

AWS CloudTrail

One CloudTrail trail is created in each account and configured to send logs to a centrally managed Amazon Simple Storage Service (Amazon S3) bucket in the log archive account, and to AWS CloudWatch Logs in the local account for local operations (with a 14-day log group retention policy).

AWS Config

AWS Config is enabled and account configuration log files are stored in a centrally managed Amazon S3 bucket in the log archive account.

AWS Config Rules

AWS Config rules are enabled for monitoring storage encryption (Amazon Elastic Block Store, Amazon S3, and Amazon Relational Database Service), AWS Identity and Access Management (IAM) password policy, root account multi-factor authentication (MFA), Amazon S3 public read and write, and insecure security group rules.

AWS Identity and Access Management

AWS Identity and Access Management is used to configure an IAM password policy.

Cross-Account Access

Cross-account access is used to configure audit and emergency security administrative access to AWS Landing Zone accounts from the security account.

Amazon Virtual Private Cloud (VPC)

An Amazon VPC configures the initial network for an account. This includes deleting the default VPC in all regions, deploying the AVM requested network type, and network peering with the Shared Services VPC when applicable.

AWS Landing Zone Notifications

Amazon CloudWatch alarms and events are configured to send a notification on root account login, console sign-in failures, and API authentication failures within an account.

Amazon GuardDuty

Amazon GuardDuty is configured to view and manage GuardDuty findings in the member account.

Routing

Hello, and welcome to this final lecture. By now, you should have a solid understanding of how to architect your VPC subnets and configure routine appropriately for various different configurations and scenarios. However, in this last lecture, I just want to reiterate some of the key points that we made throughout the previous lectures.

We started off by looking at VPC CIDR blocks. Where this lecture focused on the effects of subnetting your VPC CIDR block and here were some of the key points.

- Subnetting is the process of splitting a CIDR block into smaller CIDR blocks within the same range of the larger block.  
- Subnetting enables you to divide your VPC CIDR block into small networks.  
- The VPC CIDR block range encompasses the entire IP address space that you can use within that VPC.  
- The maximum and minimum masks for your VPC CIDR block are /16 to /28.  
- AWS reserves the first three host IP addresses of each subnet for internal AWS usage. The first host is used for the VPC router. The second is for DNS and the third is reserved for future use.  
- An IPv4 CIDR block range for your VPC is mandatory but you can also associate an IPv6 block as well.

Next we focused on the topic of why subnet your VPC? And here we looked at some of the reasons why you may want to subnet your VPC by looking at some of the advantages and benefits.

- Logical network division. When you create multiple networks it allows you to create logical network divisions between your resources.  
- Security. By having multiple subnets with similar resources grouped together, as per the previous point it allows for greater security management.  
- Accessibility. Having multiple subnets allows you to create both Private and Public subnets.  
- Communication. You may want some of your subnets to route out the internet, some to remain private and some to communicate back to your corporate-on-premise network via VPN. Subnetting allows you to do this with the help of IWS routing.  
- High availability. If your solution requires a level of high availability, it's best practice to deploy services across multiple availability zones within a region. Remember, a single subnet cannot span across two availability zones.

Once we understood the reasoning of benefits behind VPC subnets I delved into what a VPC subnet looks like within the management console and its associated components such as the network access control list. Here we learned that the subnet itself has five components.

- The summary, and this summarizes all the elements and metadata associated with the subnet. Such as the IPv4 CIDR block, the VPC the Asian and route table associated exception.  
- The Route Table. This defines which route table that the subnet is using along with its routes.  
- The NACL. The network ACSL shows the current NACL associated to the subnet displaying both Inbound and Outbound rule sets.  
- Flow Logs. Shows you any Flow Logs that you have set up and created for the subnet to capture IP traffic flow information.  
- Tags. And this allows you to add any custom key value pairs to help with the tagging of your subnet.

Following the definitions made in this lecture I then discussed the different kinds of subnets, Private and Public. Key points of this lecture.  
- Public subnets have direct access to the internet. Whereas your private subnets do not.  
- There are 2 components required to make a public subnet. An attached Internet Gateway to your VPC and a route pointing to the target Internet Gateway.  
- By default, all subnets have the automatic assigned of public IP addressing turned off  
- You can modify IP addressing behavior of a subnet using the 'modify auto-assign IP settings' option within the Management Console for your subnet.

Next, I focused on what consideration should be made when peering VPCs together from a subnet point-of-view. The key points here were that:  
- VPC peering allows you to connect two or more VPCs together using IPv4 or IPv6 as if they were a part of the same network.  
- The connectivity between the VPCs is implemented through the AWS network, and so it is highly valuable with no bandwidth bottleneck.  
- Overlapping or duplicate CIDR ranges between VPCs cannot be peered together. Each AWS VPC will only communicate with its peer.

Following this, I then covered how to monitor your subnets using Flow Logs.

-Subnet Flow Logs can help you troubleshoot networking problems.  
- They can capture IP traffic going into and out of your network interfaces.  
- The log data can be sent to a CloudWatch Logs log group where you are able to filter information and monitor specific metrics.

To end this section on subnets I then demonstrated how to set-up and configure a VPC with both public and private subnets.

Following this, I then spoke about AWS routing. Starting by looking at AWS routing fundamentals and route tables. Within this lecture, we learned that:

- Routing provides a mechanism to allow a network packets to be forwarded to the correct destination.  
- Within the VPC, all routing is configured via routing tables associated to your subnets which use virtual routers that are fully managed by AWS.  
- An implicit router will be linked to your VPC as a part of your VPC creation process.  
- A default route table, known as the Main Route Table is also created for your VPC.  
- The Main Route Table cannot be deleted.  
- When you create a new subnet, this Main Route Table will be implicitly associated to it unless you specify an alternate custom route table.  
- A subnet can only have one route table association at any one time. Although, multiple subnets can all use the same route table.  
- Subnets can either be explicitly or implicitly associated to a route table.  
- Any route table can take over as the Main Route Table.  
- A route in the route table has values for destination, target, status and propagated fields.  
- The destination field shows CIDR block ranges for any network that you need to route to outside of your current subnet.  
- The target field provides the gateway to allow you to get to that destination.  
- The status shows the states of your routes within the table.  
- The propagated field will determine which routes within the route table are propagated via a virtual private gateway.  
- The local route in every route table allows all subnets within your VPC to route to each other.  
- Subnets can be explicitly associated to a route table.  
- Route propagation via a private gateway will automatically add routes representing your VPN connection to the route table if propagation has been enabled for that virtual private gateway.

