Unit V
Implementing Scalability

Defining scalability objectives

Achieving scalability requires the underlying application architecture to be scalable in order to fully leverage the highly scalable infrastructure services provided by AWS cloud.

Setting scalability objectives for an application will depend on many factors, such as the nature of the application, the number of users (peak and average), the growth rate, the business model (subscription based, free, paid, or freemium model), SLAs, the nature of customers (businesses or the general public), and so on. It is very important to set some scalability objectives, however, as operating on the cloud doesn't require you to be absolutely accurate about hard-to-estimate parameters. On the cloud, you can always respond quickly to your rapidly evolving business. So, while it is good to have sufficient information to guide you initially, you don't need to spend excessive time and effort trying to arrive at very accurate estimates.

At a high level, your application should respond proportionally to the increase in the resources consumed, and be operationally efficient and cost-effective. For example, executing a "lift-and-shift" strategy for migrating your on-premises applications to the cloud, followed by a traditional approach of increasing the sizes of your instances to meet increasing loads, will likely become very expensive. And your application's increasing resource requirements will necessitate another move to even larger instances. In most cases, you will obtain the best results by splitting your application into smaller components, and then optimizing them using AWS infrastructural features and best practices.

In all cases, an underlying objective is to always try and squeeze as much performance out of each service/component as possible before scaling and spending additional dollars.

Designing scalable application architectures

In this section, we present some of the common approaches to designing scalable application architectures.

1) **Using AWS services for out-of-the-box scalability**

One of the simplest guidelines to follow is leveraging AWS PaaS services wherever possible to enjoy the benefits of scalability and availability out of the box, without the associated administrative headaches or design complexity. Don't reinvent the wheel when ready-to use services such as email, queuing, search, databases, monitoring, metrics, logging, and so on, are vailable to you from Amazon or other third-party vendors. For example, you can leverage the RDS or the DynamoDB services available for scalable relational and NoSQL database services, respectively. Similarly, you can leverage the AWS SQS service, as it offers a multi-AZ, scalability (unlimited messages), and a secure queuing service accessible via simple APIs without having to roll out your own implementation or managing an open-source product deployed on an EC2 cluster.

2) **Using a scale-out approach**
Designing an application that can scale horizontally allows you to distribute application components, partition your data, and follow a services-oriented design strategy. This approach will help you better leverage the elasticity of the AWS cloud infrastructure

3) **Implementing loosely-coupled components**
Loosely-coupled applications are typically implemented using message-oriented architecture. The more loosely coupled the application components are, the better they will scale. Design your application to comprise of independent components. Design everything as a black box and decouple interactions to the extent possible. You can use the AWS SQS service for this purpose. SQS queues are commonly introduced between application components to buffer messages. This ensures that the application is functional under high concurrency, unpredictable loads, and/or load spikes. Loosely-coupled components enable you to differentially scale out your architecture by deploying more instances of any given component or by provisioning more powerful instances for the components that require it.

4) **Implementing asynchronous processing**
Implementing asynchronous processing in your application can improve scalability. This is typically done using AWS SQS queues.
You can use the AWS SNS service for notifying components when a message's request processing has been completed.

==========

AWS Auto Scaling

AWS Auto Scaling enables you to configure automatic scaling for the scalable resources that are part of your application in a matter of minutes. The AWS Auto Scaling console provides a single user interface to use the auto scaling features of multiple services in the AWS Cloud. You can configure automatic scaling for individual resources or for whole applications.

With AWS Auto Scaling, you configure and manage scaling for your resources through a scaling plan. The scaling plan uses dynamic scaling and predictive scaling to automatically scale your application's resources. This ensures that you add the required computing power to handle the load on your application and then remove it when it's no longer required. The scaling plan lets you choose scaling strategies to define how to optimize your resource utilization. You can optimize for availability, for cost, or a balance of both. Alternatively, you can create custom scaling strategies.

