AWS Cloud Practitioner Essentials

Module 3 introduction

Learning objectives

In this module, you will learn how to:

- Summarize the benefits of the AWS Global Infrastructure.
- Describe the basic concept of Availability Zones.
- Describe the benefits of Amazon Cloud Front and edge locations.
- Compare different methods for provisioning AWS services.

Video transcript

I want to talk to you about high availability. Say you wanna grab a cup of coffee at the cafe, but today is not just a normal day in our city. Today, there is a parade coming to march down our street in celebration of all of the wonderful cloud migrations that have been successful. This parade is going to march right in front of our shop. This is great because who doesn't love to look at floats and balloons and have someone throw candy at them from the street? But this is also bad because while the parade is coming down the street, our customers who are driving to come get coffee will be diverted and won't be able to stop by the shop. This will drive sales down and also make customers unhappy.

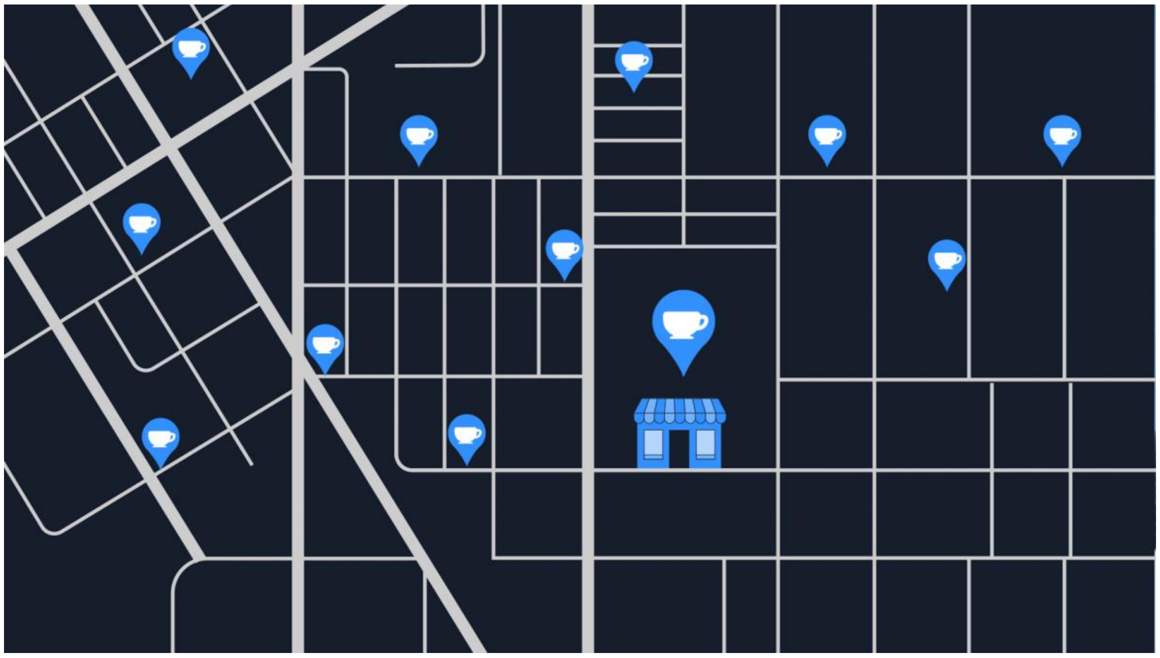
Luckily for us, we thought ahead. We thought long ago what would happen if for some reason our customers couldn't make it into the shop temporarily? Like if there was a parade or if there's a flood or some other event that blocks us from getting into the shop? Well, no matter the reason, we need to be highly available for our customers.

So, I'll let you in on a secret. This isn't our coffee shop's only location. The cafe is actually a chain and we have locations all around the city. That way, if a parade comes down the street, or if there was a power outage on our block, no worries. Customers can still get their coffee by visiting one of our shops just a few blocks away. We still make money and they get their coffee. All is well.

AWS has done a similar thing with how the AWS global infrastructure is set up. It's not good enough to have one giant data center where all of the resources are. If something were to happen to that data center, like a power outage or a natural disaster, everyone's applications would go down all at once. You need high availability and fault tolerance.

Turns out, it's not even good enough to have two giant data centers. Instead, AWS operates in all sorts of different areas around the world called Regions. We're going to talk about this in depth in upcoming videos. In the meantime, I'm gonna sit here and relax, because I know my business is highly available regardless of any parades blocking the street.

Building a global footprint



To understand the AWS global infrastructure, consider the coffee shop. If an event such as a parade, flood, or power outage impacts one location, customers can still get their coffee by visiting a different location only a few blocks away.

This is similar to how the AWS global infrastructure works.

AWS global infrastructure

Video transcript

To understand the global infrastructure AWS, I wanna begin with your fundamental business need. You have an application that you have to run, or content you need stored, or data you need analyzed. Basically you have stuff that has to live and operate somewhere. Now historically, businesses had to run applications in their own data centers, 'cause they didn't have a choice. Once AWS became available, companies like yours could now run their applications in other data centers they didn't actually own.

But the conversation is so much more than that, 'cause I want you to understand a fundamental problem with any data center, doesn't matter who built it or who owns it. Events can happen that could cause you to lose connection to that building. If you run your own data center, you have to figure out how to answer the question of what will you do when disaster strikes your building. You could run a second data center, but real estate prices alone could stop you, much less all the duplicate expense of hardware, employees, electricity, heating and cooling, security. Most businesses simply end up just storing backups somewhere, and then hope for the disaster to never come. And hope is not a good business plan.

AWS answers the question of what happens when disaster strikes by building our data centers in large groups we call Regions, and here's how it's designed.

Throughout the globe, AWS builds Regions to be closest to where the business traffic demands. Locations like Paris, Tokyo, Sao Paulo, Dublin, Ohio. Inside each Region, we have multiple data centers that have all the compute, storage, and other services you need to run your applications. Each Region can be connected to each other Region through a high speed fiber network, controlled by AWS, a truly global operation from corner to corner if you need it to be. Now before we get into the architecture of how each Region is built, it's important to know that you, the business decision maker, gets to choose which Region you want to run out of. And each Region is isolated from every other Region in the sense that absolutely no data goes in or out of your environment in that Region without you explicitly granting permission for that data to be moved. This is a critical security conversation to have.

For example, you might have government compliance requirements that your financial information in Frankfurt cannot leave Germany. Well this is absolutely the way AWS operates outta the box. Any data stored in the Frankfurt Region never leaves the Frankfurt Region, or data in the London region never leaves London, or Sydney never leaves Sydney, unless you explicitly, with the right credentials and permissions, request the data be exported.

Regional data sovereignty is part of the critical design of AWS Regions. With data being subject to the local laws and statutes of the country where the Region lives. So with that understanding, that your data, your application, lives and runs in a Region, one of the first decisions you get to make is which Region do you pick? There's four business factors that go into choosing a Region.

Number one, compliance. Before any of the other factors, you must first look at your compliance requirements. Do you have a requirement your data must live in UK boundaries? Then you should choose the London Region, full stop. None of the rest of the options matter. Or let's say you must run inside Chinese borders. Well then, you should choose on of our Chinese Regions. Most businesses, though, are not governed by such strict regulations. So if you don't have a compliance or regulatory control that dictates your Region, then you can look at the other factors.

Number two, proximity. How close you are to your customer base is a major factor because speed of light, still the law of the universe. If most of your customers live in Singapore, consider running out of the Singapore Region. Now you certainly can run out of Virginia, but the time it takes for the information to be sent, or latency, between the US and Singapore is always going to be a factor. Now we may be developing quantum computing, but quantum networking, still some ways away. The time it takes light to travel around the world is always a consideration. Locating close to your customer base, usually the right call.

Number three, feature availability. Sometimes the closest Region may not have all the AWS features you want. Now here's one of the cool things about AWS. We are constantly innovating on behalf of our customers. Every year, AWS releases thousands of new features and products specifically to answer customer requests and needs. But sometimes those brand new services take a lot of new physical hardware that AWS has to build out to make the service operational. And sometimes that means we have to build the service out one Region at a time. So let's say your developers wanted to play with Amazon Braket, our new quantum computing platform. Well then, they have to run in the Regions that already have the hardware installed. Eventually, can we expect it to be in every Region? Well, yeah, that's a good expectation, but if you wanna use it today, then that might be your deciding factor.

Number four, pricing. Even when the hardware is equal one Region to the next, some locations are just more expensive to operate in, for example, Brazil. Now Brazil's tax structure is such that it costs AWS significantly more to operate the exact same services there than in many other countries. The exact same workload in Sao Paulo might be, let's say, 50% more expensive to run than out of Oregon in the United States. Price can be determined by many factors, so AWS has very transparent granular pricing that we'll continue to discuss in this class. But know that each Region has a different price sheet. So if budget is your primary concern, even if your customers live in Brazil, you might still wanna operate in a different country, again, if price is your primary motivation.

So four key factors to choose a Region: Compliance, proximity, feature availability, and pricing. When we come back, we'll look at the moving parts inside a Region.

Selecting a Region

When determining the right Region for your services, data, and applications, consider the following four business factors.

Compliance with data governance and legal requirements

Depending on your company and location, you might need to run your data out of specific areas. For example, if your company requires all of its data to reside within the boundaries of the UK, you would choose the London Region.

Not all companies have location-specific data regulations, so you might need to focus more on the other three factors.

Proximity to your customers

Selecting a Region that is close to your customers will help you to get content to them faster. For example, your company is based in Washington, DC, and many of your customers live in Singapore. You might consider running your infrastructure in the Northern Virginia Region to be close to company headquarters, and run your applications from the Singapore Region.

