AWS 'Developing on AWS' training course DAY TWO (22/03/2023 – 24/03/2023)

Content:

This summary below is just a short & summarized version of the main points in the course.

1. Extra info: AWS 7 factors methodology – how AWS evaluates cloud-based apps:
    a. Scalability
    b. Sustainability
    c. Reliability
    d. Cost Optimization
    e. Performance efficiency
    f. Operational excellency
    g. Security
    h. The last 6 (b to g) are part of AWS Well Architected Framework
2. Module 7 (Getting started with databases)
    a. Databases can be divided into 2 broad categories:
        i. Non-managed services: Means you ownself manage your own DB services. Need to manually patch your DBs, and need to manage your DB to ensure they are scalable and highly available. For instance, you have a EC2 instance (a server/on-prem computer) then you install your DB on that server instance/on-prem instance & manage it yourself.
        ii. Managed services: Means your DB services are managed by AWS. No need to care about scalability, high availability, and patching etc… cause AWS handles it for you.
    b. Two main types of DBs:
        i. SQL (relational) → Tables & Rows → Good for 'ACID' properties (Atomicity, Consistency, Isolation, Durability) → Many companies like banks prefer to use this → Examples: MySQL, Aurora, RDS etc → Strongly supports ACID transactions
        ii. NoSQL (non-relational) → Collections & Documents/Objects/Key-value pairs → E.g. DynamoDB → Only some NoSQL DBs support ACID transactions (e.g. DynamoDB)
        iii. Static schemas tend to be associated with SQL DBs, whereas dynamic schemas tend to be associated with NoSQL DBs.
        iv. Reference: https://www.prisma.io/dataguide/intro/intro-to-schemas
    c. In DynamoDB, your data is split & stored into many partitions (see actual lecture slide diagram for more info). When you search for data in DynamoDB, it will search for the appropriate partition that contains the data you are trying to query.
    d. For DynamoDB, we can search/query for the data based on two main ways:
        i. Partition key + Sort key (Main way)
        ii. Local or Global Secondary index (Alternative way – local means query for data on **a/one** specific partition VS global means query for data across **all** partitions).
        iii. Each secondary index is essentially a second table that's created by DynamoDB for you to query/scan your data.
        iv. Local SI → same partition key, different sort key. Global SI → PK & SK different

e. To simplify, imagine you have a library of books, and you have a catalogue to search for books based on 'Book Author' (Partition Key) & 'Book Title' (Sort Key). This is the MAIN way to search for books in the library.
f. Imagine one day, a customer wants to search for books by another way – e.g. via 'Language' of book (e.g. English/Japanese/Korean etc). This way of searching the books in the library is the ALTERNATIVE way (Secondary Index).
g. We will be learning how to use **AWS Python SDK (Boto3)** to interact with DynamoDB in this course. There are other methods such as using **aws cli** or **NoSQL Workbench** that we can use to interact with DynamoDB (but need to download it).

3. Module 8 (Processing your database operations):
    a. There are two types of primary keys in DynamoDB:
        i. Simple primary key (partition key)
        ii. Composite primary key (partition key + sort key)
        iii. Usually we will be querying data using the primary key (either one above).
        iv. Reference: https://aws.amazon.com/premiumsupport/knowledge-center/primary-key-dynamodb-table/#:~:text=There%20are%20two%20types%20of,partition%20key%20and%20sort%20key.
    b. We can either query or scan the database in DynamoDB. Differences are:
        i. Query – faster; only looks through those data that matches that query/condition you've sent (SOME data looked through only)
        ii. Scan – slower; looks through ALL the data in the database
    c. In summary, this module goes through the various AWS cli commands that we would be using to interact with DynamoDB programmatically.
    d. DynamoDB caching options:
        i. Think of this way, instead of a user accessing your DB for data all the time (e.g. take milliseconds), if we have a cache (aka a copy) of your DB, instead of accessing your actual DB, the user can access the cached DB and this will improve latency time to microseconds even!
        ii. In short DB caching helps to improve speed of accessing data (improve latency) & reduce costs (won't have to keep accessing the actual DB).
        iii. Two options:
            1. Amazon DynamoDB Accelerator (DAX) - cheaper
            2. Amazon ElasticCache (memcache or redis) → FYI, can read up on a) write-through method (data gets updated in DB also gets updated in the cached DB/cache) vs b) lazy-loading method (data updated in DB but will not get updated in cached DB/cache).
    e. Lab 3 (Develop Solutions using Amazon DynamoDB):
        i. In this lab, we will be learning how to use AWS Python SDK (Boto3) to create & interact with a DynamoDB instance.
        ii. In short, writing Python code to interact with DynamoDB using Boto3 SDK
4. Module 9 (Processing your application logic – AWS Lambda):
    a. We will learn about AWS Lambda (serverless in nature – AWS manages it for you)
    b. Event source (e.g. Cloudwatch events, Cron jobs, S3 bucket) → invoke Lambda function (containing your application logic) → For example, we hit other AWS resources or hit other API endpoints → get a response to send back to the client.
    c. AWS Lambda runtime lifecycle:
        i. Event →

ii. Triggers lambda function →
iii. Create/Unfreeze the lambda runtime environment:
    1. Cold start: Download Code & Initialize environment (e.g. Node.js env)
    2. Warm start: Invoke the application logic/code & Shutdown after its done