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

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

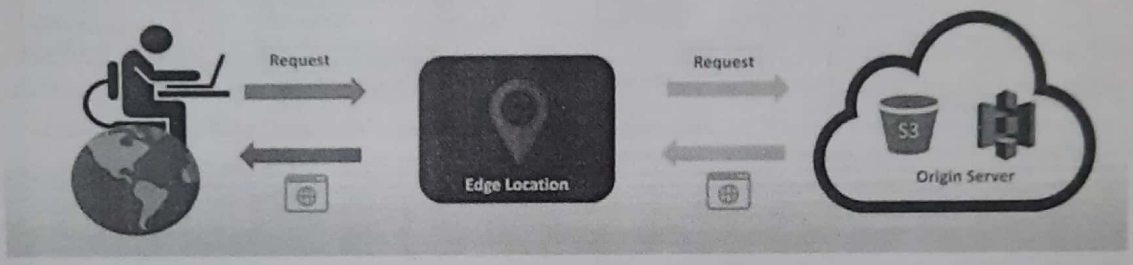
==========

Leveraging AWS infrastructure services for scalability

In this section, we will shift our focus to the strategies you can use to leverage the AWS cloud infrastructure to scale your applications.

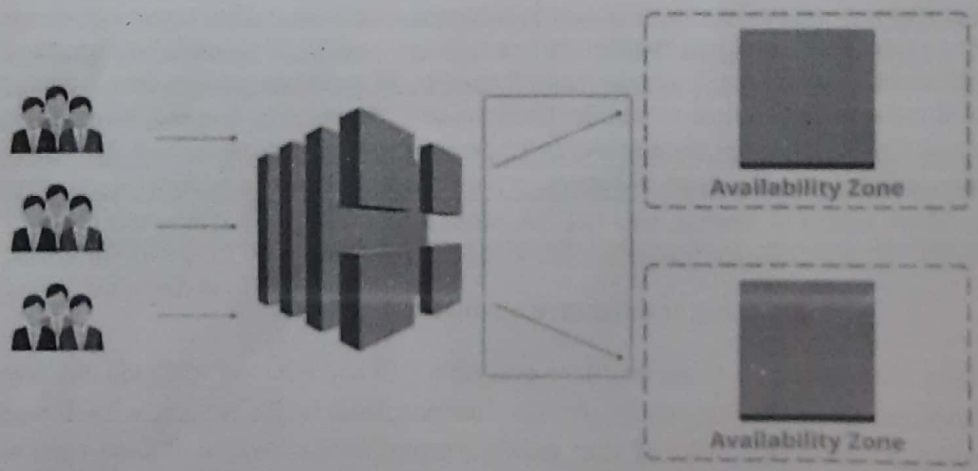
Using AWS CloudFront to distribute content

Try to offload as much content as possible to the AWS CloudFront CDN service for distribution to Amazon edge locations. This can include both static and dynamic content. For example, static content or files would include CSS, HTML, images, and so on, that are stored in Amazon S3 (and not on your web server instance). This can reduce load on your web servers and improve the efficiency of maintaining content (by storing at one S3 location) while reducing latency for your end users and overall cost (by reducing the size or the number of EC2 instances required for your web servers).



