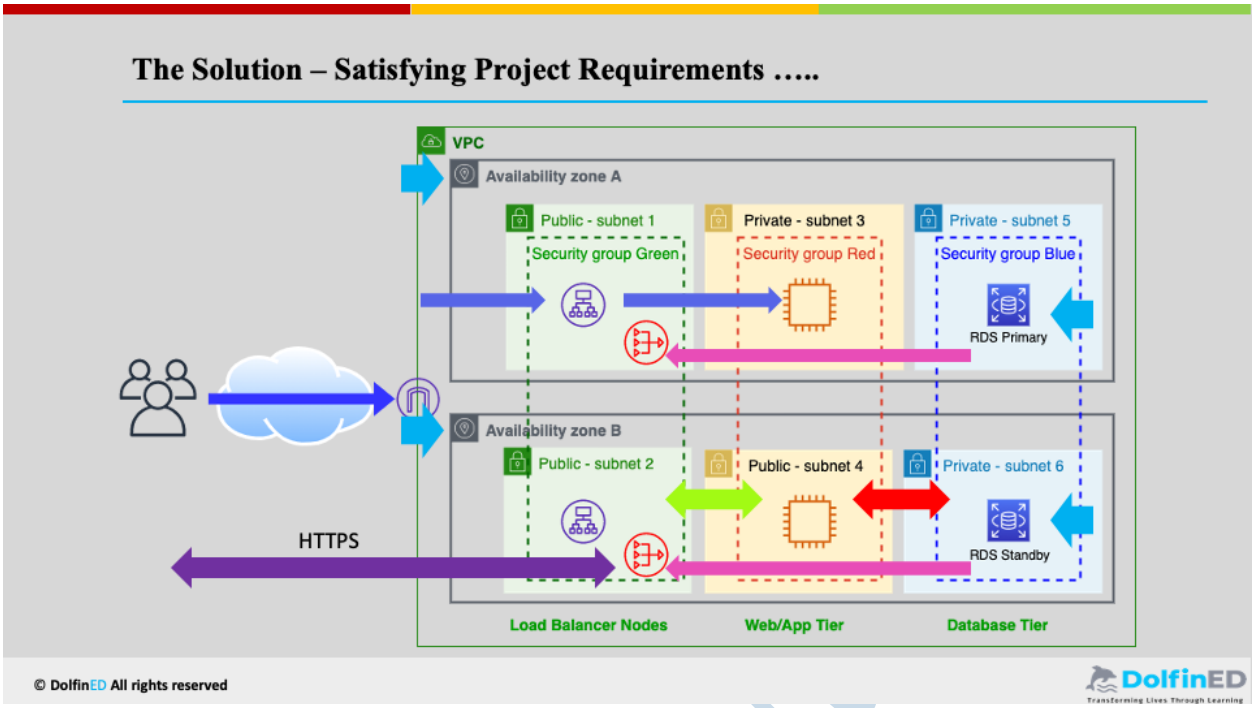


www.dolfined.com



A critical exam-taking skill is reading and analyzing simultaneously to breakdown the scenario details and visualize the solution.

www.dolfined.com



How many subnets?

Highly available and fault-tolerant in one region -> The solution must span at least two Availability zones.
We have to live with two AZs only for minimum subnets, as more AZs will mean more subnets.

Internet-facing ELB dictates a minimum of 2 public subnets for high availability.
Web/app and database cannot share route tables; they should then be on different subnets. This means separate subnets.
Web/app and database tiers must not be accessed from the internet – Private subnets, 2 each.
So a total of 4 private subnets.

RDS MySQL Multi-AZ for High Availability and Fault Tolerance.

Web/app and database tiers need internet access – NAT Gateway in HA configuration for high availability and fault tolerance.
Each AZ uses the local NAT Gateway in the same AZ for High Availability.

Traffic to the Web/App must only come from the ELB – Use Security groups – Inbound traffic permitted through the Web/App tier security group must be only from the ALB security group.
We could do the same for the RDS database security group to limit inbound traffic to only that coming from the web/app tier's security group.

Finally, Since we do not need to carry out encryption and decryption on the EC2 instances, we offload them to the ALB.

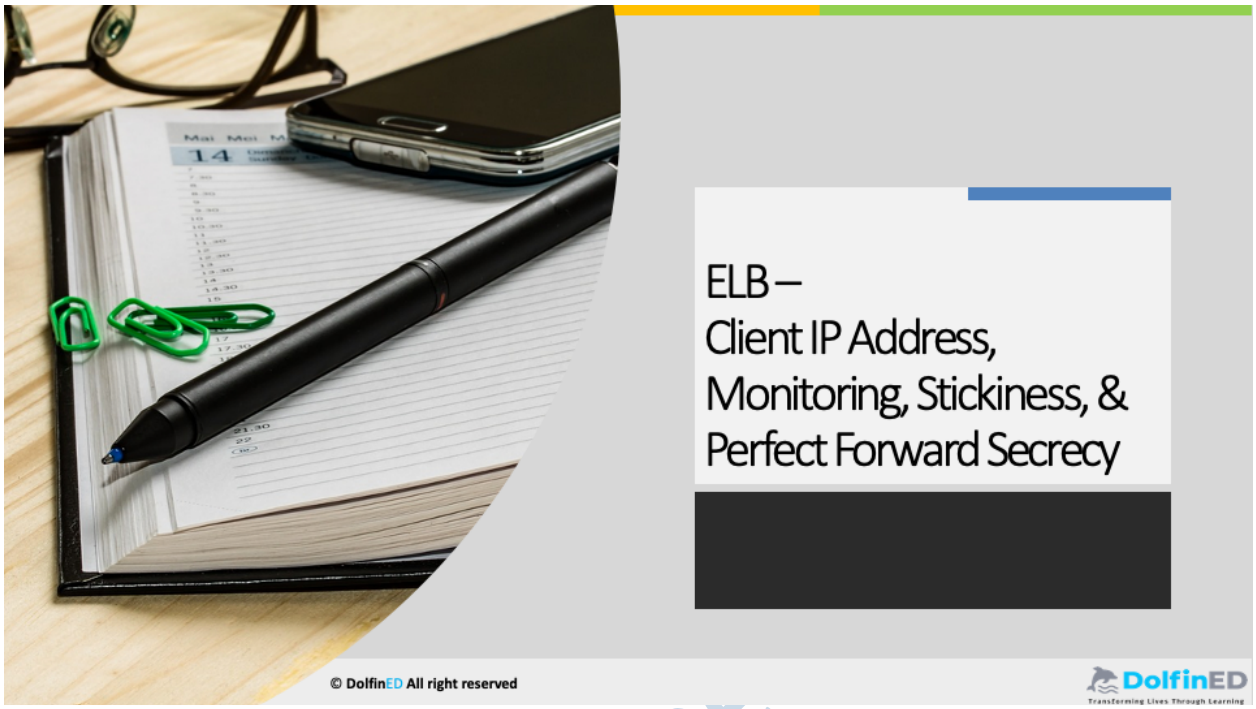
© DolfinED All rights reserved

We need to load an SSL certificate and configure HTTPS listeners on the ALB.
SSL will be between the clients and the ALB only.

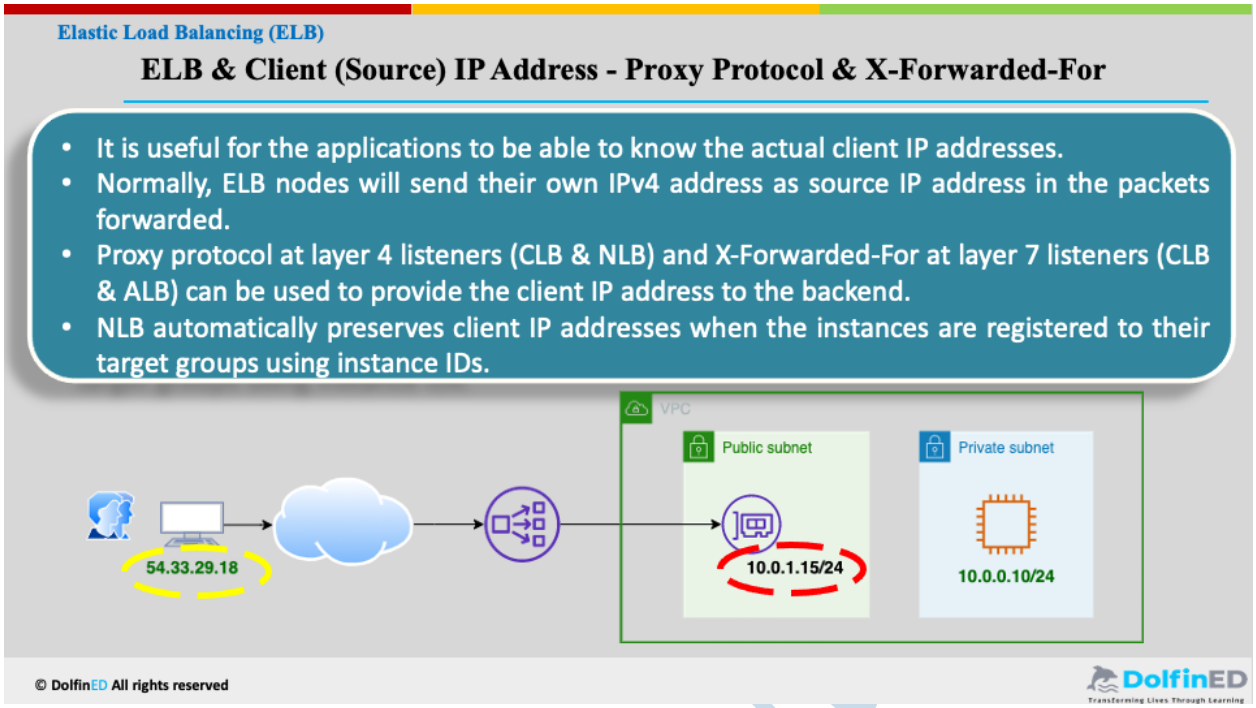
www.dolfined.com

© DolfinED All rights reserved

ELB – Client IP Address, Monitoring, Stickiness, and PFS



www.dolfined.com



Let's dive a little deeper and think about which source IP address appears in the packets received at the backend or target.

Is it the ELB node IPv4 address 10.0.1.15?

Or the actual client IP address who sent the request, which is 54.33.29.18?

It will be the ELB node's IP address. Except for one case with NLB that we will cover later.

What if we need to have the back-end application know who the actual client IP is that generated the request for whatever reason, such as security or filtering?

This is crucial when you want to whitelist or filter at the application level, not the ELB level.

How do we satisfy this requirement?

There are two ways to achieve this:

1. For layer 4 listeners: We can use Proxy Protocol

In this case, before sending the packet to the backend, the ELB will insert the actual client IP address in the packet payload, i.e., the actual data, not in the headers. This applies to both CLB and NLB.

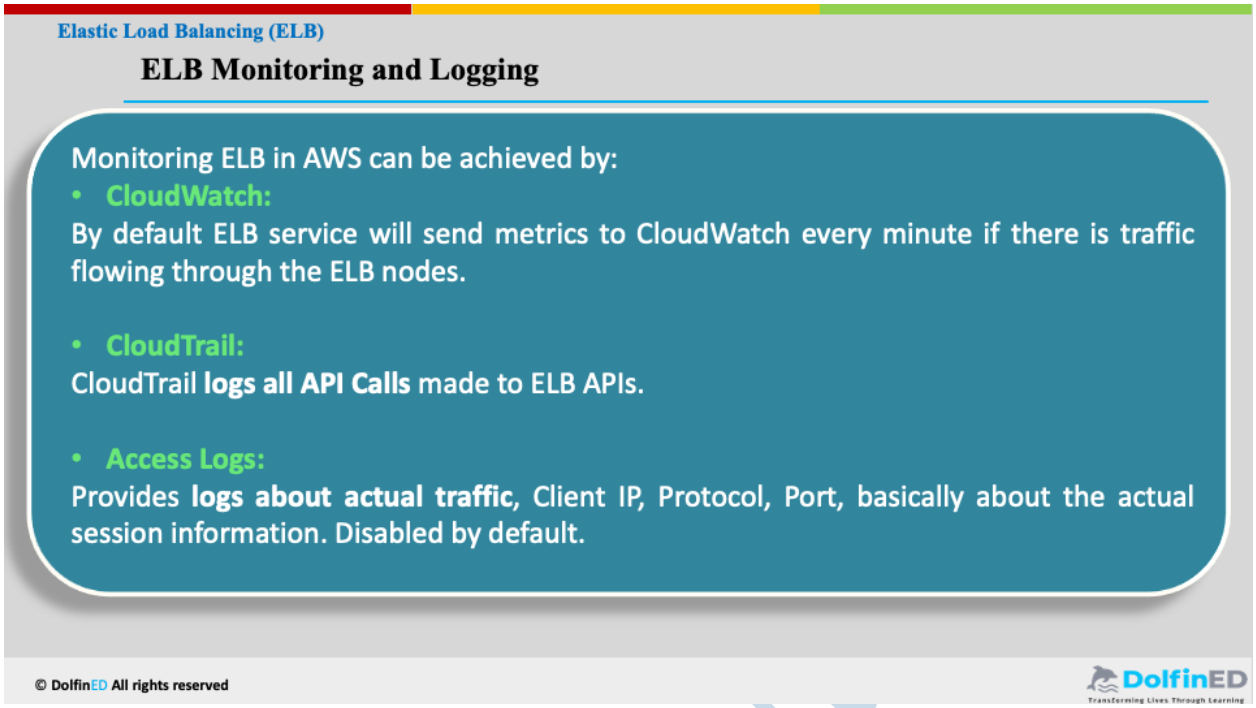
2. For layer 7 Listeners, we can use the HTTP header X-Forwarded-For

In which case, the ELB will copy the client IP address in a specific HTTP header. Of course, the backend must support X-Forwarded-For as well to be able to read it. This is natively supported by ALB and needs to be configured for CLB.

The exception to this where we do not need to do either is when using NLB with targets registered by instance IDs. The NLB will preserve the client IP addresses and not change them to the NLB IPv4 addresses before sending the packets to the backend.

www.dolfined.com

© DolfinED All rights reserved



ELB monitoring can be achieved by:

1. AWS Cloud Watch:

- AWS ELB service sends ELB metrics to cloud watch every "one minute."
- ELB service sends these metrics only if requests are flowing through the ELB.
- AWS CloudWatch can be used to trigger an SNS notification if a defined threshold is reached.