Next, routing priorities were discussed to understand how overlapping routes within the same route table were managed.

- The rule of thumb is that AWS will use the most precise route available within the table to route traffic to a specific target.  
- This is also known as the longest prefix match.  
- There are some circumstances where the longest prefix match route is not selected.  
- When propagated routes have overlapping destinations with your VPCs local route, then your VPC local route will have precedence even if your propagated routes have the longest prefix match.  
- If you have any propagated routes with the same destination as any existing static routes then the rule of longest prefix match will not be applied. Instead, the following priority is executed against any static routes that have the following gateways as the target first. Internet gateway, virtual private gateway network interface, an instance ID a VPC peering connection, a NAT gateway and, finally, a VPC endpoint.

Next I started to look at different routing configurations for different scenarios.

Starting off with VPC peering.

- A peered connection if pre-fixed with pcx in the target.  
- Routing must be set up on each side of the peer to allow each VPC to know where to route traffic to.  
- Peering can only exist if the IP CIDR blocks do not overlap in anyway.  
- Every subnet that requires connectivity to a peered VPC, a route must exist in the route table for that subnet pointing to the pcx.  
- VPC peering does not support unicast reverse path forwarding.  
- Use the longest prefix-match rule for any VPC that is peered to multiple VPCs that have the same CIDR block ranges.

Next was the routing configuration for VPN connectivity via a virtual private gateway.

-Virtual private gateways are used to help create a VPN connection between your VPC and your corporate network outside of AWS.  
- Before you can set up a route for a VPN over virtual private gateway you need to create and attach a virtual private gateway to your VPC.  
- To route across your VPN, you must update the route tables for any subnets that intend to do so using the target of your virtual private gateway.  
- By enabling propagation on this virtual private gateway your route table will automatically include the routes used by your VPN connection.  
- If you do not enable propagation, you will need to add static routes for all the networks used by the VPN connection that you want to route to.  
- AWS does not currently support IPv6 traffic across a VPN connection.

Following this, I then looked at routing for Internet gateways and NAT gateways.

-The Internet Gateway, or IGW, provides a means of communicating out to the Internet.  
- Once your internet getway is attached to your VPC, you have a gateway to the internet.  
- The destination value in a route table of 0.0.0.0/0 essentially implies that for any destinations that are not known by the route table, then use this route.  
- Any subnet which has a route table associated to that point is to an internet gateway is considered a public subnet.  
- A NAT gateway allows instances within a private subnet access to the internet. However, access to the instances within these private subnets cannot be initiated from the internet.  
- Instances in a private subnet must have a route to the NAT gateway which is located in the public subnet.  
- Routing for a NAT gateway requires the use of a public subnet to be configured.

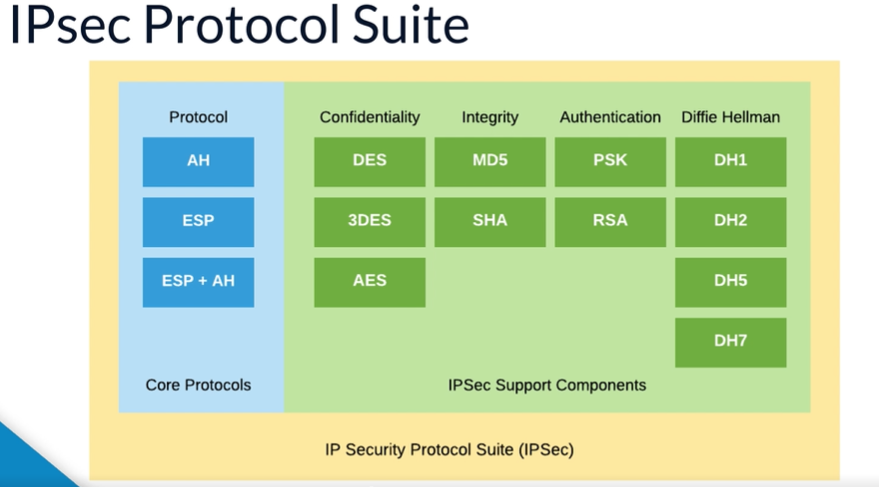
Then, finally, the last routing lecture I discussed was on VPC endpoints.

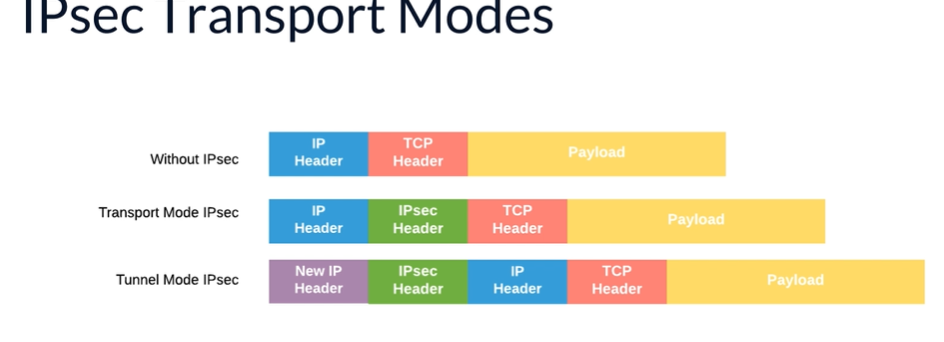
- A VPC endpoint is a virtual device which allows you to connect your VPC to another AWS service without traversing any gateway.  
- Currently, the only supported endpoint is for S3.  
- When communicating between the VPC endpoint and your VPC, the traffic remains inside the global AWS network.  
- VPC endpoint routing is implemented automatically during the creation of the endpoint.  
- The VPC endpoint creation will add a new rule with a new destination and a target value of the 'vpce ID' for the selected route table.

