Spring23 Quiz1

A. [30 points] A.1 [5] Write the code snippet to extract the public IP address and the instanceID once you submit the request to create a VM and you get the result. Your code should address the asynchronous aspect of the request you might have submitted to request a VM instance

A.1 Extracting Public IP Address and Instance ID from EC2 Instance Creation

To extract the public IP address and instance ID after submitting a request to create an EC2 instance, you need to handle the asynchronous nature of the request. Here's a code snippet that demonstrates this using the AWS SDK for .NET:

```
using Amazon.EC2;

using Amazon.EC2.Model;

using System;

using System.Threading.Tasks;


public async Task<(string InstanceId, string PublicIpAddress)>
LaunchInstanceAndGetDetails()

{

   var ec2Client = new AmazonEC2Client();

   var request = new RunInstancesRequest

   {

      ImageId = "ami-xxxxxxxx", // Replace with your AMI ID

      InstanceType = InstanceType.T2Micro,
```

```
        MinCount = 1,
        MaxCount = 1
    };

    try
    {
        var response = await ec2Client.RunInstancesAsync(request);
        var instanceId = response.Reservation.Instances[0].InstanceId;

        // Wait for the instance to be running and have a public IP
        var describeRequest = new DescribeInstancesRequest { InstanceIds = new List<string> { instanceId } };
        Instance instance;
        do
        {
            await Task.Delay(5000); // Wait for 5 seconds before checking again
            var describeResponse = await ec2Client.DescribeInstancesAsync(describeRequest);
            instance = describeResponse.Reservations[0].Instances[0];
        } while (instance.State.Name != InstanceStateName.Running || string.IsNullOrEmpty(instance.PublicIpAddress));

        return (instance.InstanceId, instance.PublicIpAddress);
    }
```

```
    catch (AmazonEC2Exception e)
    {
        Console.WriteLine($"Error: {e.Message}");
        return (null, null);
    }
}
```

This code does the following:

1. It creates an EC2 client and a request to launch an instance.
2. It sends the request asynchronously using RunInstancesAsync.
3. After getting the initial response, it extracts the instance ID.
4. It then enters a loop to wait for the instance to reach the "running" state and have a public IP address assigned.
5. The loop uses DescribeInstancesAsync to get updated instance information every 5 seconds.
6. Once the instance is running and has a public IP, it returns both the instance ID and public IP address.

A.2 [5] You want to create a backup service using AWS cloud to backup files from your local drive. Describe how you would design this and the key steps required for this. Add any code fragments (i.e., command line etc from CLI) to illustrate it

A.2 Designing a Backup Service Using AWS Cloud

To create a backup service using AWS cloud to backup files from your local drive, you can follow these key steps:

1. **Set up AWS CLI**: Install and configure AWS CLI on your local machine

4

.

2. **Create an S3 bucket**: This will store your backups.

text

```
aws s3 mb s3://my-backup-bucket
```

3. **Create a backup script**: Write a script to sync your local files to S3. Here's a sample batch script for Windows

3

:

text

```
@echo off
setlocal

set "AWS_PROFILE=your-profile-name"
set "SOURCE_DIR=C:\path\to\your\local\directory"
set "BUCKET_NAME=my-backup-bucket"

aws s3 sync "%SOURCE_DIR%" "s3://%BUCKET_NAME%" --delete
```

4. **Schedule the backup**: Use Windows Task Scheduler or cron (on Linux) to run the script regularly

3

.

5. **Set up lifecycle policies**: Configure S3 lifecycle policies to manage long-term storage and reduce costs.
6. **Implement versioning**: Enable versioning on your S3 bucket to keep multiple versions of files.
7. **Set up notifications**: Use S3 event notifications to get alerts about backup activities

3

.

8. **Secure your backups**: Use AWS IAM to manage access to your S3 bucket and encrypt your data at rest.

Key AWS CLI commands for managing backups:

- Sync files to S3:

text

```
aws s3 sync C:\your\local\path s3://my-backup-bucket --delete
```

- Copy a specific file:

text

```
aws s3 cp "C:\users\my first backup.bak" s3://my-backup-bucket/
```

- Download a file from S3:

text

```
aws s3 cp s3://my-backup-bucket/my-backup.bak ./
```

- Delete a file from S3:

text

```
aws s3 rm s3://my-backup-bucket/my-backup.bak
```

This design provides a robust, scalable, and secure solution for backing up your local files to AWS cloud using S3. The use of AWS CLI allows for easy automation and integration with existing systems.

