1. What are three advantages to using Swagger?

2. When you call the API from your frontend application hosted at http://www.yourdomain.com, you get

the following error in the developer console:

“No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin

‘http://www.yourdomain.com’ is therefore not allowed access.”

> What is the reason for this error? How do you resolve it? (describe every step)

3. You create a new Lambda function that returns the text "Hello world". You test it and it completes the

execution in a few milliseconds. You then decide to use it to store a message in a DynamoDB table. You

add code to do just that, on top of the original code. Upon testing the new version of the function, it times

out after 3 seconds, over and over again.

>What might be the issue and how can you resolve it?

4. You have an SQS queue with 100,000 messages in it. How would you go about reading the messages

on AWS? Please be descriptive in terms of the infrastructure and the permissions needed.

5. For the following questions, imagine you have the following resources on AWS: an API Gateway

deployment, a Lambda function connected to one of the API Gateway methods that responds to each

request, and a Lex bot called "Concierge". You create a new intent called HotelBookingIntent in Lex. You

add a few utterances and build it. When you start testing it in the Lex test console, everything works

as expected. You then write code to call your Lex bot from a Lambda function, but all the incoming

messages are not understood by the Lex bot.

>What is the most likely reason for Lex failing to understood messages sent through the Lambda function,

yet everything working correctly through the Lex test console?

1. **Advantages of Swagger**: Swagger provides automated documentation, which is always in sync with the code; it allows for easy testing and mocking of APIs; and it supports code generation for client SDKs, saving time on development.

2. **CORS error resolution**: This error occurs because your frontend application is trying to access resources from a different domain without the proper CORS (Cross-Origin Resource Sharing) headers. To resolve it, add the Access-Control-Allow-Origin header in the server's response to allow the frontend's domain, either by modifying the API backend code or configuring API Gateway to include the necessary CORS settings.

3. **Lambda timeout issue**: The Lambda function likely exceeds the default 3-second execution time due to delays in writing to DynamoDB. To resolve this, increase the Lambda timeout value in the AWS Lambda console and ensure that the DynamoDB operation completes within the new timeout limit.

4. **Reading SQS messages**: To read 100,000 messages from an SQS queue, create an AWS Lambda function triggered by the SQS queue, with a batch size to process multiple messages at once. Ensure the Lambda function has permissions (via an IAM role) to access the SQS queue and process the messages.

5. **Lex failure through Lambda**: The issue could be related to the format or structure of the request sent from the Lambda function. Ensure that the message is correctly formatted and that the Lambda function's permissions are set up to allow it to invoke the Lex bot. Additionally, verify that the correct botAlias and botName are being used in the Lambda function.

A. Questions based on Assignment 1 [25 Points]

1. You have a payment API that calls a third party API service to make payments. The API receives millions of requests per hour. At some point, the payment service, starts failing intermittently due to the high volume of transactions.

> How would you re-architect the system to ensure that all payments are processed without an impact to the customer experience?

2. What steps would you take in API Gateway to secure your API? Your goal is to both authenticate and authorize the users. (describe every step)

You now decide to create an Amazon Lex bot and call the bot from your Lambda function. Once you finished the setup, you start making API calls from your frontend to your API again, but you realize that the Lambda function that is supposed to call Lex times out after 30 seconds

3. What is the most likely issue with the Lambda function's timeout?

You fixed the timeout issue and you start testing the bot through your frontend. After a few tries you notice that your bot is not picking up the RestaurantRecommendationIntent that you set up in Lex. You check the Lex console on AWS and confirm that the intent is there and that you can test it through the AWS console.

4. What is the most likely reason for the missing RestaurantRecommendationIntent? How would you solve it?

5. When would you use Elastic vs DynamoDB? Give examples to illustrate when one is preferable over the other.

1. **Re-architecting payment API**: To handle high volume and ensure reliability, use a message queue (e.g., SQS) to decouple the payment processing, so requests are placed in a queue and processed asynchronously. Add retries with exponential backoff to handle failures, and use Lambda to process payments in batches.

2. **Securing API Gateway**: Implement API keys or OAuth 2.0 for authentication. For authorization, use AWS IAM roles and policies, or integrate a service like Amazon Cognito to manage user identities. Configure API Gateway to enforce these authentication and authorization mechanisms for incoming requests.

3. **Lambda timeout issue**: The Lambda function may be waiting too long for a response from the Lex bot. This could happen if the Lex bot's response time exceeds the Lambda function's timeout setting. Increase the Lambda function's timeout to accommodate the longer response time.

4. **Missing intent issue**: The Lambda function may not be correctly configured to handle the RestaurantRecommendationIntent. Ensure the intent name is correctly referenced in the Lambda code and that the Lex bot is properly integrated with the Lambda function.

5. **Elastic vs DynamoDB**: Use **ElasticSearch** for full-text search, complex queries, and when you need advanced search capabilities (e.g., logs, documents). Use **DynamoDB** for high-performance, scalable NoSQL storage, particularly when you need low-latency access for simple queries or key-value lookups (e.g., user profiles, session data).