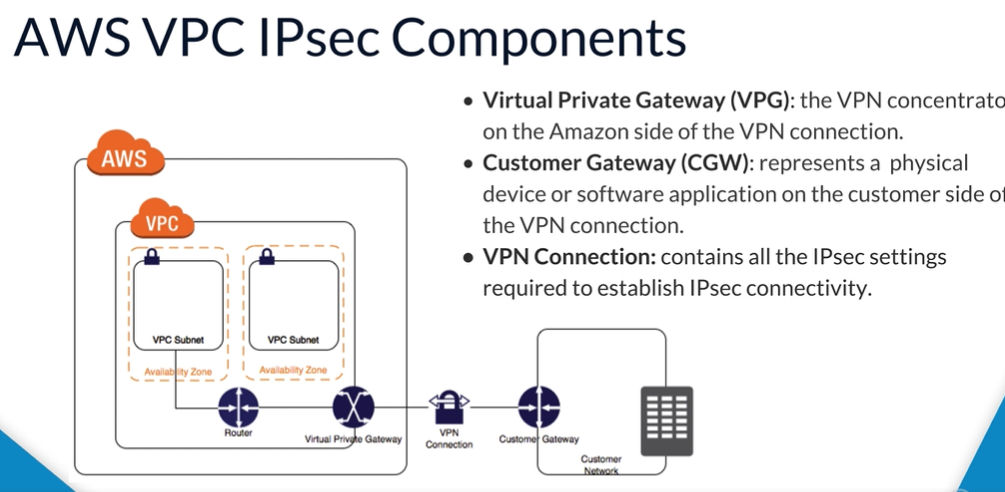
That has now brought me to the end of this lecture, and to the end of this course.

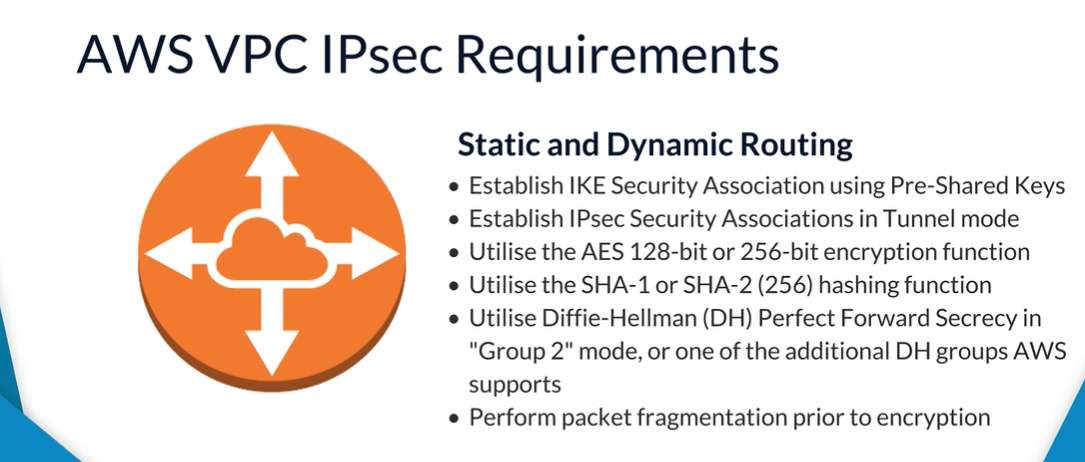
I hope it has given you a good understanding of some of the advanced AWS network concepts surrounding subnets and routing and has left you comfortable enough to start designing and architecting your own AWS network.

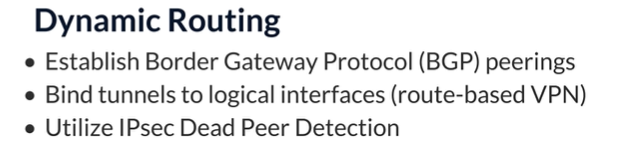
IPSec











When building on Amazon's Virtual Private Cloud, you have the capability to extend your local corporate on-prem network into the AWS Cloud securely. An AWS VPC can be set up with a VPN concentrator known as a virtual private gateway. The VPG will leverage IPsec to establish a pair of redundant IPsec tunnels. Let's now take some time here and detail some of the important features of IPsec. IPsec is a framework of protocols used to ensure data authentication, integrity, and confidentiality of data as it moves across IP networks. IPsec protects data against potential security exposures by protecting it while in transit through the use of various cryptographic services. IPsec is an open standard framework based on standards developed and administered by the IPsec working group within the Internet Engineering Task Force. IPsec is designed to provide security services that operate at the IP layer of the TCP/IP stack, or equivalently, the network layer in the OSI model. Since IPsec operates below the transport layer, it's effectively transparent to applications, meaning no changes are required in the application layers for them to be able to leverage the security functions of IPsec if enabled. From a data in transit perspective, IPsec provides three important and primary security services. Confidentiality: when data is in transit between two communication network endpoints, IPsec can be used to encrypt the data, ensuring that if anyone were to intercept the traffic in between, then they would not be able to view nor understand its contents. Integrity: when data is in transit between two communication network endpoints, IPsec can be used to ensure the integrity of the data, ensuring that if anyone were to attempt to alter the content of the payload, then this will be flagged at the receiving end. Authentication: when data is in transit between two communication network endpoints, IPsec can be used to mutually assure that both ends are indeed who they say they are before the data is sent, guaranteeing authenticity of both sender and receiver. Additionally, IPsec can be used to prevent replay attacks of data and non-repudiation of data. With non-repudiation of data, the sender of the data cannot, in any future point in time, refute sending that piece of data. Next, let's focus on the intention and use cases of IPsec, who benefits from using IPsec, and how they benefit from using IPsec. Use cases of IPsec technology are widely varied, from corporate online financial institutions to cloud platform providers such as AWS, Azure, and GCP. Basically, any data flow requiring security performed over an open network such as the internet is a good candidate for IPsec. IPsec can be used to provide site-to-site or host-to-host protected and secured network pathways over open networks such as the internet. As we'll see later in this course when we demonstrate setting up IPsec VPN tunnels, AWS VPCs can be implemented in a site-to-site configuration with on-prem corporate networks using IPsec. In doing so, this ensures any and all information in transit between a corporate network and an AWS VPC to be authenticated and protected from both data theft and data corruption. In the corporate world, IPsec is typically implemented through the deployment and installation of trusted firewall appliances supplied from specializing vendors such as Cisco, Juniper, Check Point, to name but a few. We'll now briefly touch on how AWS performs IPsec tunnels and integrates them with a VPC. Later on in this course, we'll dive deeper into the functions of each individual components, but for now, let's just list the required components to be deployed within a VPC. To integrate virtual private cloud with an on-prem corporate network in a private and secured manner, you must provision and deploy a virtual private gateway, customer gateway, and VPN connection. In this diagram, the AWS VPN connection is simplified as a single, logical connection between the gateways. In actuality, a pair of redundant IPsec VPN tunnels are created between the endpoints, as we'll be showing in the next slide. As previously mentioned, each AWS VPN connection created results in a pair of IPsec tunnels. This is done to provide redundancy. When one tunnel becomes unavailable; for example, down for maintenance; network traffic is automatically routed to the available tunnel for that specific VPN connection. AWS allows you to configure either a one-to-one arrangement with a single VPN connection between a virtual private gateway and customer gateway, or, as seen in this slide, a one-to-many arrangement with multiple VPN connections between a single virtual private gateway and many customer private gateways.

