AWS Basics

General Concepts:

A regions is a physical location in the world that comprises cluster of data centres

Within each region there are AZs(Availability Zones) which consists of 1 - 6 Data centres.

Apart of AZs, there are Edge Locations which are mainly used to service Amazon cloud front and Amazon route 53.

A. AWS CloudFront

B. STORAGE AND CONTENT DELIVERY

Storage offerings of AWS can be divided in 3 categories

1. **Object** — An object is a piece of data, like a document, image, or video that is stored with some metadata in a flat structure. As a example you can easily develop a web application which can call (API)content on top of Amazon S3
2. **File** — In file storage, data is presented via a file system interface and with file system semantics to instances.
3. **Block** — In block storage, data is presented to your instance as a disk volume.

B.1.1 Elastic Block Storage
B.1.2 Elastic File System
B. 1.3 Amazon Simple Shared Storage(S3)
B. 1.4. AWS Storage Gateway

C. Compute

C.1 -> EC2

AWS has different types of Ec2 instances **1.On demand 2. Reserved 3. Spot 4. Dedicated Hosts**

- Standard Reserved Instances cannot be moved between regions.

- Spot Instances pricing on the lower range.

D. Networking

E. SECURITY AND COMPLIANCE

========== ===== ====== ===== ====

AWS LAMBDA

50 interview questions related to AWS Lambda:

1) What is AWS Lambda, and what problem does it solve?
2) Explain the basic concept of "serverless" computing in AWS Lambda.
3) What are the programming languages supported by AWS Lambda?
4) How do you trigger an AWS Lambda function?
5) What is an event source in AWS Lambda?
6) Can you explain the key components of an AWS Lambda function?
7) How do you control the concurrency of an AWS Lambda function?

8) What is the maximum execution time allowed for an AWS Lambda function?
9) How is the billing calculated for AWS Lambda functions?
10) What are the deployment packages in AWS Lambda?
11) How can you share code and resources between multiple AWS Lambda functions?
12) What is the significance of the "Handler" in an AWS Lambda function?
13) How can you pass data to an AWS Lambda function from the event source?
14) How does AWS Lambda handle dead-letter queues?
15) Can you explain the concept of "environment variables" in AWS Lambda?
16) What is an AWS Lambda function alias, and how is it useful?
17) How can you perform error handling in AWS Lambda?
18) What is the difference between AWS Lambda and EC2?
19) How does AWS Lambda ensure high availability and fault tolerance?
20) What are the IAM (Identity and Access Management) permissions required for an AWS Lambda function?
21) How can you optimize the cold start time of an AWS Lambda function?
22) What is the significance of the "context" object in an AWS Lambda function?
23) Explain the concept of "provisioned concurrency"

in AWS Lambda.

24) How do you enable VPC (Virtual Private Cloud) access for an AWS Lambda function?

25) Can you invoke one AWS Lambda function from another?

26) How do you debug an AWS Lambda function?

27) What is the difference between synchronous and asynchronous invocation of AWS Lambda functions?

28) How do you package and deploy external dependencies with an AWS Lambda function?

29) What is the relationship between AWS API Gateway and AWS Lambda?

30) Can you explain the concept of "Lambda Layers"?

31) How can you set up environment-specific configurations in AWS Lambda?

32) What is the impact of not handling exceptions properly in an AWS Lambda function?

33) How can you ensure secure communication between an AWS API Gateway and AWS Lambda?

34) What are the best practices for monitoring AWS Lambda functions?

35) How does AWS Lambda manage scalability and concurrency?

36) What is the purpose of the "RequestResponse" and "Event" invocation types in AWS Lambda?

37) Can you explain the concept of "reserved concurrency" in AWS Lambda?

38) How do you manage secrets and sensitive information in an AWS Lambda function?

39) What are the limitations of AWS Lambda?

40) How can you configure an AWS Lambda function

to be triggered by a scheduled event?

41) What is the "X-Ray" service in AWS Lambda, and how can it be used for debugging?

42) How can you handle long-running tasks in AWS Lambda?

43) What are the options for logging from an AWS Lambda function?

44) How do you handle binary data in an AWS Lambda function?

45) Can you explain the "Step Functions" integration with AWS Lambda?

46) How does AWS Lambda handle concurrent function executions?

47) What are the best practices for managing state in AWS Lambda?

48) How do you handle cross-origin resource sharing (CORS) in an AWS Lambda function?

49) Can you explain the impact of the AWS Lambda execution environment on function performance?

50) How can you secure an AWS Lambda function to prevent unauthorized access?

