AWS User Cases



AWS Region: cluster of data centers. AWS services are regoin-scoped

Availability Zones: An Availability Zone consists of one or more data centers with redundant power, networking, and connectivity. These data centers operate in discrete facilities in undisclosed locations. They are connected using redundant high-speed and low-latency links.



How to choose the regions in AWS?

1. Complaince

If company demands to keep data in certain place then choose region within those place no other factors matter.

Enterprise companies often must comply with regulations that require customer data to be stored in a specific geographic territory. If applicable, choose a Region that meets your compliance requirements.

1. Latency :  (the delay between a request for data and the response)

If your application is sensitive to latency (the delay between a request for data and the response), choose a Region that is close to your user base. This helps prevent long wait times for your customers. Synchronous applications such as gaming, telephony, WebSockets, and Internet of Things (IoT) are significantly affected by high latency. Asynchronous workloads, such as ecommerce applications, can also suffer from user connectivity delays.Pricing

1. Pricing

Pricing can vary from region to region. Some regions can be expensive due to different taxation.

Instead of charging a flat rate worldwide, AWS charges based on the financial factors specific to each Region.

1. Service Availability

Some services might not be available in some Regions

Any new service is lauched it is not available at all regions at 1st day.

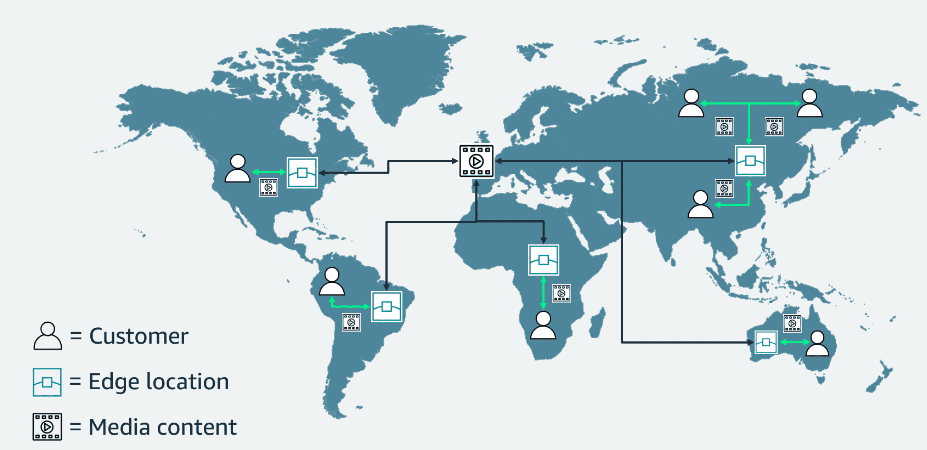
**Maintaining resiliency**

To keep your application available, you must maintain high availability and resiliency. A well-known best practice for cloud architecture is to use Region-scoped, managed services. These services come with availability and resiliency built in. When that is not possible, make sure your workload is replicated across multiple Availability Zones. At a minimum, you should use two Availability Zones. That way, if an Availability Zone fails, your application will have infrastructure up and running in a second Availability Zone to take over the traffic.

Edge Locations

Edge locations are global locations where content is cached. For example, if your media content is in London and you want to share video files with your customers in Sydney, you could have the videos cached in an edge location closest to Sydney. This would make it possible for your customers to access the cached videos more quickly than accessing them from London. Currently, there are over 400+ edge locations globally.

Amazon CloudFront delivers your content through a worldwide network of edge locations. When a user requests content that is being served with CloudFront, the request is routed to the location that provides the lowest latency. So that content is delivered with the best possible performance. CloudFront speeds up the distribution of your content by routing each user request through the AWS backbone network to the edge location that can best serve your content.



**Interacting with AWS**

**AWS Management Console**: Manage through web based console.

**AWS CLI:**

Consider the scenario where you run many servers on AWS for your application’s frontend. You want to run a report to collect data from all the servers. You need to do this programmatically every day because the server details might change. Instead of manually logging in to the console and then copying and pasting information, you can schedule an AWS CLI script with an API call to pull this data for you.

For example, you run the following API call against a service, using the AWS CLI:  
aws s3api list-buckets  
You will get a response similar to the following one, listing the buckets in your AWS accounts

**AWS SDK**

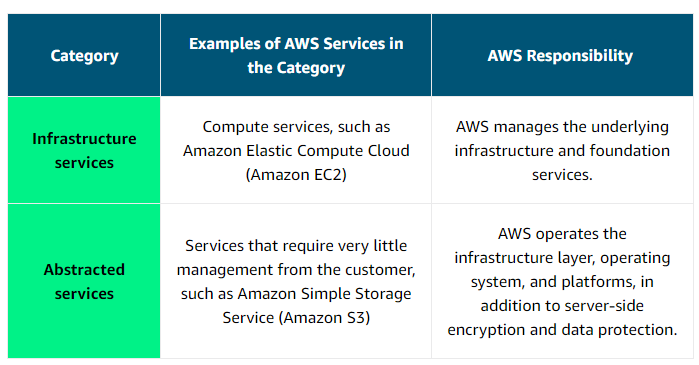
API calls to AWS can also be performed by running code with programming languages. You can do this by using AWS SDKs. SDKs are open source and maintained by AWS for the most popular programming languages, such as C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python, Ruby, Rust, and Swift.