The one-to-many arrangement can be used to facilitate securely connection multiple independent department networks back to the same VPC.

Let's move our focus back to IPsec and review the individual components that make up the framework. IPsec uses two distinct protocols,

Authentication Header and Encapsulating Security Payload. Together, the pair of protocols are often referred to as the IPsec Core Protocols.

The Authentication Header, or AH in shortened form, provides a mechanism for authentication only. It allows the recipient of a data grab to verify that the supposed originator of a message is, in fact, the one that sent it. The recipient can also verify that the data within the datagram has not been changed by any adversary en route.

Additionally, it can provide protection against replay attacks, where a message is captured by an unauthorized user and resent. By itself, AH does not ensure privacy of data within the datagram while in transit. Encapsulating Security Payload, or ESP in shortened form, provides both data encryption and authentication. Encryption translates a readable message into an unreadable format to hide the message content, ensuring privacy and confidentiality.

Additional to the core protocols, IPsec provides a standard set of encryption and hashing algorithms, security policies, and security associations, and finally, key management functions. We'll dive a little deeper into the composition of the IP Protocol Suite in the next slide.

As you now see, the IPsec protocol suite is decomposed and classified into the following group security functions. Confidentiality, which gives us privacy, is performed through the use of encryption algorithms such as **DES, 3DES, and AES**. Integrity is performed using the **hashing algorithms, MD5 and SHA**. Authentication is performed by using **pre-shared keys and RSA public key cryptosystem. Key Exchange. It's performed by using Diffie-Hellman key exchange algorithms**.

Before establishing a secure IPsec VPN tunnel between two communication endpoints, formalities are required to ensure both endpoints are identifiable and trusted.

This process is performed and orchestrated by the Internet Key Exchange protocol, and it's done in two distinct phases. IKE phase one: the two endpoints authenticate one another and negotiate keying material