ANSWERS :

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). It allows developers to run code without provisioning or managing servers, thus providing a cost-effective and scalable solution. Here's a compact summary of the 50 interview questions and their answers:

1) AWS Lambda is a serverless compute service that enables running code without managing servers.

2) "Serverless" refers to the concept of abstracting infrastructure management from developers, allowing them to focus on code.

3) AWS Lambda supports various programming languages, including Node.js, Python, Java, Ruby, Go, and .NET Core.

4) AWS Lambda functions are triggered by various event sources, such as API Gateway, S3, SNS, and more.

5) The event source is responsible for triggering the AWS Lambda function.

6) Key components of an AWS Lambda function include the function code, handler ,runtime, and associated configuration. (More details after this section)

7) Concurrency in AWS Lambda can be controlled to manage the number of simultaneous function executions. (More details after this section) Provisoned Concurrency

8) AWS Lambda functions have a maximum execution time, typically ranging from a few seconds to 15 minutes.

9) Billing for AWS Lambda is based on the number of requests and the time taken to execute those requests.

10) Deployment packages in AWS Lambda consist of the function code and any external dependencies.

11) Code and resources can be shared between AWS Lambda functions using layers.

12) The "Handler" in an AWS Lambda function specifies the entry point to the function code.

13) Data can be passed to an AWS Lambda function through the event object.

14) Dead-letter queues in AWS Lambda are used to handle failed asynchronous invocations.

15) Environment variables in AWS Lambda allow storing configuration information securely.

16) AWS Lambda function aliases enable pointing to different versions of a function.

17) Error handling in AWS Lambda involves logging errors and using retry mechanisms for synchronous invocations.

18) AWS Lambda is serverless, while EC2 requires provisioning and managing servers.

19) AWS Lambda ensures high availability and fault tolerance by running functions across multiple Availability Zones.

20) IAM permissions are required for AWS Lambda functions to interact with other AWS services.

21) Cold start time can be optimized by reducing the size of the deployment package and using provisioned concurrency.

22) The "context" object provides runtime information to an AWS Lambda function.

23) Provisioned concurrency allows pre-warming AWS Lambda function instances to reduce cold start latencies.

24) VPC access can be enabled for AWS Lambda functions to access resources within a VPC.

25) AWS Lambda functions can invoke other Lambda functions using their ARNs.

26) AWS Lambda functions can be debugged using logging and monitoring tools like CloudWatch.

27) Synchronous invocation waits for a response from the function, while asynchronous invocation does not.

28) External dependencies can be packaged with the function code or placed in Lambda Layers.

29) AWS API Gateway integrates with AWS Lambda

to create serverless APIs.

30) Lambda Layers allow sharing code libraries and resources across multiple functions.

31) Environment-specific configurations can be set using environment variables.

32) Improper exception handling can lead to unhandled errors and unexpected behavior in AWS Lambda functions.

33) Secure communication between API Gateway and AWS Lambda is achieved using IAM roles.

34) Monitoring AWS Lambda functions involves using CloudWatch metrics and logs.

35) AWS Lambda automatically scales to handle varying workloads and concurrent requests.

36) AWS Lambda offers "RequestResponse" and "Event" invocation types.

37) Reserved concurrency can be used to limit the number of concurrent executions of a function.

38) Secrets and sensitive information can be managed using AWS Key Management Service (KMS) and environment variables.

39) AWS Lambda has some limitations related to

execution time, memory, and disk space.

40) Scheduled events can trigger AWS Lambda functions at specified intervals.

41) AWS X-Ray can be used to debug and trace requests through AWS Lambda functions.

42) Long-running tasks can be handled using asynchronous invocations and Step Functions.

43) Logging options in AWS Lambda include CloudWatch Logs and custom logging libraries.

44) Binary data can be handled in AWS Lambda by encoding it as base64.

45) AWS Step Functions allow orchestrating and coordinating AWS Lambda functions in workflows.

46) AWS Lambda handles concurrent function executions by spinning up multiple instances.

47) State management in AWS Lambda functions can be achieved using external storage services.

48) Cross-Origin Resource Sharing (CORS) can be handled in an API Gateway configuration.

49) The AWS Lambda execution environment affects function performance.

50) AWS Lambda functions can be secured using IAM roles and resource policies.

Key Components of AWS Lambda

Q. Can you explain the key components of an AWS Lambda function?

Ans : Certainly! An AWS Lambda function consists of several key components that work together to execute the code when triggered by an event source. The main components of an AWS Lambda function are:

A. **Function Code:** The function code is the heart of an AWS Lambda function. It contains the actual business logic that the function performs when triggered. The code is written in one of the supported programming languages, such as Node.js, Python, Java, Ruby, Go, or .NET Core.

