AWS Solution Architect Associate

Version : C03

Domain 3

Task 3

Determine high-performing database solutions

Caching strategies and services

- **Amazon ElastiCache** : Fully managed in-memory caching service compatible with Redis & Memcached. It allows to deploy, operate and scale caches
- **Redis** : Open-source in-memory data structure store that can be used as a cache, message broker or data store
- **Memcached** : Open-source, distributed memory caching system commonly used to speed up dynamic web applications by alleviating database load
- **Amazon CloudFront** : Content delivery network service that accelerates the delivery of web content to end users around the world. It caches content at edge locations close to the user and can cache both static and dynamic content
- **Amazon API Gateway caching** : Built-in caching functionality for RESTful APIs. Supports both stage and method level caching with configurable time-to-live (TTL) settings
- **Elastic Load Balancer (ELB) caching**: Caching can be implemented at the application layer behind an ELB. The load to backend servers can be reduced by configuring the application servers to cache responses
- **Amazon RDS Read Replicas** :Improves read scalability and reduce read latency for database queries.Performance for read-heavy workloads can be improved by offloading read traffic to replicas

Caching strategies types

Caching Strategies :

- ➔ Database Caching
- ➔ CDN Caching
- ➔ Web Caching
- ➔ General Application Caching
- ➔ Session Caching
- ➔ Object Caching

Caching strategies and services

Caching Strategies - Process of storing frequently accessed data in a temporary storage area, which allows for faster retrieval when requested. Caching strategies-

- ➔ **Cache-Aside (Lazy Loading)**- Data is fetched from the cache only when needed. Data is fetched from the primary data store and stored in the cache for future use if not there in the cache
- ➔ **Write-Through**- Data is written to both cache and the primary data store simultaneously. This ensures cache is always up-to-date but may introduce latency due to the additional write operation.
- ➔ **Write-Behind (Write-Behind Caching)**- Data is initially written only to the cache,later cache asynchronously writes data to the primary data store. This improves write performance but may risk data loss if the cache fails before data is written to the primary store
- ➔ **Cache-Aside with Read-Through**- Data is fetched from the primary data store and stored in the cache for future reads if not found in the cache

Caching strategies and services

Capacity Units - Capacity units refer to the resources allocated to a cache instance such as memory, CPU and network bandwidth. The number of capacity units determines the performance and scalability of the caching service. Increasing capacity units typically improves cache performance and can handle more requests simultaneously

Instance Types - Caching services often offer different instance types with varying levels of compute, memory, and networking capabilities. Instance types are designed to accommodate different workloads and performance requirements. Common instance types include small, medium, large and extra-large with corresponding resource allocations

Provisioned IOPS (Input/Output Operations Per Second) - Provisioned IOPS is a performance metric used for storage volumes in cloud computing environments. It represents the maximum number of read/write operations that a storage volume can perform per second. Provisioned IOPS is particularly important for caching services that rely on fast access to data. By provisioning a sufficient number of IOPS, you ensure that the cache can handle high levels of read and write operations without experiencing performance degradation.

Data access patterns (Read Vs Write Intensive)

Data access patterns varies depending on the nature of the application and requirements of the workload. They can be classified into two broad categories - Read-Intensive and Write-Intensive

- **Read-Intensive Patterns**
  - Higher frequency of read operations compared to write operations.
  - Example - CDN, Static website hosting, read-heavy analytics and data warehousing
  - AWS services - S3 for storing static content, CloudFront for CDN services, Redshift for data warehousing and Elasticsearch Service for log analysis and search

- **Write-Intensive Patterns**
  - Higher frequency of write operations compared to reads
  - Example - transactional db's, logging systems and real-time analytics
  - AWS services - RDS for transactional db's, DynamoDB for NoSQL db's requiring high throughput, Kinesis for real-time data streaming and Aurora for high-performance relational databases.