Available services within a Region

Sometimes, the closest Region might not have all the features that you want to offer to customers. AWS is frequently innovating by creating new services and expanding on features within existing services. However, making new services available around the world sometimes requires AWS to build out physical hardware one Region at a time.

Suppose that your developers want to build an application that uses Amazon Braket (AWS quantum computing platform). As of this course, Amazon Braket is not yet available in every AWS Region around the world, so your developers would have to run it in one of the Regions that already offers it.

Pricing

Suppose that you are considering running applications in both the United States and Brazil. The way

Brazil’s tax structure is set up, it might cost 50% more to run the same workload out of the São Paulo Region compared to the Oregon Region. You will learn in more detail that several factors determine pricing, but for now know that the cost of services can vary from Region to Region.

Video transcript

If a Region is where all of the pieces and parts of your application live, some of you might be thinking that we never actually solved the problem that we presented in the last video. Let me restate the problem. You don't want to run your application in a single building because a single building can fail for any number of unavoidable reasons.

You may be thinking, if my business needs to be disaster proof, then it can't run in just one location. Well, you're absolutely correct. AWS agrees with that statement and that's why our Regions are not one location. To begin with, AWS has data centers, lots of data centers, all around the world, and each Region is made up of multiple data centers.

AWS calls a single data center or a group of data centers, an Availability Zone or AZ. Each Availability Zone is one or more discrete data centers with redundant power, networking, and connectivity. When you launch an Amazon EC2 instance, it launches a virtual machine on a physical hardware that is installed in an Availability Zone. This means each AWS Region consists of multiple isolated and physically separate Availability Zones within a geographic Region.

But we don't build Availability Zones right next to each other because if a large scale incident were to occur, like a natural disaster, for example, you could lose connectivity to everything in that Availability Zone. The question what happens in case of a disaster matters and if you are familiar with disaster recovery planning, you might even have an idea of where we are going with this.

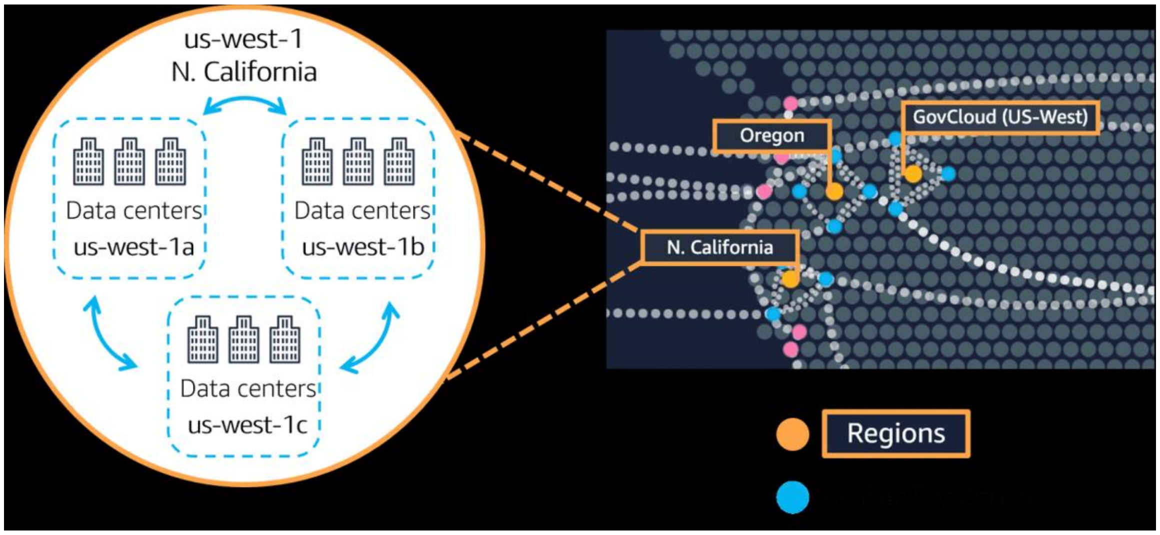
If you only run one EC2 instance, it only runs in one building, or one Availability Zone and a large scale disaster occurs, will your application still be able to run and serve your business? The obvious solution to this is to run multiple EC2 instances, just like we showed in the scaling example earlier. But the main thing is don't run them in the same building. Don't even run them in the same street, push them as far apart as you can before the speed of light tells you to stop if you still want low latency communication. Turns out that the speed of light will let us move these Availability Zones tens of miles apart from each other and still keep single digit millisecond latency between these Availability Zones. Now, if a disaster strikes, your application continues just fine because this disaster only knocked over some of your capacity, not all.

And as we saw in the last section, you can rapidly spin up more capacity in the remaining Availability Zones, thus allowing your business to continue operating without interruption. And as a best practice with AWS, we always recommend you run across at least two Availability Zones in a Region. This means redundantly deploying your infrastructure in two different AZs.

But there's more to Regions than just places to run EC2. Many of the AWS services run at the Region level, meaning they run synchronously across multiple AZs without any additional effort on your part. Take the ELB we talked about previously. This is actually a regional construct. It runs across all Availability Zones, communicating with the EC2 instances that are running in a specific Availability Zone. Regional services are by definition already highly available at no additional cost of effort on your part.

So as you plan for high availability, any service that is listed as a regionally scoped service, you'll already have that box checked. When we come back, let's look at going outside the Regions for additional solutions.

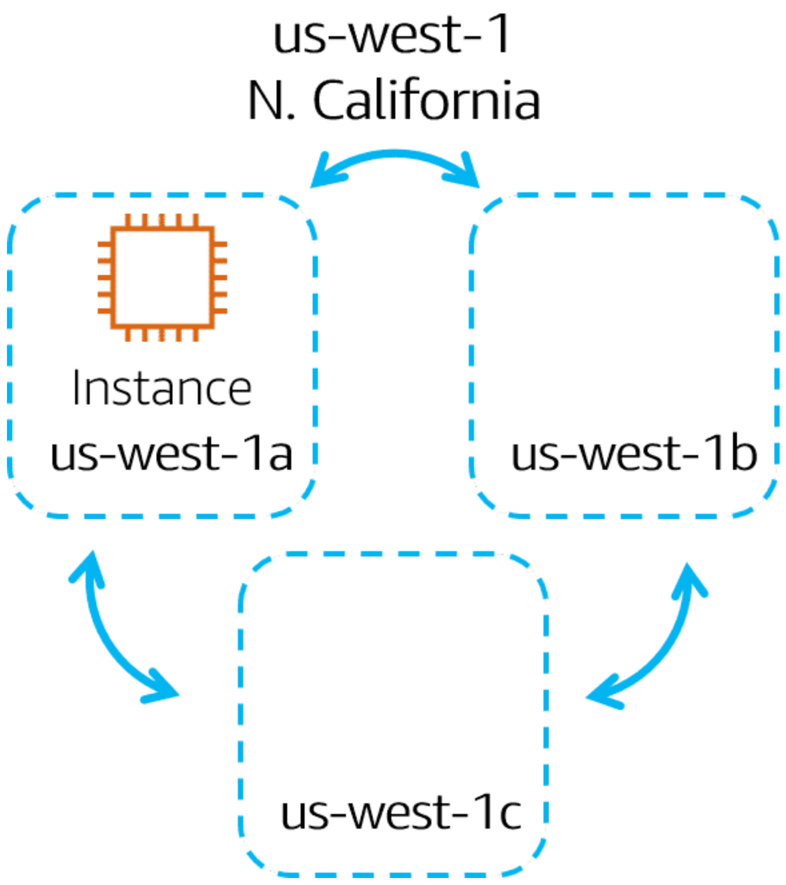
Availability Zones



An Availability Zone is a single data center or a group of data centers within a Region. Availability Zones are located tens of miles apart from each other. This is close enough to have low latency (the time between when content requested and received) between Availability Zones. However, if a disaster occurs in one part of the Region, they are distant enough to reduce the chance that multiple Availability Zones are affected.

Running Amazon EC2 instances in multiple Availability Zones

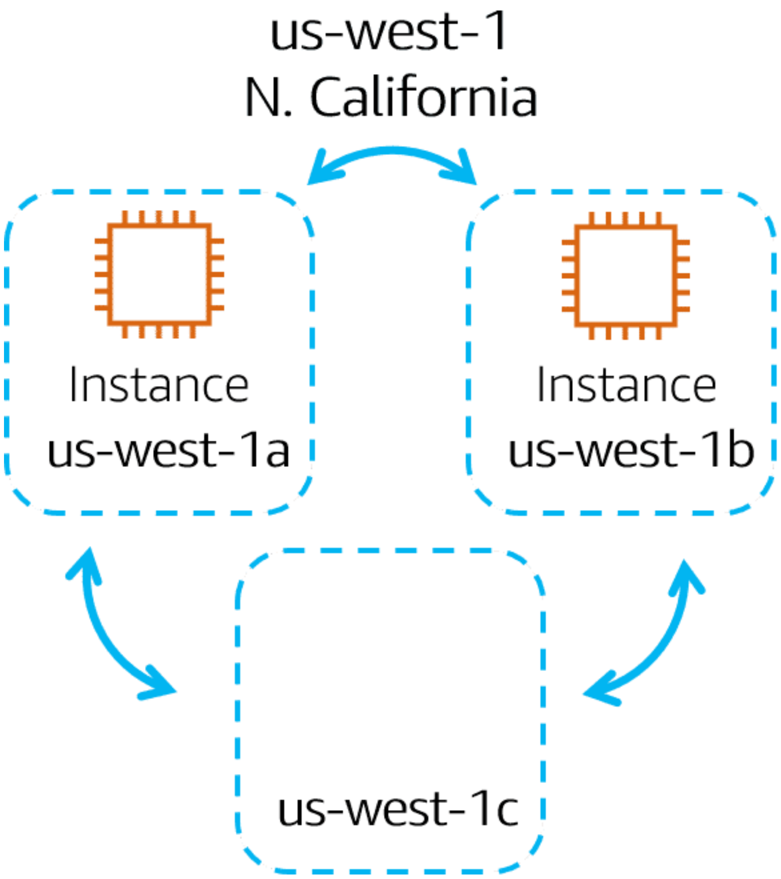
Amazon EC2 instance in a single Availability Zone



Suppose that you're running an application on a single Amazon EC2 instance in the Northern California Region. The instance is running in the us-west-1a Availability Zone. If us-west-1a were to fail, you would lose your instance.