**Security**

**AWS Roles:**

AWS is responsible for security of the cloud. This means that AWS protects and secures the infrastructure that runs the services offered in the AWS Cloud. AWS is responsible for the following:

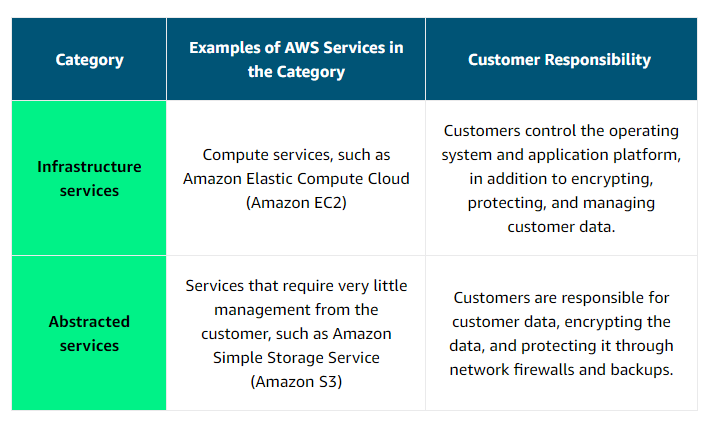
* Protecting and securing AWS Regions, Availability Zones, and data centers, down to the physical security of the buildings
* Managing the hardware, software, and networking components that run AWS services, such as the physical servers, host operating systems, virtualization layers, and AWS networking components

****

**Customer Responsibilty**

A key concept is that customers maintain complete control of their data and are responsible for managing the security related to their content. For example, you are responsible for the following:

* Choosing a Region for AWS resources in accordance with data sovereignty regulations
* Implementing data-protection mechanisms, such as encryption and scheduled backups
* Using access control to limit who can access your data and AWS resources



**Access Management**

Authentication ensures that the user is who they say they are. User names and passwords are the most common types of authentication.

Authorization is the process of giving users permission to access AWS resources and services. Authorization determines whether a user can perform certain actions, such as read, edit, delete, or create resources.

**IAM**

AWS Identity and Access Management (IAM) is an AWS service that helps you manage access to your AWS account and resources.

With IAM, you can share access to an AWS account and resources without sharing your set of access keys or password. You can also provide granular access to those working in your account, so people and services only have permissions to the resources that they need. For example, to provide a user of your AWS account with read-only access to a particular AWS service, you can granularly select which actions and which resources in that service that they can access.

arn:aws:iam::922354750072:<>/<>

**AMAZON EC2**

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. With this service, you can provision virtual servers called EC2 instances.

With Amazon EC2, you can do the following:

* Provision and launch one or more EC2 instances in minutes.
* Stop or shut down EC2 instances when you finish running a workload.
* Pay by the hour or second for each instance type (minimum of 60 seconds).

 You can create and manage EC2 instances through the AWS Management Console, AWS CLI, AWS SDKs, automation tools, and infrastructure orchestration services.

To create an EC2 instance, you must define the following:

**Hardware specifications:**CPU, memory, network, and storage

**Logical configurations:**Networking location, firewall rules, authentication, and the operating system of your choice

**AMAZON MACHINE IMAGE**

An AMI is a template that contains the software configuration(operating systems, application servers, and applications) required to launch your instance.

When launching an EC2 instance, the first setting you configure is which operating system you want by selecting an Amazon Machine Image (AMI).

An AMI includes the operating system, storage mapping, architecture type, launch permissions, and any additional preinstalled software applications.

**AMI & EC2 Relation**

EC2 instances are live instantiations (or versions) of what is defined in an AMI.

If you are familiar with software development, you can also see this kind of relationship between a class and an object.

The AMI is how you model and define your instance. The EC2 instance is the entity you interact with, where you can install your web server and serve your content to users.