Data access patterns (Read Vs Write Intensive)

Design Considerations

- **Read Intensive**
  - Optimize read performance with caching mechanisms to reduce the load on underlying storage
  - AWS services - CloudFront for caching frequently accessed content closer to end-users.
  - Consider read replicas for databases to distribute read traffic and scale read operations horizontally.
- **Write Intensive**
  - Optimize for high throughput and low latency writes.
  - Utilize partitioning and sharding techniques to distribute write load evenly across resources.
  - AWS services - DynamoDB automatically handles scaling to accommodate write-heavy workloads.
  - Consider asynchronous processing mechanisms to decouple write operations from time-sensitive tasks

Scaling Strategies

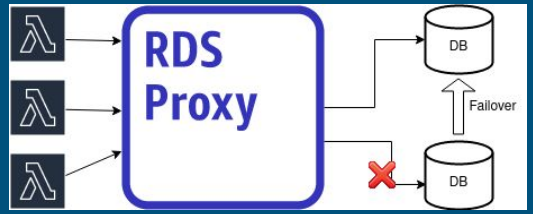
- **Read Intensive**
  - Horizontal scaling by adding more read replicas or using scalable storage services like Amazon S3.
  - Implementing caching strategies to reduce the load on the backend systems.
  - Leveraging content delivery networks (CDNs) to cache and serve frequently accessed content.
- **Write Intensive**
  - Vertical scaling by increasing the capacity of the underlying resources
  - Horizontal scaling through sharding or partitioning data across multiple nodes or partitions.
  - AWS services - Amazon DynamoDB or Amazon Aurora

Database capacity planning

- Capacity planning for AWS databases involves estimating resource requirements to ensure optimal performance, scalability and cost-effectiveness. Approach for capacity planning

  - **Understand workloads**
  - **Appropriate database types**
  - **Storage requirements**
  - **Instance type and size**
  - **Replication and Availability**
  - **Monitoring and Alerts**
  - **Horizontal and Vertical scaling**
  - **Query performance**
  - **Regular review and adjust**
  - **Backup and disaster recovery**

Database connections and proxies



- **RDS (Relational Database Service)**
  - AWS takes care of the scaling, patching and backups
  - AWS provides options for controlling access to the RDS instance such as security groups and IAM roles
- **ProxySQL**
  - Proxy for MySQL db's that help with pooling and managing connections to the database servers
  - Can be deployed and configured on EC2 instances to route traffic to RDS\self-managed db instances
- **Amazon Aurora**
  - MySQL and PostgreSQL-compatible relational database built for the cloud
  - Built-in connection pooling capabilities that manage connections efficiently, reducing the load on the database instances
- **Amazon DocumentDB**
  - Fully managed document database service compatible with MongoDB.
  - Offers high availability and scalability
  - Standard MongoDB drivers can be used to connect to DocumentDB
  - Handles connection pooling and failover internally
- **Amazon Neptune**
  - Fully managed graph db service that supports popular graph models
  - Manage connections using IAM-based authentication which provides fine-grained control

Database replication

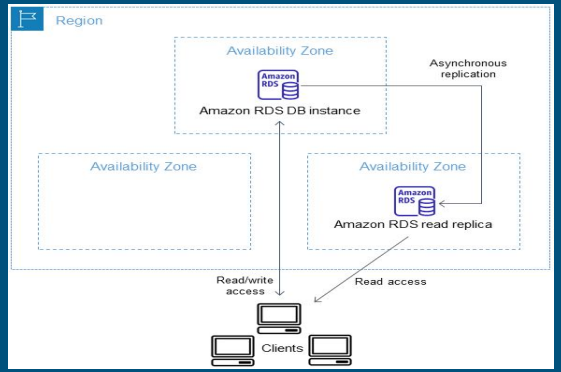
- **Amazon RDS Multi-AZ Deployments**
  - Supports Multi-AZ deployments for high availability and failover.
  - RDS automatically replicates db to a standby instance in a different AZ.
  - In the event of a primary instance failure, RDS automatically fails over to the standby instance with minimal downtime
- **Amazon RDS Read Replicas**
  - Supports read replicas for scaling read-heavy workloads.
  - Asynchronously replicate data from the primary RDS instance to read-only replicas. This offloads read traffic from the primary instance and improves read scalability
  - In case of a primary instance failure,Read replicas can be promoted to standalone instances
- **Amazon Aurora Global Database**
  - MySQL and PostgreSQL-compatible relational database engine that offers a Global Database feature for cross-region replication
  - Allows to replicate a primary Aurora db cluster across multiple AWS regions with low latency
  - Enables to build disaster recovery solutions and serve read traffic from geographically distributed replicas