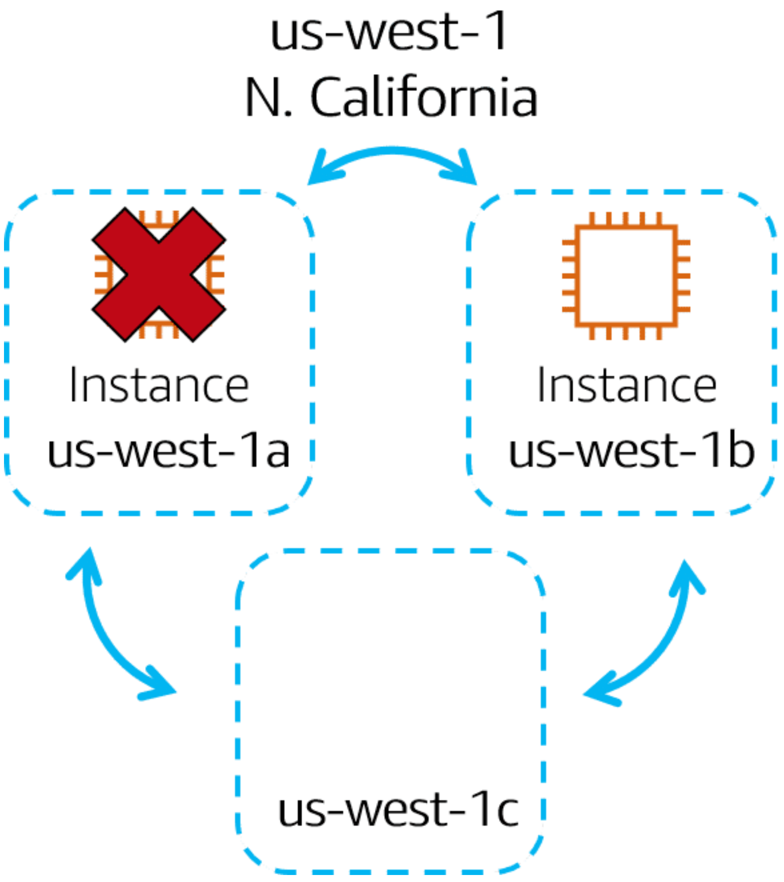
Amazon EC2 instances in multiple Availability Zones

A best practice is to run applications across at least two Availability Zones in a Region. In this example, you might choose to run a second Amazon EC2 instance in us-west-1b.Availability Z



Availability Zone failure

If us-west-1a were to fail, your application would still be running in us-west-1b.



Summary

Planning for failure and deploying applications across multiple Availability Zones is an important part of building a resilient and highly available architecture.

Edge locations

Video transcript

One of the great things about the AWS global infrastructure, is the way it's engineered to help you better serve your customers. Remember when selecting a Region, one of the major criteria was proximity to your customers, but what if you have customers all over the world or in cities that are not close to one of our Regions? Well, think about our coffee shop. If you have a good customer base in a new city, you can build a satellite store to service those customers.

You don't need to build an entire new headquarters or from an IT perspective, if you have customers in Mumbai who need access to your data, but the data is hosted out of the Tokyo Region, rather than having all the Mumbai-based customers, send requests all the way to Tokyo, to access the data, just place a copy locally or cache a copy in Mumbai. Caching copies of data closer to the customers all around the world uses the concept of content delivery networks, or CDNs.

CDNs are commonly used, and on AWS, we call our CDN Amazon CloudFront. Amazon CloudFront is a service that helps deliver data, video, applications, and APIs to customers around the world with low latency and high transfer speeds. Amazon CloudFront uses what are called Edge locations, all around the world, to help accelerate communication with users, no matter where they are.

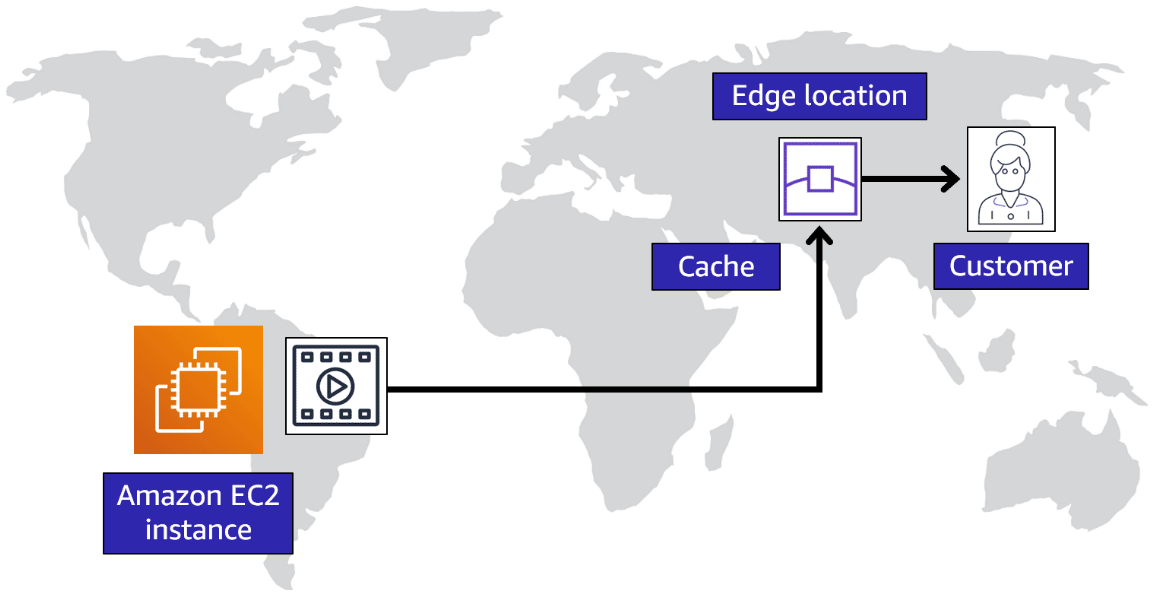
Edge locations are separate from Regions, so you can push content from inside a Region to a collection of Edge locations around the world, in order to accelerate communication and content delivery. AWS Edge locations, also run more than just CloudFront. They run a domain name service, or DNS, known as Amazon Route 53, helping direct customers to the correct web locations with reliably low latency.

But what if your business wants to use, AWS services inside their own building? Well sure. AWS can do that for you. Introducing AWS Outposts, where AWS will basically install a fully operational mini Region, right inside your own data center. That's owned and operated by AWS, using 100% of AWS functionality, but isolated within your own building. It's not a solution most customers need, but if you have specific problems that can only be solved by staying in your own building, we understand, AWS Outposts can help.

All right, there is so much more that we can say about AWS global infrastructure, but let's keep it simple and stop here. So here's the key points. Number one, Regions are geographically isolated areas, where you can access services needed to run your enterprise. Number two, Regions contain Availability Zones, that allow you to run across physically separated buildings, tens of miles of separation, while keeping your application logically unified. Availability Zones help you solve high availability and disaster recovery scenarios, without any additional effort on your part, and number three, AWS Edge locations run Amazon CloudFront to help get content closer to your customers, no matter where they are in the world.

Edge locations

An edge location is a site that Amazon CloudFront uses to store cached copies of your content closer to your customers for faster delivery.



How to provision AWS resources

Video transcript

We've been talking about a few different AWS resources as well as the AWS global infrastructure. You may be wondering, how do I actually interact with these services? And the answer is APIs. In AWS, everything is an API call. An API is an application programming interface. And what that means is, there are pre determined ways for you to interact with AWS services. And you can invoke or call these APIs to provision, configure, and manage your AWS resources.

For example, you can launch an EC2 instance or you can create an AWS Lambda function. Each of those would be different requests and different API calls to AWS. You can use the AWS Management Console, the AWS Command Line Interface, the AWS Software Development Kits, or various other tools like AWS CloudFormation, to create requests to send to AWS APIs to create and manage AWS resources.

First, let's talk about the AWS Management Console. The AWS Management Console is browser-based. Through the console, you can manage your AWS resources visually and in a way that is easy to digest. This is great for getting started and building your knowledge of the services. It's also useful for building out test environments or viewing AWS bills, viewing monitoring and working with other non technical resources. The AWS Management Console is most likely the first place you will go when you are learning about AWS.

However, once you are up and running in a production type environment, you don't want to rely on the point and click style that the console gives you to create and manage your AWS resources. For example, in order to create an Amazon EC2 Instance, you need to click through various screens, setting all the configurations you want, and then you launch your instance. If later, you wanted to launch another EC2 Instance, you would need to go back into the console and click through those screens again to get it up and running. By having humans do this sort of manual provisioning, you're opening yourself up to potential errors. It's pretty easy to forget to check a checkbox or misspell something when you are doing everything manually.

The answer to this problem is to use tools that allow you to script or program the API calls. One tool you can use is the AWS Command Line Interface or CLI. The CLI allows you to make API calls using the terminal on your machine. This is different than the visual navigation style of the Management Console. Writing commands using the CLI makes actions scriptable and repeatable. So, you can write and run your commands to launch an EC2 Instance. And if you want to launch another, you can just run the pre-written command again. This makes it less susceptible to human error. And you can have these scripts run automatically, like on a schedule or triggered by another process.