l. This results in an encrypted tunnel used by phase two for negotiating security associations. Phase one sets up mutual authentication with the peers, negotiates cryptographic parameters, and creates session keys. IKE phase two: the two endpoints use the secure tunnel created in phase one to negotiate SAs. The SAs are what are used to encrypt the actual user data that's passed between the two endpoints. Phase two negotiates an IPsec tunnel by creating keying material for the IPsec tunnel to use, either by using the IKE phase one keys as a base or by performing a new key exchange. A key consideration when deciding to work with IPsec is to determine, choose, and configure the correct IPsec transport mode. IPsec supports two modes, transport and tunnel. The choice of transport or tunnel mode depends on the network topology, and it's dependent on the logical connectivity to be established between communication endpoints. In transport mode, IPsec encrypts only the payload and ESP trailer, so the IP header of the original packet is not encrypted. Transport mode is often implemented for client-to-site VPN scenarios. Transport mode encapsulation retains the original IP header; therefore, when transport mode is used, the IP header reflects the original source and destination of the packet. Transport is most often used in a host-to-host scenario where the data endpoints and the security endpoints are the same. A transport mode encapsulated datagram is rooted or transported in the same manner as the original packet. In tunnel mode, IPsec protects the internal routing information by encrypting the IP header of the original packet. The original packet is encapsulated by another set of IP headers. Tunnel mode is implemented often in site-to-site VPN scenarios, as is the case with AWS VPCs. Tunnel mode encapsulation builds a new IP header containing the source and destination address of the security endpoints. When tunnel mode is used, the out IP header reflects the source and destination of the security endpoints, which might or might not be the same as the original source and destination IP address of the data connection. Note the AWS virtual private gateway VPN concentrator only supports tunnel mode. The manner in which the original in packet is modified depends on the encapsulation mode used. There are two encapsulation modes used by AH and ESP, transport and tunnel. Again, for clarity, in transport mode, encapsulation retains the original IP header; therefore, when transport mode is used, the IP header reflects the original source and destination of the packet. In tunnel mode, IPsec protects the internal routing information by encrypting the IP header of the original packet. The original packet is encapsulated by another set of IP headers. Let's now elaborate our understanding of how we would establish IPsec connectivity to our AWS VPC. As earlier described, to integrate a virtual private cloud with on-prem corporate network in a private and secured manner, we need to provision and deploy the following components. One, virtual private gateway, or VPG in shortened form, represents the VPN concentrator on the Amazon side of the VPN connection. Two, customer gateway, or CGW in shortened form, represents the physical device or software application on the customer's side of the VPN connection. The customer gateway is configured with externally-facing public IP address configured on the device. Three, VPN connection represents an actual connection between a customer gateway and a virtual private gateway. A VPN connection, when created, requires reference to a single customer gateway and a single virtual private gateway. In addition to deploying and configuring the previously-discussed AWS components, we need to ensure that our customer gateway device or appliance can negotiate the following IPsec protocol requirements for both statically and dynamically rooted VPNs: establish IKE security association using pre-shared keys; establish IPsec security associations in tunnel mode; utilize the AES 128-bit or 256-bit encryption function; utilize the SHA-1 or SHA-2 hashing function; utilize Diffie-Hellman Perfect Forward Secrecy in Group 2 mode, or one of the additional DH groups AWS supports; and perform packet fragmentation prior to encryption. Additionally, [dynamically-routed IPsec VPNs](https://cloudacademy.com/course/amazon-vpc-ipsec-vpns-understanding-building-and-configuring/amazon-vpc-ipsec-vpns-dynamic-bgp-routing-demo/) have the following extra requirements: must be able to establish BGP peerings; must be able to bind tunnels to logical interfaces and act as a route-based VPN; and must be able to perform IPsec Dead Peer Detection. Pricing for AWS IPsec VPNs is very simple. It costs five cents per VPN connection-hour. You're charged for each VPN connection-hour that your VPN is provisioned and available. Please note, a charge still applies even if your VPN and IPsec tunnels haven't yet completed the internet key exchange phases successfully and are, as such, still in a down status. Additionally, standard AWS data transfer charges for all data transferred by the VPN connection apply. It's worth noting a few default soft limits AWS imposes on us when provisioning and running an IPsec VPN: 50 customer gateways per region; five virtual private gateways per region; 50 VPN connections per region; 10 VPN connections per VPC. Each limit mentioned here is a soft limit and can be increased upon request. One other notable limit to mention is that you may only have one virtual private gateway attached to a VPC at any given time. Let's now walk through the high-level provisioning sequence required to stand up a VPN IPsec connection. The following sequence need to be performed: one, the VPC owner creates a virtual private gateway and attaches it to their VPC. Two, the VPC owner creates a customer gateway. This represents the corporate network peripheral firewall device. The key attribute set on the customer gateway is the externally-facing public IP address of the peripheral firewall device. Three, the VPC owner creates a VPN connection referencing both the virtual private gateway and customer gateway components. Four, the VPC owner downloads a matching vendor-specific VPN config file and provides it to the administrator of the corporate hardware device for importing. Five, customer gateway administrator imports and configures VPN settings on the firewall device. Six, the customer gateway device initiatives connectivity to the virtual private gateway. In terms of timing, the first four steps can be completed in a matter of minutes. Steps five and six either also happen very quickly or can sometimes take longer due to troubleshooting and/or other dependencies, such as device maintenance windows, et cetera. Having completed this sequence, if all goes well, a pair of redundant IPsec tunnels will be negotiated. On the AWS side, the status of the tunnels can be examined within the VPC console, as will be demonstrated later in this course. Likewise, on the customer gateway side, an administrator will have some form of ability to run diagnostic checks on the firewall device to determine their status of the tunnels. One key point to reiterate about this sequence is the following: a customer gateway device always initiates connectivity to the virtual private gateway. Let's have a quick discussion on when it makes sense to use IPsec VPNs. For obvious reasons, any time we have a need to secure data in transit over private networks, IPsec VPNs are a great solution. Mostly, IPsec VPN connections can be configured and operational within a matter of minutes and are a good choice if you have an immediate need. IPsec VPN tunnels work well with low to modest bandwidth requirements and can tolerate the inherent variability in internet-based connectivity. Inversely, the following reasons may preclude you from using IPsec VPNs: AWS currently does not support IPv6 traffic through a VPN connection. AWS only supports IPsec connections in tunnel mode. If you're moving massive amounts of data, it may make more commercial sense to provision a direct connect connection, which costs more per month for the service itself but has a cheaper data transfer rate, meaning there is a break even point between the two options that you should appreciate. Finally, the complexity built within the IPsec framework and all of its settings and configuration may be prohibitive and counterproductive to those unfamiliar with it.

ELB Hello and welcome to this lecture, which is going to focus on what the AWS Elastic Load Balancer service is and does.

Now the main function of an Elastic Load Balancer, commonly referred to as an ELB, is to help manage and control the flow of inbound requests destined to a group of targets by distributing these requests evenly across the targeted resource group. These targets could be a fleet of EC2 instances, Lambda functions, a range of IP addresses, or even Containers. The targets defined within the ELB could be situated across different Availability Zones for additional resiliency or all placed within a single Availability Zone. Let's look at this from a typical scenario.

Let's suppose you have just created a new application, which is currently residing on a single EC2 instance within your environment, and this is being accessed by a number of users. At this stage, your architecture can be logically summarized as shown. If you are familiar with architectural design and best practices, then you would realize that using a single instance approach isn't ideal although it would certainly work and provide a service to your users. However, this infrastructure layout brings some challenges. For example, the single instance where your application is located can fail, perhaps from a hardware or software fault. And if that happens, your application will be down and unavailable to your users. Also, if you experience a sudden spike in traffic, your instance may not be able to handle the additional load based on its performance limitations. As a result, to strengthen your infrastructure and help remediate these challenges, the unpredictable traffic spikes and high availability, et cetera, you should introduce an Elastic Load Balancer, an additional instance that's running your application into the design as shown.

As you can see, in this design, the AWS Elastic Load Balancer will act as the point for receiving incoming traffic from users and evenly distribute the traffic across a greater number of instances. By default, the ELB is highly available as this is a managed service provided by AWS. And so, they ensure its resilience so we don't have to. Although it might seem that ELB is a single point of failure, the ELB is in fact comprised of multiple instances managed by AWS. Also in this scenario, we now have three instances running our application. Now let me revisit the challenges we discussed previously. If any of these three instances fail, the ELB will automatically detect the failure based on defined metrics and divert any traffic to the remaining two healthy instances. Also if you experience a surge in traffic, then the additional instances running your application would help with the additional load. One of the many advantages of using ELB is the fact that it is managed by AWS, and it is, by definition, elastic. This means that it will automatically scale to meet your incoming traffic as the incoming traffic scales both up and down.

If you are system administrator or DevOps engineer running your own load balancer by yourself, then you would need to worry about scaling your load balancer and enforcing high availability. With an AWS ELB, you can create your load balancer and enable dynamic scaling with just a few clicks. Depending on your traffic distribution requirements, there are three ELBs available within AWS to choose from.

Firstly, the Application Load Balancer. This provides a flexible feature set for your web applications running the HTTP or HTTPS protocols. The Application Load Balancer operates at the request level, and it also provides advanced routing, TLS termination, and visibility features targeted at application architectures, allowing you to route traffic to different ports on the same EC2 instance.

