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

Course objectives:

1. Configure IAM permissions to support your development environment. (Setup env)
2. Design, diagram, build & deploy a cloud-native application using AWS SDKs. (Developing)
3. Monitor and maintain an application by using AWS resources. (Monitoring)

Course layout:

1. This course is split into 2 parts – theory & practical (labs)
2. https://online.vitalsource.com/reader/books/200-DODEVA-43-EN-SG-E/pageid/0 (Student Guide - Theory lecture slides/notes; Can download the app to have the book valid for 3 years)
3. https://us-east-1.student.classrooms.aws.training/class/eNLEW7WXzUi6waHTdF4AEE (Labs link - Practical labs)

Content:

For details – pls refer to the actual lecture notes/slides

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

1. Module 1 (Course Overview)
    a. We will be learning how to use various AWS resources & SDKs in this course
    b. We will be going through the theory & doing practical labs to apply what we learnt
2. Module 2 (Building a web app on AWS):
    a. Various AWS resources which we will learn:
        i. AWS IAM (use this to manage access to AWS resources securely
        ii. Amazon S3 (We will use this to host our front-end website, and use it to store user files)
        iii. Amazon DynamoDB (Serverless DB to store user data)
        iv. Amazon Lambda (will contain our business logic to do CRUD operations)
        v. Amazon API Gateway (creates an API layer for user to interact with our business logic hosted on aws lambdas or in servers)
        vi. Amazon Cognito (used for authentication & authorization of users)
        vii. Amazon CloudWatch & AWS X-ray (Performance monitoring)
3. Module 3 (Getting started with development on AWS):
    a. Low level APIs vs High level APIs:
        i. Low level → Dives deep into the nitty gritty logic
        ii. High level → More abstraction of logic
    b. When to use low level vs high level apis?
        i. Low level → Need for performance & optimization, if you need higher control of your application
        ii. High level → If just need something quick and simple
    c. Setting up your AWS IDEs (e.g. setup VSCode or other code editor to make it ready for you to start developing your apps using AWS SDKs):
        i. In this course, we will be learning to use AWS Cloud9 (a cloud based IDE)
4. Module 4 (Getting started with permissions):
    a. In summary, in AWS:
        i. Users – are like you & me (individual users)
        ii. User groups (are made up of individual users)

iii. Role - Each user has their own role which then will have different policies/permissions
iv. Policy/Permissions – the ability/rights to use a particular AWS resource

b. AWS Policy are like JSON files (main thing “Effect”, “Action”, “Resource” keys):
   i. Effect – allow or deny access to that AWS resource
   ii. Action – which action to do to that resource
   iii. Resource – which AWS resource you are allowing access to
   iv. More information: https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_effect.html

c. Proceeded to do Lab 1 (Configuring the IDE environment for AWS development):
   i. All Labs Link: https://us-east-1.student.classrooms.aws.training/class/eNLEW7WXzUi6waHTdF4AEE
   ii. Lab 1 link: https://us-east-1.durian.bkr.team.aws.training/session/iVQ1aHEgmPQkEUNhhbhVVb?locale=en-US&reference=eNLEW7WXzUi6waHTdF4AEE%3A%3A814fc721-977a-4bab-96a7-261991ef3bc1#task-1-review-the-development-environment