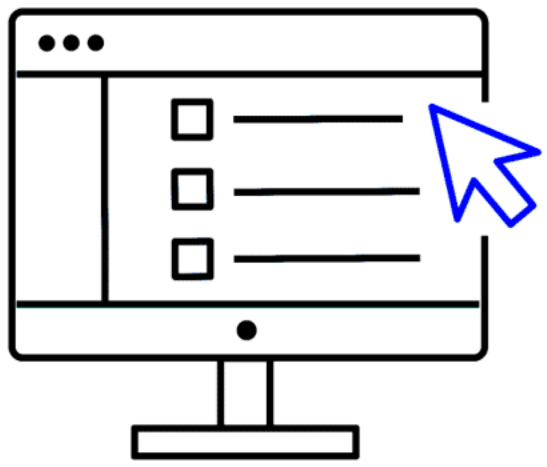
Automation is very important to having a successful and predictable cloud deployment over time. Another way to interact with AWS is through the AWS Software Development Kits or SDKs. The SDKs allow you to interact with AWS resources through various programming languages. This makes it easy for developers to create programs that use AWS without using the low level APIs, as well as avoiding that manual resource creation that we just talked about. More on that in a bit.

Ways to interact with AWS services

AWS Management Console

The AWS Management Console is a web-based interface for accessing and managing AWS services. You can quickly access recently used services and search for other services by name, keyword, or acronym. The console includes wizards and automated workflows that can simplify the process of completing tasks.

You can also use the AWS Console mobile application to perform tasks such as monitoring resources, viewing alarms, and accessing billing information. Multiple identities can stay logged into the AWS Console mobile app at the same time.

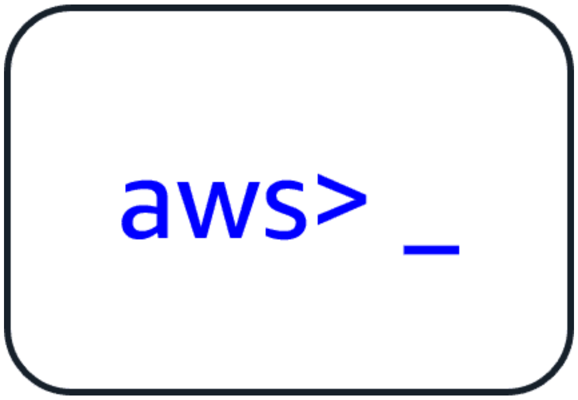


AWS Command Line Interface

To save time when making API requests, you can use the AWS Command Line Interface (AWS CLI). AWS CLI enables you to control multiple AWS services directly from the command line within one tool. AWS

CLI is available for users on Windows, macOS, and Linux.

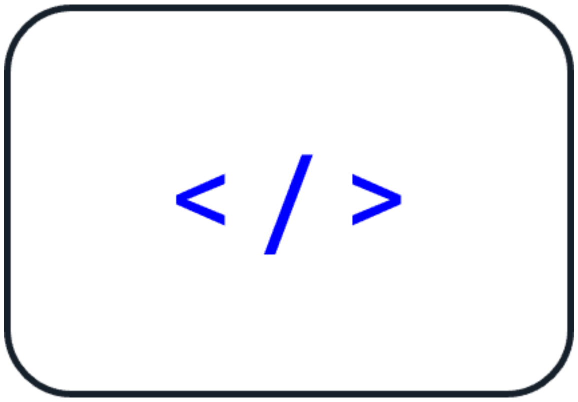
By using AWS CLI, you can automate the actions that your services and applications perform through scripts. For example, you can use commands to launch an Amazon EC2 instance, connect an Amazon EC2 instance to a specific Auto Scaling group, and more.



software development kits

Another option for accessing and managing AWS services is the software development kits (SDKs). SDKs make it easier for you to use AWS services through an API designed for your programming language or platform. SDKs enable you to use AWS services with your existing applications or create entirely new applications that will run on AWS.

To help you get started with using SDKs, AWS provides documentation and sample code for each supported programming language. Supported programming languages include C++, Java, .NET, and more.



Video transcript

All right, let's continue to talk about how to interact with AWS. You have the AWS Management Console, the CLI, and the SDKs, which are all sort of do it yourself ways to provision and manage your AWS environment. If you want to provision a resource, you can log into the console and point and click, you can write some commands, or you can write some programs to do it. There are also other ways you can manage your AWS environment using managed tools like AWS Elastic Beanstalk, and AWS CloudFormation.

AWS Elastic Beanstalk is a service that helps you provision Amazon EC2-based environments. Instead of clicking around the console or writing multiple commands to build out your network, EC2 instances, scaling and Elastic Load Balancers, you can instead provide your application code and desired configurations to the AWS Elastic Beanstalk service, which then takes that information and builds out your environment for you. AWS Elastic Beanstalk also makes it easy to save environment configurations, so they can be deployed again easily. AWS Elastic Beanstalk gives you the convenience of not having to provision and manage all of these pieces separately, while still giving you the visibility and control of the underlying resources. You get to focus on your business application, not the infrastructure.

Another service you can use to help create automated and repeatable deployments is AWS CloudFormation. AWS CloudFormation is an infrastructure as code tool that allows you to define a wide variety of AWS resources in a declarative way using JSON or YAML text-based documents called CloudFormation templates. A declarative format like this allows you to define what you want to build without specifying the details of exactly how to build it. CloudFormation lets you define what you want and the CloudFormation engine will worry about the details on calling APIs to get everything built out.

It also isn't just limited to EC2-based solutions. CloudFormation supports many different AWS resources from storage, databases, analytics, machine learning, and more. Once you define your resources in a CloudFormation template, CloudFormation will parse the template and begin provisioning all the resources you defined in parallel. CloudFormation manages all the calls to the backend AWS APIs for you. You can run the same CloudFormation template in multiple accounts or multiple regions, and it will create identical environments across them. There is less room for human error as it is a totally automated process.

To recap, the AWS Management Console is great for learning and providing a visual for the user. The AWS Management Console is a manual tool. So right off the bat, it isn't a great option for automation. You can instead use the CLI to script your interactions with AWS using the terminal. You can use the SDKs to write programs to interact with AWS for you or you can use manage tools like AWS Elastic Beanstalk or AWS CloudFormation.

AWS Elastic Beanstalk

With AWS Elastic Beanstalk, you provide code and configuration settings, and Elastic Beanstalk deploys the resources necessary to perform the following tasks:

- Adjust capacity
- Load balancing
- Automatic scaling

- Application health monitoring

AWS CloudFormation

With AWS CloudFormation, you can treat your infrastructure as code. This means that you can build an environment by writing lines of code instead of using the AWS Management Console to individually provision resources.

AWS CloudFormation provisions your resources in a safe, repeatable manner, enabling you to frequently build your infrastructure and applications without having to perform manual actions. It determines the right operations to perform when managing your stack and rolls back changes automatically if it detects errors.

Module 4 introduction

Learning objectives

In this module, you will learn how to:

- Describe the basic concepts of networking.
- Describe the difference between public and private networking resources.
- Explain a virtual private gateway using a real life scenario.
- Explain a virtual private network (VPN) using a real life scenario.
- Describe the benefit of AWS Direct Connect.
- Describe the benefit of hybrid deployments.
- Describe the layers of security used in an IT strategy.
- Describe the services customers use to interact with the AWS global network.

Video transcript

If we think back to our coffee shop or AWS account, things by now should be running smoothly. Although, what if we had a few eager customers who wanted to give their orders directly to the baristas instead of the cashier out in front? Well, it doesn't make sense to allow every customer to be able to interact with our baristas since they are focused on brewing some fresh caffeinated beverages. So what do we do?

That's right, kids, say it with me, Amazon Virtual Private Cloud, or VPCs, as they're affectionately known. A VPC lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. These resources can be public facing so they have access to the internet, or private with no internet access, usually for backend services like databases or application servers. The public and private grouping of resources are known as subnets and they are ranges of IP addresses in your VPC.

Now, in our coffee shop, we have different employees and one is a cashier. They take customers' orders and thus we want customers to interact with them, so we put them in what we call a public subnet. Hence they can talk to the customers or the internet. But for our baristas, we want them to focus on making coffee and not interact with customers directly, so we put them in a private subnet.

Okay, let's get into the next video where we'll dive into networking.

Connectivity to AWS

Video transcript

A VPC, or Virtual Private Cloud, is essentially your own private network in AWS. A VPC allows you to define your private IP range for your AWS resources, and you place things like EC2 instances and ELBs inside of your VPC.

Now you don't just go throw your resources into a VPC and move on. You place them into different subnets. Subnets are chunks of IP addresses in your VPC that allow you to group resources together. Subnets, along with networking rules we will cover later, control whether resources are either publicly or privately available. What we haven't told you yet is there are actually ways you can control what traffic gets into your VPC at all. What I mean by this is, for some VPCs, you might have internet-facing resources that the public should be able to reach, like a public website, for example.

However, in other scenarios, you might have resources that you only want to be reachable if someone is logged into your private network. This might be internal services, like an HR application or a backend database. First, let's talk about public-facing resources. In order to allow traffic from the public internet to flow into and out of your VPC, you must attach what is called an internet gateway, or IGW, to your VPC.

An internet gateway is like a doorway that is open to the public. Think of our coffee shop. Without a front door, the customers couldn't get in and order their coffee, so we install a front door and the people can then go in and out of that door when coming and going from our shop. The front door in this example is like an internet gateway. Without it, no one can reach the resources placed inside of your VPC.