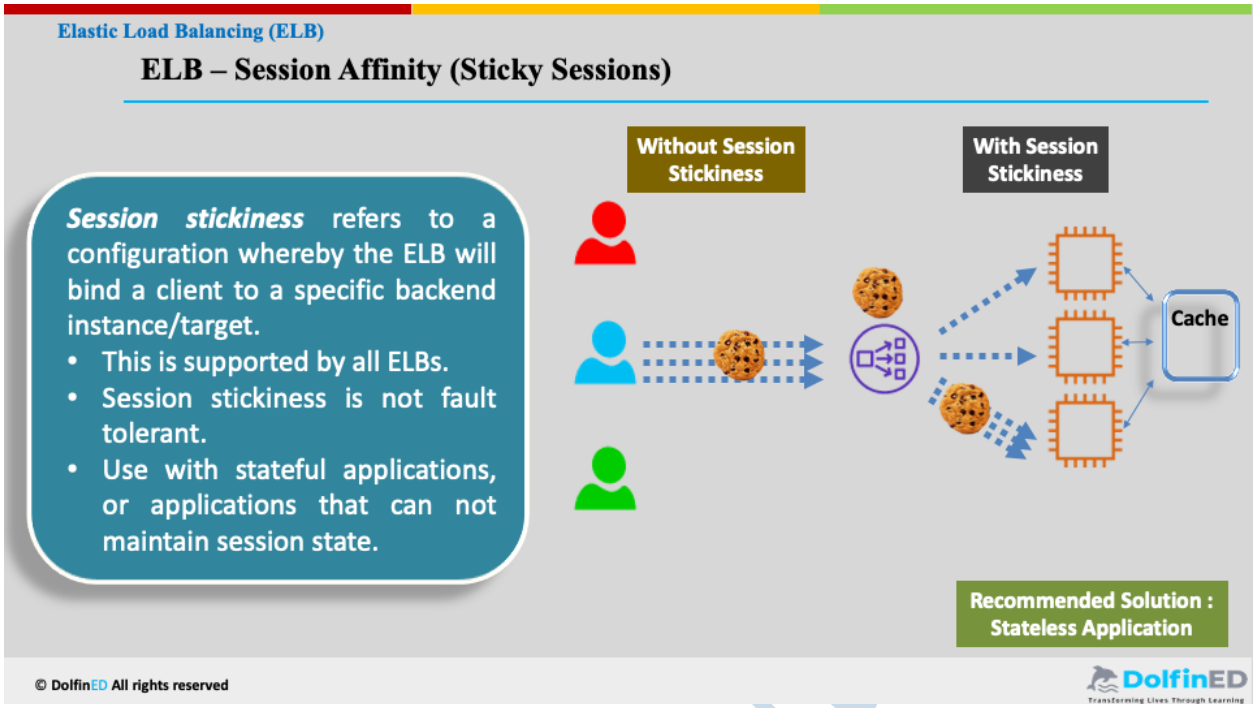
2. AWS Cloud Trail:

- You can use it to capture all API calls for your ELB, who did the call, what it was about, at what time, from which IP, etc.
- You can store these logs in an S3 bucket that you specify.

3. Access Logs:

- We can use it to obtain request information such as requester, time of the request, requester IP, request type, etc.
- This is optional and is disabled by default.
- We can choose to store the access logs in an S3 bucket; Storage costs will apply.
- There is *no* extra charge for enabling access logs.

www.dolfined.com



Session stickiness or **session affinity** is a mechanism to route client traffic to the same backend instance or target in a target group. This is useful for applications or servers that maintain state information to provide a continuous experience to clients.

Session stickiness is not fault-tolerant; if the backend instance to which the sessions are directed fails, all users will have their sessions interrupted.

For session stickiness to work, it requires a cookie inserted in the response of the first request to bind future requests from the same client to the same backend EC2 instance. Clients must support cookies for this to work.

The duration of the session stickiness is either:

- ELB duration-based, the ELB inserts a cookie and tracks it to maintain stickiness, supported by the CLB and ALB.
- Or application-cookie based, which is supported by CLB only.

For NLB and ALB, the session stickiness is configured at the target group level.

(Click on without a cookie)

Without Session affinity or sticky sessions, if the same client keeps sending traffic, the traffic gets distributed among backend instances or targets in a round-robin fashion.

However, if we configure session stickiness

The way this works is as follows:
When a load balancer first receives a request from a client, it routes the request to a target,

The ELB generates a cookie that encodes information about the selected target, encrypts the cookie, and includes the cookie in the response to the client.

The client should include the cookie that it receives in subsequent requests to the load balancer. When the load balancer receives a request from a client that contains the cookie, if sticky sessions are enabled for the target group, and the request goes to the same target group, the load balancer detects the cookie and routes the request to the same target.

Since stateful application architecture is not fault-tolerant, what can we recommend to build a tolerant one?

The recommendation would be to rewrite the application to become stateless, i.e., maintain a state in a cache or a database of some sort, and have the application write and read the session state from that cache. This way session state is maintained in a fast cache (called *in-memory cache*), and sessions do not have to be bound to a target anymore.

All targets using application cookies can read state information from memory. If any target or backend instance fails, any other one can pick the session up from where the failed one left off and continue without a user impact.

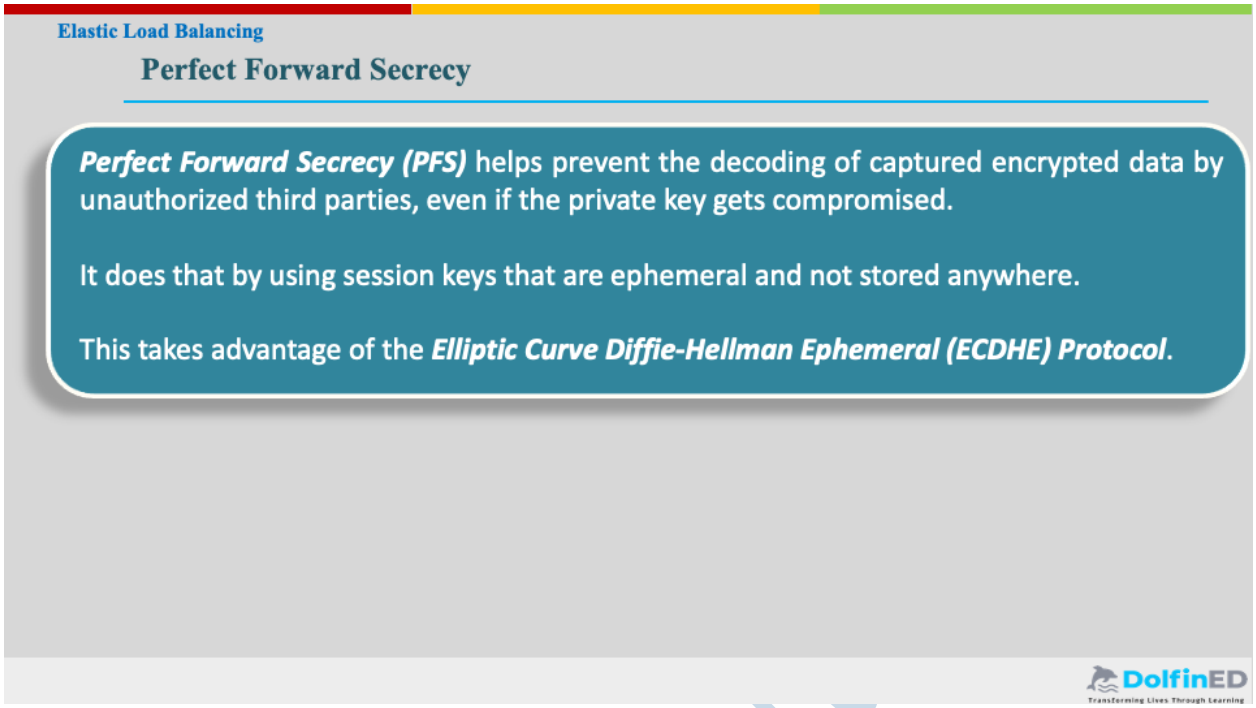
This way, it becomes a fault-tolerant and stateless application architecture.

Of course, examining the cache fault tolerance needs to be done as well, as if the cache fails, all state information is gone too.

We will go over that later in the course.

www.dolfined.com

© DolfinED All rights reserved



Amazon ELB now offers advanced cipher suites that use the **Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)** protocol.

ECDHE allows SSL/TLS clients to provide Perfect Forward Secrecy, which provides safeguards against eavesdropping on encrypted data using session keys that are ephemeral and not stored anywhere.

This is done by generating a unique session key for every session a user initiates. This way, the compromise of a single session key will not affect any data other than that exchanged using that key.

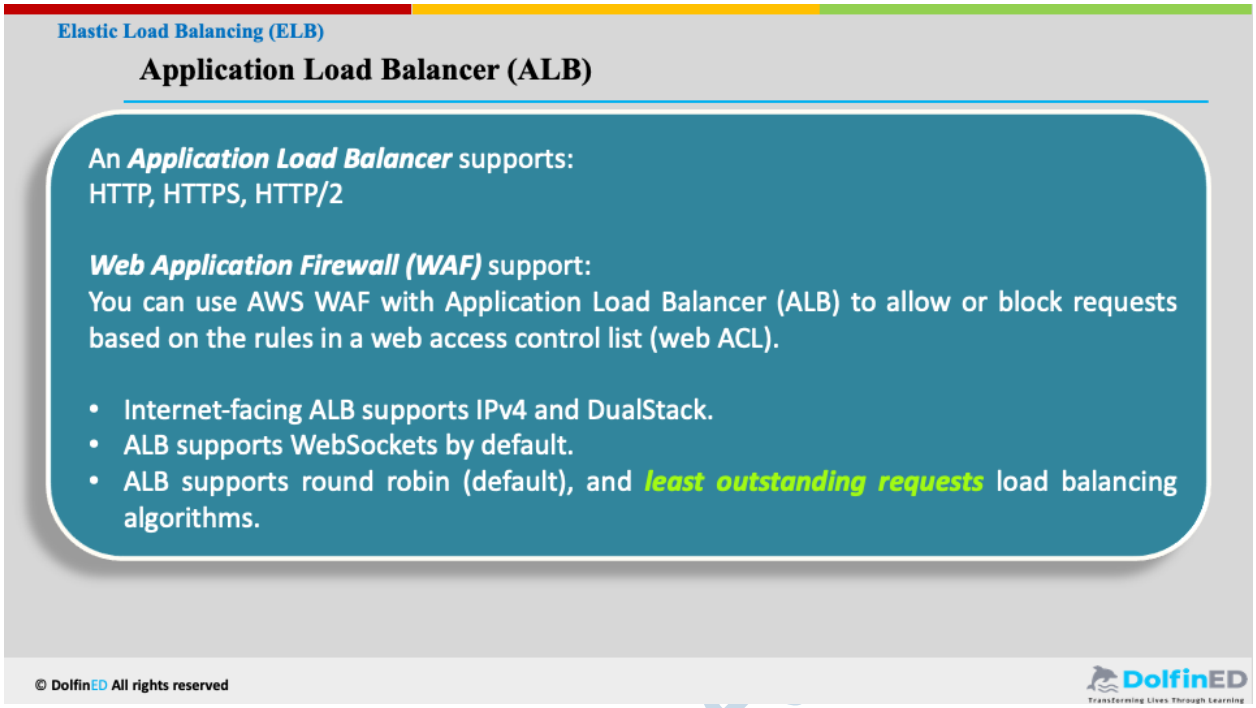
This helps prevent the decoding of captured data by unauthorized third parties, even if the server's private key itself is compromised.

This is supported by ALB, NLB, and CloudFront, as will be explained later.

© DolfinED All rights reserved

Application Load Balancer

www.dolfined.com



ALB supports HTTP, HTTPS, and HTTP/2 native support with HTTPS listeners.

This allows clients to send up to 128 requests in parallel using one HTTP/2 connection. While in HTTP 1.1, a client can send one request per connection.

In HTTP/2, when the ALB gets these requests on the same connection and from the same client, it converts them to individual HTTP1.1 requests and distributes them across targets.

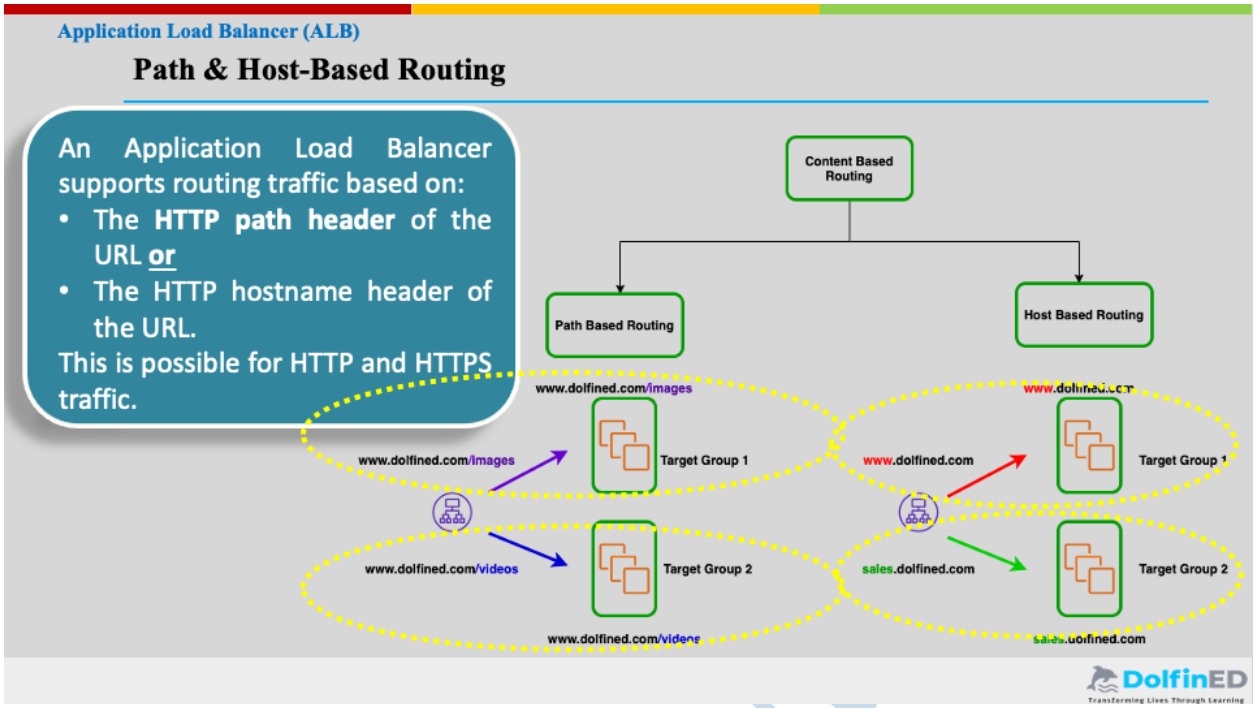
ALB can be integrated with AWS **Web Application Firewall (WAF)** to protect at layer 7. (More on this later on).

ALB supports IPv4 and dual-stack or IPv4 and IPv6.

- It also supports *WebSockets* protocol by default, which allows for full-duplex communication between the client and server over a TCP connection.
- We can use WebSockets with both HTTP and HTTPS listeners.
- WebSockets connections are inherently sticky.

The ALB supports two algorithms for load balancing.

- The default is **round-robin**.
- But you can also switch to the **least outstanding requests**, where the next request gets sent to the target with the least number of outstanding requests.



Content-Based Routing

An Application Load Balancer can route a request to a service based on the content of the request. Multiple types of content routing are supported on the ALB: host-based and path-based.

1. Path-based Routing (paths within the same domain name)

Where the ALB routes incoming HTTP and HTTPS traffic based on the path element of the URL path in the HTTP header of the request.

For example, let's say we want to serve images from a target group and videos from another target group both under www.dolfined.com.

We can send any request to www.dolfined.com/**images** to one target group and www.dolfined.com/**videos** to another target group using path-based routing.

Segmenting traffic this way gives us the ability to control the processing environment for each category of requests.

Why should we do this?

© DolfinED All rights reserved

Well, perhaps **/images** requests are best processed on a specific type of EC2 instance, while **/videos** requests are best handled by graphics-optimized instances.