Next, there is a Network Load Balancer. This is used for ultra-high performance for your application while at the same time managing very low latencies. It operates at connection level, routing traffic to targets within your VPC, and it's also capable of handling millions of requests per second.

Finally, the Classic Load Balancer. This is primarily used for applications that were built in the existing EC2 Classic environment and operates at both the connection and request level. We'll now talk a little bit about the components of an AWS ELB and some of the principles behind them.

Listeners. For every load balancer, regardless of the type used, you must configure at least one listener. The listener defines how your inbound connections are routed to your target groups based on ports and protocols set as conditions. The configurations of the listener itself differs slightly depending on which ELB you have selected. I will dive into the configuration of these as I discuss each ELB in further detail in upcoming lectures.

Target groups. A target group is simply a group of resources that you want your ELB to route requests to, for example a fleet of EC2 instances. You can configure your ELB with a number of different target groups, each associated with a different listener configuration and associated rules. This enables you to route traffic to different resources based upon the type of request. Rules.

Rules are associated to each listener that you have configured within your ELB, and they help to define how an incoming request gets routed to which target group. As you can see, your ELB can contain one or more listener. And each listener can contain one or more rules, and each rule can contain more than one condition, and all conditions in the rule equal a single action. An example rule could look as follows where the if statement resembles the conditions and the then statement acts as the action if all the conditions are met. So, depending on which listener request was responded to by the ELB, a rule based upon a priority listing would be associated containing these conditions and actions. If the request came from within the 10.0.1.0/24 network range, which is the first condition, and was trying to carry a HTTP PUT request, the second condition, then the request would be sent to the target group entitled Group1, which is the action.

Health checks. The ELB associates a health check that is performed against the resources defined within the target group. These health checks allow the ELB to contact each target using a specific protocol to receive a response. If no response is received within a set of thresholds, then the ELB will mark the target as unhealthy and stop sending traffic to that target.

Internal or Internet-facing ELBs. There are two different schemes that can be used for your load balancers, either internal or Internet-facing. Internet-facing, as the name implies, the nodes of the ELBs that are defined as Internet-facing are accessible via the Internet and so have a public DNS name that can be resolved with public IP address. This would be in addition to an internal IP address as well. This allows the ELB to serve incoming requests from the Internet before distributing and routing the traffic to your target groups, which in this instance could be a fleet of web servers receiving HTTP or HTTPS requests. When your Internet-facing ELB communicates with its target group, it will only use the internal IP address, meaning that your target group does not need public IP addresses. An internal ELB only has an internal IP address. This means that it can only serve requests that originate from within your VPC itself. For example, you might have an internal load balancer sitting between your web servers in the public subnet and your application servers in the private subnet.

ELB nodes. During the creation process of your ELBs, you're required to define which Availability Zone you'd like your ELB to operate within. For each Available Zone selected, an ELB node will be placed within that Availability Zone. As a result, you need to ensure that you have an ELB node associated to any Availability Zones for which you want to route traffic to. Without the Availability Zone associated, the ELB will not be able to route traffic to any targets within that Availability Zone even if they are defined within the target group. This is because the nodes are used by the ELB to distribute traffic to your target groups.

Cross-Zone load balancing. Depending on which ELB option you select, you may have the option of enabling and implementing Cross-Zone load balancing within your environment. Let's presume you have two Availability Zones activated for your ELB with each associated load balancer receiving equal amount of traffic. One Availability Zone has six targets, and the other has four as shown. When Cross-Zone load balancing is disabled, each ELB and its associated AZ would distribute its traffic with the targets within that Availability Zone only. As we can see from the image, this results in an uneven distribution of traffic for each target across the Availability Zones. With Cross-Zone load balancing enabled, regardless of how many targets are in an associated Availability Zone, the ELBs would distribute all incoming traffic evenly between all targets, ensuring each target across the Availability Zones have an even distribution.

That now brings me to the end of this lecture. In the lecture, I shall be discussing server certificates and how they are used with load balancers to help terminate encrypted requests.

ELB

Hello and welcome to this lecture, which is going to focus on what the AWS Elastic Load Balancer service is and does.

Now the main function of an Elastic Load Balancer, commonly referred to as an ELB, is to help manage and control the flow of inbound requests destined to a group of targets by distributing these requests evenly across the targeted resource group. These targets could be a fleet of EC2 instances, Lambda functions, a range of IP addresses, or even Containers. The targets defined within the ELB could be situated across different Availability Zones for additional resiliency or all placed within a single Availability Zone. Let's look at this from a typical scenario.

Let's suppose you have just created a new application, which is currently residing on a single EC2 instance within your environment, and this is being accessed by a number of users. At this stage, your architecture can be logically summarized as shown. If you are familiar with architectural design and best practices, then you would realize that using a single instance approach isn't ideal although it would certainly work and provide a service to your users. However, this infrastructure layout brings some challenges. For example, the single instance where your application is located can fail, perhaps from a hardware or software fault. And if that happens, your application will be down and unavailable to your users. Also, if you experience a sudden spike in traffic, your instance may not be able to handle the additional load based on its performance limitations. As a result, to strengthen your infrastructure and help remediate these challenges, the unpredictable traffic spikes and high availability, et cetera, you should introduce an Elastic Load Balancer, an additional instance that's running your application into the design as shown.

As you can see, in this design, the AWS Elastic Load Balancer will act as the point for receiving incoming traffic from users and evenly distribute the traffic across a greater number of instances. By default, the ELB is highly available as this is a managed service provided by AWS. And so, they ensure its resilience so we don't have to. Although it might seem that ELB is a single point of failure, the ELB is in fact comprised of multiple instances managed by AWS. Also in this scenario, we now have three instances running our application. Now let me revisit the challenges we discussed previously. If any of these three instances fail, the ELB will automatically detect the failure based on defined metrics and divert any traffic to the remaining two healthy instances. Also if you experience a surge in traffic, then the additional instances running your application would help with the additional load. One of the many advantages of using ELB is the fact that it is managed by AWS, and it is, by definition, elastic. This means that it will automatically scale to meet your incoming traffic as the incoming traffic scales both up and down.