Section A - Assignments (5 x 5)

1. What protocol governs the permissions granted by a web server to web browsers when accessing resources from different domains? Explain how this works.

2. Write a Kubernetes Service manifest to expose a Deployment named "my-app" on port 8080 within the cluster.

3. Explain the concept of serverless architecture and its benefits. How does it relate to the implementation of the Dining Concierge chatbot?

4. Design a data model for the DynamoDB table "yelp-restaurants" to efficiently store and query restaurant information. Discuss the choice of partition keys, sort keys, and secondary indexes to support various query patterns, including location-based searches and cuisine filtering.

5. In a microservices architecture, when choosing between AWS EC2 and AWS Lambda for handling tasks within a service, what distinguishes their use cases?

A.3 [20] Multiple Choice Answers

1. What is the difference between a Docker container and a virtual machine?

- B) A Docker container shares the same kernel as the host machine, while a virtual machine has its own kernel

2. What is the purpose of a Dockerfile?

- D) To define the Docker image and its dependencies

3. Which Docker component is responsible for managing images?

- B) Docker Registry

4. What is the main benefit of AWS Lambda?

- A) It eliminates the need for server administration

5. How does AWS Lambda handle state management?

- B) It does not handle state management, and requires the use of external storage systems

6. Which of the following is true about Elasticsearch?

- C) Both A and B

7. What is the purpose of throttling in API Gateway?

- a) To prevent excessive API calls that may degrade API performance or availability.

8. Which of the following is not a benefit of using API caching in API Gateway?

- d) Reduced API security risks

9. What is the purpose of session data in API Gateway?

- b) To store user session information, such as user ID and access token

10. DynamoDB uses eventually consistent and strongly consistent reads to support dynamic application needs.

- A. TRUE

1. **CORS (Cross-Origin Resource Sharing)** governs permissions for web browsers when accessing resources from different domains. It works by having the server include specific headers (Access-Control-Allow-Origin, Access-Control-Allow-Methods, etc.) to define which domains are allowed to access its resources, enabling secure cross-origin requests.

```
apiVersion: v1

kind: Service

metadata:

  name: my-app-service

spec:

  selector:

    app: my-app

  ports:

    - protocol: TCP

      port: 8080

      targetPort: 8080

  type: ClusterIP
```

3. **Serverless architecture** eliminates the need for managing servers, as it automatically scales based on usage. Benefits include reduced operational costs, easier scaling, and simplified management. For the Dining Concierge chatbot, serverless (using AWS Lambda) would allow for automatic scaling and reduced overhead for API calls and responses.

4. **Data model for "yelp-restaurants" DynamoDB**:

- **Partition Key**: restaurant_id (unique identifier for each restaurant)
- **Sort Key**: location (city, state, or region for location-based queries)
- **Global Secondary Index (GSI)**: cuisine for filtering by cuisine
- **Local Secondary Index (LSI)**: rating for sorting by ratings

This design supports efficient queries by restaurant ID, location, cuisine, and rating.

5. **EC2 vs Lambda in Microservices**:

- **EC2** is ideal for long-running services or when you need complete control over the infrastructure (e.g., complex configurations).
- **Lambda** is best for event-driven, short-duration tasks that can be triggered by events (e.g., API calls), offering automatic scaling and cost-efficiency for variable workloads.

A.2 [5] Designing a backup service using AWS Cloud

To create a backup service for files from your local drive to AWS:

1. **Amazon S3** will be used for storing backups.
2. **AWS CLI** or **AWS SDK** can be used to interact with S3 for file uploads.
3. Use **AWS Lambda** to automate periodic backups (triggered by an AWS EventBridge schedule).
4. Set up an **IAM Role** with proper permissions to allow access to S3.

Steps:

- Create an S3 bucket for storing backups.
- Install and configure the AWS CLI on your local machine.
- Write a script to upload files to S3.

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.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.

Example backup command using AWS CLI:

bash Copy

```bash
aws s3 cp /path/to/local/files s3://my-backup-bucket/ --recursive
```

Automate backups with Lambda (Python):

python Copy

```python
import boto3
import os
from datetime import datetime

s3 = boto3.client('s3')
bucket_name = 'my-backup-bucket'

def lambda_handler(event, context):
    local_path = '/path/to/local/files'
    files = os.listdir(local_path)

    for file in files:
        s3.upload_file(os.path.join(local_path, file), bucket_name, f"backup/{datetime.now().strftime('%Y-%m-%d')}/{file}")
```

A.1 [5] Code snippet to extract the public IP address and instanceID after VM creation

python Copy

```python
import boto3
import time

# Create EC2 client
ec2 = boto3.client('ec2')

# Start an EC2 instance (asynchronous request)
response = ec2.run_instances(
    ImageId='ami-12345678',  # Use appropriate AMI ID
    InstanceType='t2.micro',
    MinCount=1,
    MaxCount=1
)

# Extract the Instance ID from the response
instance_id = response['Instances'][0]['InstanceId']

# Wait until the instance is running
ec2.get_waiter('instance_running').wait(InstanceIds=[instance_id])

# Fetch the public IP address of the instance
instance_info = ec2.describe_instances(InstanceIds=[instance_id])
public_ip = instance_info['Reservations'][0]['Instances'][0].get('PublicIpAddress')

print(f"Instance ID: {instance_id}")
print(f"Public IP: {public_ip}")
```

This code waits for the instance to be in the ‘running’ state before fetching the public IP address.