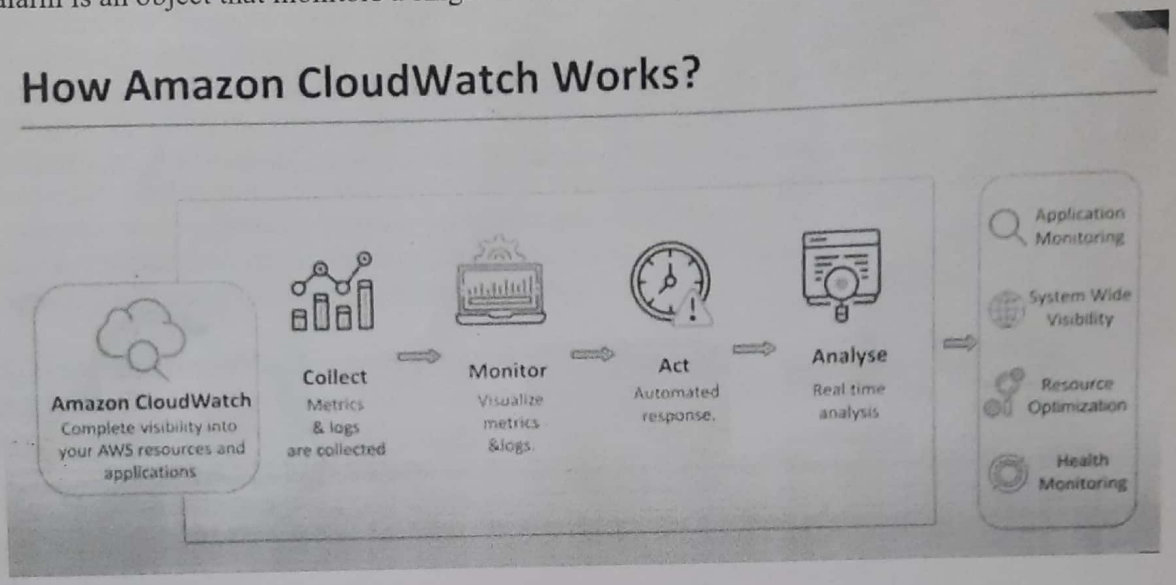
Using AWS ELB to scale without service interruptions

Configure an AWS ELB in your deployment architecture even if you are using a single EC2 instance behind it. This will ensure you are ready to scale up or down without interrupting your services. ELB ensures the CNAME application access point remains the same, as you auto scale the number of servers or even replace a fleet of servers behind it. This can also help you systematically roll out new versions of your application behind the ELB without service interruptions for your customers.



Using Amazon CloudWatch for Auto Scaling

Amazon CloudWatch is a web service that enables you to monitor and manage various metrics, and configure alarm actions based on the metrics. A metric is a variable that you want to monitor, for example, CPU usage or incoming network traffic. A CloudWatch alarm is an object that monitors a single metric over a specific period.



Scaling data services

There are several options for data services optimized for specific use cases available from AWS. Choose the most appropriate one for your application needs. For example, you can choose the RDS service for using MySQL databases, and create read replicas for use in your reporting applications. Read replicas not only serve your application needs efficiently but also help you reduce the size and number of RDS instances required.

Scaling proactively

You can proactively scale your applications in response to known traffic patterns or special events. For example, if you have cyclical patterns (daily, weekly, or monthly) in the usage of your application, then you can leverage that information to scale up or down the number of instances at the appropriate time to handle the increase or decrease in demand, respectively. You can also rapidly scale up just minutes in advance of special events, such as flash sales or in response to some major breaking news, to handle a huge surge in traffic.

Evolving architecture against increasing loads

Auto scaling is not the single thing that fixes everything. In real life, you will probably evolve your architecture against an increasing number of users or load. In this section, we will suggest actions you can take at each stage of your growth starting from a handful of users right up to tens of millions of users for a typical web stack

Scaling from one to half a million users

In the beginning, you can get started with a single EC2 instance that hosts your web service and the DB on the same instance. You can provision an EIP and use Route53 for DNS services. This should be sufficient to handle a typical website or service for a new business.

As your number of users increases to several hundreds or thousands, there are several easy-to-upgrade options available to scale your infrastructure. The simplest first step is to get a bigger EC2 instance (scale vertically). You can also leverage instance types (high I/O,memory, compute, storage intensive) based on your specific workloads. Additionally, increasing PIOP settings can give you the results you need to handle an increasing number of users. However, be aware that you will hit an upper limit in terms of how far you canscale this way. Additionally, having no redundancy or failover mechanisms as the number of users increases will become a major cause for worry.