B. System Design [30] You will enhance the system and architecture of the Assignment 1 with these additional features. Following are the additional features you need to support: 1. You need to support user authentication. Each user will have their own profile - you may assume whatever information you want to keep in the profile. 2. When the user logs in, this should recommend a list of restaurants based on past usage and other factors such as what their friends are sharing about their restaurant choices. 3. You need to have support for a Friends' List - you could assume that a user will have a set of friends who are using this App. You need to support what the user's friends are recommending with their reviews. You need to state the details of the following to support these enhancements: 1. Extended architecture on top of the architecture for Assignment 1. 2. Additional data models you would need. 3. List of APIs and Lambda functions to support the above. 4. E2e interaction of components for illustrating how your proposed recommendation works. 5. E2e interaction of components for illustrating how your friends' recommendations surfaced on the user's app. Your extended system should scale, handle failures, and should utilize appropriate data stores as needed

To enhance the system and architecture of Assignment 1 with the additional features, I'll provide a detailed system design that addresses each of the requirements.

1. Extended Architecture

To support the new features, the following components need to be added to the existing architecture:

- **Authentication Service**: An AWS Cognito User Pool for managing user authentication, along with an Identity Pool for granting temporary, secure access to AWS resources.
- **User Profile Service**: A DynamoDB table for storing user profiles, including preferences, past restaurant visits, friends' lists, and other personal information.
- **Friends' List Service**: Another DynamoDB table for managing friends' relationships.
- **Recommendation Engine**: A Lambda function (LF3) that processes user and friends' data to provide personalized recommendations.
- **Social Feed**: An SQS queue (Q2) for friends' restaurant recommendations and reviews, triggering a Lambda function (LF4) for real-time updates to the user's feed.
- **Caching Layer**: Use ElastiCache (Redis) to cache frequent user data for faster access.

Additional Data Models

1. **User Profile**:
   - UserID (Primary Key)
   - Email
   - Name
   - PreferredCuisines (List)
   - LastLoginTimestamp

2. **Friend Relationship**:
   - UserID
   - FriendID
   - RelationshipStatus

3. **User Activity**:
   - UserID
   - RestaurantID
   - ActivityType (View, Book, Review)
   - Timestamp

4. **Restaurant Review**:
   - ReviewID (Primary Key)
   - UserID
   - RestaurantID
   - Rating
   - ReviewText
   - Timestamp

APIs and Lambda Functions

1. **User Authentication API**:
   - `/auth/login`: Lambda function to handle user login

- `/auth/register`: Lambda function for user registration
- `/auth/logout`: Lambda function to handle user logout

2. **User Profile API**:
- `/profile/get`: Lambda function to retrieve user profile
- `/profile/update`: Lambda function to update user profile

3. **Friends API**:
- `/friends/list`: Lambda function to get user's friends list
- `/friends/add`: Lambda function to add a friend
- `/friends/remove`: Lambda function to remove a friend

4. **Recommendations API**:
- `/recommendations/personal`: Lambda function to get personalized recommendations
- `/recommendations/friends`: Lambda function to get friends' recommendations

5. **Review API**:
- `/review/create`: Lambda function to create a review
- `/review/get`: Lambda function to get reviews for a restaurant

6. **Enhanced LF2 (Recommendation Worker)**:
- Update to include personalized recommendations and friend activity

E2E Interaction for Personalized Recommendations

1. User logs in through the frontend, authenticated by Amazon Cognito.

2. The frontend calls the `/recommendations/personal` API.

3. The API Gateway triggers the corresponding Lambda function.

4. The Lambda function:

   a. Retrieves the user's profile from DynamoDB.

   b. Fetches the user's activity data from DynamoDB.

   c. Calls Amazon Personalize to generate personalized recommendations.

   d. Queries ElasticSearch for restaurant details based on the recommendations.

   e. Retrieves full restaurant information from DynamoDB.

5. The Lambda function returns the personalized recommendations to the frontend.

6. The frontend displays the recommendations to the user.

E2E Interaction for Friends' Recommendations

1. User navigates to the friends' recommendations section in the frontend.

2. The frontend calls the `/recommendations/friends` API.

3. The API Gateway triggers the corresponding Lambda function.

4. The Lambda function:

   a. Queries Amazon Neptune to get the user's friends list.

   b. For each friend, retrieves their recent reviews and activities from DynamoDB.

   c. Aggregates and sorts the friends' recommendations.

   d. Fetches restaurant details from ElasticSearch and DynamoDB.

5. The Lambda function returns the aggregated friends' recommendations to the frontend.

6. The frontend displays the friends' recommendations to the user.

Scalability, Fault Tolerance, and Data Store Utilization

1. **Scalability**:

   - Use AWS Auto Scaling for Lambda functions and ElasticSearch.

   - Implement caching with Amazon ElastiCache to reduce database load.

   - Utilize Amazon CloudFront for content delivery to reduce latency.

2. **Fault Tolerance**:

   - Implement retry mechanisms and circuit breakers in Lambda functions.

   - Use multi-AZ deployments for DynamoDB, ElasticSearch, and Neptune.

   - Implement dead-letter queues for failed SQS messages.

3. **Data Store Utilization**:

- DynamoDB: For user profiles, restaurant details, and user activity.

- ElasticSearch: For efficient restaurant search and filtering.

- Neptune: For managing friend relationships and social graph queries.

- S3: For storing static assets and frontend files.