If you are system administrator or DevOps engineer running your own load balancer by yourself, then you would need to worry about scaling your load balancer and enforcing high availability. With an AWS ELB, you can create your load balancer and enable dynamic scaling with just a few clicks. Depending on your traffic distribution requirements, there are three ELBs available within AWS to choose from.

Firstly, the Application Load Balancer. This provides a flexible feature set for your web applications running the HTTP or HTTPS protocols. The Application Load Balancer operates at the request level, and it also provides advanced routing, TLS termination, and visibility features targeted at application architectures, allowing you to route traffic to different ports on the same EC2 instance.

Next, there is a Network Load Balancer. This is used for ultra-high performance for your application while at the same time managing very low latencies. It operates at connection level, routing traffic to targets within your VPC, and it's also capable of handling millions of requests per second.

Finally, the Classic Load Balancer. This is primarily used for applications that were built in the existing EC2 Classic environment and operates at both the connection and request level. We'll now talk a little bit about the components of an AWS ELB and some of the principles behind them.

Listeners. For every load balancer, regardless of the type used, you must configure at least one listener. The listener defines how your inbound connections are routed to your target groups based on ports and protocols set as conditions. The configurations of the listener itself differs slightly depending on which ELB you have selected. I will dive into the configuration of these as I discuss each ELB in further detail in upcoming lectures.

Target groups. A target group is simply a group of resources that you want your ELB to route requests to, for example a fleet of EC2 instances. You can configure your ELB with a number of different target groups, each associated with a different listener configuration and associated rules. This enables you to route traffic to different resources based upon the type of request. Rules.

Rules are associated to each listener that you have configured within your ELB, and they help to define how an incoming request gets routed to which target group. As you can see, your ELB can contain one or more listener. And each listener can contain one or more rules, and each rule can contain more than one condition, and all conditions in the rule equal a single action. An example rule could look as follows where the if statement resembles the conditions and the then statement acts as the action if all the conditions are met. So, depending on which listener request was responded to by the ELB, a rule based upon a priority listing would be associated containing these conditions and actions. If the request came from within the 10.0.1.0/24 network range, which is the first condition, and was trying to carry a HTTP PUT request, the second condition, then the request would be sent to the target group entitled Group1, which is the action.

Health checks. The ELB associates a health check that is performed against the resources defined within the target group. These health checks allow the ELB to contact each target using a specific protocol to receive a response. If no response is received within a set of thresholds, then the ELB will mark the target as unhealthy and stop sending traffic to that target.

Internal or Internet-facing ELBs. There are two different schemes that can be used for your load balancers, either internal or Internet-facing. Internet-facing, as the name implies, the nodes of the ELBs that are defined as Internet-facing are accessible via the Internet and so have a public DNS name that can be resolved with public IP address. This would be in addition to an internal IP address as well. This allows the ELB to serve incoming requests from the Internet before distributing and routing the traffic to your target groups, which in this instance could be a fleet of web servers receiving HTTP or HTTPS requests. When your Internet-facing ELB communicates with its target group, it will only use the internal IP address, meaning that your target group does not need public IP addresses. An internal ELB only has an internal IP address. This means that it can only serve requests that originate from within your VPC itself. For example, you might have an internal load balancer sitting between your web servers in the public subnet and your application servers in the private subnet.

ELB nodes. During the creation process of your ELBs, you're required to define which Availability Zone you'd like your ELB to operate within. For each Available Zone selected, an ELB node will be placed within that Availability Zone. As a result, you need to ensure that you have an ELB node associated to any Availability Zones for which you want to route traffic to. Without the Availability Zone associated, the ELB will not be able to route traffic to any targets within that Availability Zone even if they are defined within the target group. This is because the nodes are used by the ELB to distribute traffic to your target groups.

Cross-Zone load balancing. Depending on which ELB option you select, you may have the option of enabling and implementing Cross-Zone load balancing within your environment. Let's presume you have two Availability Zones activated for your ELB with each associated load balancer receiving equal amount of traffic. One Availability Zone has six targets, and the other has four as shown. When Cross-Zone load balancing is disabled, each ELB and its associated AZ would distribute its traffic with the targets within that Availability Zone only. As we can see from the image, this results in an uneven distribution of traffic for each target across the Availability Zones. With Cross-Zone load balancing enabled, regardless of how many targets are in an associated Availability Zone, the ELBs would distribute all incoming traffic evenly between all targets, ensuring each target across the Availability Zones have an even distribution.

That now brings me to the end of this lecture. In the lecture, I shall be discussing server certificates and how they are used with load balancers to help terminate encrypted requests

ALB

Hello and welcome to this lecture covering the Application Load Balancer, the ALB. The first of the three load balancers that I shall be discussing. If you are familiar with the open systems interconnection model, the OSI model, then you won't be surprised that the ALB operates at layer seven, the application layer. The application layer of the OSI model serves as the interface for users and application processes to access network services. Everything at this layer is application specific. The application layer of the model helps to provide network services to the applications. And examples of the application process or services it offers are http, ftp, smtp and nfs. For more information on the OSI model, please see our existing course [here](https://cloudacademy.com/course/osi-and-tcp-ip-networking-models/osi-and-tcp-ip-networking-models/).

As you can see AWS suggests you use the application load balancer if you need to provide a flexible feature set including advanced routing and visibility features aimed purely for application architectures such as microservices and containers when used in HTTP or HTTPS. Before configuring your ALB, it's good practice to set up your target groups. Now I explained in a previous lecture that a target group is simply a group of resources that you want your ALB to route requests to. You might want to configure different target groups depending on the nature of your requests. For example, let's say you had an internet-facing ALB, you might want a target group allocated to handle and process HTTP port 80 requests and a different target group configured to process requests from the secure HTTPS protocol using port 443. In this scenario, you could configure two different target groups and then route traffic, depending on the request, to different targets through the use of listeners and rules.

I now want to demonstrate how to configure an ALB and in this demonstration, I will also show you how to set up target groups as well. Let's take a look.