Next, let's talk about a VPC with all internal private resources. We don't want just anyone from anywhere to be able to reach these resources. So we don't want an internet gateway attached to our

VPC. Instead, we want a private gateway that only allows people in if they are coming from an approved network, not the public internet. This private doorway is called a virtual private gateway, and it allows you to create a VPN connection between a private network, like your on-premises data center or internal corporate network to your VPC.

To relate this back to the coffee shop, this would be like having a private bus route going from my building to the coffee shop. If I want to go get coffee, I first must badge into the building, thus authenticating my identity, and then I can take the secret bus route to the internal coffee shop that only people from my building can use. So if you want to establish an encrypted VPN connection to your private internal AWS resources, you would need to attach a virtual private gateway to your VPC.

Now the problem with our super secret bus route is that it still uses the open road. It's susceptible to traffic jams and slowdowns caused by the rest of the world going about their business. The same thing is true for VPN connections. They are private and they are encrypted, but they still use a regular internet connection that has bandwidth that is being shared by many people using the internet.

So what I've done to make things more reliable and less susceptible to slowdowns is I made a totally separate magic doorway that leads directly from the studio into the coffee shop. No one else driving around on the road can slow me down because this is my direct doorway; no one else can use it. What, did you not have a secret magic doorway into your favorite coffee shop? All right, moving on. The point being is you still want a private connection, but you want it to be dedicated and shared with no one else. You want the lowest amount of latency possible with the highest amount of security possible.

With AWS, you can achieve that using what is called AWS Direct Connect. Direct Connect allows you to establish a completely private, dedicated fiber connection from your data center to AWS. You work with a Direct Connect partner in your area to establish this connection, because like my magic doorway, AWS Direct Connect provides a physical line that connects your network to your AWS VPC. This can help you meet high regulatory and compliance needs, as well as sidestep any potential bandwidth issues. It's also important to note that one VPC might have multiple types of gateways attached for multiple types of resources all residing in the same VPC, just in different subnets.

Thanks for listening. I'm gonna sit here and keep ordering coffees from my magic door. See ya.

Amazon Virtual Private Cloud (Amazon VPC)

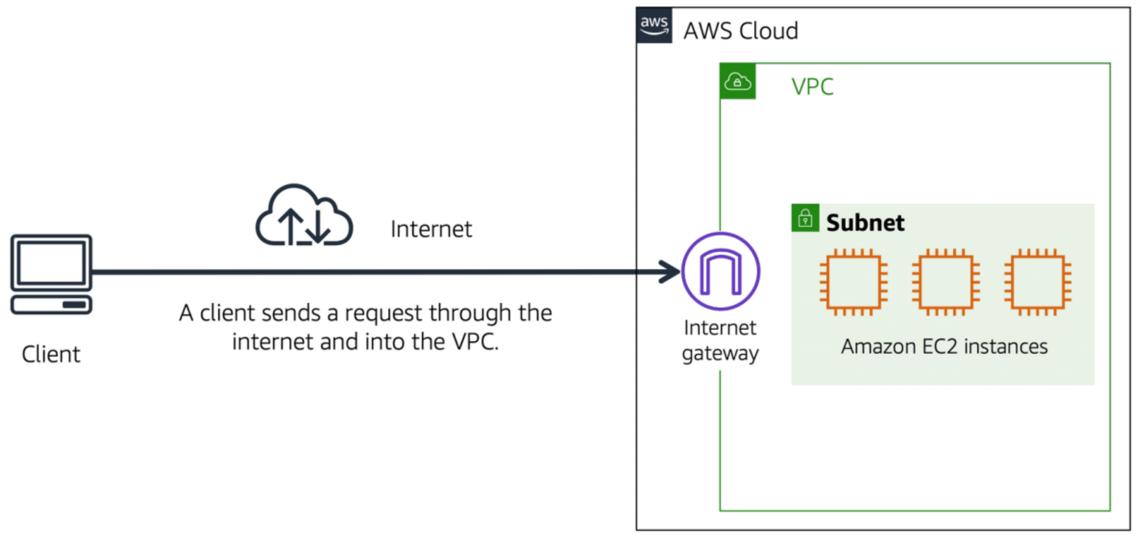
Imagine the millions of customers who use AWS services. Also, imagine the millions of resources that these customers have created, such as Amazon EC2 instances. Without boundaries around all of these resources, network traffic would be able to flow between them unrestricted.

A networking service that you can use to establish boundaries around your AWS resources is Amazon Virtual Private Cloud (Amazon VPC).

Amazon VPC enables you to provision an isolated section of the AWS Cloud. In this isolated section, you can launch resources in a virtual network that you define. Within a virtual private cloud (VPC), you can organize your resources into subnets. A subnet is a section of a VPC that can contain resources such as Amazon EC2 instances.

Internet gateway

To allow public traffic from the internet to access your VPC, you attach an internet gateway to the VPC.



An internet gateway is a connection between a VPC and the internet. You can think of an internet gateway as being similar to a doorway that customers use to enter the coffee shop. Without an internet gateway, no one can access the resources within your VPC.

What if you have a VPC that includes only private resources?

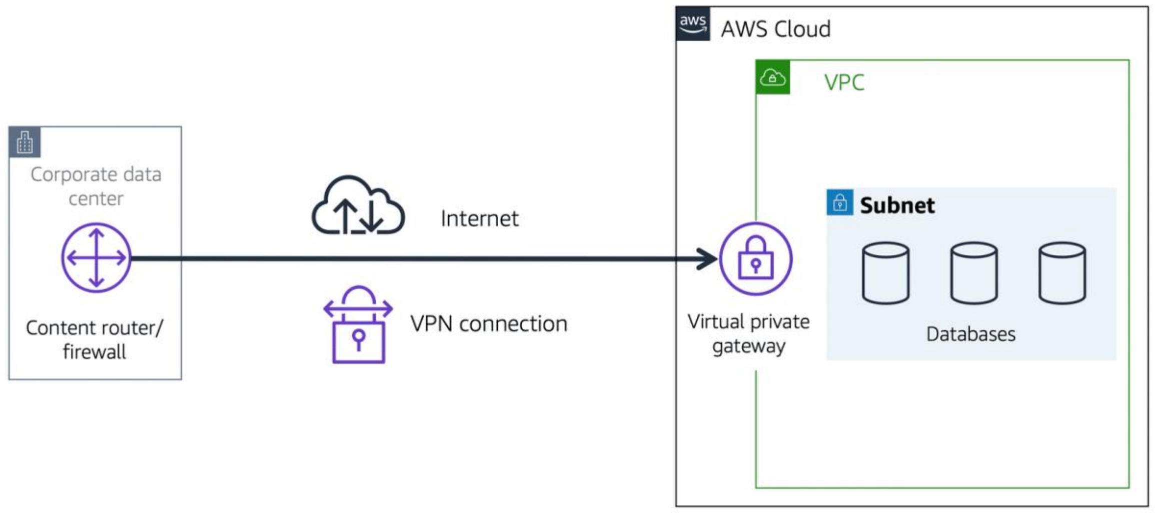
Virtual private gateway

To access private resources in a VPC, you can use a virtual private gateway.

Here’s an example of how a virtual private gateway works. You can think of the internet as the road between your home and the coffee shop. Suppose that you are traveling on this road with a bodyguard to protect you. You are still using the same road as other customers, but with an extra layer of protection.

The bodyguard is like a virtual private network (VPN) connection that encrypts (or protects) your internet traffic from all the other requests around it.

The virtual private gateway is the component that allows protected internet traffic to enter into the VPC. Even though your connection to the coffee shop has extra protection, traffic jams are possible because you’re using the same road as other customers.



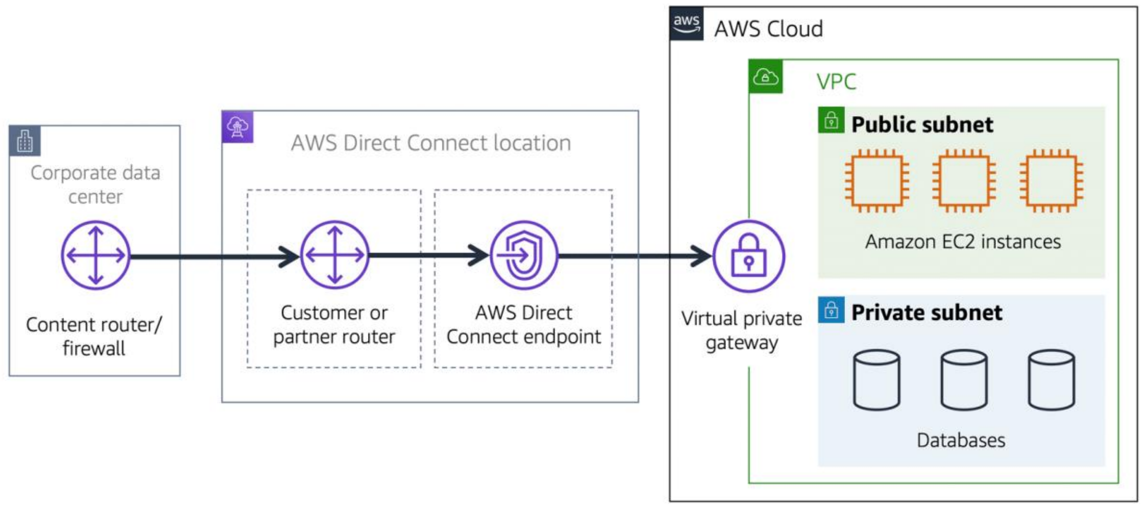
A virtual private gateway enables you to establish a virtual private network (VPN) connection between your VPC and a private network, such as an on-premises data center or internal corporate network. A virtual private gateway allows traffic into the VPC only if it is coming from an approved network.