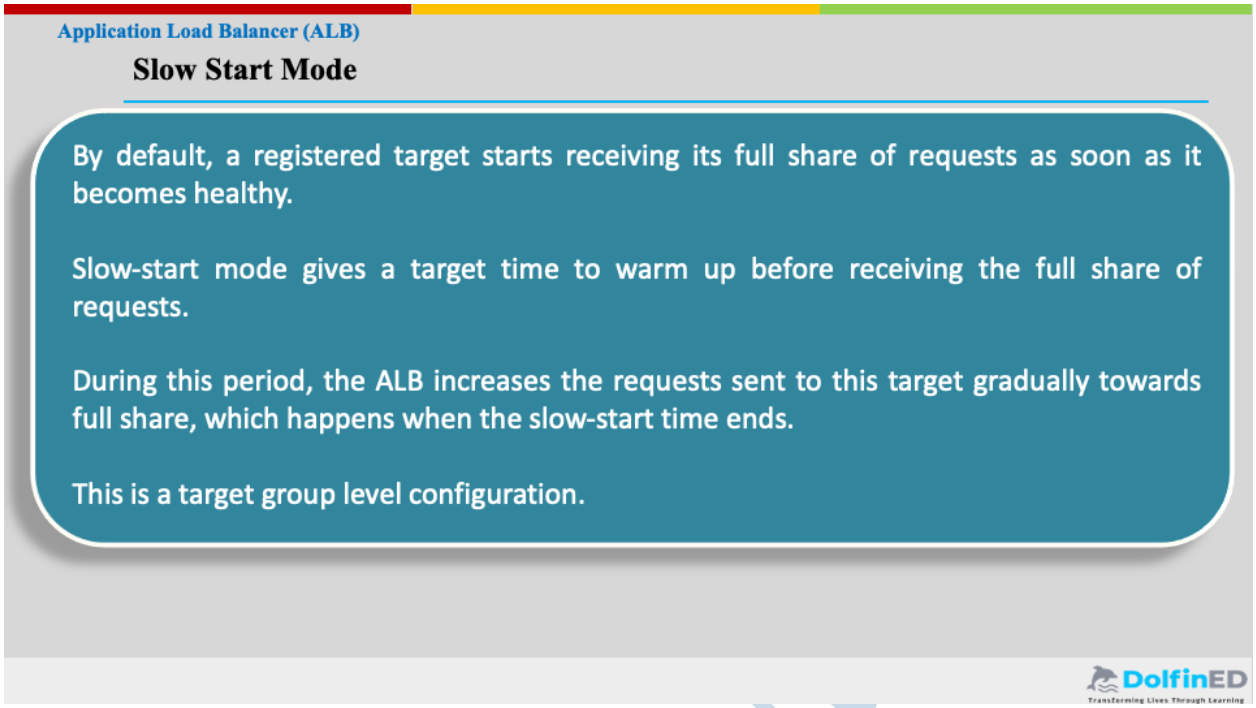
2. The second type of content routing is Host-based (Domain name based) Routing.

We can create ALB rules to route a client request based on the HTTP header's domain name host field, allowing us to route to multiple domains from the same load balancer.

HTTP request – Host field:

Requests to sales.dolfined.com can be sent to a target group, while requests to www.dolfined.com are sent to another.
The ALB supports routing based on Source IP address, the HTTP request method, Query String, or Host header.

www.dolfined.com



By default, a target starts to receive its full share of requests as soon as it is registered with a target group and passes an initial health check.

Using **slow-start mode**, targets get additional time to warm up before the load balancer sends them a full share of requests.

The load balancer linearly increases the number of requests sent to a target in slow-start mode. After a healthy target exits slow-start mode, the load balancer can send it a full share of requests.

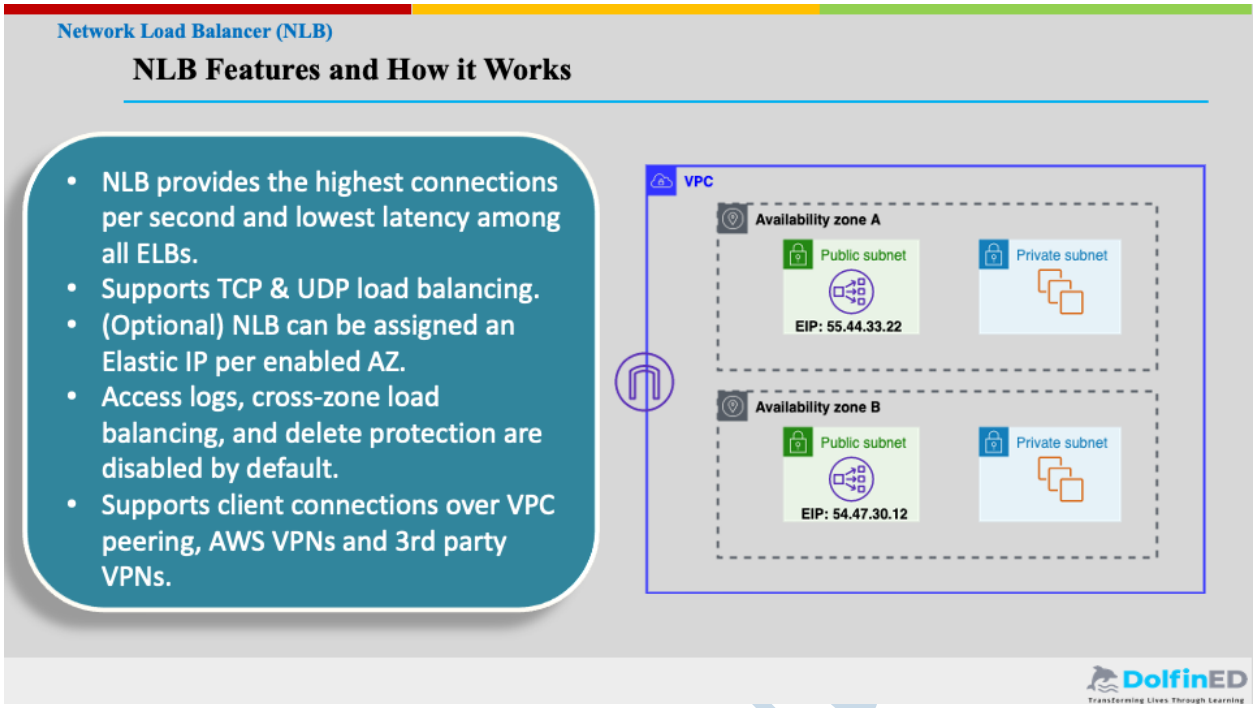
You can enable slow start for a target group.

www.dolfined.com

Network Load Balancer

Let us now have a closer look at the **Network Load Balancer** (abbreviated NLB).

www.dolfined.com



NLB can handle millions of requests per second and provides much lower latencies compared to other ELBs.

- Unlike other ELB types, NLB supports both TCP and UDP load balancing.
- Like the ALB, however, it supports routing requests to multiple applications on a single EC2 instance.

We can register each instance or IP address with the same target group using multiple ports.

NLB supports a **static or Elastic IP address** for the load balancer ENI created in the subnet where NLB is enabled (one subnet per AZ also means one IP per enabled AZ).

Any connections/requests traffic sent to the NLB's IP address will be distributed across the VPC subnets' targets in that AZ.

Assigning an elastic IP to the NLB in each AZ can be used in situations where:

- IP addresses need to be hard-coded into DNS records,
- Customer firewall rules (whitelisting) or similar needs.
-

Access logs, delete protection, and cross-zone load balancing is disabled by default on the NLB.

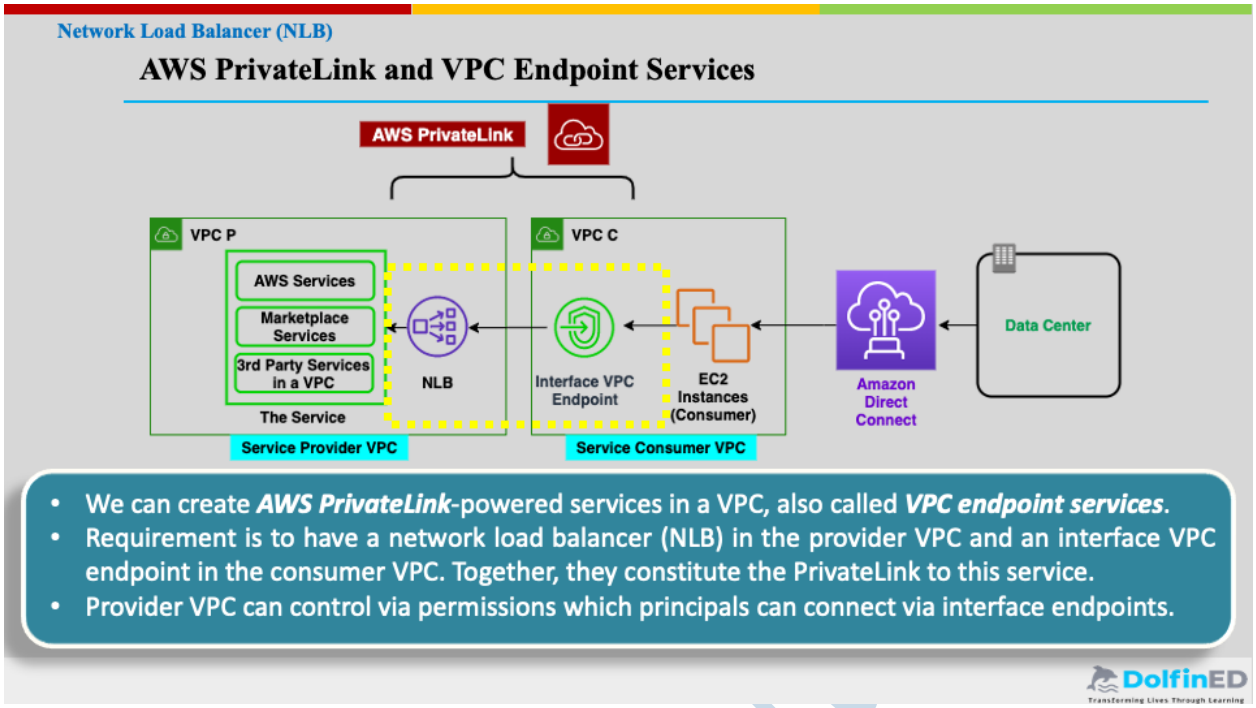
Network load balancers support connections from clients over:

- VPC peering,
- AWS VPNs, and

© DolfinED All rights reserved

- Third-party VPN solutions.

www.dolfined.com



Now let us talk about AWS PrivateLink and VPC endpoint services.
Let us consider a use case where a company A on AWS would like to share some critical business and marketing data with other companies as a service. The companies interested in this service may exist in AWS or on-premises.

The VPC that will host the services is called the **service provider VPC**. Whereas the ones that will use or consume the service are called **consumer VPCs**.

The solution must be easy to use and should not require updating the route tables, security groups, NACLs, or peering between VPCs.

It also can't use the internet for security, compliance, and regulatory requirements.

This can be achieved using VPC endpoint services, basically PrivateLink-powered services.

With endpoint services, a customer can create an application in a VPC and configure it as an AWS PrivateLink-powered service (also referred to as an *endpoint service*).

Other AWS consumers or principals, from an IAM perspective, based on permission granted to them, can create a connection from their VPCs to the endpoint service using an Interface VPC Endpoint.

Let us better understand the AWS Privatelink and how it works.

© DolfinED All rights reserved

We need to launch an NLB that load-balances to a target group hosting the EC2 instance where the application is deployed. We need to create an endpoint service and attach the NLB to that service, which indirectly attaches the targets or EC2 instance. Hence, the application or service.

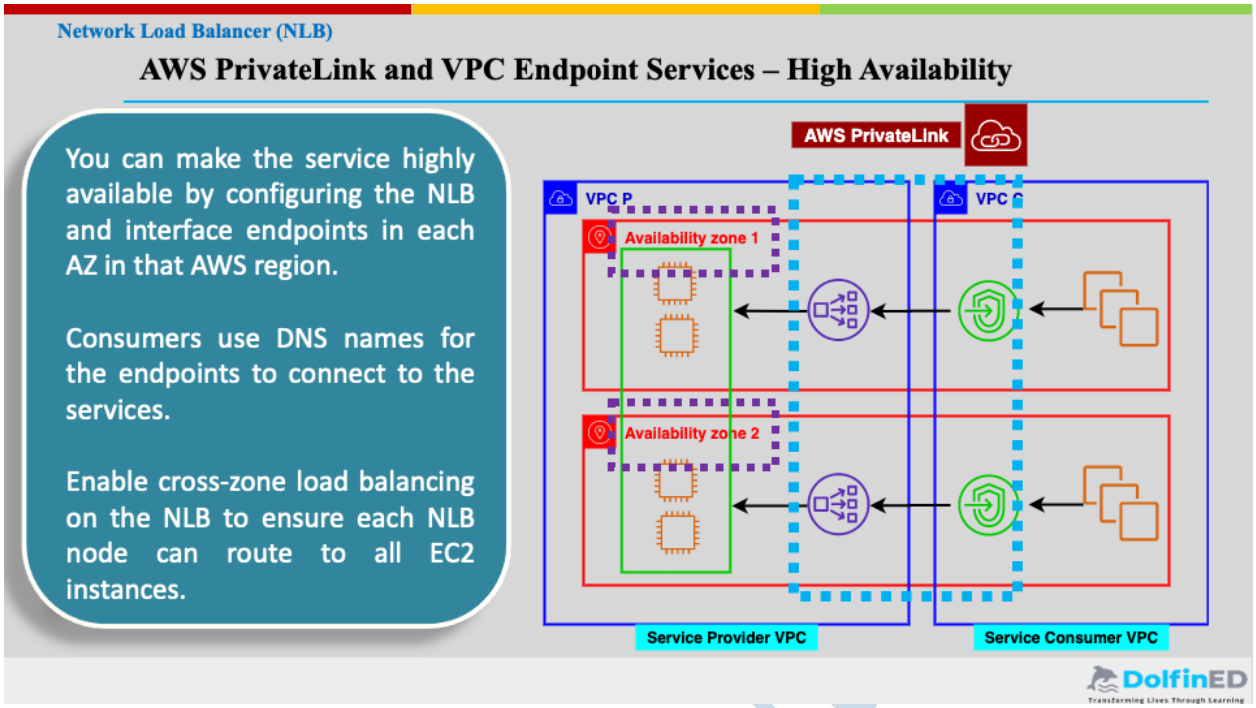
Decide on which principals (AWS accounts, IAM role, or IAM users) will be permitted to create VPC interface endpoints in their VPCs to connect to the NLB and thus to the service.