The first change to the architecture is to separate the DB from the web tier. This step can instantly improve overall scalability, as it enables the web and data tiers to be independently scaled. Web and data tiers don't have the same requirements, so having the ability to scale them separately gives you more flexibility to align the architecture more closely to your workloads.

Typically, you would start with a relational database because you are probably most familiar and comfortable with it. It is a well established and well-worn technology; there is lots of existing sample code available, active communities, many reference books, and tools to help you with anything you are likely to face. For most workloads, you aren't going to break RDBMS down to several million users. There are well-documented patterns for scaling relational databases.

However, there may be exceptions where relational databases are not suitable for your first year of operations, and deploying a NoSQL database makes more sense. These use cases could be due to massive data volumes (>5 TB) expected in the first year itself. Other reasons for deploying a NoSQL solution could be due to requirements such as super low-latency use cases, highly non-relational data, or unstructured data requiring schema-less data constructs, rapid ingestion of streaming data (thousands of records per second), and so on. In many applications, the functional and non-functional requirements will most likely be best served by having both SQL and NoSQL databases in your application.

As the number of users goes over a thousand, you will want to consider a more distributed architecture for higher scalability and High Availability. For example, consider distributing the web tier across two AZs, and a multi-AZ DB deployment to enable replicated DBs across the two AZs. You can also include a classic elastic load balancer to distribute the load evenly and get HA out of the box.

Scaling from half a million to a million users

As we cross half a million users, we definitely need to implement auto scaling for the automatic resizing of compute clusters.

At the most basic level, we need to be clear on three parameters for auto scaling: the minimum, maximum, and the desired number of instances. We should always keep the minimum number of instances running, and launch or terminate new instances to meet desired capacity. As a practice, we never start more than a maximum number of instances,and as much as possible we keep the instances balanced across the AZs.

We will need to define launch configurations to determine what is going to be launched(EC2 instance type and size) and the AMI to be used along with the security groups, SSH keys, IAM instance profile, and users' data (essentially any arbitrary data). Bootstrapping the infrastructure requires the installation and setup to be fully automated.

If we are terminating an instance due to reduced traffic then we also need to de-register the instance from the ELB, select a target instance, and then terminate it.

Scaling plans determine when an Auto Scaling group scales in or scales out. If the desired capacity is greater than the current capacity then we launch new instances, and if the desired capacity is less than the current capacity then we terminate instances.
There are different types of scaling plans. Default pans and also you can have your own scaling plans

Scaling from a million to ten million users

As you grow your user base in the range of 5 to 10 million you will need to focus a lot more on the data tier. This would be the time to think about database federation, that is, splitting the DB into multiple DBs based on the function/purpose, sharding (splitting a dataset into multiple parts), and moving some of the functionalities to other types of specialized DBs(such as NoSQL and graph databases).

==========

Designing for and Implementing Security

Defining security objectives

Security on the cloud should be a primary area of focus for you because it is a top-of-the-mind issue for your customers. You will need to robustly address these concerns because of the following:

Customer trust: Customers come to our site expecting us to protect their information and keep it safe. We need to live up to that trust.

Regulatory compliance: Increasingly, various regulations and compliance requirements are putting security front and center, especially for cloud-based data storage and applications. And data privacy is a huge part of that.

In order to protect your assets and data on the cloud, you will need to define an Information Security Management System (ISMS), and implement security policies and processes for your organization. While larger companies may have well-defined security controls already defined for their on-premises environments, start-up organizations may be starting from scratch. However, in all cases, your customers will demand to understand your security model and require strong assurances before they use your cloud-based applications; especially in cases of SaaS or multi-tenanted applications, it can be extremely challenging to collate security-related documentation to meet varying demands, specifications, and the standards of your customers.

There are several information security standards available, for example, the ISO 27000family of standards can help you define your ISMS. In addition, you should budget for the expenses and effort required to conduct regular vulnerability assessments and audits. In some cases, be prepared to share these audit reports with your major customers.