B. **Handler:** The handler is the entry point to the AWS Lambda function's code. It is a specific function within the function code that AWS Lambda calls when an event occurs. The handler function receives two parameters: the event data (input) and a context object (runtime information).

C. **Runtime:** The runtime is the environment in which the AWS Lambda function runs. It determines the execution environment and provides the necessary runtime dependencies for the function code. AWS Lambda supports various runtimes corresponding to the supported programming languages.

D. **Event Sources:** AWS Lambda functions are

event-driven, meaning they are triggered by event sources such as AWS services (e.g., S3, DynamoDB, SNS) or custom event sources (e.g., API Gateway, CloudFront, Step Functions). The event source passes the event data to the Lambda function.

E. **Context Object:** The context object provides runtime information to the AWS Lambda function during its execution. It contains details like the function's AWS request ID, remaining time before the function times out, and information about the function's execution environment.

F. **Configuration:** AWS Lambda functions have various configuration settings. These include memory allocation, timeout duration, IAM (Identity and Access Management) role, VPC (Virtual Private Cloud) configuration, environment variables, and function concurrency settings.

G. **Execution Role:** An execution role is an IAM role assigned to the AWS Lambda function. It grants necessary permissions to access AWS services and resources while the function is executing. The execution role is defined in the function's configuration.

H. **Versioning and Aliases:** AWS Lambda supports function versioning, which allows you to manage different versions of the same function. Additionally, function aliases provide a way to decouple the application code from a specific function version, making it easier to implement blue-green deployments and

traffic shifting.

I. **Logs and Monitoring:** AWS Lambda automatically streams logs to Amazon CloudWatch, which provides real-time monitoring and troubleshooting capabilities. Developers can also set up custom logging and monitoring using CloudWatch and other AWS services.

J. **Concurrency and Scaling:** AWS Lambda automatically manages the concurrency and scaling of function instances based on incoming requests. Developers can configure concurrency limits and provisioned concurrency to control the number of concurrent executions.

K. **Triggers:** Triggers are the mechanisms that invoke the AWS Lambda function. Event sources like S3 bucket changes, API Gateway requests, and CloudWatch Events can trigger the Lambda function.

By understanding and leveraging these key components, developers can create powerful serverless applications using AWS Lambda, focusing on writing business logic while leaving the operational aspects to AWS.

Concurrency in AWS Lambda

Ans :

AWS Lambda is designed to automatically manage scalability and concurrency to handle varying levels of incoming traffic and workload. Here's how it achieves this:

1. Event-Driven Architecture: AWS Lambda is triggered by various event sources, such as API

Gateway requests, S3 bucket changes, SNS notifications, and more. When an event occurs, Lambda automatically spins up instances of the function to process the events concurrently.

2. Automatic Scaling: AWS Lambda automatically scales out to handle increased request rates. When the number of incoming events increases, Lambda detects the load and provisions additional instances of the function to distribute the workload.

3. Parallel Execution: Each instance of a Lambda function can handle a single request at a time. However, Lambda can create multiple instances of the function to process multiple requests concurrently, thus achieving parallel execution.

4. Concurrency Limits: AWS Lambda allows you to set concurrency limits for your functions. Concurrency limits define the maximum number of instances that can run in parallel. By setting concurrency limits, you can control the number of concurrent executions and avoid potential overloading of downstream resources.

5. **Provisioned Concurrency:** To reduce cold start latencies and ensure consistent performance for critical applications, AWS Lambda offers "provisioned concurrency." With provisioned concurrency, you can specify a number of instances that are pre-initialized and kept ready to respond to events immediately. This reduces the initial overhead and ensures low-latency responses.

6. Auto Scaling Integration: AWS Lambda can be integrated with Amazon CloudWatch and AWS Auto Scaling. CloudWatch allows you to monitor the number of concurrent executions and other metrics, while AWS Auto Scaling can automatically adjust the provisioned concurrency based on the configured scaling policies.

7. Stateless Architecture: AWS Lambda functions are stateless, which means they don't retain any data between invocations. This statelessness allows Lambda to spin up multiple instances without the need to consider session affinity or data synchronization.

8. Shared Responsibility: While AWS takes care of the underlying infrastructure scaling, it is essential to design your Lambda functions with concurrency in mind. Avoid global variables, utilize local variables within the function, and be mindful of potential contention issues when accessing shared resources.