(Click on the AWS private Link RED rectangle)

The Interface VPC endpoint connection to the NLB is the PrivateLink

The consumer VPCs can configure one Interface VPC Endpoint per AZ. So how can we make this highly available?

www.dolfined.com



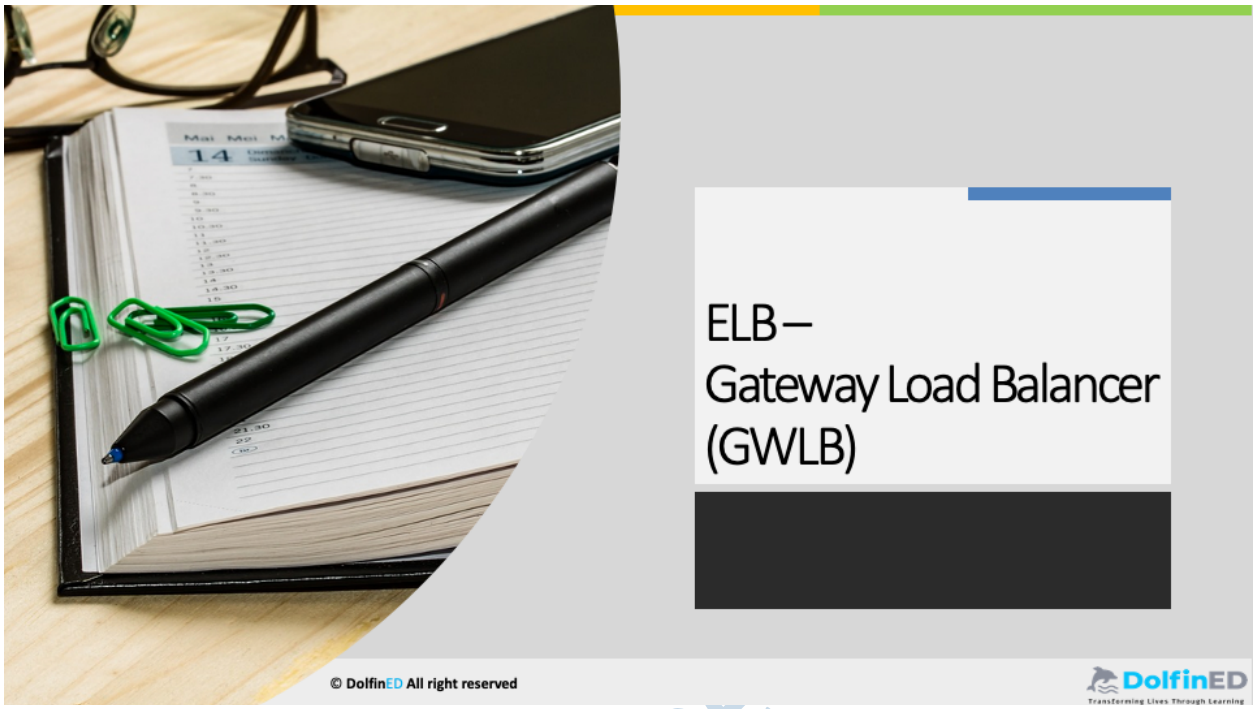
To make the Endpoint service access highly available,
We need to ensure that the application is deployed across multiple availability zones. Here we are showing two availability zones

Enable the NLB in those availability zones.
Register the EC2 instances with the Target group under the NLB.
In the consumer VPCs,
Configure an Interface VPC endpoint per AZ where the NLB exists.
Consumers can use Zonal DNS hostnames for the respective Interface VPC endpoint, and cross-zone load balancing is enabled on the NLB; we are assured that NLB nodes will route traffic to healthy targets in either AZ.

This way, we can have a highly available setup for the Endpoint Services, protected against an AZ failure.

© DolfinED All rights reserved

ELB – Gateway Load Balancer (GWLB)

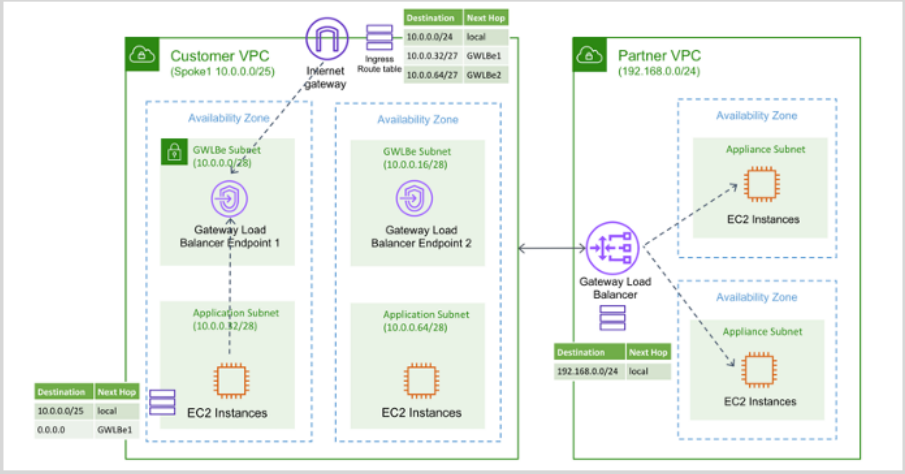


www.dolfined.com

Gateway Load Balancer

Gateway Load Balancer

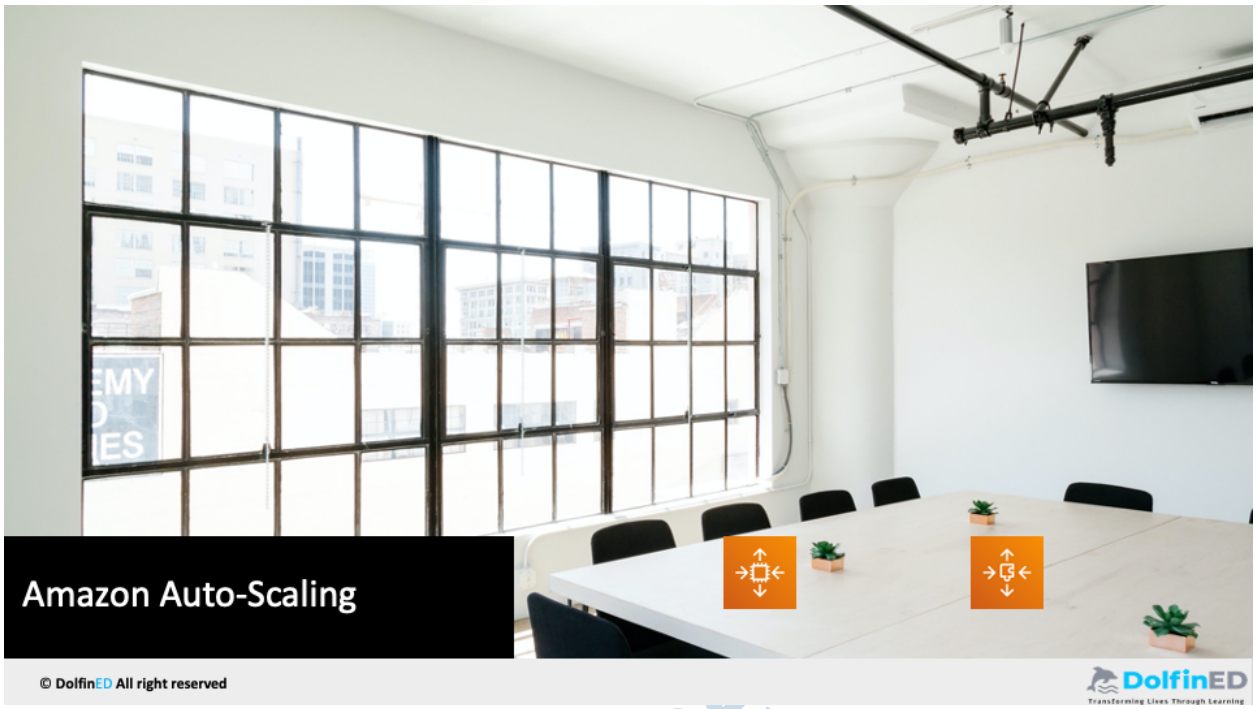
It is a service that makes it easy and cost-effective to deploy, scale and manage the availability of third-party virtual appliances such as Firewalls, Inrusion detection and prevention systems, and deep packet inspection systems in the cloud.



Source : https://aws.amazon.com/

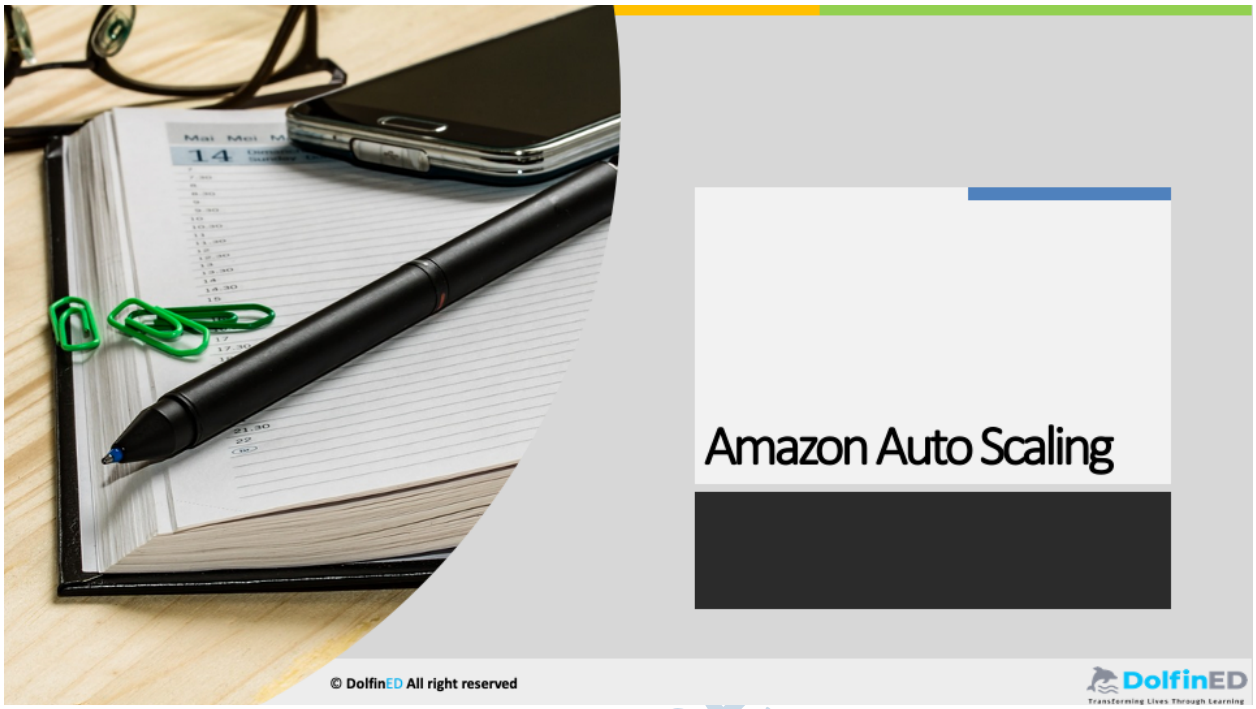
www.dolfined.com

Amazon Auto Scaling

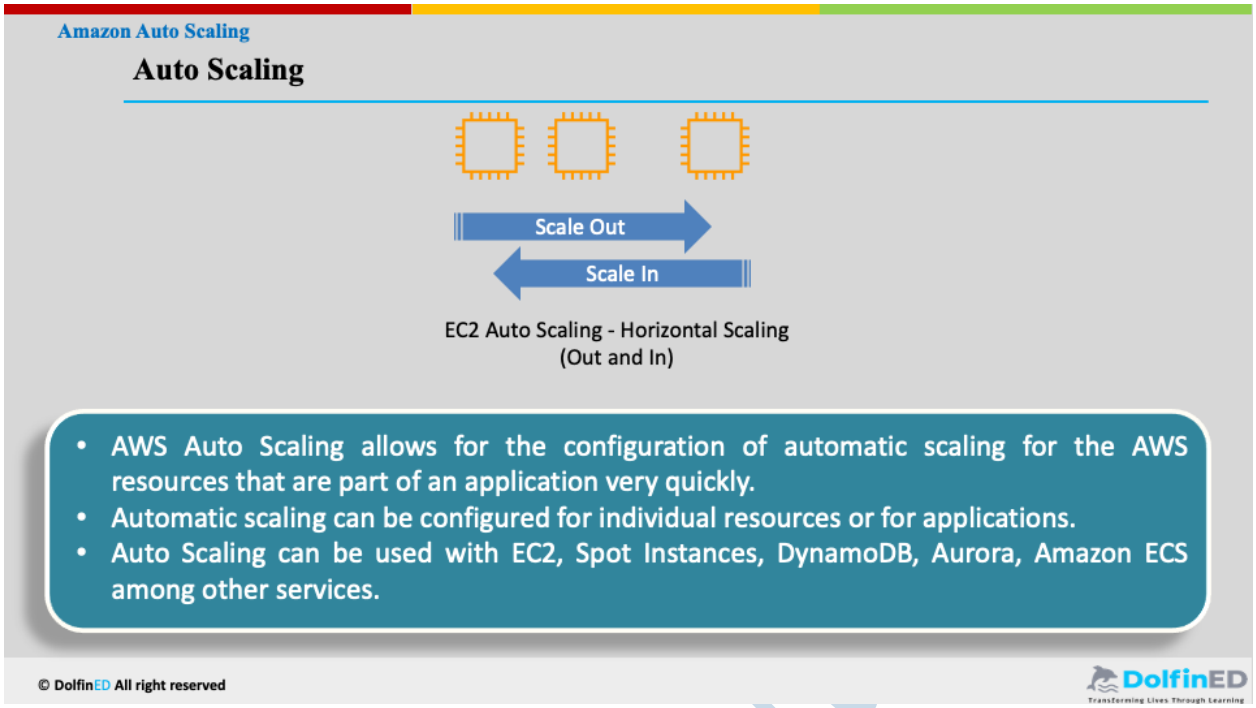


www.dolfined.com

Auto Scaling



www.dolfined.com



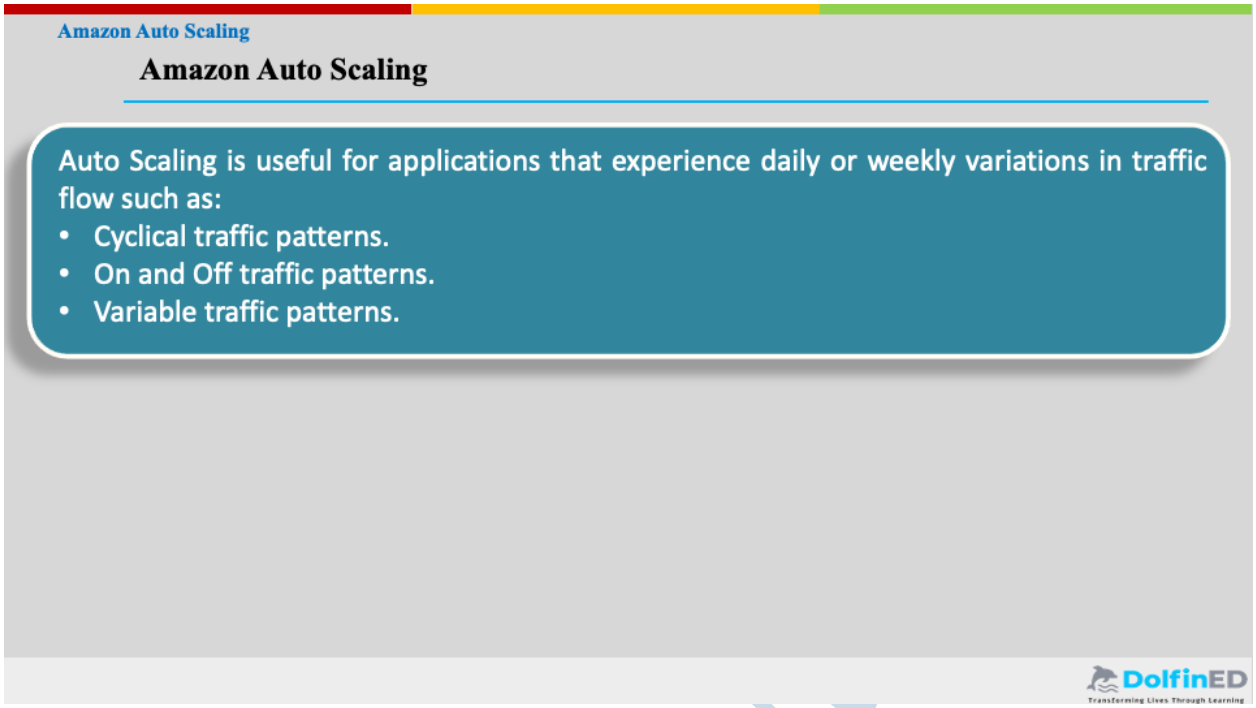
AWS Auto Scaling is different from EC2 auto-scaling. However, they both serve a similar purpose: to enable automatic scaling for AWS resources that are part of the architecture or application.