Understanding the security responsibilities

AWS security operates on a shared responsibility model comprising of parts to be managed by you and parts managed by AWS. This model consists of three parts—infrastructure security, application security, and services security:

Infrastructure security: AWS has a whole host of industry recognized compliance certifications against various security-centric standards such as Payment Card Industry (PCI), NIST, SSAE, and ISO, as well as PCI DSS 2.0 Level1, ISO 9001, 27001, 27017, 27018, and so on.

Application security: Services that support security implementation—such as IAM policies, origin protection, ACM integration, keys/certificate rotation, and soon—in applications makes them more secure without sacrificing performance.

Services security: This includes a set of things that Amazon provides by default and what you can do with them to make your applications more secure. For example, the security options and features available on CloudFront across a growing number of edge locations and regions across cities, countries, and continents include a standardized implementation of security features.

Best practices in implementing AWS security

The AWS Identity and Access Management (IAM) service is central to implementing security for your applications on AWS cloud. Some of the main activities and best practices for AWS IAM are listed below:

- Use IAM to create users, groups, and roles, and to assign appropriate permissions.
- Manage permissions using groups. You assign permissions to groups and then assign individuals to them. While assigning permissions to groups, always grant the least privilege. AWS provides several policy templates for each of their services.

Use these policy templates, as they are a great starting point for setting up the permissions for AWS services.

- In your ISMS, you will need to define a set of roles and responsibilities and assign specific owners to particular security-related tasks and controls. Depending on your choices, these owners may be a combination of people within your organization, or AWS partners, third-party service providers, and vendors. Map each of these owners to appropriate AWS IAM roles.
- Use IAM roles to share access. Never share your credentials for giving access, temporarily or otherwise. Restrict privileged access further by using IAM conditions, and reduce or eliminate the use of root credentials.
- Use IAM roles for getting your access keys to various EC2 instances. This eases rotation of keys as the new set keys can be accessed via a web service call in your application.
- Enable multi-factor authentication for privileged users. For example, users that have permissions to terminate instances.
- Rotate security credentials regularly. Rotate both your passwords and access keys

---

Implementing High Availability

Defining availability objectives

It is easy to get confused between HA and cost optimization objectives because product owners will often push for cost optimization while ignoring their availability requirements until something fails. As a standard practice, remember to always prioritize availability goals and only then look at ways to optimize your costs.

Achieving high availability can be costly. Therefore, it is important to ensure that you align your application's availability requirements with your business objectives. There are several options to achieve the level of availability that is right for your application. Hence, it is essential to start with a clearly defined set of availability objectives and then make the most prudent design choices to achieve those objectives at a reasonable cost. Typically, all systems' functionality and services do not need to achieve the highest levels of availability possible, but at the same time, ensure that you do not introduce a single point of failure in your architecture through dependencies between your components.

The best way to approach high availability design is to assume that anything can fail, at anytime, and then consciously design against it.

Availability is something you should consider early in your application design process, as it can be hard to modify the same later. In addition, it is important to understand that

availability objectives can influence and or impact your design, development, test, and running of your system on the cloud

Finally, ensure that you proactively test all your design assumptions and reduce uncertainty by injecting or forcing failures instead of waiting for random failures to occur.

AWS high availability architecture

High availability refers to the ability to avoid unplanned outages by eliminating single-point-of-failure (SPOF). High availability systems are understood as capable of continuing to function even when critical components fail, without interrupting service or losing data, and recovering seamlessly from failure. This is a measure of the hardware, operating system, middleware, and database management software reliability.