- Amazon Personalize: For generating personalized recommendations.

This enhanced system design builds upon the existing architecture while adding new components and functionality to support user authentication, personalized recommendations, and friend-based recommendations. The design ensures scalability, fault tolerance, and efficient data management across various AWS services.

D. Lectures [20]

D.1 What is live VM migration? Describe the process through "iterative copy" with an illustration? Is shared storage between source and destination VM required for live migration?

D.1 Live VM Migration

Live VM Migration or **Live Migration** is the process of moving a virtual machine (VM) from one host to another without interrupting access to the VM. Here's how the process works through **iterative copy**:

- **Pre-Migration (Stage 0)**: The VM is running normally on Host A. If needed, a target host (Host B) is preselected for migration.

- **Reservation (Stage 1)**: A request is issued to migrate the VM from Host A to Host B. Resources on Host B are confirmed and reserved.
- **Iterative Pre-Copy (Stage 2)**: Memory pages are transferred iteratively from Host A to Host B. Initially, all pages are transferred, then only those modified during the previous transfer phase are copied in subsequent iterations.
- **Stop-and-Copy (Stage 3)**: The VM is briefly suspended on Host A, and any remaining memory pages are transferred. Network traffic is redirected to Host B.
- **Commitment (Stage 4)**: Host B indicates that it has received a consistent OS image. Host A releases the VM state and confirms the migration.
- **Activation (Stage 5)**: The VM starts on Host B, resumes normal operation, and connects to local devices.

Illustration:

text

```
graph TD
    A[Active VM on Host A] --> B(Reservation)
    B --> C(Iterative Pre-Copy)
    C --> D(Stop-and-Copy)
    D --> E(Commitment)
    E --> F[VM running normally on Host B]
```

Is shared storage required for live migration?Live migration can be performed without shared storage, but it is more common and simpler with shared storage. Shared storage allows for easier access to the VM's disk image during migration, but it's not strictly necessary. Virtualization platforms like VMware vSphere support live

migration without shared storage through features like standard vMotion, which requires access to both source and destination datastores

D.2 What is the use of Secondary Indexes in DynamoDB tables? List and explain the different types of secondary indexes available in AWS DynamoDB

D.2 Secondary Indexes in DynamoDB

Secondary Indexes in AWS DynamoDB are used to provide more querying flexibility on a table beyond the primary key:

- **Local Secondary Index (LSI)**: An index that has the same partition key as the base table but a different sort key. It must be defined at table creation and allows for queries on attributes other than the primary key.
- **Global Secondary Index (GSI)**: Allows for a different partition key and sort key from the base table. GSIs can be added or removed at any time, enabling queries on different attributes across the entire table, not just within a partition.

Benefits:

- **Query Flexibility**: Secondary indexes allow for more diverse query patterns.
- **Performance**: By adding indexes, you can reduce the need to scan the entire table, improving query performance.
- **Scalability**: GSIs can be used to distribute data more evenly across partitions.

D.3 What is the benefit of a private cloud? Provide three reasons why Enterprises are adopting this even though they need to invest in the infrastructure. How does Hybrid cloud help

D.3 Benefits of Private Cloud

Benefits of Private Cloud:

1. Control and Security: Enterprises have full control over the infrastructure, security policies, and data, ensuring compliance with regulations and data privacy concerns.
2. Customization: Private clouds can be tailored to specific business needs, providing a customized environment for applications and workloads.
3. Performance: Since resources are not shared with other organizations, performance can be optimized for the specific workloads of the enterprise.

Adoption Reasons:

- Data Sensitivity: Companies handling sensitive data prefer private clouds for better control over data security.
- Regulatory Compliance: Certain industries require data to reside within the company's infrastructure for compliance reasons.
- Performance and Reliability: Dedicated resources ensure high performance and reliability.

Hybrid Cloud Help:

- Scalability: Allows for scaling out to the public cloud when additional capacity is needed.
- Cost Efficiency: Utilizes the public cloud for non-sensitive workloads or for burst capacity, reducing costs.

- Seamless Integration: Provides a unified platform where applications can move between private and public clouds seamlessly.

D.4 What is the key difference between Full and Para virtualization

D.4 Key Difference Between Full and Para Virtualization

Full Virtualization:

- **Complete Isolation**: The VM is fully isolated from the host, simulating complete hardware, allowing for unmodified guest OS to run.
- **Performance Overhead**: There can be a performance penalty due to the emulation of hardware.
- **Flexibility**: Any OS can be run, even those not aware of being virtualized.

Para Virtualization:

- **OS Modifications**: The guest OS is modified to be aware of the virtualization layer, communicating directly with the hypervisor for certain operations.
- **Performance**: Better performance as the guest OS can directly interact with the hypervisor, reducing the overhead of full hardware emulation.
- **Compatibility**: Requires changes to the guest OS, which might limit compatibility with some operating systems.

In summary, full virtualization provides better compatibility and isolation, while para virtualization offers improved performance at the cost of OS modifications.