d. Every AWS Lambda function is essentially a handler function which takes into two arguments – an event object (contains information from the invoker) and a context object (contains information about the runtime env).
e. The handler function contains your main application logic.
f. For more info: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html
g. For AWS Lambda, remember 2 main permissions:
    i. Invocation permissions (who can invoke your lambda?)
    ii. Processing permissions (after your lambda is invoked, is it allowed to access certain AWS resources?)
h. Various AWS lambda optimization techniques:
    i. Reduce cold start time (e.g. by regularly executing the lambda function to keep it "warm" and "not cold")
    ii. Concurrency execution
    iii. Reduce function duration
    iv. Memory allocation
i. Lab 4 (Python) - Develop Solutions Using AWS Lambda:
    i. The main objective of this lab is to learn how to create a lambda function, add processing logic (your main application logic) in your lambda function, then publish the lambda function, and then learn how to invoke it.
    ii. All these are done using the **AWS CLI** & **AWS Python Boto3 SDK**.
    iii. In this practical, we **programmatically invoke the lambda function** (which was already created, added with processing logic, published/deployed) with the help of an **event.json** object. Alternatively, we can also **manually invoke the lambda function** on AWS Console.
    iv. In short, we have an event object (event.json file) that looks something like:
        1. { "UserId": "test", "NoteId": "1", "VoiceId": "John" }
        2. Then we use aws cli command: 'aws lambda invoke \ --function-name <NAME_OF_LAMBDA_FN> \ --payload fileb://event.json response.txt'
        3. That aws cli command essentially invokes a lambda function of a particular name, and triggers it with a payload/event (event.json), and when successful it will generate a response.txt file locally
        4. In this example of the practical, based on our lambda_handler(event, context) function, it essentially contains logic which grabs a note record from DynamoDB → process that note & converts it into a MP3 audio file using AWS Polly → then uploads that MP3 file to S3 bucket → before FINALLY generating a pre-signed URL that is packaged into a response.txt file → Then user can open that file manually to find that presigned URL so as to access/download that particular MP3 file.

5. Final URL generated which when accessed will download an MP3 file: https://tinyurl.com/bdcwz3kj (converted using TinyURL as original URL was too long)

5. Module 10 (Managing the APIs – Amazon API Gateway):
    a. Amazon API Gateway is a service provided by Amazon to essentially "Create, Manage, and Deploy APIs".
    b. Amazon API Gateway offers three main services:
        i. HTTP API (Similar to ii – but AWS says this option faster/better latency/cheaper)
        ii. REST API (Similar to i actually)
        iii. WebSocket APIs (client apps & backend apps communicate in REAL-TIME → e.g. messaging/chat apps like WhatsApp/Telegram)
        iv. All three services in short:
            1. User → submit request to a particular backend API endpoint → forward request to API Gateaway → which then it will hit the respective lambda function for instance → then send response to client
    c. Can use either **aws cli** or **AWS Console** to create the AWS API Gateway instance/layer
    d. Lab 5 - Develop Solutions Using Amazon API Gateway:
        i. The main purpose of this lab is to learn how to create & deploy an Amazon API Gateway layer that will process requests from the user/client, then route that request to the appropriate backend AWS services (e.g. Lambda functions in our use case), which will retrieve/insert/update/delete the necessary record or data in the database, and then process those records into a JSON response or appropriate format which will then be sent to our client.
        ii. We will also earn how to configure our API layer to allow CORS, and add various other config settings using the AWS Console.
        iii. Summary:
            1. Create a Amazon API Gateway layer/instance using the AWS console website
            2. Name your API layer
            3. Add a resource to your API layer (a resource is like an endpoint – e.g. /notes endpoint)
            4. Add a method to that resource/endpoint (e.g. GET method to /notes)
            5. When the client sends a request to /GET notes → it will forward the request to our API layer (method request) → After which, AWS API gateway can further modify that request (Under 'Integration Request') by applying certain mapping templates to map/transform that request body into an appropriate format required by our backend services.
            6. The reverse is true as well → the response received from our backend services, can be transformed by various mapping templates (under 'Integration Response' in AWS Gateway console) into an appropriate response format which will then be forwarded to our client subsequently. Through this, we can reduce the amount of

redundant data sent back to the client, thus reducing the size of the response sent back to the client. This helps to improve latency, performance & reduces our data transfer (AWS) costs.

7. These mapping templates are written using AWS's Velocity Template Language (VTL).
8. We can also configure /POST methods on AWS API Gateway, and enforce a Model (essentially a data schema that validates the shape or format of our payload), and if the data structure/shape of the payload is correct, then we invoke our AWS backend or lambda services.
9. This is done using 'Models' to create the model schema of our expected JSON request body. And then going to the /POST – Method Request interface, and we edit the 'Request Validator' settings to 'Validate Body'. Then we click below on the 'Request Body' tab & tag/add the appropriate Model we've created earlier.
10. To enable CORS in AWS API Gateway, click on 'Actions' button > 'Enable CORS' > Select the necessary settings you need (i.e. which routes you want to enable CORS) > Click 'Ok' to save the settings.
11. To deploy the API layer, similarly click 'Actions' > 'Deploy API' > Choose 'New Stage' > Key in the name of your API stage name (e.g. dev/prod) > 'Deploy'.
12. Deployed API URL: https://e203cyqg8e.execute-api.ap-northeast-1.amazonaws.com/Prod/notes (Link will not work once lab session ends FYI).

iv. For more info, please refer to the actual lab or refer to lecture notes.

6. AWS Quiz for Day 2:
   a. Got **second place** for the Day 2 AWS Quiz 😊