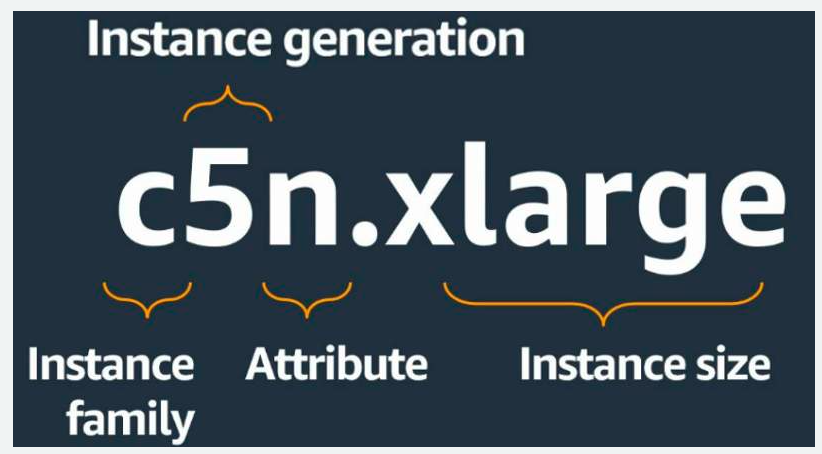
When you launch a new instance, AWS allocates a virtual machine that runs on a hypervisor. Then the AMI that you selected is copied to the root device volume, which contains the image that is used to boot the volume. In the end, you get a server that you can connect to and install packages and additional software on.

One advantage of using AMIs is that they are reusable.

You could create an AMI from your running instance and use the AMI to start a new instance.

**AWS Instance Types**

EC2 instances are a combination of virtual processors (vCPUs), memory, network, and, in some cases, instance storage and graphics processing units (GPUs).

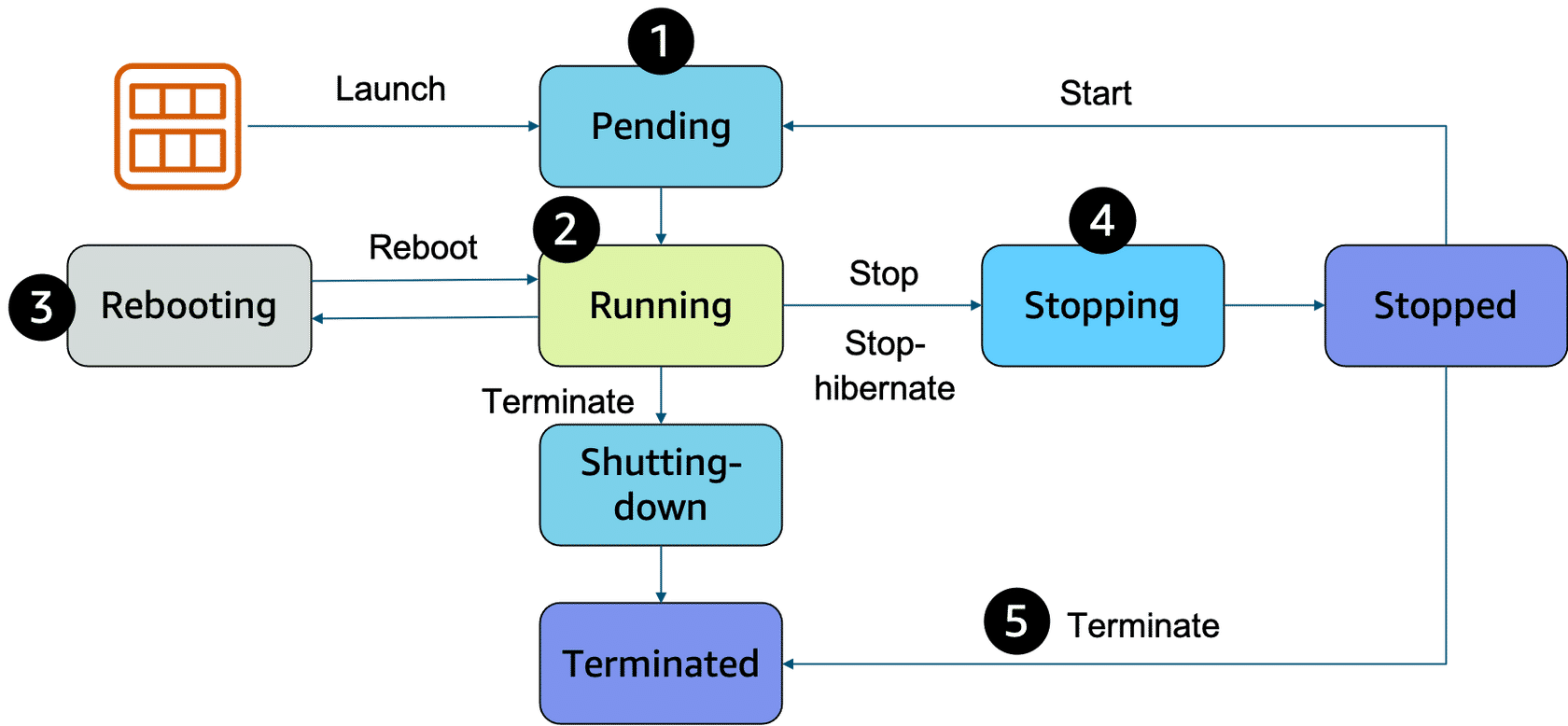
****

**EC2 Instances Locations**

Unless otherwise specified, when you launch EC2 instances, they are placed in a default virtual private cloud (VPC). The default VPC is suitable for getting started quickly and launching public EC2 instances without having to create and configure your own VPC.