AWS helps you achieve high availability for cloud workloads across three different dimensions:

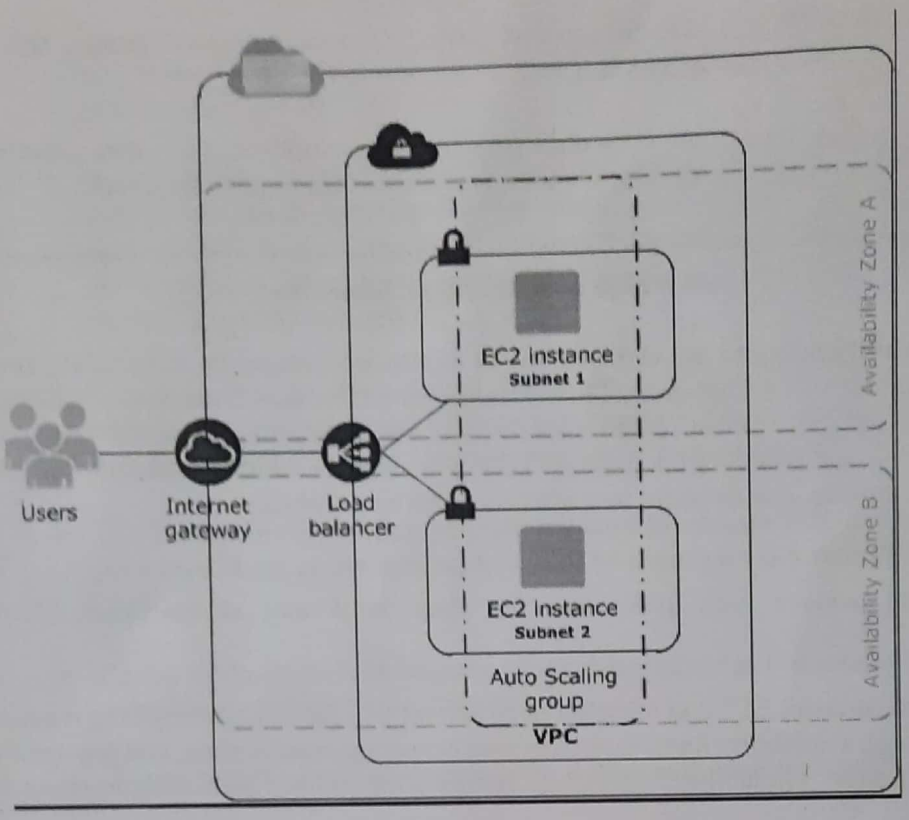
- **Compute** Amazon EC2 and other services that let you provision computing resources provide high availability features such as load balancing, auto-scaling, and provisioning across Amazon Availability Zones (AZ), representing isolated parts of an Amazon data center.
- **Database:** Amazon RDS and other managed SQL databases provide options for automatically deploying databases with a standby replica in a different AZ.
- **Storage:** Amazon storage services, such as S3, EFS, and EBS, provide built-in high availability options. S3 and EFS automatically store data across different AZs, while EBS enables the deployment of snapshots to different AZs.

AWS High Availability for EC2 Instances

If you are running instances on Amazon EC2, Amazon provides several built-in capabilities to achieve high availability:

- **Elastic Load Balancing:** Can launch several EC2 instances and distribute traffic between them.
- **Availability Zones:** Can place instances in different AZs.
- **Auto Scaling:** Detect when loads increase, and then dynamically add more instances.

These capabilities are illustrated in the diagram below. The **Elastic Load Balancing** distributes traffic between two or more EC2 instances, each of which can potentially be deployed in a separate subnet that resides in a separate Amazon *Availability Zone*. These instances can be part of an *Auto-Scaling Group*, with additional instances launched on-demand.



AWS High Availability for SQL Databases on Amazon RDS

- **Availability Zone:**

Each **Availability Zone** will have one or more data centers. The Availability Zone was created to ensure that all the essential components of AWS are always active and have sufficient emergency backup capacity.

Each Availability Zone operates independently but is still connected through its own high-speed fiber-optic network to facilitate data movement between Availability Zones within the same region.

- **Region:**

Each **AWS Region** concentrates at least 02 Availability Zones (North Virginia is currently the most AWS Availability Zones with 6 zones). In addition to backing up data between Availability Zones in the same region, users can also choose to back up data between different regions and use public or private networks to ensure that business operations always go smoothly and never interrupted.