By employing these strategies and features, AWS Lambda efficiently manages scalability and concurrency, allowing developers to focus on writing the business logic and leaving the operational aspects to AWS.

Q. What is Dead-letter queue in AWS Lambda?
Ans:
In AWS Lambda, a Dead-Letter Queue (DLQ) is a feature that helps capture and handle failed events from asynchronous invocations. When an AWS Lambda

function is configured with a DLQ, any event that fails to be processed successfully by the function is sent to the specified DLQ instead of being discarded or lost.
The DLQ acts as a safety net to capture events that couldn't be processed due to errors, throttling, or other issues. It allows developers to inspect and analyze the failed events, perform retries, and take appropriate corrective actions. DLQs are especially useful when using AWS Lambda to process events from event sources like Amazon SNS (Simple Notification Service), Amazon Kinesis, and Amazon SQS (Simple Queue Service).
Here's how the Dead-Letter Queue works in AWS Lambda:

- Event Processing Failure: When AWS Lambda is triggered by an event source (e.g., an SNS message or an SQS message), it attempts to process the event using the function code.

- Failure Handling: If the function encounters an error during processing and is unable to complete successfully, it can explicitly raise an exception or error, or it might fail due to various reasons like code issues, throttling, or resource limitations.

- DLQ Configuration: To handle these failed events, you can configure a DLQ for the AWS Lambda function. **The DLQ can be an Amazon SQS queue or an Amazon SNS topic.**

- Sending Failed Events to DLQ: When an event

fails to be processed successfully, AWS Lambda automatically sends the failed event (along with the error details) to the specified DLQ.

- Retries and Error Handling: Once the event is in the DLQ, you can implement retry mechanisms, investigate the cause of failure, and take corrective actions to address the underlying issues.

- Monitoring and Troubleshooting: AWS Lambda automatically logs information about events sent to the DLQ, which enables monitoring and troubleshooting of failed event processing.

By using a Dead-Letter Queue, developers can prevent the loss of critical events and ensure that failed events are captured and processed appropriately. This is particularly important in scenarios where data integrity and reliability are crucial, and it helps to maintain the overall resilience and robustness of serverless applications built with AWS Lambda.

How to call AWS lambda function calling from another

==========

AWS Lambda functions can invoke other Lambda functions, allowing you to build modular and reusable serverless applications. This approach is commonly used for breaking down complex tasks into smaller,

manageable functions, each responsible for a specific piece of functionality. To call one Lambda function from another, you can use either synchronous or asynchronous invocation methods.

Synchronous Invocation:

In synchronous invocation, the calling Lambda function waits for the response from the invoked Lambda function before proceeding. This method is suitable when you need the result of the invoked function immediately.

Here's how to achieve synchronous invocation between Lambda functions:

- Using SDKs: If you are using an AWS SDK (e.g., Boto3 for Python, AWS SDK for JavaScript) or AWS CLI, you can invoke a Lambda function synchronously using the invoke method. The invoke method waits for the function to complete and returns the response.

- Invocation Payload: When invoking the function, you can pass data to the invoked function through the payload. The payload should be formatted as per the input expected by the invoked function's handler.

- Response Handling: The invoked Lambda function's handler returns a response, which can be processed by the calling Lambda function. You can extract the data or status from the response to use it further in your logic.

Asynchronous Invocation:

In asynchronous invocation, the calling Lambda function

sends the request to the invoked function but does not wait for the response. This method is suitable when you don't need immediate results and want to decouple the functions for better scalability.
Here's how to achieve asynchronous invocation between Lambda functions:

- Using SDKs: Similar to synchronous invocation, you can use AWS SDKs or AWS CLI to invoke the Lambda function asynchronously. The invoke method with the appropriate invocation type (Event or RequestResponse) is used for asynchronous invocation.

- Invocation Payload: You can pass data to the invoked function through the payload, just like in synchronous invocation. However, since the caller does not wait for the response, the payload is typically used to pass input data without expecting a return value.

- No Immediate Response: When invoking a Lambda function asynchronously, the calling function does not receive an immediate response. The invoked function processes the event independently.

- Error Handling: If the invoked Lambda function fails, AWS will automatically retry the function a few times. If the retries fail, the event may be sent to a Dead-Letter Queue (DLQ) if configured.

By using synchronous or asynchronous invocation between Lambda functions, you can create more flexible and modular serverless applications, enabling better

code organization and reusability. Choose the appropriate invocation method based on your application's requirements and use case.

Best Approach

Using Amazon SNS (Simple Notification Service) is another common method to invoke AWS Lambda functions. SNS allows you to decouple the event source from the Lambda function, making it easier to integrate multiple functions and event sources. SNS acts as an intermediary between the event source and the Lambda function, enabling a more flexible and scalable architecture.