d. Summary of Lab 1:
   i. In short is learning how to use the AWS CLI (various command lines for AWS)
   ii. AWS STS (AWS Security Token Service), a service provided by AWS to provide temporary credentials to users so that they can have temp access to your resources. The temp access automatically expire after a certain period; keeps your resources secure.
   iii. Examples of various AWS CLI commands (From ‘AWS CLI Reference’):
      1. python –version
      2. aws –version
      3. aws configure
      4. aws sts get-caller-identity
      5. aws s3 ls
      6. bucketToDelete=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `deletemebucket`) == `true`] | [0].Name') → creating a variable called bucketToDelete which contains a query to delete a s3 bucket later
      7. aws s3 rb s3://$bucketToDelete OR aws s3 rb s3://$bucketToDelete –debug
      8. policyArn=$(aws iam list-policies --output text --query 'Policies[?PolicyName == `S3-Delete-Bucket-Policy`].Arn') → creating a variable called policyArn which will be used to query AWS on a list of policies of that name in that account
      9. aws iam get-policy-version --policy-arn $policyArn --version-id v1 (get a particular policy with that query & review it if it has the necessary permissions – recall ‘effect’, ‘action’, ‘resource’)
      10. aws iam attach-role-policy --policy-arn $policyArn --role-name notes-application-role (attaching a policy to a particular role)
      11. aws iam list-attached-role-policies --role-name notes-application-role (check if that policy has been added to your role)

e. What is difference between AWS CDK and SDK?

i. CDK (Cloud Development Kit) is Infrastructure as Code, which means you can write code to **create** AWS resources like API Gateway, DynamoDB or Lambda.
ii. On the other hand, SDK (Software Development Kit) helps to **interact** with these AWS resources for ex: fetch the items from DynamoDB or list the contents of S3 bucket.

5. Module 5 (Getting started with Storage):
    a. Types of storages:
        i. Block storage – takes your data and "chop" it up into small blocks/volumes and saves it.
            1. Example of block storage – SSD, HDD drives, AWS Elastic Block Storage, AWS Instance store
            2. AWS EBS (persistent) vs AWS Instance store (non-persistent…recall AWS EC2 Instance they each have their own non-persistent instance storage)
            3. Think of it this way an AWS EC2 instance (a server instance) has their own instance store (the default HDD/SDD in the server instance) which is non-persistent in nature, we can also attach AWS EBS (additional SDD/HDD drives) to get more storage space, and this type is persistent.
        ii. Object storage:
            1. Stores your objects – e.g. MP3 songs, MP4 videos, photos, pictures
            2. Examples of object based storage – Google Drive, One Drive, Dropbox, AWS S3 buckets
            3. Tends to be bit slow…but gets the job done
            4. WORM (Write Once Read Many) → Once you upload (write) the file once, many people can then read (access) the file.
            5. Properties of AWS S3 (Simple Storage Service):
                a. Serverless storage
                b. Create a bucket to store your file
                c. Security (Will it be a public or private bucket?)
                d. Encryption – various encryption types:
                    i. SSE – S3 (AWS manages the encryption/decryption for you)
                    ii. SSE – KMS (Manage the encryption/decryption yourself)
                    iii. SSE – C (Manage the encryption/decryption yourself ;more complicated)
                e. Versioning (we can enable or disable it): E.g. at first we add photo.jpg, next we reupload photo.jpg. The first photo.jpg is version 1, the latest new photo.jpg is like version 2. Basically can 'get back' the original v1 photo.jpg (if needed). But take note, versioning of the files COST MONEY…may lead to unnecessary costs. We can either clear your old versioned files manually or use Life Cycle Policies (FYI).
                f. Access points or ACL (Access Control List) – e.g. a S3 bucket can have 3 different access points (e.g. Finance department can only READ the bucket, Operations department can

READ, CREATE the bucket, Management department can CREATE,READ,UPDATE,DELETE files in the bucket)

iii. File storage:
    1. E.g. Multiple EC2 instances in the same VPC (same network) share the same filesystem (file storage)
    2. Examples of file storage protocols – SMB, NFS, CIFS
    3. Network file system (NFS), server message block (SMB) and common internet file system (CIFS) are all file access storage protocols
    4. Used to organize & store data on a computer hard drive or network
    5. Example of file storage: AWS EFS (Elastic File System)

iv. Database:
    1. SQL → Tables & Rows → Example: RDS, Aurora, Maria, PostgresSQL, MySQL, Oracle etc
    2. NoSQL → Key value pair, objects/documents {…} → Example: DynamoDB, MongoDB etc

b. AWS S3 has different storage classes → Need to set the appropriate storage class based on your own usage pattern → Selecting the correct or optimal storage class of your S3 bucket can help you save unnecessary costs incurred.