Amazon Auto Scaling is a service you can use with EC2, databases, and other services or applications to ensure that the solutions are elastic and can adapt to workload changes.

If we can automate scale-out and scale-in processes based on load, then we have an elastic solution/design.

The Auto Scaling service in AWS can cooperate with many AWS services, like EC2, DynamoDB, Aurora, and others.

www.dolfined.com

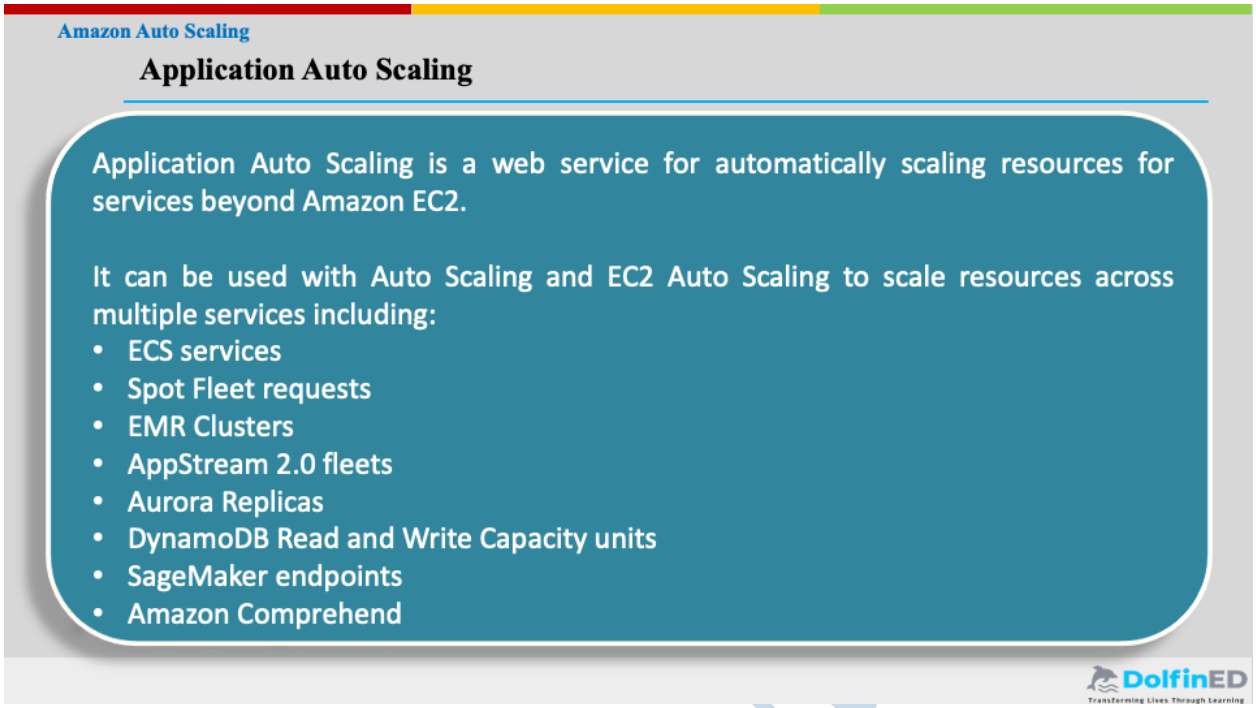


Amazon Auto Scaling is useful for applications that experience daily or weekly variations in traffic flow, including:

- Cyclical traffic such as high use of resources during regular business hours and low use of resources overnight.
- On and off workload patterns, such as batch processing, testing, or periodic analysis.
- Variable traffic patterns, such as marketing campaigns with periods of spiky growth.

www.dolfined.com

© DolfinED All rights reserved



Application Auto Scaling is a web service for developers and system administrators who need a solution for automatically scaling their scalable resources for individual AWS services beyond Amazon EC2.

Application autoscaling allows you to configure automatic scaling for the following resources:

- Amazon ECS services
- Spot Fleet requests
- Amazon EMR clusters
- AppStream 2.0 fleets
- DynamoDB tables and global secondary indexes
- Aurora replicas
- Amazon SageMaker endpoint variants
- Custom resources provided by your applications or services.
- Amazon Comprehend document classification endpoints
- Lambda function provisioned concurrency

We will go over Auto Scaling for some of these services as we move along the course.

Application Auto Scaling allows us to automatically scale our scalable resources according to conditions that we define.

- Target tracking scaling—Scale a resource based on a target value for a specific CloudWatch metric.

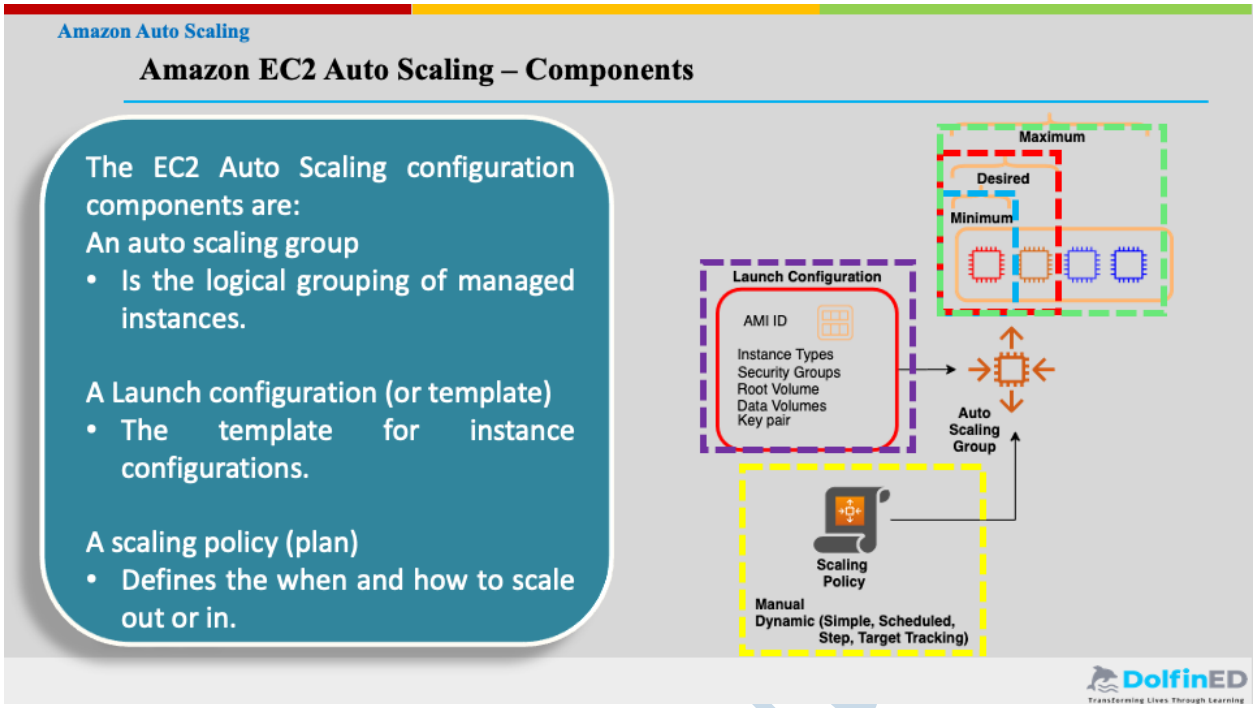
© DolfinED All rights reserved

- Step scaling— Scale a resource based on a set of scaling adjustments that vary based on the size of the alarm breach.
- Scheduled scaling—Scale a resource based on the date and time.

Customers can suspend and resume any of their AWS Application Auto Scaling actions on any of these resources without having to delete the scaling policies or scheduled activities. Customers can suspend and later resume any scheduled scaling actions, dynamic scaling-in actions, or dynamic scaling-out actions.

www.dolfined.com

© DolfinED All rights reserved



An **auto-scaling group** is a virtual grouping of EC2 instances managed under the AS Group.

An auto-scaling group defines the minimum, desired, and maximum EC2 instances.

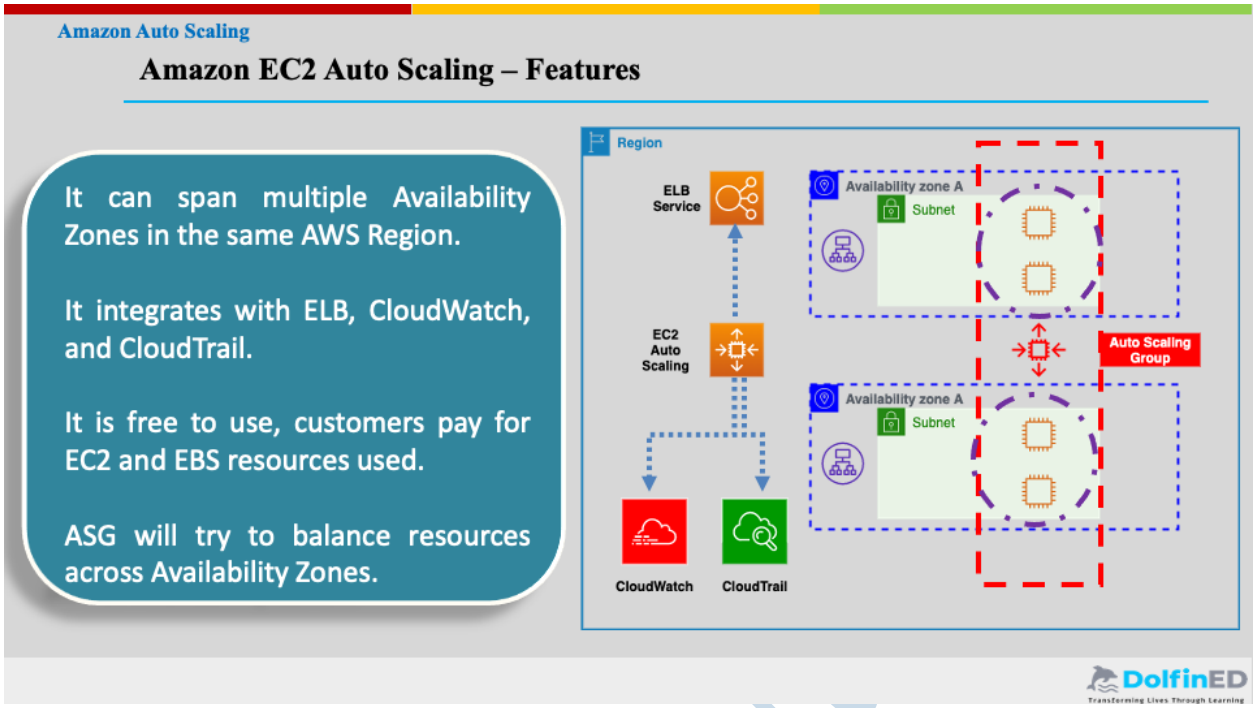
Minimum Capacity:

Specifies the minimum EC2 capacity of the AS Group. AWS EC2 Auto Scaling ensures that the number of instances available never goes below this size.

Desired Capacity:

Is the actual number of instances available at any given time based on requirements. It is also the starting number of EC2 instances when the group is created.

www.dolfined.com



Auto Scaling can span Multi-AZs within the *same* AWS region, but *not* across regions.

It can be used to create fault-tolerant architectures within a region in AWS.

You can determine which subnets auto-scaling groups will use to launch new instances in each Availability Zone.

- It works well with AWS ELB, Cloud Watch, and Cloud Trail.
- Auto Scaling is PCI DSS compliant.
- There is no additional cost for launching AS Groups.

AZ Rebalancing

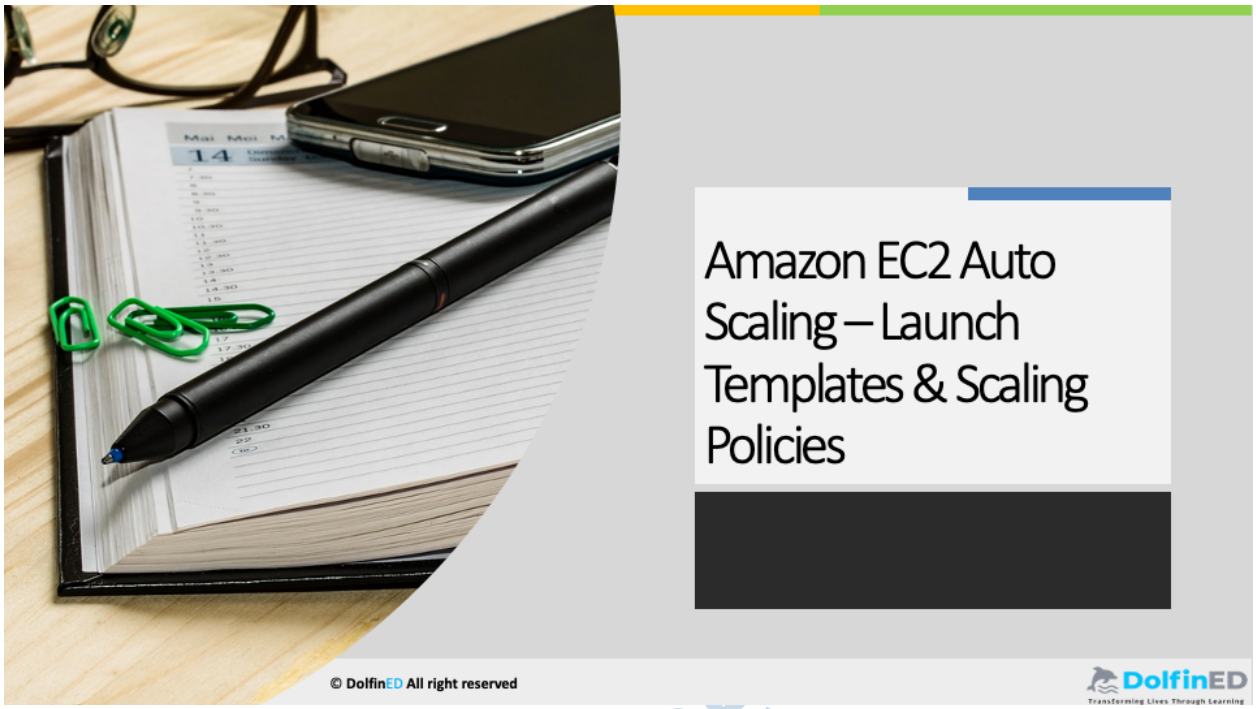
The Auto Scaling service always tries to distribute EC2 instances evenly across AZs where it is enabled.

Suppose Auto Scaling finds that the number of EC2 instances launched by an Auto Scaling group into subject Availability Zones is not balanced. In that case, Auto Scaling will initiate a rebalancing activity to fix that.