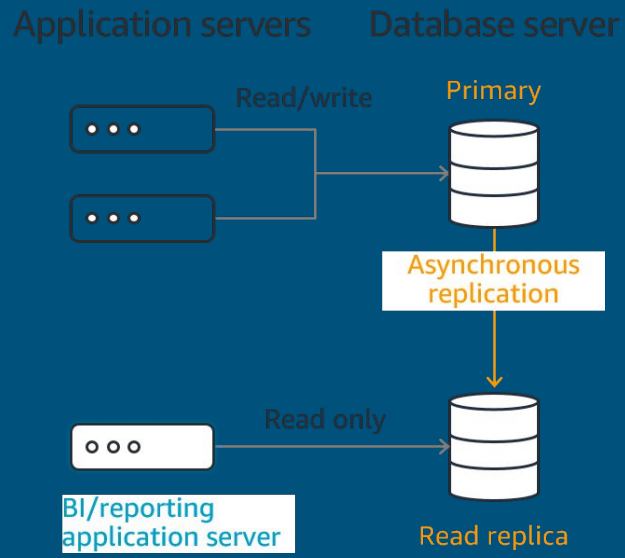
Database replication

- **Amazon DynamoDB Streams and Cross-Region Replication**
  - Fully managed NoSQL database service offering DynamoDB Streams for capturing changes to data in real-time.
  - DynamoDB Streams can be used to replicate data across multiple DynamoDB tables or to external systems
  - Supports cross-region replication
- **Amazon Redshift Cluster Replication**
  - Fully managed data warehousing service
  - Supports cluster replication for data redundancy and disaster recovery
  - Cross-region snapshots can be created and restored to a new cluster in a different region
  - Enables querying data directly from Amazon S3
- **Third-Party Replication Solutions**
  - AWS Database Migration Service (DMS) can be used to replicate data between different database engines (e.g., MySQL to PostgreSQL) or between on-premises databases and AWS cloud databases.

Read replicas

- A read replica is a read-only copy of a DB instance
- Load on the primary DB instance can be reduced by routing queries from applications to the read replica
- Can be elastically scaled beyond the capacity constraints of a single DB instance for read-heavy database workloads
- To create a read replica from a source DB instance, Amazon RDS uses the built-in replication features of the DB engine
- After the read replica is created from source DB, the source becomes the primary DB instance
- When the updates are made to the primary DB instance, Amazon RDS copies them asynchronously to the read replica





Read replicas setup

- **Choose database engine**
  - AWS RDS supports various database engines -MySQL, PostgreSQL, Oracle, SQL Server,Amazon Aurora
- **Create primary instance**
  - Start by creating primary database instance where the data will be written to
- **Enable Multi-AZ deployment**
  - Multi-AZ deployment automatically provisions and maintains a synchronous standby replica in a different Availability Zone
- **Create read replicas**
  - Read replicas are separate db instances that replicate data from primary instance asynchronously
- **Configure read replica settings**
  - Settings like instance class, storage type, availability zone, and the number of replicas
- **Security and access control**
  - Appropriate security group settings and access control policies to restrict access as needed.
- **Connect applications to read replicas**
  - Connect applications to read replicas to offload read-only queries from the primary instance, distributing the read workload and improving overall performance
- **Monitor and scale**
  - CloudWatch can be used to monitor the performance of read replicas
  - Depending on workload, read replicas can be scaled up or down to meet demand

Designing database architectures

Designing a database architecture on AWS involves considering various factors such as scalability, availability, performance, security and cost-effectiveness.