As you can see, I'm in the AWS management console and the first thing we want to do is create our target groups and I can do this by going into the EC2 service which is found here under compute. And then if I scroll down on the left-hand side, I'll get to the load balancing section here. Then in here, I have load balancers and target groups, but first I want to set up our target groups. So, if you select target group, as you can see there's no target groups currently configured. So if I click on the blue button, Create target group, I now have a page of information that I need to complete.

So, firstly the target group name and I'm going to call this Web Servers. And then I have my target type and here I can specify it by instance, IP or Lambda function. I'm going to leave it as instance, then we can select what protocol we want. As this is going to be a web service, I'll leave it as HTTP on port 80 and here we can select our VPC that we want this target group to exist in. So, select my appropriate virtual private cloud there. At the bottom we just have some health check settings and this is the path and protocol that the load balance will use when performing its health checks. So, for the path I'll just put in index.html as an example. If we take a look at the advanced health check settings, here we have a number of other options that we can select.

This value specifies the healthy threshold which means the load balancer will have to receive five responses from the instance before deeming the previously unhealthy target healthy again, and the unhealthy threshold means that the load balancer only has to receive two failures before marking the instance as unhealthy. The timeout is simply the number of seconds that the load balancer will wait for a response, and the interval is how many seconds between each health check. Once we're happy with our configuration, we'll click on Create. Looks like I left this space in the name and you can't have any spaces. So let's just delete that and click on Create. And there you go, our target group was successfully created, and now we can see it in our list of target groups.

However, we don't have any targets associated with this group yet. That was just simply the configuration of the group. So, down here we have the Description, the Targets, the Health checks, Monitoring and Tags, but if we click on the Targets tab, then here we can start adding our targets associated with this target group. And if we click on Edit, we can see here at the bottom that there's two instances that I have running, web server one, and web server two. Now here I can select which instance I want to add and associate to this target group. For this demonstration, I'm going to select both instances and then add these as registered targets to this group. And as you can see, these two instances have now been added under the Registered targets section. Click on Save. And we can see here, that we have two registered targets which are the ones I just added, web server one and web server two, now associated to this target group.

Let's just quickly look at these other tags here as well, the Health checks, that's the health check information that we configured during the creation of the target group; Monitoring, this shows a number of CloudWatch metrics associated with the target group such as number of healthy hosts and unhealthy hosts et cetera. And then we also have Tags if you wanted to create a key-value pair for your target group and you can do so here. So, as you can see, it's very easy to create different target groups as you need to for your load balancing.

Let us now go ahead and create an Application Load Balancer. So, back on the left-hand side here, again under Load Balancing, we have Load Balancers. So if you select that. Now I don't have any load balancers configured here. So if I click on Create Load Balancer, and I can create an Application Load Balancer and Network Load Balancer or the Classic. In this example, I'm going to create the Application Load Balancer. So, click on Create. Now here we have a number of different steps. Firstly, we need to give it a name. So this would be WebServerALB, and we'll have this as internet-facing using ipv4. Now down here we have our listeners. So this is the port and protocol that we want the load balancer to listen on and as this is our web server, let's leave it as HTTP on port 80. If you want to add additional listeners, then you can do so just by selecting Add Listener and selecting the different protocols et cetera. Now if we scroll down to the bottom here, we can select our Availability Zones that we want to enable for our load balancer. So for eu-west-1a, let me select this subnet and for eu-west-1b I, shall select this subnet. So there are the two subnets that I want to associate with the load balancer, and each of them are in a set per availability zone as you can see here.

Now I need to go and configure my security settings. And I have a message here to say that the load balancer security is not using a secure listener. Now, if we were to go back and change that to HTTPS, then we would be using a secure listener and we'd also have to set up server certificates as well, but for this demonstration, I just want to show you how to create the Application Load Balancer, but generally in a wide-scale production environment, if you're creating a load balancer for your internet-facing resources, then you'd probably want to use https for that additional security. Click on next.

Now we'll need to select the security group that is going to be associated to our load balancer. So we could create a new security group, call this our Application Load Balancer. So we'll have HTTP from any IP address, and we click on that Next Configure Reading. And this is where we can specify our target group for the Application Load Balancer. So we can create a new target group here and go through the same process as we did earlier or click on the dropdown list and select an existing target group and here we can see that we have our WebServers target group that we created earlier with all the settings already pre-filled. Click on Next Register Targets, and here we can see that these are the two targets associated with the target group. Click on Next Review. And this is just a review of all the configuration options that we made during the creation of this. Once you're happy with that, simply click on Create. And then we have it.

We have our new load balancer, our WebServerALB was successfully created. So, let's take a look. This might take it a couple of minutes for it to be provisioned. While that's being provisioned, if we take a look at the bottom here, we can see that we have some basic configuration that we've set up with the availability zones, the fact that it's internet-facing, and we have the ARN et cetera as well. We have our listener configuration here that we can change if we need to. As we can see, at the minute we're listening on port 80. Again, we have some monitoring metrics here being carried out by CloudWatch. We'll see a number of different CloudWatch metrics. Actually, we can now see that that the state is now active. So that's our Application Load Balancer set up and configured.

Now before I finish this quick demonstration, I just want to show you the rules that I mentioned earlier with regards to listeners. So, if we go down to our Listeners tab here, we can see that we can view and edit our rules for our listener. So if we click on that, at the moment we can see that we have our default action here listening on port 80. We can see that this rule cannot be moved or deleted. That's basically saying that this listener is listening on port 80 and for any requests then forward it to the WebServers. But we can add additional rules in here. So let's take a look. So if we click on this plus button, we can see that we now have this option here of Insert Rule. So, let me select that, and that allows me to add a new rule in. So, first we need to add a condition. So, for example, let's have the condition of a Source IP, then we just put in a random IP address here. So, this is saying if the source IP is this IP address, then add the following action, and here we could choose to forward it to another target group. I mean, I've only got one target group configured at the minute called WebServers, but if I had other target groups here with different instances associated to those target groups, then I could select a different target group to forward any requests that are received from this IP address. So that allows you to customize how your load balancer directs traffic, depending on what rules you create with your listeners. So, when I was talking about conditions and rules in a previous lecture, then this is the section that I was referring to. So I just wanted to show you that quickly within this demonstration, where you can edit your rules and add customization with conditions and actions.

Okay, and that's the end of this demonstration.