AWS Direct Connect

AWS Direct Connect is a service that enables you to establish a dedicated private connection between your data center and a VPC.

Suppose that there is an apartment building with a hallway directly linking the building to the coffee shop. Only the residents of the apartment building can travel through this hallway.

This private hallway provides the same type of dedicated connection as AWS Direct Connect. Residents are able to get into the coffee shop without needing to use the public road shared with other customers.



The private connection that AWS Direct Connect provides helps you to reduce network costs and increase the amount of bandwidth that can travel through your network.

Subnets and network access control lists

Video transcript

Welcome to your VPC. You can think of it as a hardened fortress where nothing goes in or out without explicit permission. You have a gateway on the VPC that only permits traffic in or out of the VPC. But that only covers perimeter, and that's only one part of network security that you should be focusing on as part of your IT strategy.

AWS has a wide range of tools that cover every layer of security: network hardening, application security, user identity, authentication and authorization, distributed denial-of-service or DDoS prevention, data integrity, encryption, much more. We're gonna talk about a few of these key pieces. And if you really wanna know more about security, please follow the links in this page to be directed to more information and additional training on how to lock your infrastructure down tighter than Aunt Robin's secret lemon pie recipe, and ain't nobody getting that recipe.

Today, I wanna talk about a few aspects of network hardening looking at what happens inside the VPC. Now, the only technical reason to use subnets in a VPC is to control access to the gateways. The public subnets have access to the internet gateway; the private subnets do not. But subnets can also control traffic permissions. Packets are messages from the internet, and every packet that crosses the subnet boundaries gets checked against something called a network access control list or network ACL. This check is to see if the packet has permissions to either leave or enter the subnet based on who it was sent from and how it's trying to communicate.

You can think of network ACLs as passport control officers. If you're on the approved list, you get through. If you're not on the list, or if you're explicitly on the do-not-enter list, then you get blocked. Network ACLs check traffic going into and leaving a subnet, just like passport control. The list gets checked on your way into a country and on the way out. And just because you were let in doesn't necessarily mean they're gonna let you out. Approved traffic can be sent on its way, and potentially harmful traffic, like attempts to gain control of a system through administrative requests, they get blocked before they ever touch the target. You can't hack what you can't touch.

Now, this sounds like great security, but it doesn't answer all of the network control issues. Because a network ACL only gets to evaluate a packet if it crosses a subnet boundary, in or out. It doesn't evaluate if a packet can reach a specific EC2 instance or not. Sometimes, you'll have multiple EC2 instances in the same subnet, but they might have different rules around who can send them messages, what port those messages are allowed to be sent to. So you need instance level network security as well.

To solve instance level access questions, we introduce security groups. Every EC2 instance, when it's launched, automatically comes with a security group. And by default, the security group does not allow any traffic into the instance at all. All ports are blocked; all IP addresses sending packets are blocked. That's very secure, but perhaps not very useful. If you want an instance to actually accept traffic from the outside, like say, a message from a front end instance or a message from the Internet. So obviously, you can modify the security group to accept a specific type of traffic. In the case of a website, you want web-based traffic or HTTPS to be accepted but not other types of traffic, say an operating system or administration requests.

If NACLs are a passport control, a security group is like the doorman at your building, the building being the EC2 Instance, in this case. The doorman will check a list to ensure that someone is allowed to enter the building but won't bother check the list on the way out. With security groups, you allow specific traffic in and by default, all traffic is allowed out. Now, wait a minute, Blaine. You just described two different engines each doing the exact same job. Let good packets in, keep bad packets out. The key difference between a security group and a network ACL is the security group is stateful, meaning, as we talked about, it has some kind of a memory when it comes to who to allow in or out, and the network ACL is stateless, which remembers nothing and checks every single packet that crosses its border regardless of any circumstances.

You know, this metaphor is important to understand. So I wanna illustrate the round trip of a packet as it goes from one instance to another instance in a different subnet. Now, this traffic management, it doesn't care about the contents of the packet itself. In fact, it doesn't even open the envelope, it can't. All it can do is check to see if the sender is on the approved list.

All right. Let's start with instance A. We wanna send a packet from instance A to instance B in a different subnet, same VPC, different subnets. So instance A sends the packet. Now, the first thing that happens is that packet meets the boundary of the security group of instance A. By default, all outbound traffic is allowed from a security group. So you can walk right by the doormen and leave, cool, right. The packet made it past the security group of instance A. Now it has to leave the subnet boundary. At the boundary, the passport must now make it through passport control, the network ACL. The network ACL doesn't care about what the security group allowed. It has its own list of who can pass and who can't. If the target address is allowed, you can keep going on your journey, which it is.

So now we have exited the original subnet and now the packet goes to the target subnet where instance B lives. Now at this target subnet, once again, we have passport control. Just because the packet was allowed out of its home country does not mean that it can enter the destination country or subnet in this case. They both have unique passport officers with their own checklists. You have to be approved on both lists, or you could get turned away at the border. Well, turns out the packet is on the approved list for this subnet, so it's allowed to enter through the network ACL into the subnet. Almost there. Now, we're approaching the target instance, instance B. Every EC2 Instance has their own security group. You wanna enter this instance, the doorman will need to check to see if you're allowed in, and we're on the list. The packet has reached the target instance.

Once the transaction's complete, now it's just time to come home. It's the return traffic pattern. It's the most interesting, because this is where the stateful versus stateless nature of the different engines come into play. Because the packet still has to be evaluated at each checkpoint. Security groups, by default, allow all return traffic. So they don't have to check a list to see if they're allowed out. Instead, they automatically allow the return traffic to pass by no matter what. Passport control here at the subnet boundary, these network ACLs do not remember state. They don't care that the packet came in here. It might be on a you-can't-leave list. Every ingress and egress is checked with the appropriate list. The package return address has to be on their approved list to make it across the border. Made it to the border of the origin subnet, but we have to negotiate passport network ACL control here as well. Stateless controls, means it always checks its list.

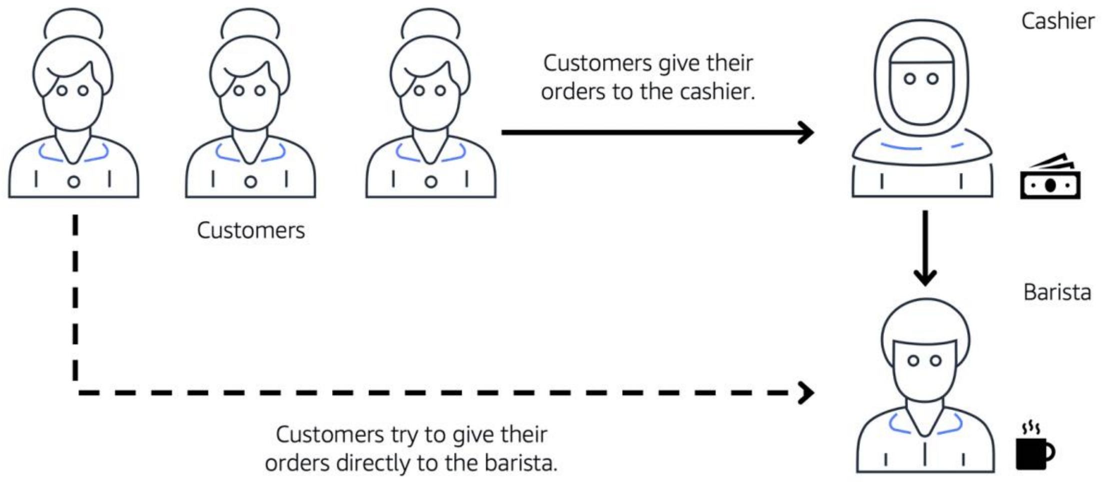
The packet pass the network ACL, the subnet level, which means the packets now made it back to instance A but the security group, the doorman is still in charge here. The key difference though is these security groups are stateful. The security group recognizes the packet from before. So it doesn't need to check to see if it's allowed in. Come on home.

Now, this may seem like we spent a lot of effort just getting a packet from one instance to another and back. You might be concerned about all the network overhead this might generate. The reality is all of these exchanges happen instantly as part of how AWS Networking actually works. If you wanna know the technology that makes all that possible, well, then you need to sign up for a completely different set of trainings. Good network security will take advantage of both network ACLs and security groups, because security in-depth is critical for modern architectures.

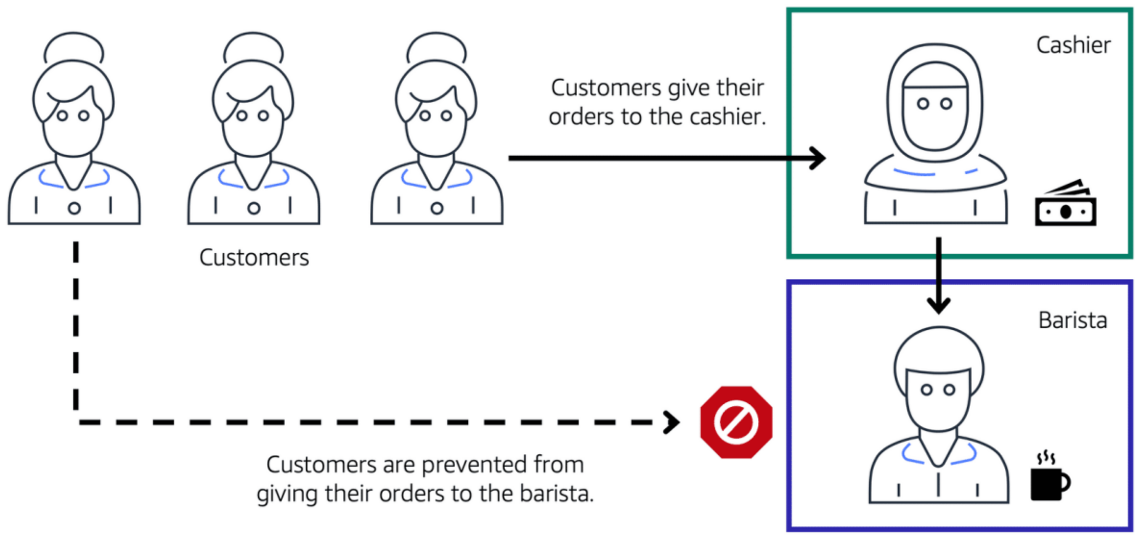
To learn more about the role of subnets within a VPC, review the following example from the coffee shop.