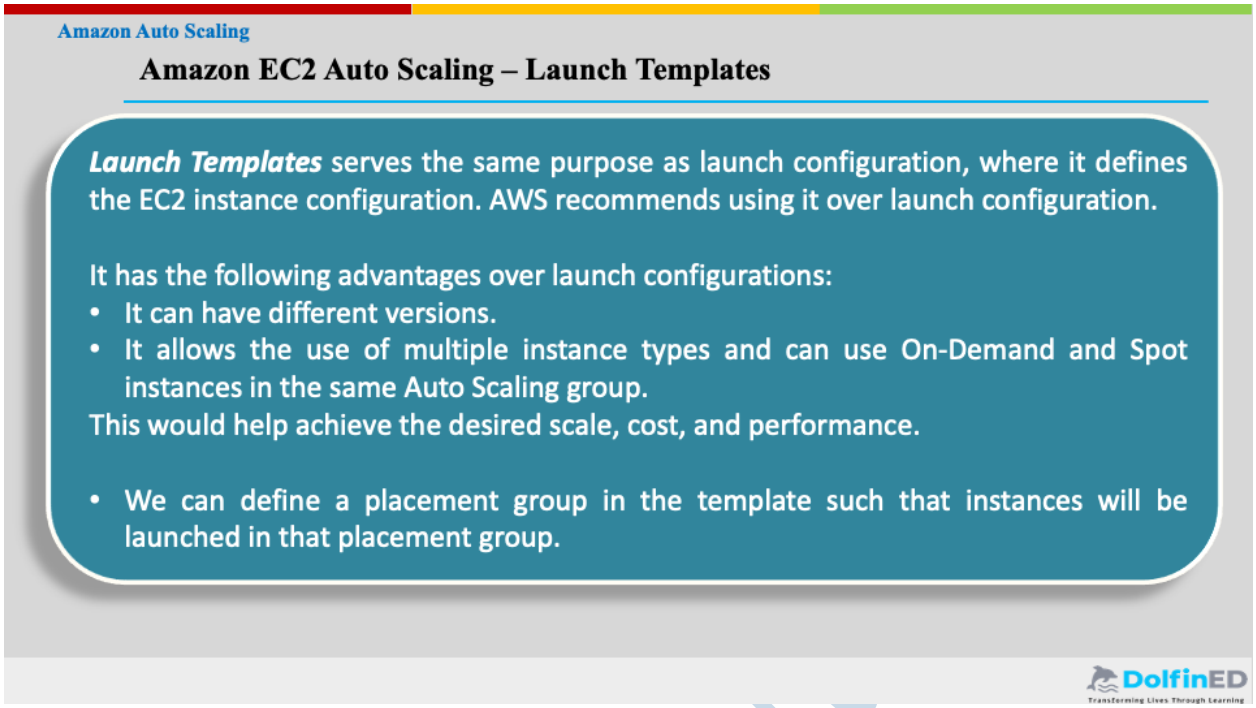
If Auto Scaling fails to launch instances in an Availability Zone (in the event of an Availability Zone failure or capacity unavailability, etc.), it will try in the other Availability Zones defined for this AS Group until it succeeds.

© DolfinED All rights reserved

Amazon Auto Scaling – Launch Templates and Scaling Policies



www.dolfined.com



Launch Templates serve the same purpose as a launch configuration; however, it has the following differences:

- It can have multiple versions of a template.
- With versioning, we can create a subset of the full set of parameters and then reuse it to create other templates or template versions.
- With launch templates, we can launch multiple instance types using both on-demand and spot instances to achieve the desired cost, performance, and scale.

AWS recommends Launch Templates over launch configurations to ensure that we can use the latest features of Amazon EC2.

What if a customer is using launch configurations currently? How can they move to Launch Templates?

They can simply specify a Launch Template when they update an auto-scaling group created using a launch configuration.

We can create a template from scratch, create a new version of an existing one, or copy the parameters from a launch configuration, another template, or an EC2 instance.

Launch Templates cannot be changed/edited after being created, but additional versions can be created.

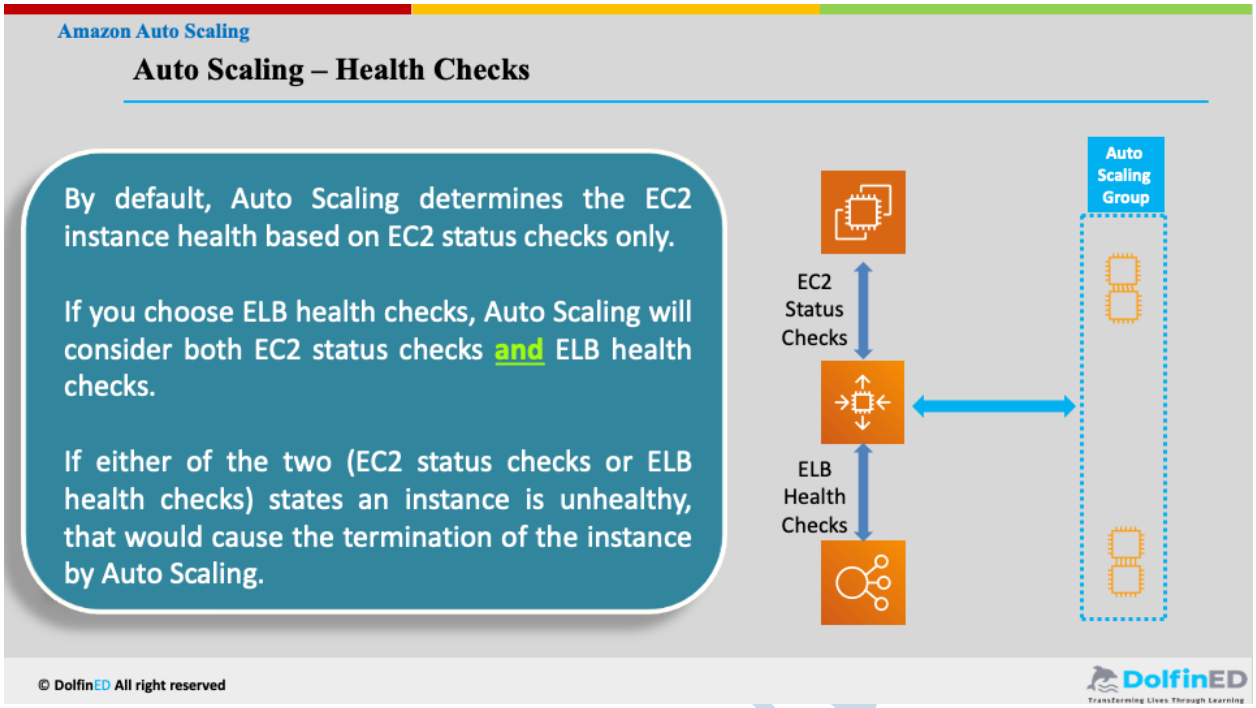
© DolfinED All rights reserved

Whenever we create an auto-scaling group, we must specify a launch configuration, a Launch Template, or an EC2 instance.

When we create an auto-scaling group using an EC2 instance, Amazon EC2 Auto Scaling automatically creates a launch configuration and associates it with the auto-scaling group.
We can define a placement group in the template such that instances will be launched in that placement group.

www.dolfined.com

© DolfinED All rights reserved



Now let's elaborate a little bit on Auto Scaling health checks.

Amazon EC2 Auto Scaling can determine the health status of an instance using one or more of the following sources:

1. EC2 Status checks. This includes both instance status checks and system status checks to identify hardware and software issues that may impair an instance. The default health checks for an Auto Scaling group are EC2 status checks only.

2. Elastic Load Balancing (ELB). If enabled, Auto Scaling can use the health checks provided by ELB. This can be configured by choosing the ELB option under Auto Scaling health checks. If we choose it, then ELB and EC2 status checks feed or notify Auto Scaling of an unhealthy instance.

3. Custom health checks.

One important note here is, if the ELB is not enabled as a health check source for Auto Scaling, Auto Scaling will NOT consider ELB health checks in its decision as to whether an instance is healthy.

In this case, the ELB may believe an instance is unhealthy and will NOT send traffic its way, while Auto Scaling does *not* believe the instance is unhealthy because EC2 status checks did not report the instance as unhealthy. This would result in effectively having EC2 instances running but not utilized for load distribution, which means paying for something not used.

www.dolfined.com

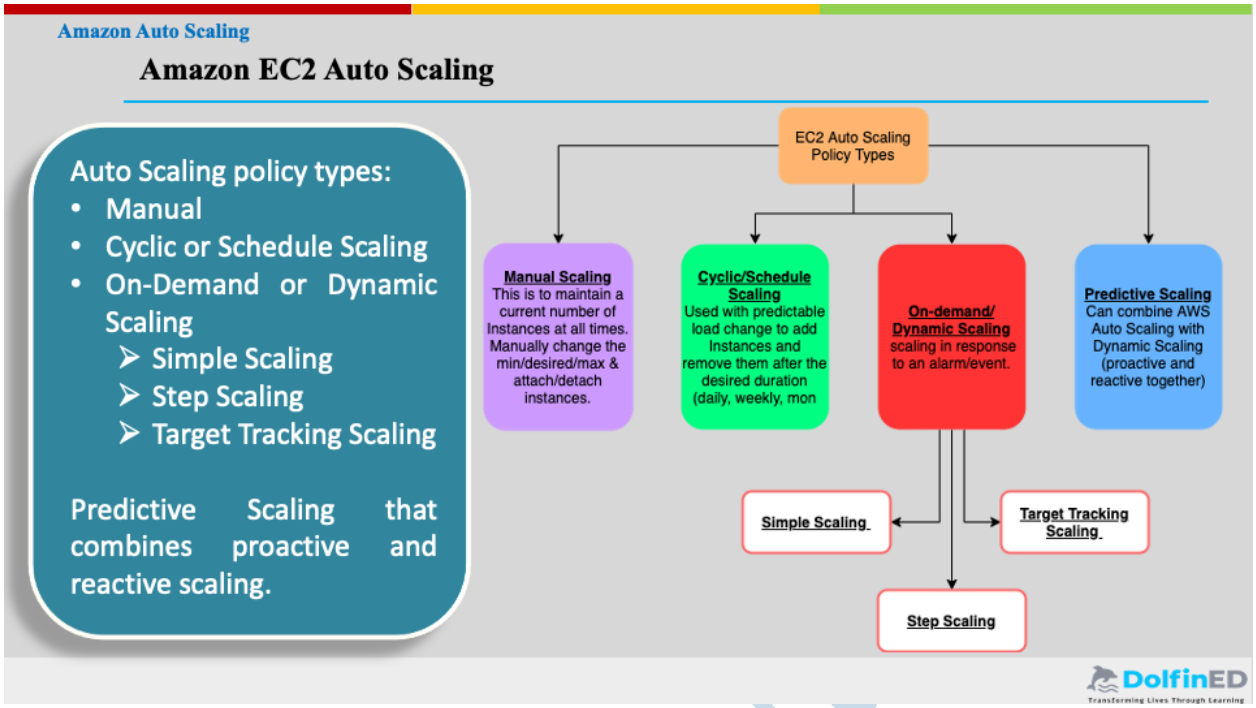
Both should feed into Auto Scaling, which means we have to enable ELB health checks in Auto Scaling

By integrating the ELB health checks into Auto Scaling, we would gain detailed information with Auto Scaling regarding when to terminate and replace an instance.

If an instance is determined to be unhealthy, Auto Scaling will terminate the unhealthy one first and then create a new one to replace it.

www.dolfined.com

© DolfinED All rights reserved



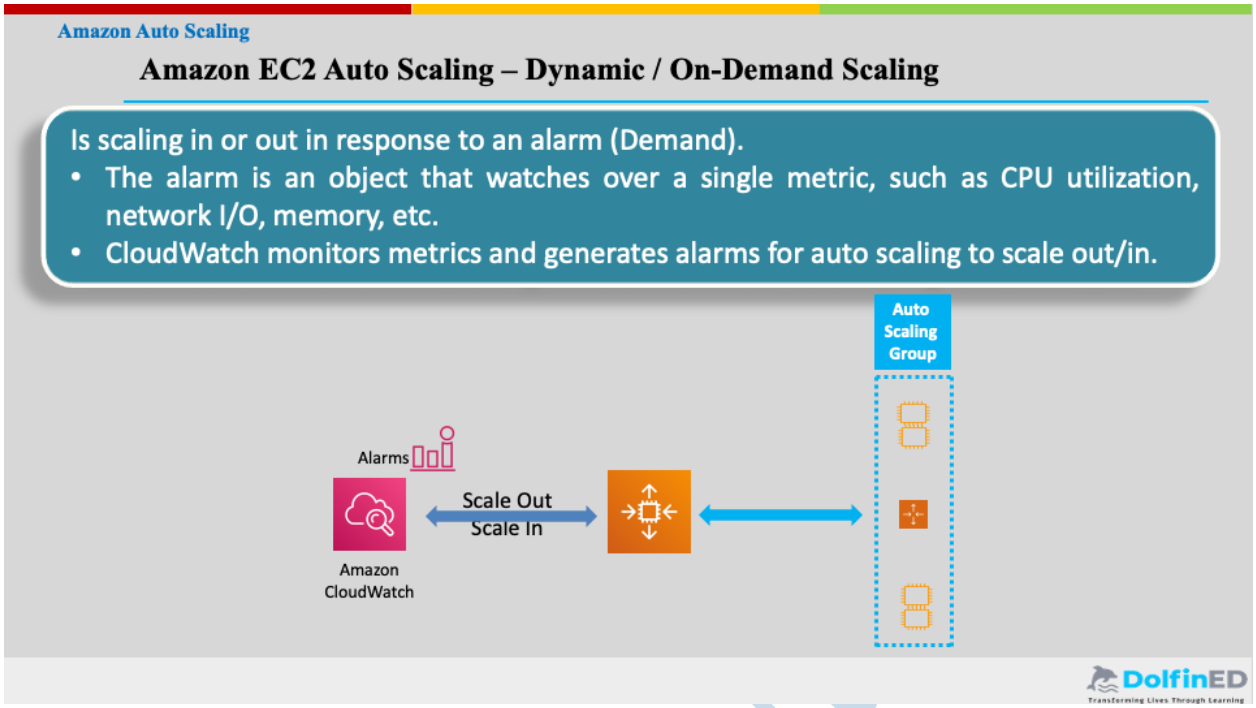
EC2 Auto Scaling policies:

Maintain a current number of instances at all times.

- **Manual Scaling** - Manually change ASG's min/desired/max and attach/detach instances.
- **Cyclic (schedule-based) scaling** - Used for predictable load change.
- **On-Demand/Dynamic -** (event-based) scaling - Scaling in response to an event/alarm.
- **Predictive scaling** - Combines AWS auto-scaling with On-Demand scaling (Proactive and reactive together).

An ASG can have multiple policies attached to it at any time.