Currently, AWS has about 77 Availability Zones spread over 24 different geographical regions. This number is expected to increase in the future to provide customers with richer service options.

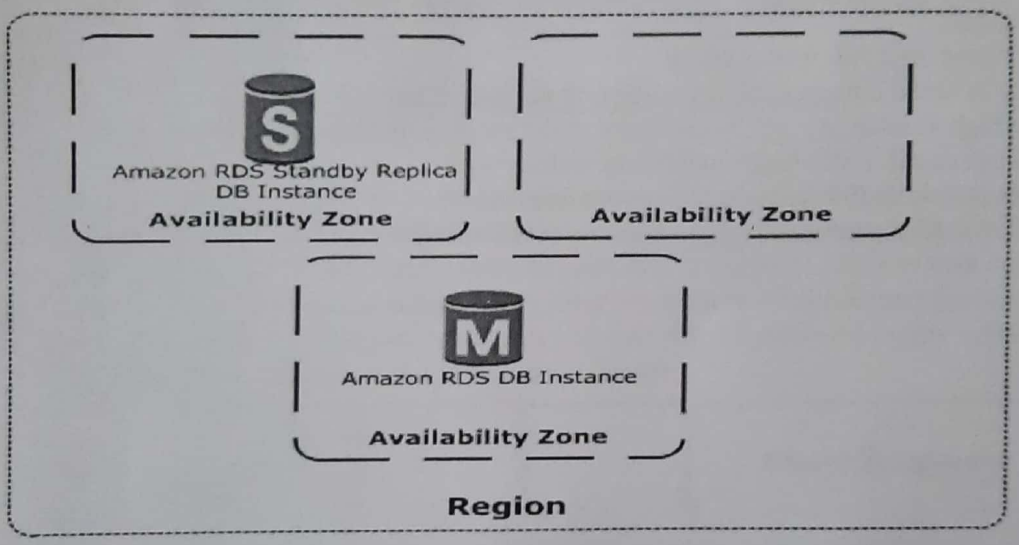
AWS High Availability for SQL Databases on Amazon RDS

RDS provides high availability using Multi-Availability Zone (Multi-AZ) deployments. This means RDS automatically provisions a synchronous replica of the database in a different availability zone. When the main database instance goes down, users are redirected transparently to the other availability zone.

This provides two levels of redundancy:

- **In case the active database fails**, there is a standby replica ready to receive requests.
- **In case of a disruption in the AZ**, your main database instance runs in; there is a failover to another AZ.

The following diagram illustrates the Multi-AZ database deployment.



Note that Multi-AZ deployment is not supported for read-only instances, and you should use read replicas to enable high availability for those instances.

AWS High Availability for Storage Services

Here is a summary of the high availability capabilities Amazon offers for other popular storage services:

- **Amazon S3:**

S3 guarantees 99.999999999% (eleven 9's) durability by redundantly storing objects on multiple devices across a minimum of three AZs in an Amazon S3 Region.

- **Amazon EFS:**

EFS guarantees up to 99.9% availability; otherwise, between 10-100% of the service fee is discounted. Every file system object is redundantly stored across multiple AZs.

- **Amazon EBS:**

EBS volumes are created in a specific AZ. You can make a volume available in another AZ, and it can then be attached to other instances in that same Availability Zone. To make a volume available outside the AZ or create redundancy, you can create a snapshot and restore it in another AZ within the same region. You can also copy snapshots to other AWS regions to create redundancy across Amazon Web Services data centers.

IMPORTANT QUESTIONS

a) Explain in detail how to define scalability objectives
b) Explain in detail how to design scalable application architectures in AWS
c) Explain the strategies used to leverage the AWS cloud infrastructure to scale your applications.
d) Write a note on AWS Auto scaling
e) Explain in detail how to evolve architecture against increasing loads
f) Define high availability and discuss how to define high availability in cloud
g) Explain in detail AWS high availability architecture
h) Explain how to define Security objectives in cloud
i) Explain the Best practices in implementing AWS security