Any resource that you put inside the default VPC will be public and accessible by the internet, so you shouldn’t place any customer data or private information in it.

**Life Cycle of EC2 Instances**



1. When you launch an instance, it enters the **pending**state. When an instance is pending, billing has not started. At this stage, the instance is preparing to enter the running state. Pending is where AWS performs all actions needed to set up an instance, such as copying the AMI content to the root device and allocating the necessary networking components.
2. When your instance is **running**, it's ready to use. This is also the stage where billing begins. As soon as an instance is running, you can take other actions on the instance, such as reboot, terminate, stop, and stop-hibernate.
3. When you reboot an instance, it’s different than performing a stop action and then a start action. **Rebooting**an instance is equivalent to rebooting an operating system. The instance keeps its public DNS name (IPv4) and private and public IPv4 addresses. An IPv6 address (if applicable) remains on the same host computer and maintains its public and private IP address, in addition to any data on its instance store volumes.
4. When you stop your instance, it enters the **stopping** and then **stopped** state. This is similar to when you shut down your laptop. You can stop and start an instance if it has an Amazon Elastic Block Store (Amazon EBS) volume as its root device. When you stop and start an instance, your instance can be placed on a new underlying physical server. Your instance retains its private IPv4 addresses and if your instance has an IPv6 address, it retains its IPv6 address. When you put the instance into stop-hibernate, the instance enters the stopped state, but saves the last information or content into memory, so that the start process is faster.
5. When you **terminate**an instance, the instance stores are erased, and you lose both the public IP address and private IP address of the machine. Termination of an instance means that you can no longer access the machine. As soon as the status of an instance changes to **shutting down**or **terminated,**you stop incurring charges for that instance.

**Stop v/s Stop-Hibernate**

When you stop an instance, it enters the stopping state until it reaches the stopped state. AWS does not charge usage or data transfer fees for your instance after you stop it. But storage for any Amazon EBS volumes is still charged. While your instance is in the stopped state, you can modify some attributes, like the instance type. When you stop your instance, the data from the instance memory (RAM) is lost.

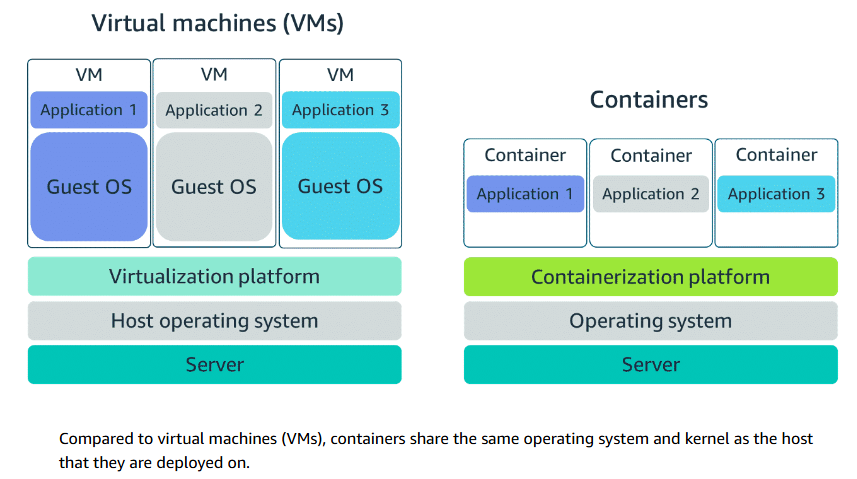
When you stop-hibernate an instance, Amazon EC2 signals the operating system to perform hibernation (suspend-to-disk), which saves the contents from the instance memory (RAM) to the EBS root volume. You can hibernate an instance only if hibernation is turned on and the instance meets the hibernation prerequisites.

**Container**

A container is a standardized unit that packages your code and its dependencies. This package is designed to run reliably on any platform, because the container creates its own independent environment. With containers, workloads can be carried from one place to another, such as from development to production or from on-premises environments to the cloud.

An example of a containerization platform is Docker. Docker is a popular container runtime that simplifies the management of the entire operating system stack required for container isolation, including networking and storage. Docker helps customers create, package, deploy, and run containers.

**Difference VM v/s Containers**



Containers share the same operating system and kernel as the host that they exist on. But virtual machines contain their own operating system. Each virtual machine must maintain a copy of an operating system, which results in a degree of wasted resources.

When we say that **containers are lightweight**, we mean that they are efficient in terms of resource usage and performance.

1. **Minimal Resource Usage**: Containers share the host system's operating system kernel, which means they don't need to include an entire OS. This reduces their size and resource consumption compared to virtual machines
2. **Quick Startup and Shutdown**: Containers can start up and shut down almost instantly because they don't have the overhead of booting up a full operating system.
3. **Efficient Use of Resources**: Since containers use the host's resources directly, they are more efficient in using CPU, memory, and storage. Multiple containers can run on a single host without significant performance degradation.
4. **Portability and Consistency**: Containers package the application along with its dependencies. This ensures that the application runs consistently across different environments, reducing compatibility issues.
5. **Isolation**: Containers provide process and file system isolation, allowing multiple applications to run on the same host without interfering with each other.