© DolfinED All rights reserved



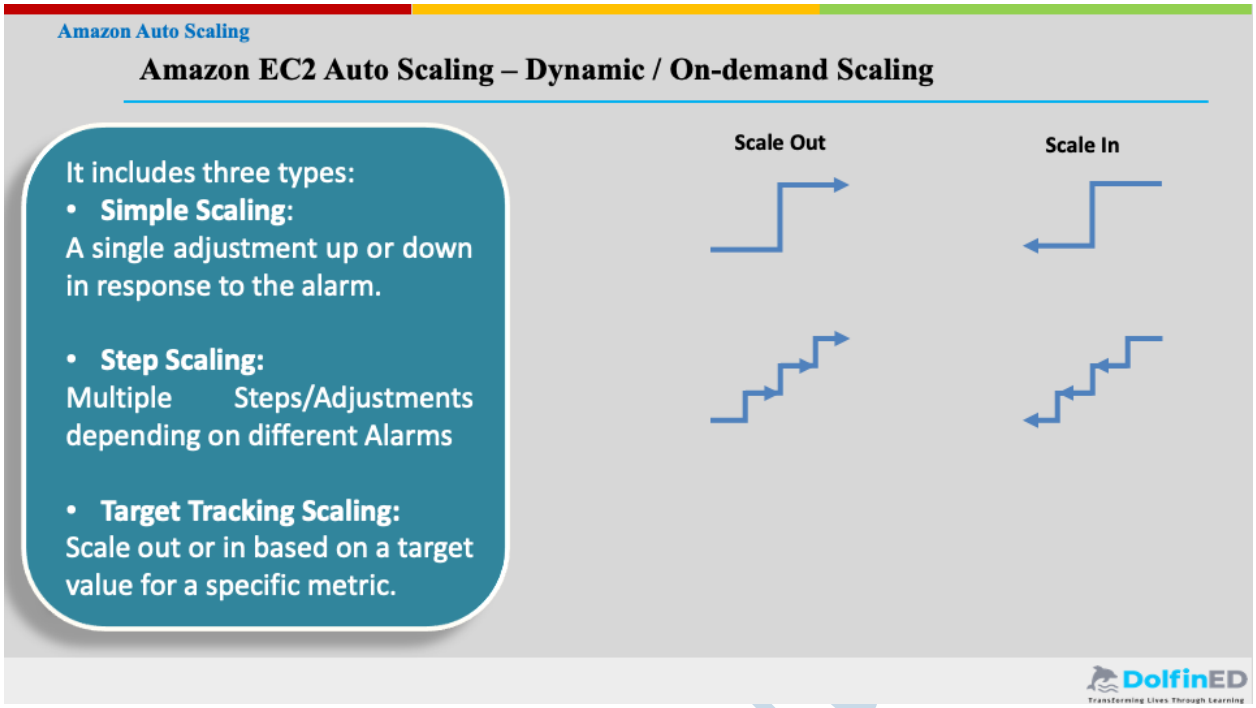
Dynamic or On-Demand scaling is scaling out (or in) in response to an alarm (demand).

An *alarm* is an object that watches over a single metric, such as CPU utilization, memory, or network in/out.

We need to have scale-out and scale-in policies configured. These will instruct Auto Scaling in what to do (scale-out or in) in response to alarms

CloudWatch alarms are used to monitor a metric and to alert Auto Scaling if a threshold is crossed. Auto Scaling can then determine, based on the configured policy, to scale out or scale in as a reaction to that alarm.

www.dolfined.com



There are three On-demand scaling policy types:

Simple Scaling:
Single adjustment (up or down) in response to an alarm

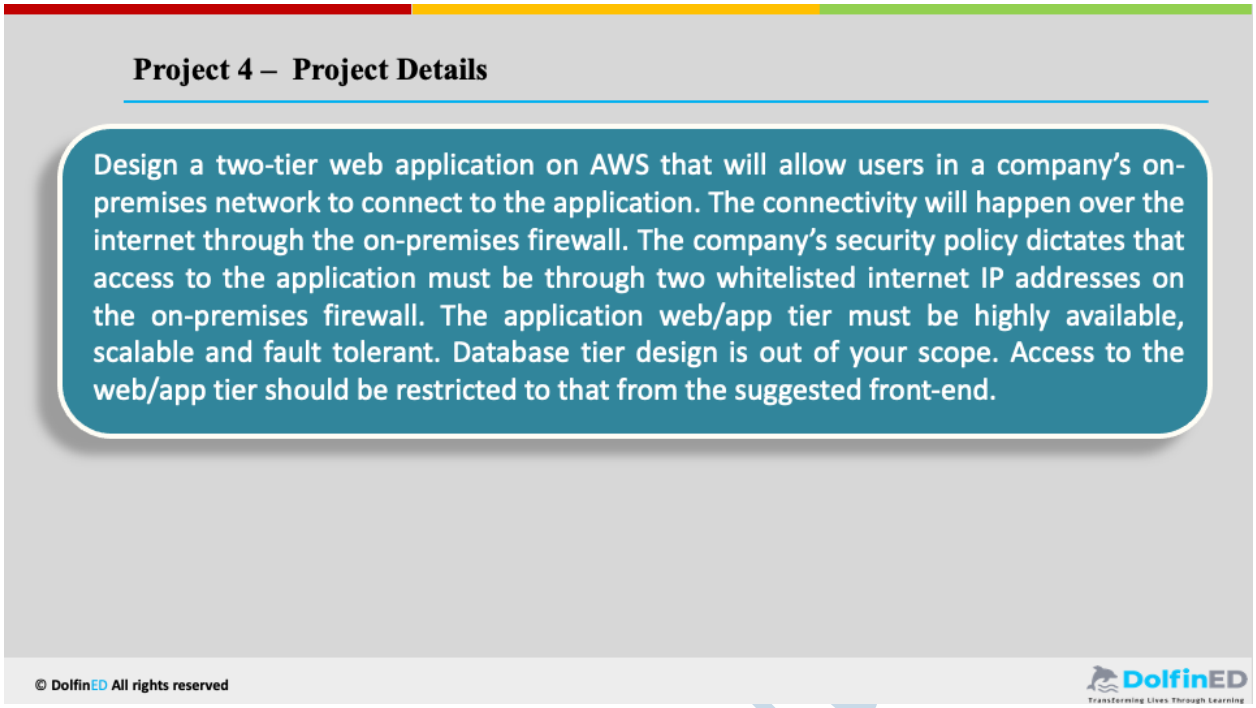
Step Scaling:
Multiple steps/adjustments based on multiple thresholds and different alarms

Target Tracking Scaling:
Increase or decrease the current capacity of the group based on a target value for a specific metric.
For example, to maintain the average group CPU utilization at 50%, in this case, we are not sure what is the number of EC2 instances required; however, we know that we need to keep the average CPU utilization across all instances at 50%.
AWS strongly recommends using a target tracking scaling policy to scale on a metric like average CPU utilization. **This helps ensure that Amazon EC2 Auto Scaling follows the demand curve for your applications closely. ….**
Moreover, to save costs, AWS recommends using Target Tracking Auto Scaling policies.

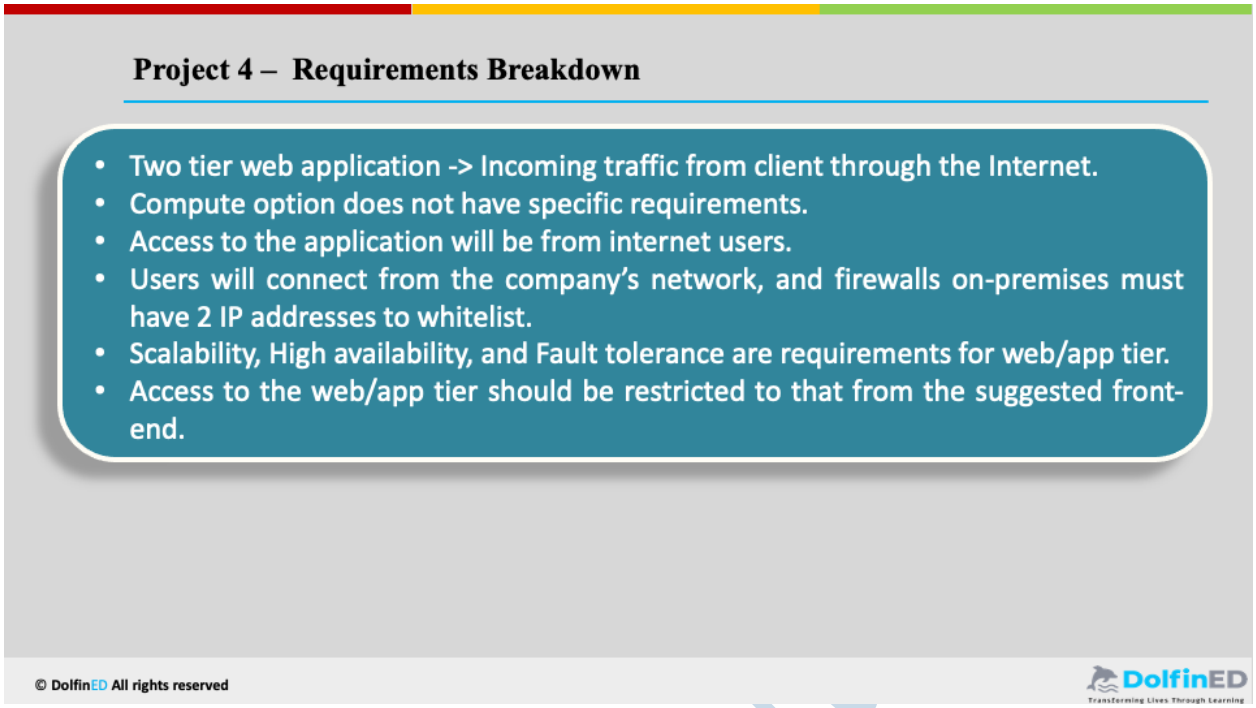
© DolfinED All rights reserved

Knowledge In Action – Project 4

www.dolfine



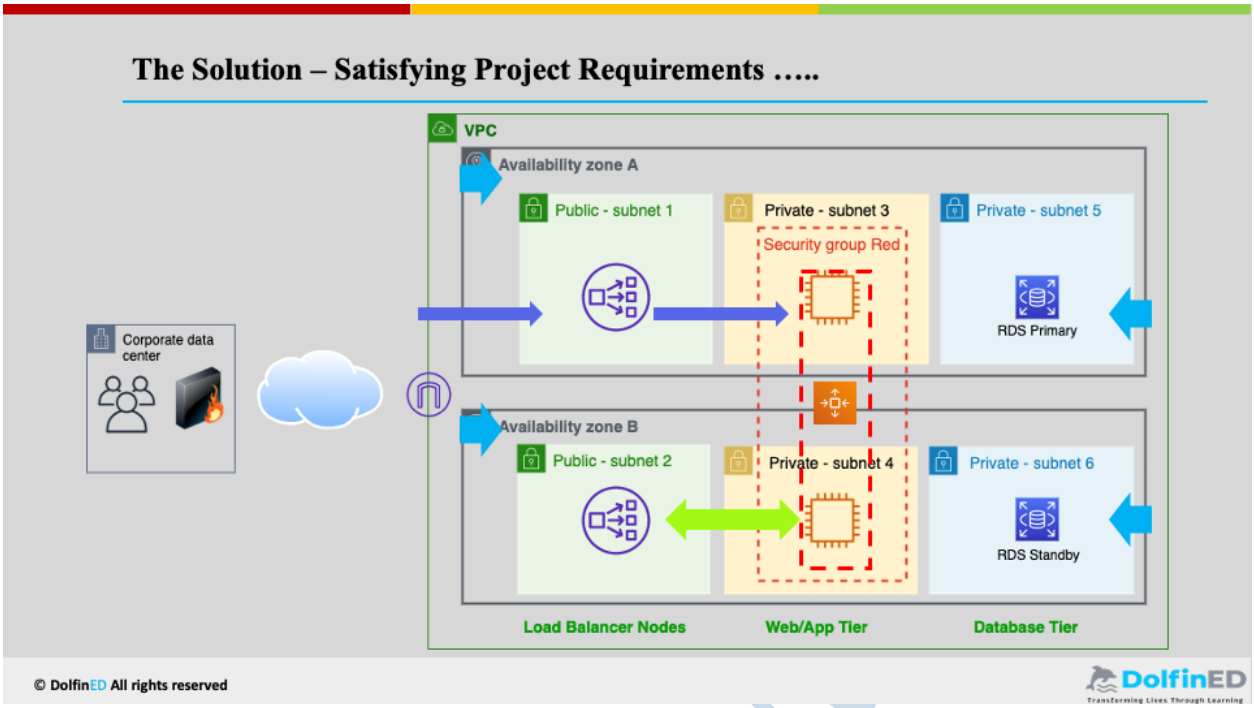
www.dolfined.com



A critical skill to take the exam with AWS is to start building and breakdown details as you read.

Going through the question carefully and analytically, you should have come out with this understanding and having those logical bullets in your mind. If you did not get all of them, that's fine; it's a skill that we will establish as we practice more.

www.dolfined.com



How many subnets do we need? Ideally, place the Web/App tier in a private subnet. This was not a requirement here, but you can do it; if you did not and included them in a public subnet, that's fine too, but understand that this will be less secure.

Since we need Highly available and fault-tolerant -> The solution must span at least two Availability zones.

Since the solution needs to be scalable, we need auto-scaling for the web app tier.
Since the solution needs to be highly available and fault-tolerant, we need to ensure we have an Internet-facing ELB enabled in the two-availability zones.
The same for the Auto Scaling group.

The minimum we need to achieve this is 2 public subnets for the ELB and two private subnets for the Web/app tier. The database is out of scope; however, we could add two more private subnets if we needed to.

Since we need to whitelist internet IP addresses for the application's web/app tier, then we need a Network Load balancer and not an ALB. Because we can have one Elastic IP address per AZ for the NLB, then the on-premises firewall can whitelist those IP addresses.

To restrict traffic to the web/app tier from the NLB only, we need to ensure that the security group of the web/app instances will only allow traffic from the subnets of the NLB, the private subnets.