First, customers give their orders to the cashier. The cashier then passes the orders to the barista. This process allows the line to keep running smoothly as more customers come in.

Suppose that some customers try to skip the cashier line and give their orders directly to the barista. This disrupts the flow of traffic and results in customers accessing a part of the coffee shop that is restricted to them.


Cashier
Customers give their orders to the cashier.
Customers
Barista
Customers try to give their orders directly to the barista.

To fix this, the owners of the coffee shop divide the counter area by placing the cashier and the barista in separate workstations. The cashier’s workstation is public facing and designed to receive customers. The barista’s area is private. The barista can still receive orders from the cashier but not directly from customers.

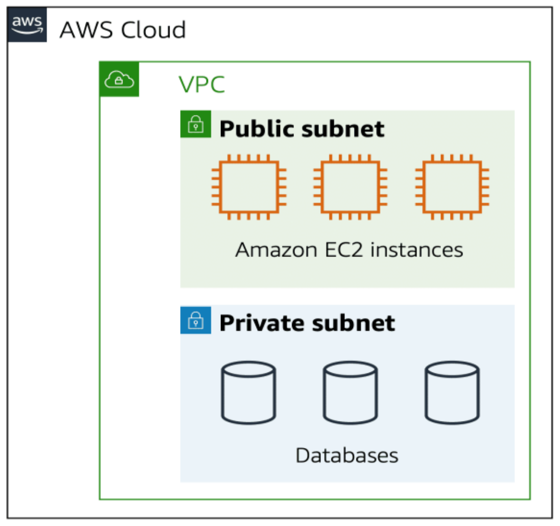

Customers give their orders to the cashier.
Cashier
Customers
Barista
Customers are prevented from giving their orders to the barista.

This is similar to how you can use AWS networking services to isolate resources and determine exactly how network traffic flows.

In the coffee shop, you can think of the counter area as a VPC. The counter area divides into two separate areas for the cashier's workstation and the barista's workstation. In a VPC, subnets are separate areas that are used to group together resources.

Subnets

A subnet is a section of a VPC in which you can group resources based on security or operational needs. Subnets can be public or private.

 AWS Cloud VPC Public subnet Amazon EC2 instances Private subnet Databases	**Public subnets** contain resources that need to be accessible by the public, such as an online store's website. **Private subnets** contain resources that should be accessible only through your private network, such as a database that contains customers' personal information and order histories. In a VPC, subnets can communicate with each other. For example, you might have an application that involves Amazon EC2 instances in a public subnet communicating with databases that are located in a private subnet.

Network traffic in a VPC

When a customer requests data from an application hosted in the AWS Cloud, this request is sent as a packet. A packet is a unit of data sent over the internet or a network.

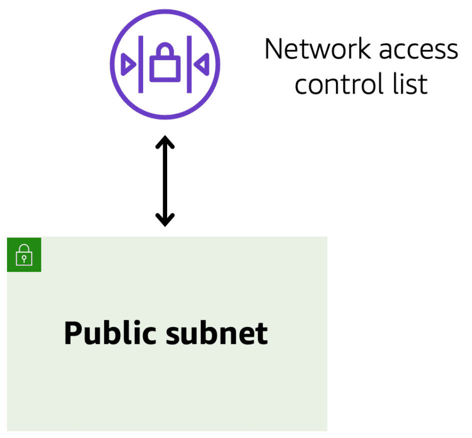
It enters into a VPC through an internet gateway. Before a packet can enter into a subnet or exit from a subnet, it checks for permissions. These permissions indicate who sent the packet and how the packet is trying to communicate with the resources in a subnet.

The VPC component that checks packet permissions for subnets is a network access control list (ACL).

Network access control lists (ACLs)

A network access control list (ACL) is a virtual firewall that controls inbound and outbound traffic at the subnet level.

For example, step outside of the coffee shop and imagine that you are in an airport. In the airport, travelers are trying to enter into a different country. You can think of the travelers as packets and the passport control officer as a network ACL. The passport control officer checks travelers' credentials when they are both entering and exiting out of the country. If a traveler is on an approved list, they are able to get through. However, if they are not on the approved list or are explicitly on a list of banned travelers, they cannot come in.



Each AWS account includes a default network ACL. When configuring your VPC, you can use your account's default network ACL or create custom network ACLs.

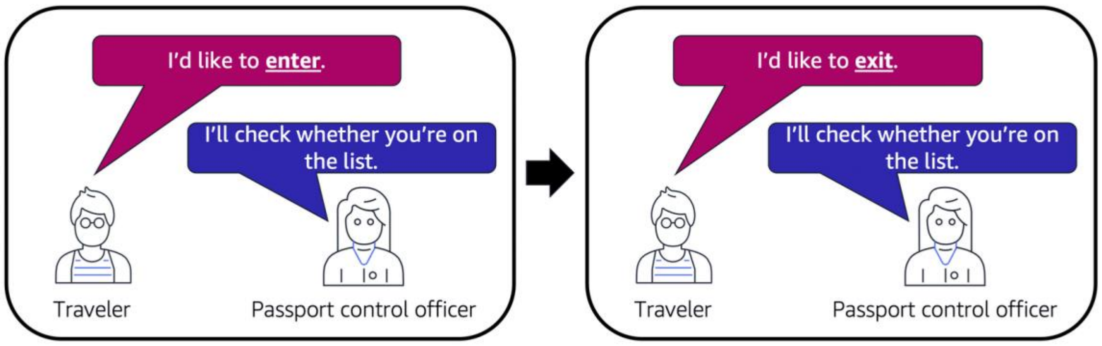
By default, your account's default network ACL allows all inbound and outbound traffic, but you can modify it by adding your own rules. For custom network ACLs, all inbound and outbound traffic is denied until you add rules to specify which traffic to allow. Additionally, all network ACLs have an explicit deny rule. This rule ensures that if a packet doesn't match any of the other rules on the list, the packet is denied.

Stateless packet filtering

Network ACLs perform stateless packet filtering. They remember nothing and check packets that cross the subnet border each way: inbound and outbound.

Recall the previous example of a traveler who wants to enter into a different country. This is similar to sending a request out from an Amazon EC2 instance and to the internet.

When a packet response for that request comes back to the subnet, the network ACL does not remember your previous request. The network ACL checks the packet response against its list of rules to determine whether to allow or deny.

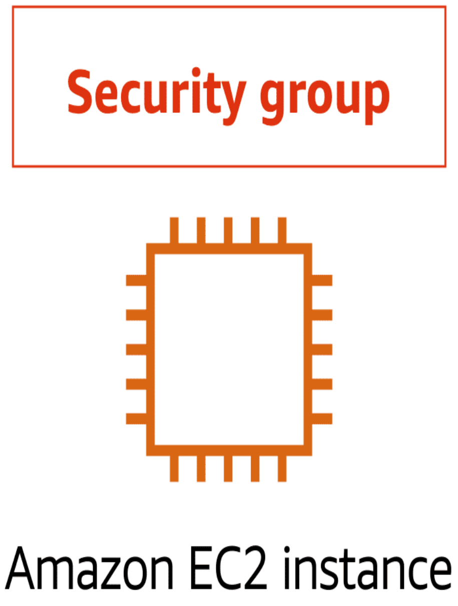


After a packet has entered a subnet, it must have its permissions evaluated for resources within the subnet, such as Amazon EC2 instances.

The VPC component that checks packet permissions for an Amazon EC2 instance is a security group.

Security groups

A security group is a virtual firewall that controls inbound and outbound traffic for an Amazon EC2 instance.

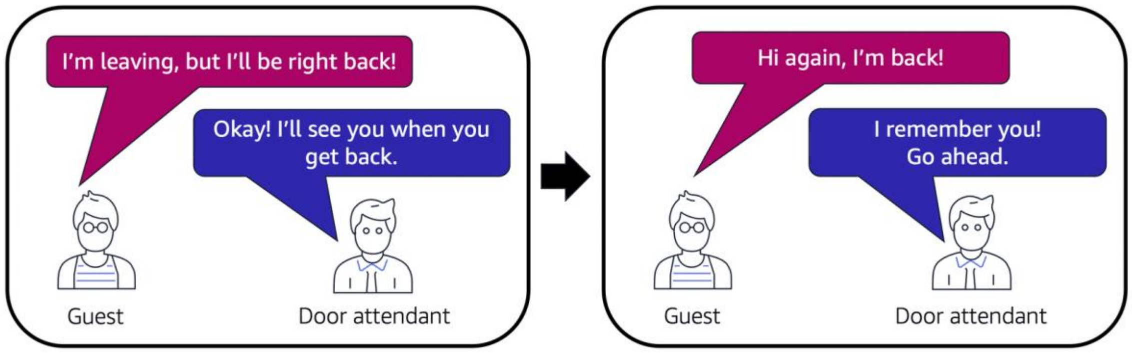
	By default, a security group denies all inbound traffic and allows all outbound traffic. You can add custom rules to configure which traffic to allow or deny. For this example, suppose that you are in an apartment building with a door attendant who greets guests in the lobby. You can think of the guests as packets and the door attendant as a security group. As guests arrive, the door attendant checks a list to ensure they can enter the building. However, the door attendant does not check the list again when guests are exiting the building If you have multiple Amazon EC2 instances within a subnet, you can associate them with the same security group or use different security groups for each instance.