- **Identify Requirements** : Understand the application's data requirements, including the volume of data, access patterns, query latency and the types of data being stored
- **Choose the Right Database Service** : Select the service that best fits your needs based on factors like data structure, scalability, consistency and query requirements
- **Data Modeling** : Design the schema for your database based on the identified requirements. This includes defining tables, columns, indexes, relationships and data types
- **Scalability** : Features like auto-scaling, read replicas, sharding and partitioning to distribute load effectively can be used to ensure the db architecture can scale to handle increasing workload demands.
- **High Availability and Disaster Recovery** : Features like Multi-AZ deployments, backups and replication can be used to ensure data durability and recoverability

Designing database architectures

- **Performance Optimization** : Caching mechanisms, indexing and query optimization techniques can be used to improve query response times
- **Security** : Features like encryption at rest and in transit, IAM roles, VPC security groups and fine-grained access controls can be used to implement robust security measures to protect data and ensure compliance with regulations
- **Monitoring and Management** : AWS CloudWatch, AWS CloudTrail and third-party monitoring tools can be used to monitor performance and troubleshoot issues.
- **Cost Optimization** : DB architecture should be optimized for cost-effectiveness by selecting the right instance types, storage options, and reserved capacity. Over-provisioning can be avoided by monitoring usage
- **Testing and Optimization** : Database architecture should be tested under different scenarios to ensure it meets performance, scalability and availability requirements

Database engines

- **Amazon RDS (Relational Database Service)**
  - Supports various relational db engines like MySQL, PostgreSQL, MariaDB, Oracle and Microsoft SQL Server
  - Automates time-consuming tasks such as hardware provisioning, database setup, patching and backups
  - Ideal for applications that require a traditional relational database structure
- **Amazon Aurora**
  - MySQL and PostgreSQL compatible relational database built for the cloud
  - Fully managed, highly scalable and durable
  - Offers performance and availability enhancements for the supported db's
  - Automatic failover, continuous backup to Amazon S3 and read replicas for scaling reads.
- **Amazon DynamoDB**
  - Fully managed NoSQL database service
  - Designed for applications that require single-digit millisecond latency at any scale
  - Supports document\key-value data models
  - Features like automatic scaling, encryption at rest and in transit and built-in security

Database engines

- **Amazon Redshift**
  - Fully managed data warehouse service
  - Designed for running complex queries on large datasets.
  - Optimized for high-performance analysis of structured data using SQL.
  - Scales easily to petabytes of data and allows integration with other AWS services
- **Amazon Neptune**
  - Fully managed graph database service
  - Supports popular graph models Property Graph and W3C's RDF and their respective query languages
  - Designed to store and navigate relationships between highly connected datasets
- **Amazon ElastiCache**
  - Fully managed in-memory data store and cache service
  - Supports two popular open-source in-memory data stores :
    - Redis
    - Memcached.
  - Designed to improve the performance of web applications by caching frequently accessed data.

Database Type/Usecases/AWS Services

Type of Database	Use Cases	AWS Services
Key-Value	Real-time bidding, Ecommerce shopping carts,Product Catalogs,Customer preference	Amazon DynamoDB
Document	Cataloging,Content Management System, Customer profiles and personalization, Mobile Apps	Amazon DocumentDB
In-memory	Caching , Session Stores, Gaming , Leaderboards, Geospatial services, Pub/Sub messaging, Real time streaming	Amazon ElasticCache for Memcached Amazon ElasticCache for Redis
Graph	Fraud detection, Social Networking, Recommendation Engines, Knowledge graphs, Data lineage	Amazon Neptune
Time-series	Devops, Application Monitoring, Industrial telemetry, IoT applications	Amazon Timestream
Ledger	Finance, Manufacturing, Insurance Claims, HR and payroll, Retail inventories	Amazon Quantum Ledger

AWS Solution Architect Associate

Version : C03

Domain 3

Task 3

The END