Containers can provide speed, but virtual machines offer the full strength of an operating system and more resources, like package installation, dedicated kernel, and more.

**Orchestrating Containers**

**Container orchestration** is the process of automating the deployment, management, scaling, and networking of containers. It helps streamline the lifecycle of containerized applications, making it easier to manage complex container architectures

**Provisioning and Deployment**: Automatically provisions and deploys containers across different environments.

**Scaling**: Manages the scaling of containers up or down based on demand.

**Networking**: Handles the networking between containers to ensure they can communicate effectively.

**Health Monitoring**: Monitors the health of containers and restarts them if they fail.

**Load Balancing**: Distributes workloads evenly across containers to optimize resource use.

AWS offers two container orchestration services: Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS).

**Managing Containers with Amazon ECS**

Amazon ECS is an end-to-end container orchestration service that helps you spin up new containers.

(To **spin up a container** means to create and start a new instance of a containerized application)

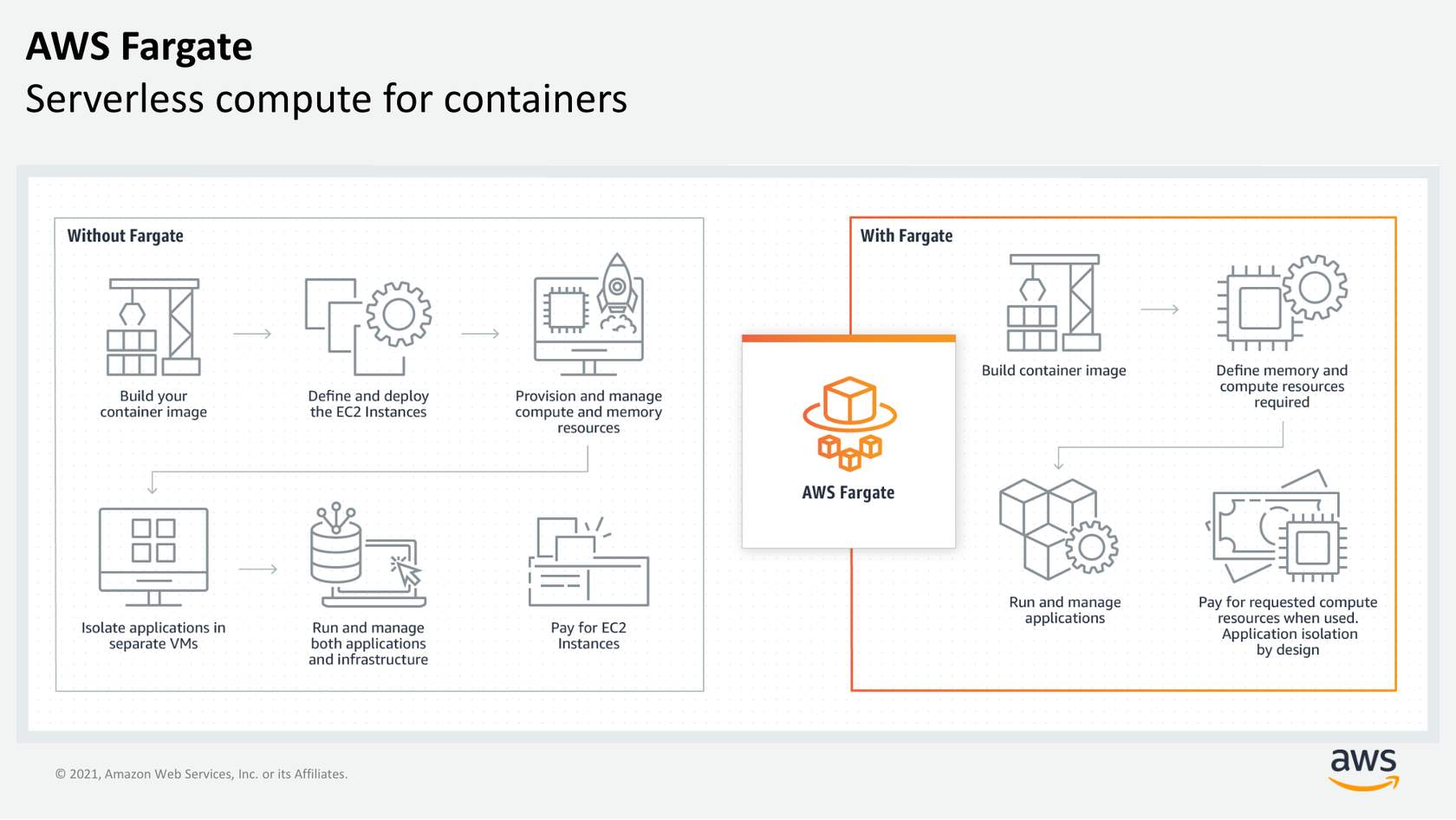
With Amazon ECS, your containers are defined in a task definition that you use to run an individual task or a task within a service.