Stateful packet filtering

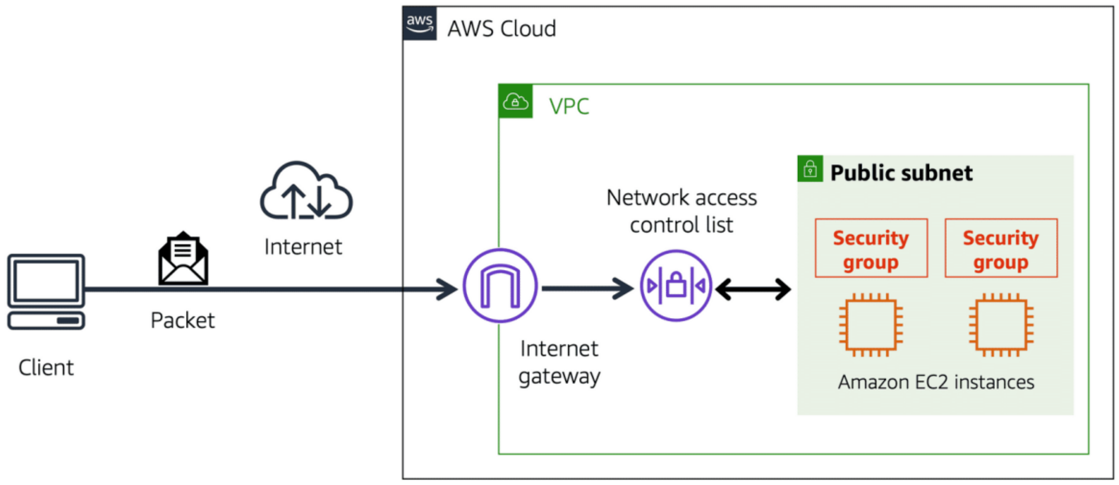
Security groups perform stateful packet filtering. They remember previous decisions made for incoming packets.

Consider the same example of sending a request out from an Amazon EC2 instance to the internet.

When a packet response for that request returns to the instance, the security group remembers your previous request. The security group allows the response to proceed, regardless of inbound security group rules.


I'm leaving, but I'll be right back!
Okay! I'll see you when you get back.
Guest
Door attendant
Hi again, I'm back!
I remember you! Go ahead.
Guest
Door attendant

Both network ACLs and security groups enable you to configure custom rules for the traffic in your VPC. As you continue to learn more about AWS security and networking, make sure to understand the differences between network ACLs and security groups.



Global networking

Video transcript

We've been talking a lot about how you interact with your AWS infrastructure. But how do your customers interact with your AWS infrastructure? Well, if you have a website hosted at AWS, then customers usually enter your website into their browser, hit Enter, some magic happens, and the site opens up.

But how does this magic work? Well, just like this magic coin that I have here, you know, you take a bite, and it's, it's back. Well, not quite like that, but I'm going to take you through two services, which would help in the website case. The first one being Route 53. Route 53 is AWS's domain name service, or DNS, and it's highly available and scalable. But wait, what is DNS, you say? Think of DNS as a translation service. But instead of translating between languages, it translates website names into IP, or Internet Protocol, addresses that computers can read.

For example, when we enter a website address into our browser, it contacts Route 53 to obtain the IP address of the site, say, 192.1.1.1, then it routes your computer or browser to that address. If we go further, Route 53 can direct traffic to different endpoints using several different routing policies, such as latency-based routing, geolocation DNS, geoproximity, and weighted round robin. If we take geolocation DNS, that means we direct traffic based on where the customer is located. So traffic coming from say North America is routed to the Oregon Region, and traffic in Ireland is routed to the Dublin Region, as an example.

You can even use Route 53 to register domain names, so you can buy and manage your own domain names right on AWS.

Speaking of websites, there is another service which can help speed up delivery of website assets to customers, Amazon CloudFront. If you remember, we talked about Edge locations earlier in the course, these locations are serving content as close to customers as possible, and one part of that, is the content delivery network, or CDN. For those who need reminding, a CDN is a network that helps to deliver edge content to users based on their geographic location.

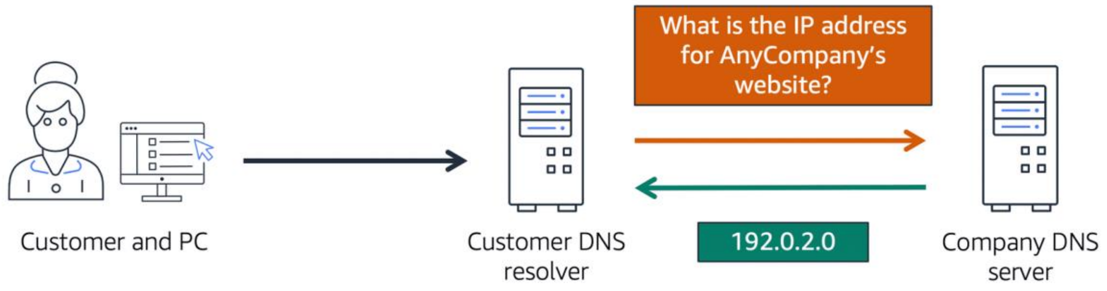
If we go back to our North America versus Ireland example, say we have a user in Seattle, and they want to access a website, to speed this up, we host the site in Oregon, and we deploy our static web assets, like images and GIFs in CloudFront in North America. This means they get content delivered as close to them as possible, North America in this case, when they access the site. But for our Irish users, it doesn't make sense to deliver those assets out of Oregon, as the latency is not favorable. Thus we deploy those same static assets in CloudFront, but this time in the Dublin Region. That means they can access the same content, but from a location closer to them, which in turn improves latency.

I hope you're content after learning about these two services. Thanks for following along, and I'm going to disappear just like this red cloth. Tada!

Domain Name System (DNS)

Suppose that AnyCompany has a website hosted in the AWS Cloud. Customers enter the web address into their browser, and they are able to access the website. This happens because of Domain Name System (DNS) resolution. DNS resolution involves a customer DNS resolver communicating with a company DNS server.

You can think of DNS as being the phone book of the internet. DNS resolution is the process of translating a domain name to an IP address.



For example, suppose that you want to visit AnyCompany's website.

1. When you enter the domain name into your browser, this request is sent to a customer DNS resolver.
2. The customer DNS resolver asks the company DNS server for the IP address that corresponds to AnyCompany's website.
3. The company DNS server responds by providing the IP address for AnyCompany's website, 192.0.2.0.

Amazon Route 53

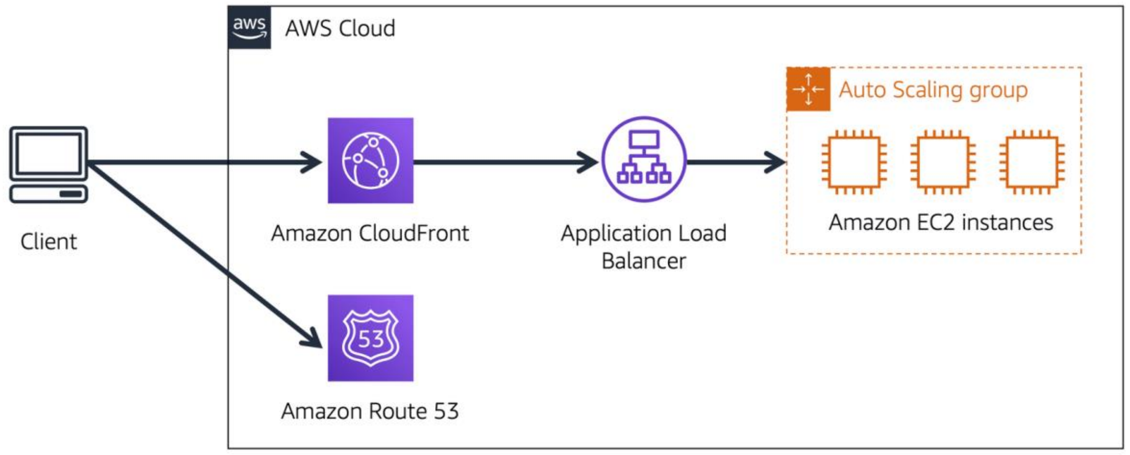
Amazon Route 53 is a DNS web service. It gives developers and businesses a reliable way to route end users to internet applications hosted in AWS.

Amazon Route 53 connects user requests to infrastructure running in AWS (such as Amazon EC2 instances and load balancers). It can route users to infrastructure outside of AWS.

Another feature of Route 53 is the ability to manage the DNS records for domain names. You can register new domain names directly in Route 53. You can also transfer DNS records for existing domain names managed by other domain registrars. This enables you to manage all of your domain names within a single location.

In the previous module, you learned about Amazon CloudFront, a content delivery service. The following example describes how Route 53 and Amazon CloudFront work together to deliver content to customers.

Example: How Amazon Route 53 and Amazon CloudFront deliver content



Suppose that AnyCompany's application is running on several Amazon EC2 instances. These instances are in an Auto Scaling group that attaches to an Application Load Balancer.

1. A customer requests data from the application by going to AnyCompany's website.
2. Amazon Route 53 uses DNS resolution to identify AnyCompany.com's corresponding IP address, 192.0.2.0. This information is sent back to the customer.
3. The customer's request is sent to the nearest edge location through Amazon CloudFront.
4. Amazon CloudFront connects to the Application Load Balancer, which sends the incoming packet to an Amazon EC2 instance.