1. Reasons for using API Gateway instead of directly invoking the Lambda function

Using API Gateway provides better **security**, **scalability**, and **API management**. It allows for easy routing of requests, integrates with authentication methods like Cognito, provides rate limiting and throttling, and offers built-in monitoring with CloudWatch. API Gateway also enables versioning and input validation, which are harder to implement directly in Lambda.

---

2. What is a session in Lex? How can it be useful?

A **session** in Lex is a temporary storage that holds the conversation context, such as user data and intents, during an interaction. It helps maintain continuity across multiple requests in a conversation, allowing the bot to reference past interactions and provide more contextually accurate responses.

---

3. How to troubleshoot documents not indexed in Elasticsearch

- **Document format**: Ensure that the documents are in the correct format (JSON) and follow Elasticsearch's expected structure.
- **Indexing failures**: Check if Elasticsearch is returning any HTTP status codes or responses indicating issues, such as 400 or 500 errors. Also, verify that the index mapping is correct and supports the document schema.

---

4. Preflight requests in API Gateway

A **preflight request** is an automatic CORS request sent by the browser to check if the server will allow the actual request. It's important for web security because it ensures that the server permits specific types of requests (like PUT, DELETE) with non-standard headers. To troubleshoot, you would ensure that your API Gateway is configured to handle CORS properly by enabling CORS on the methods that need it and allowing the correct headers.

---

5. What could go wrong with insertedTimestamp as a primary or secondary key in DynamoDB?

Using insertedTimestamp as the **primary key** can lead to issues with **hot partitions**, as records inserted around the same time may end up in the same partition, leading to uneven load distribution. As a **secondary key**, it could still cause issues with **query performance** if many records share the same timestamp, resulting in inefficient queries. It's better to use a unique identifier or business logic-based partitioning strategy.

1. Lambda function timing out after adding code for DynamoDB

The issue might be that the Lambda function is taking longer to execute due to the additional code that interacts with DynamoDB. By default, Lambda has a timeout of 3 seconds, which is too short for operations like writing to DynamoDB.

- **Resolution**:
- Increase the Lambda timeout setting in the AWS Lambda console.
- Check if the DynamoDB call is blocking or if it can be optimized.

---

2. CORS error: "No 'Access-Control-Allow-Origin' header is present"

This error happens because the API does not allow cross-origin requests from your frontend domain (http://www.yourdomain.com). This is related to Cross-Origin Resource Sharing (CORS) settings.

- **Steps to resolve**:

1. In the **API Gateway**, enable CORS on the endpoint.
2. Make sure to configure the Access-Control-Allow-Origin header in the response. This can be done by allowing the specific domain (http://www.yourdomain.com) or using * for all domains.
3. You also need to add other CORS-related headers like Access-Control-Allow-Methods and Access-Control-Allow-Headers in the API Gateway.

---

3. Purpose of a Dockerfile

The correct answer is:

- **d. To define the Docker image and its dependencies**

A Dockerfile is used to define the environment and dependencies required for a Docker image, including the base image, libraries, and configurations needed for your application.

---

4. Which Docker component is responsible for managing images?

The correct answer is:

- **b. Docker Registry**

A Docker Registry is responsible for storing, managing, and distributing Docker images. Docker Hub is a public registry, but you can also use private registries.

---

5. Main benefit of AWS Lambda

The correct answer is:

- **a. It eliminates the need for server administration**

AWS Lambda is a serverless computing service, which means you don't have to manage servers. It automatically scales based on usage, and you only pay for the execution time.

---

6. How does AWS Lambda handle state management?

The correct answer is:

- **b. It does not handle state management, and requires the use of external storage systems**

AWS Lambda is stateless, meaning it does not persist any data between invocations. To manage state, you would need to use external systems like DynamoDB or S3.

---

7. Which breakdown fields values of a document into a stream and creates inverted indexes?

The correct answer is:

- **d. Tokenizer**

A Tokenizer is responsible for breaking down the text of a document into a stream of tokens, which are then used to create inverted indexes.

---

8. Which is a collection of fields in a specific manner defined in JSON format?

The correct answer is:

- **c. Index**

An Index in Elasticsearch is a collection of documents that share the same data structure, defined in a specific manner in JSON format.

---

9. Which of the following runs on each node and ensures containers are running in a pod?

The correct answer is:

c. Kubelet

The Kubelet runs on each node in a Kubernetes cluster and ensures that the containers in the pod are running and healthy.

---

10. Which field in a Service YAML file specifies which pods the service should route traffic to?

The correct answer is:

a. selector

The selector field in a Kubernetes Service YAML file specifies which pods the service should route traffic to based on labels.