To run your tasks and services on a serverless infrastructure that's managed by another AWS service called AWS Fargate.

Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of EC2 instances that you manage.

**Serverless with AWS FARGET**

AWS Fargate is a purpose-built serverless compute engine for containers. AWS Fargate scales and manages the infrastructure, so developers can work on what they do best, application development. It achieves this by allocating the right amount of compute. This eliminates the need to choose and manage EC2 instances, cluster capacity, and scaling. Fargate supports both Amazon ECS and Amazon EKS architecture and provides workload isolation and improved security by design.



**AWS Lambda**

**AWS Lambda** is a serverless compute service provided by Amazon Web Services (AWS). It allows you to run code without needing to provision or manage servers. Here are some key features and benefits of AWS Lambda:

1. **Serverless**: You don't need to manage servers. AWS Lambda handles the infrastructure, including server and operating system maintenance, capacity provisioning, and automatic scaling2.
2. **Event-Driven**: AWS Lambda runs your code in response to events, such as changes to data in an S3 bucket, updates to a DynamoDB table, or HTTP requests via Amazon API Gateway1.
3. **Scalability**: Automatically scales up or down based on the number of events. You only pay for the compute time you consume, with no charge when your code is not running2.
4. **Cost-Effective**: Pay-as-you-go pricing model. You are charged based on the number of requests for your functions and the time your code executes.
5. **High Availability**: AWS Lambda runs your code on a high-availability compute infrastructure, ensuring reliability and performance.
6. **Use Cases**: Ideal for various applications, including file processing, stream processing, web applications, IoT backends, and mobile backends.

Triggers describe when a Lambda function should run. A trigger integrates your Lambda function with other AWS services(s3 bucket, cloudwatch, etc) and event source mappings. So you can run your Lambda function in response to certain API calls or by reading items from a stream or queue. This increases your ability to respond to events in your console without having to perform manual actions.

**Amazon VPC**

A virtual private cloud (VPC) is an isolated network that you create in the AWS Cloud, similar to a traditional network in a data center.