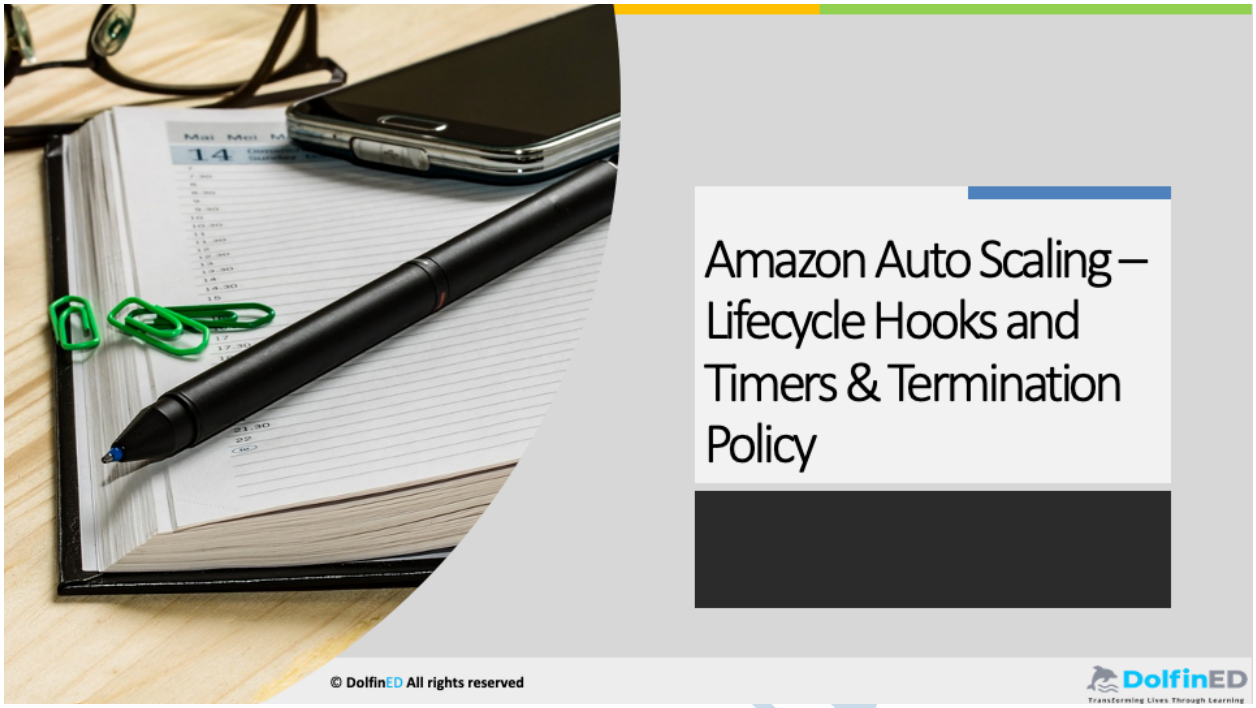
Although out of our scope, if it was in scope, we could do RDS multi-AZ.

www.dolfined.com

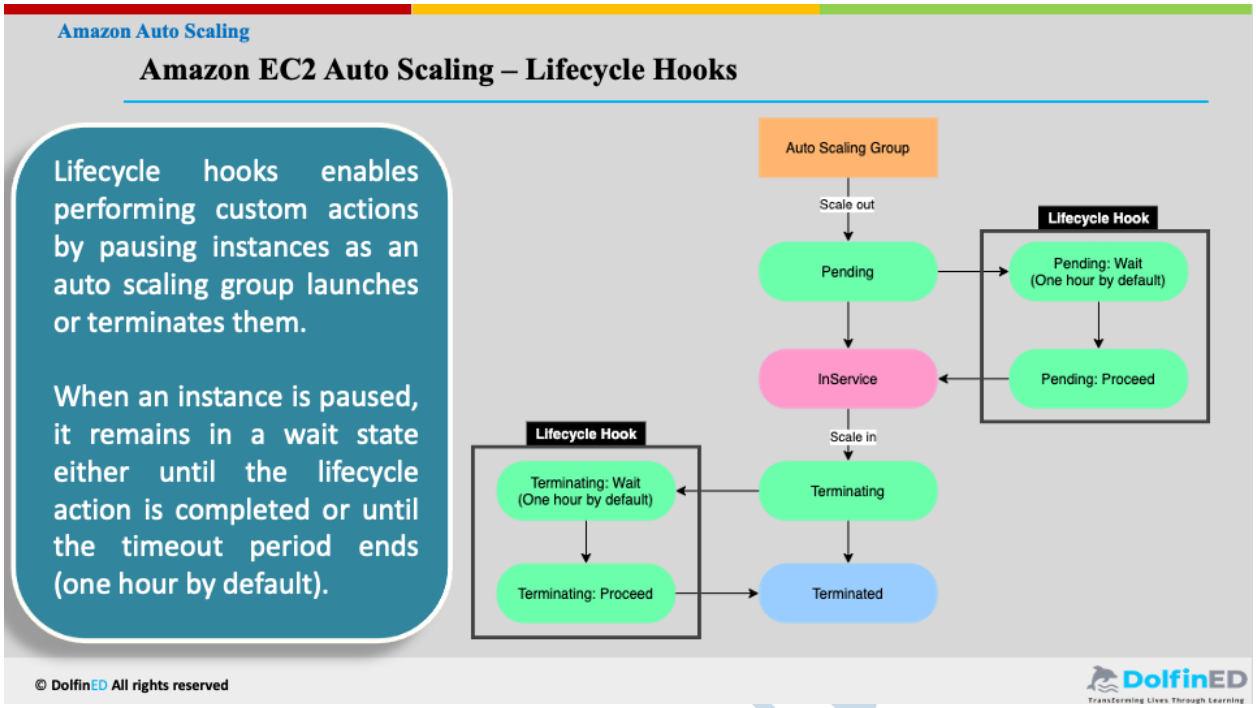
I hope this was beneficial.
If you figured out the solution correctly, this is awesome; well done. If you did not or did not capture most of the points, that is perfectly fine as well. We are learning as we go, and if you missed anything, please review the relevant lectures, then re-do the project; re-read it analytically and slowly. Practice extracting the important information and make sure you 100% get it before you proceed. We need to minimize these gaps as we go, so we save time in the end and lessen the required review before the exam.

www.dolfined.com

© DolfinED All rights reserved



www.dolfined.com



The typical lifecycle flow of an instance launched by Auto Scaling is to go through *pending* to *in-service* during a scale-out action, then *terminating* to *terminated* during a scale-in action.

What if we need to check that the instance is fully ready? Or update/validate some of its applications, software, or OS before it goes into in-service and starts receiving traffic?

How can we do this?

What if the instance is being terminated, but we need to inspect it and download logs before it is terminated?

This is where **lifecycle hooks** come in handy.

Lifecycle hooks enable you to perform custom actions by *pausing* instances as an Auto Scaling group launches or terminates them.

When the instance is paused, it remains in a *wait* state either until you complete the lifecycle action or until the timeout period ends (which is one hour by default).

You can add a lifecycle hook such that the instance moves from *pending* to *pending:wait*. In this case, it will remain in this state by default until you manually continue through *pending:proceed*, or until the one hour expires, when it will move to *in-service* as shown.

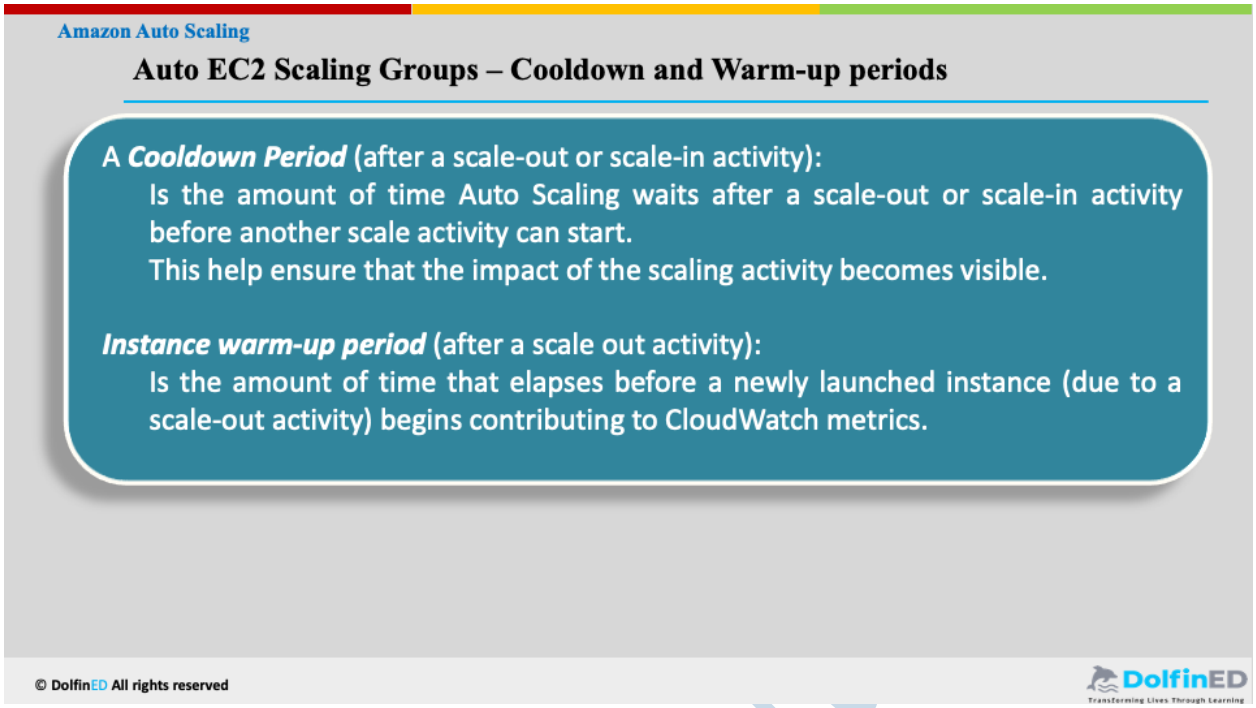
Also,

www.dolfined.com

You can add a lifecycle hook where the instance moves from *terminating* into *terminating:wait*. It will then stay there for one hour by default or until you allow it to proceed as shown.

www.dolfined.com

© DolfinED All rights reserved



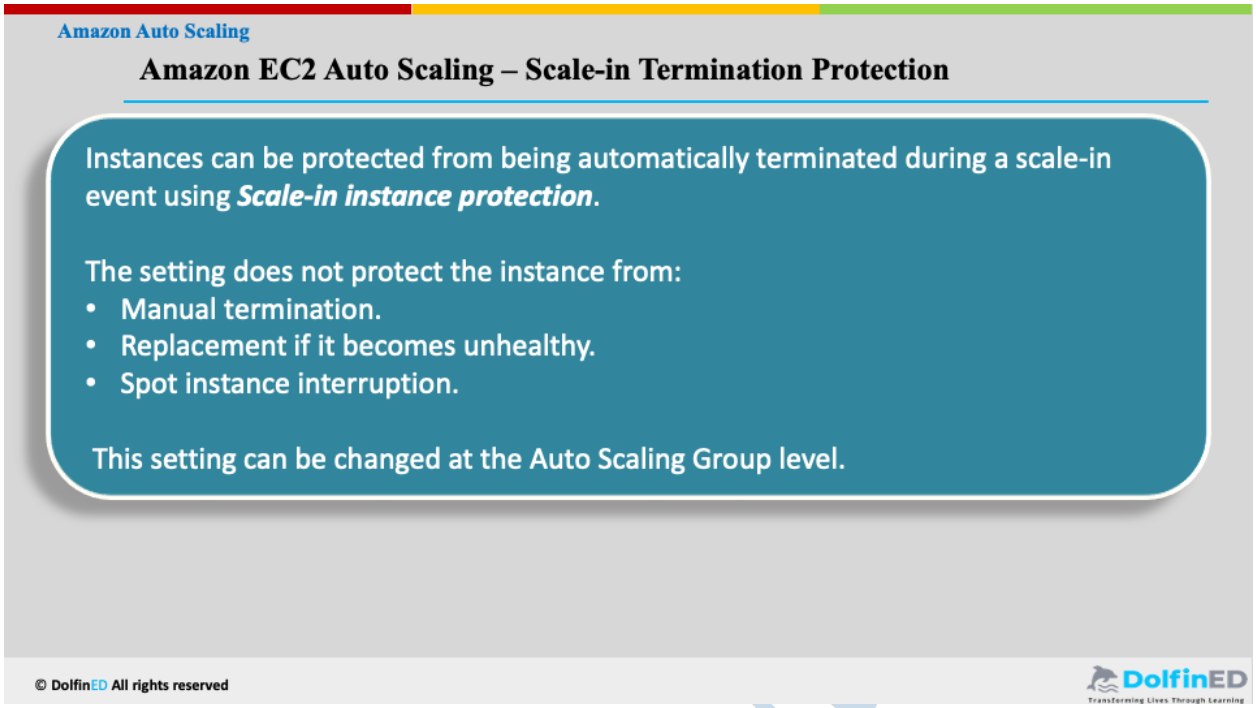
Cooldown (with step scaling):

- Creates scale-out and scale-in cooldown periods.
- A cooldown period is the amount of time after a scale-out, or scale-in activity completes before another activity can start.
- The intention is to give newly provisioned resources time to start handling demand before triggering a new scaling action.
- It is used with step scaling.

Instance warmup:

Controls the amount of time that elapses before a newly launched instance begins contributing to the CloudWatch metrics.

www.dolfined.com



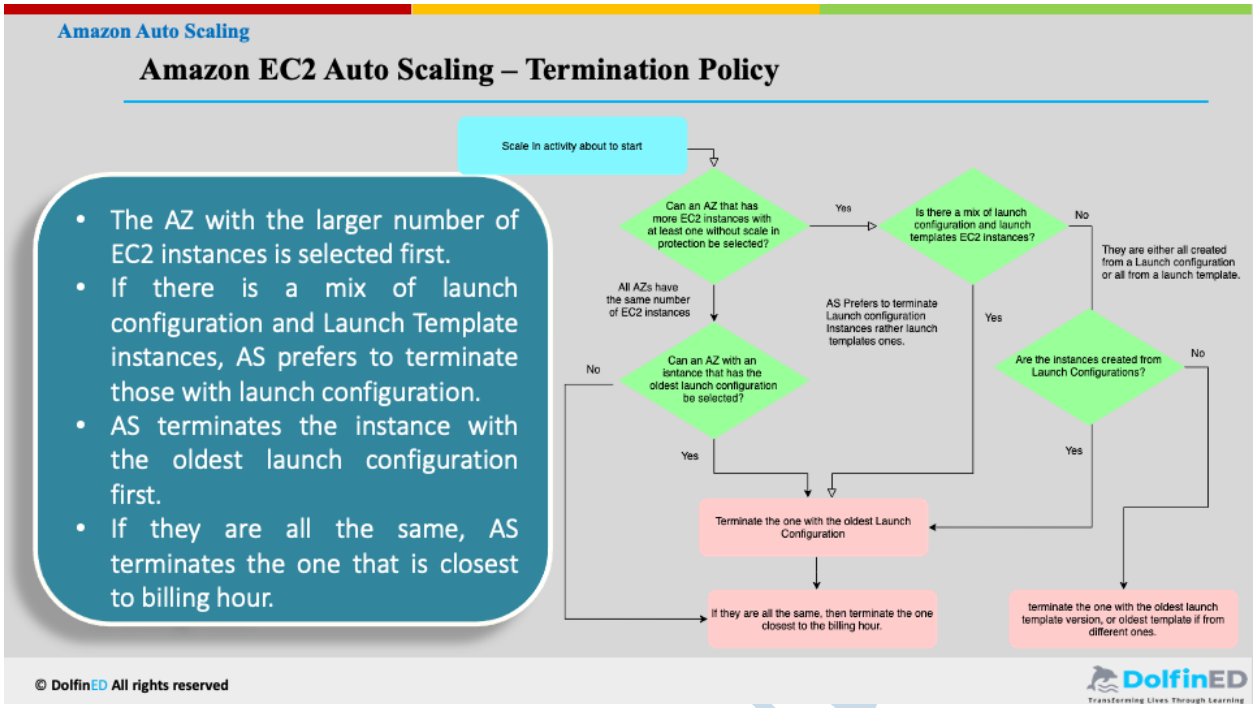
Scale-in Instance protection allows us to control whether an Auto Scaling group can terminate a particular instance when scaling-in action happens.

This is an Auto Scaling group setting, and any instance launched in a scale-out inherits the setting.

However, this does not protect the instance from manual termination, spot instance interruptions, or replacement if the instance becomes unhealthy.

We can change the instance scale-in protection setting for an Auto Scaling group or an Auto Scaling instance at any time.

www.dolfined.com



Auto Scaling has a default **termination policy** it follows in deciding which instance to terminate in a scale-in action. This is strictly for scale in-actions and has nothing to do with terminating an instance that is found to be unhealthy.

First, it starts looking for an AZ with more instances than other AZs, and with at least one instance that does not have scale-in termination protection.

Once an AZ is selected, it looks at the available instances. It will choose to terminate launch configuration-based instances if there is a mix of launch configuration instances and Launch Template instances.

If there is no mix, it will terminate the instance that has been created by the oldest launch configuration (or Launch Template if they all were created from a template).

If they are all created of the same launch configuration (or template), it will terminate the one closest to the billing hour.

If they are all the same, then it will pick one at random.

You can customize a termination policy if default the one is not meeting your requirements.

www.dolfined.com