Here's how you can use Amazon SNS to invoke AWS Lambda functions:

- Configure SNS Topic: First, you need to create an SNS topic that will act as the message bus for the events. This topic will be the trigger for the Lambda function.

- Subscribe Lambda Function: Next, you subscribe the AWS Lambda function to the SNS topic. This subscription tells SNS to deliver any messages published to the topic to the subscribed Lambda function.

- Event Source to SNS: When an event occurs in your system (e.g., a new item in an S3 bucket or a record inserted into a DynamoDB table), you send a message (event) to the configured SNS topic.

- Message Format: The message sent to the SNS topic can contain information relevant to the event, such as the event type, data payload, and other metadata.

- Message Delivery: SNS delivers the message to all the subscribed Lambda functions asynchronously. The message is placed in an SNS topic, and each subscription receives a copy of the message.

- Lambda Function Execution: The subscribed Lambda function processes the message and performs the desired actions based on the event data.

Using Amazon SNS with AWS Lambda provides several advantages:

- Decoupling: SNS decouples the event source from the Lambda function, enabling multiple functions to subscribe to the same topic and process events independently.

- Fan-Out: SNS supports fan-out patterns, where one event can be distributed to multiple Lambda functions, allowing parallel processing and scaling.

- Filtering: SNS allows you to apply message filtering policies to control which messages are delivered to specific subscriptions, providing a powerful way to handle different event types.

- Retries and Error Handling: SNS automatically retries message delivery to Lambda functions in case of failures, ensuring reliable event processing.

- Event Aggregation: SNS can aggregate events from different sources into a single topic, simplifying event handling for downstream services.

- Cross-Account Integration: SNS supports cross-account integration, enabling different AWS accounts to publish events to a shared SNS topic and invoke Lambda functions in another account.

By leveraging Amazon SNS to invoke AWS Lambda functions, you can build highly scalable and loosely coupled event-driven architectures that can handle a wide variety of events from different sources.

Q. Different use cases where lambda can be used

AWS Lambda is a versatile and powerful serverless compute service that can be used in a wide range of use cases. Here are some common scenarios where Lambda is often employed:

1. Real-time Data Processing: Lambda can process real-time data streams from services like Amazon Kinesis or AWS IoT Core, allowing you to perform real-time analytics, aggregations, and filtering on incoming data.

2. Web and Mobile Backend: Lambda can serve as the backend for web and mobile applications, handling HTTP requests from APIs like Amazon API Gateway and responding with processed data or results.

3. Scheduled Tasks: Lambda can be scheduled to run at specific intervals using Amazon CloudWatch Events, making it perfect for automating periodic tasks, data backups, and log rotation.

4. Data Transformation: Lambda can transform data

from one format to another, making it ideal for ETL (Extract, Transform, Load) tasks and data integration.

5. Microservices: Lambda can be used to implement individual microservices in a microservices architecture, providing a scalable and cost-effective solution for different application components.

6. Image and Media Processing: Lambda can process and manipulate images or videos, performing tasks like thumbnail generation, resizing, or watermarking.

7. Real-time File Processing: Lambda can process files as they are uploaded to Amazon S3, enabling real-time transformations or aggregations.

8. Chatbots and Natural Language Processing: Lambda can be integrated with services like Amazon Lex or Amazon Comprehend to build chatbots and perform natural language processing tasks.

9. Internet of Things (IoT): Lambda can be used with AWS IoT Core to process and react to IoT device data, enabling smart and responsive applications.

10. Event-Driven Architecture: Lambda is at the core of event-driven architectures, allowing services to react to events from various sources and take appropriate actions.

11. Authentication and Authorization: Lambda can be used for custom authentication and authorization logic, enhancing the security of applications and APIs.

12. Data Validation and Error Handling: Lambda can validate incoming data and perform error handling, ensuring data integrity and consistency.

13. Machine Learning Inference: Lambda can be integrated with Amazon SageMaker to perform real-time inference on machine learning models.

14. Monitoring and Log Analysis: Lambda can be used to monitor logs, trigger alerts, and take remedial actions based on predefined conditions.

15. Geospatial Processing: Lambda can handle geospatial data and perform tasks like location-based services and geofencing.

These are just a few examples of how AWS Lambda can be used. Its flexibility, scalability, and cost-effectiveness make it suitable for a wide range of applications, enabling developers to focus on writing code and delivering business value without worrying about managing infrastructure.

========== ===== ====== ===== ====