**Amazon VPC Route**

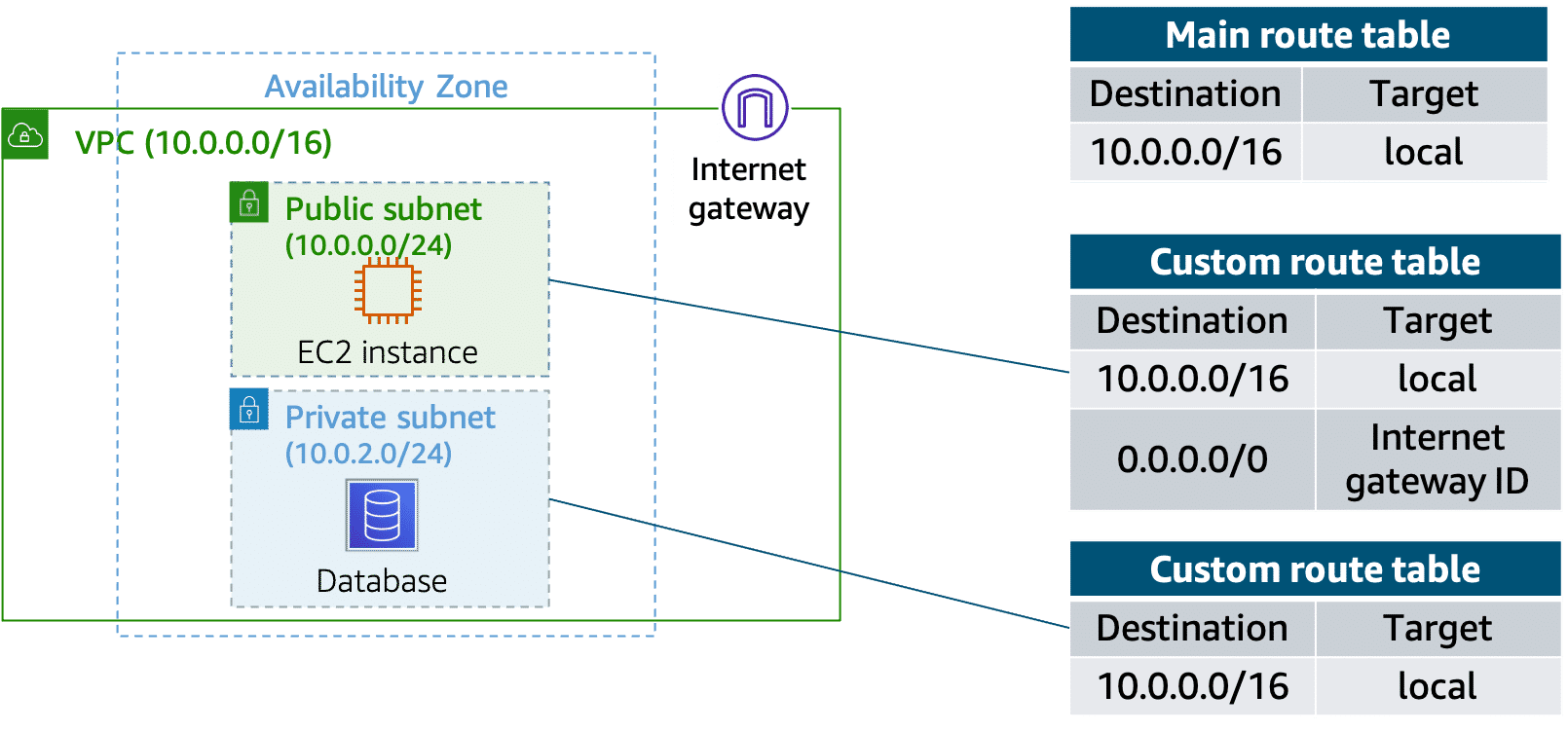
**Main route table**

When you create a VPC, AWS creates a route table called the main route table. A route table contains a set of rules, called routes, that are used to determine where network traffic is directed. AWS assumes that when you create a new VPC with subnets, you want traffic to flow between them. Therefore, the default configuration of the main route table is to allow traffic between all subnets in the local network. The following rules apply to the main route table:

* You cannot delete the main route table.
* You cannot set a gateway route table as the main route table.
* You can replace the main route table with a custom subnet route table.
* You can add, remove, and modify routes in the main route table.
* You can explicitly associate a subnet with the main route table, even if it's already implicitly associated.

**Custom route tables**

The main route table is used implicitly by subnets that do not have an explicit route table association. However, you might want to provide different routes on a per-subnet basis for traffic to access resources outside of the VPC. For example, your application might consist of a frontend and a database. You can create separate subnets for the resources and provide different routes for each of them.  
  
If you associate a subnet with a custom route table, the subnet will use it instead of the main route table. Each custom route table that you create will have the local route already inside it, allowing communication to flow between all resources and subnets inside the VPC. You can protect your VPC by explicitly associating each new subnet with a custom route table and leaving the main route table in its original default state.

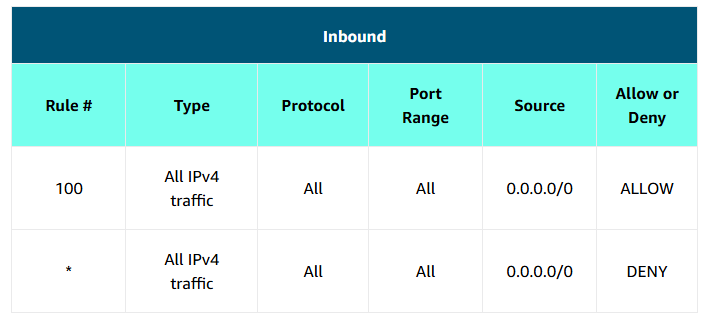


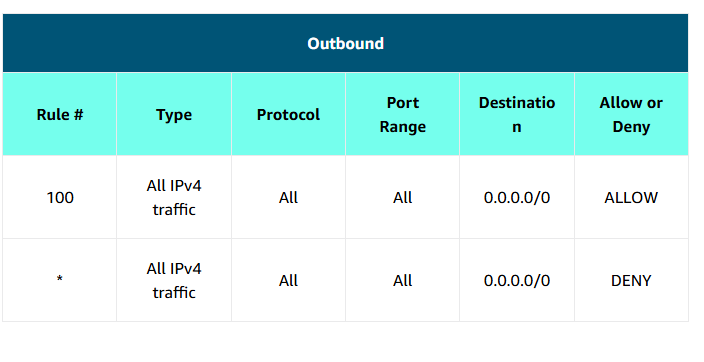
**Internet Traffic**

**Internet traffic** refers to the amount of data being transferred over the internet at any given time. This data includes everything sent and received across the internet, such as emails, website visits, file downloads, streaming videos, social media interactions, and more. Here are some key aspects of internet traffic:

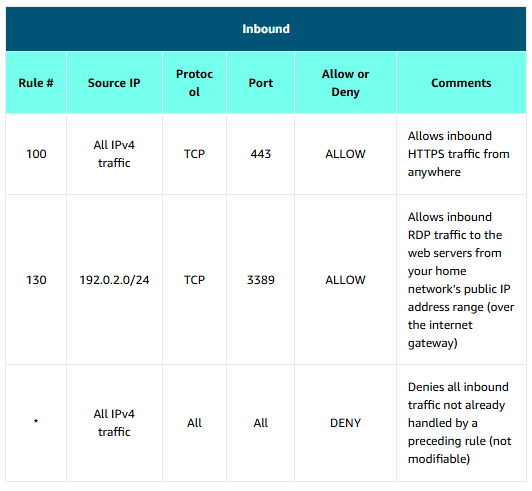
1. **Data Transfer**: Internet traffic encompasses the volume of data moving between devices, such as computers, smartphones, and servers, over the internet.
2. **Types of Traffic**:
   * **HTTP/HTTPS Traffic**: Generated by web browsing and accessing web applications.
   * **Email Traffic**: Includes sending and receiving emails.
   * **Streaming Traffic**: Includes video and audio streaming services like YouTube, Netflix, and Spotify.
   * **File Transfers**: Data moved through protocols like FTP (File Transfer Protocol) or P2P (Peer-to-Peer) networks.
   * **Social Media Traffic**: Data from interactions on social media platforms like Facebook, Twitter, and Instagram.
3. **Traffic Measurement**: Internet traffic is often measured in terms of data volume (gigabytes, terabytes) or data rate (megabits per second, gigabits per second).
4. **Traffic Analysis**: Analyzing internet traffic helps understand usage patterns, identify bottlenecks, enhance security, and optimize network performance.

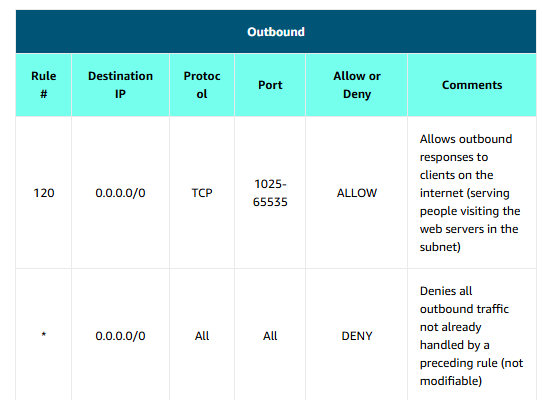
**Default ACL Network**

****

****

**Custom ACL Network**

****

****

**Secure EC2 instances with security groups**

The next layer of security is for your EC2 instances. Here, you can create a virtual firewall called a security group. The default configuration of a security group blocks all inbound traffic and allows all outbound traffic.

If you want your EC2 instance to accept traffic from the internet, you must open up inbound ports. If you have a web server, you might need to accept HTTP and HTTPS requests to allow that type of traffic into your security group. You can create an inbound rule that will allow port 80 (HTTP) and port 443 (HTTPS), as shown.  
  
Expand the block below for security group inbound rules.

**Security group inbound rules**

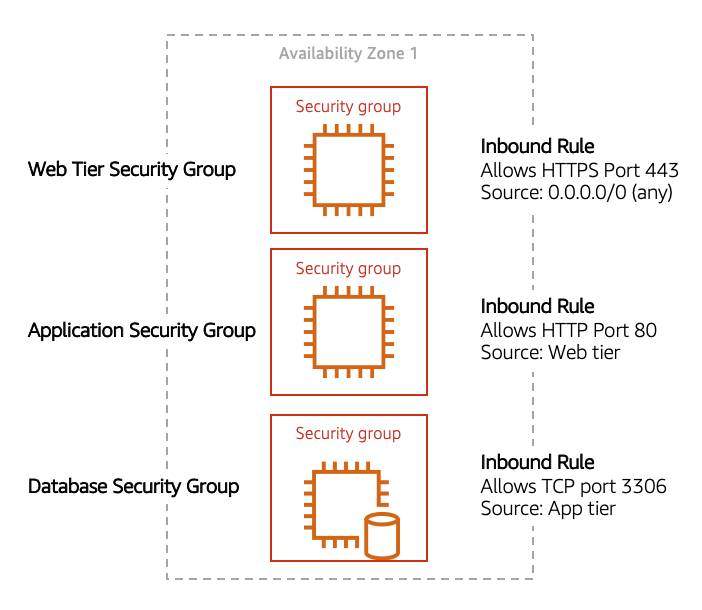
\*\*Note: For users with screen readers, use table mode to read the table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Inbound rules** | | | |
| **Type** | **Protocol** | **Port Range** | **Source** |
| HTTP (80) | TCP (4) | 80 | 0.0.0.0/0 |
| HTTP (80) | TCP (6) | 80 | ::/0 |
| HTTPS (443) | TCP (4) | 443 | 0.0.0.0/0 |
| HTTPS (443) | TCP (6) | 443 | ::/0 |

Subnets can be used to segregate traffic between computers in your network.

Security groups can be used in the same way.

A common design pattern is to organize resources into different groups and create security groups for each to control network communication between them.



This example defines three tiers and isolates each tier with defined security group rules. In this case, internet traffic to the web tier is allowed over HTTPS. Web tier to application tier traffic is allowed over HTTP, and application tier to database tier traffic is allowed over MySQL. This is different from traditional on-premises environments, in which you isolate groups of resources with a VLAN configuration. In AWS, security groups allow you to achieve the same isolation without tying the security groups to your network.