c. For example:
    i. 'Standard' storage class: Use this if you frequently access the file (tends to be the 2nd highest cost). Fast latency (milliseconds). For 'Standard', we do not get charged when retrieving the data.
    ii. 'Standard-IA' (IA mean Infrequently Accessed): If you are not going to access the file much, use this. Fast latency too; but much cheaper than 'Standard' class. BUT when you retrieve data from that bucket, you WILL be charged! SO eventually if you realize you frequently access data using this option, you would be charged more money than option 1 standard class.
    iii. 'One Zone-IA': Similar to ii Standard-IA, but instead of 3 AZs for ii, this option only has 1 Availability Zone.
    iv. 'Glacier' classes (Got 3 other glacier subclasses/subtypes): The term 'glacier' means it takes quite some time to fetch and get the data.
        1. Glacier Instant Retrieval → miliseconds to fetch data → Very expensive ($$$)
        2. Glcarier Flexible Retrieval → mins to hours to fetch data → $$
        3. Glacier Deep Archive → much more hours (longest) to fetch data → least exp amongst the 3 cause its slowest ($)
        4. These options are usually used for retrieval of archived data/files (basically files or data that's not accessed much – e.g. your degree/transcript; all these kind of files you only look at once a year or once every few years for instance).
    v. 'Intelligent-Tiering': The class of your S3 storage will change based on your data usage patterns. In short, AWS will change your S3 storage class based on their analysis of your data/file usage patterns. E.g. you don't use much → they shift your storage class to 'Glacier' etc. If you use a lot, then AWS will shift your S3 storage class to 'Standard'. <u>In short, lecturer recommended this, especially if you are not sure of your current or future file/data usage patterns.</u>
    vi. And many more storage classes for S3 (can refer AWS docs if needed)…

    d. S3 bucket policies (same thing – 'Effect', 'Action', 'Resource' + new one 'Principal' key). If 'Principal': '*' (wildcard means everyone can do that particular effect, action on that resource – recall EAR + P).
6. Module 6 (Processing your storage operations):
    a. S3 can do SINGLE UPLOAD or MULTIPART UPLOAD
    b. E.g. Small file size (e.g. <= 5GB) → Single upload
    c. Large file size (e.g,. <= 5TB) → we can do multi part upload → like split into 3 different parts → and upload them
    d. Best practice: If you need to manage large files, use the MULTIPART UPLOAD
    e. How to do multipart upload (Use AWS SDK): https://docs.aws.amazon.com/AmazonS3/latest/userguide/mpu-upload-object.html
    f. We can grant temporary permission to our S3 buckets by using pre-signed URLs
        i. User requests link to upload or download
        ii. An Amazon EC2 instance containing the logic/app code to generate the presigned URL.
        iii. Returns presigned URLs to use to grant PUT or GET access, and specifies expiration date, and valid to which objects in the bucket
        iv. User then can access the URL to GET or PUT object in your s3 bucket
        v. Can refer slide 189 for more information
    g. We can also use S3 bucket to host static websites. Cannot use to host dynamic websites. In short, S3 can be used as a storage service & web hosting service.
    h. Static website vs Dynamic website:
        i. Static website only has FIXED content
        ii. Dynamic website means that the content is generated in real time → e.g. content obtained or fetched from a database/CMS etc
        iii. So S3 not just used for storing files & file backup, we can also use it to store the static website files & then host our static websites subsequently.
    i. Lab 2 (Developing solutions using Amazon S3 & AWS Boto3 Python SDK):
        i. Setup config file → verifyBucketName → createBucket → verify bucket created → upload files → host your static website using S3 bucket
        ii. Hosted website from the lab using AWS S3 & AWS CLI: http://notes-bucket-melvin-123456.s3-website-ap-northeast-1.amazonaws.com/
7. Recap of the day:
    a. AWS STS (Security Token Service) – can provide temporary security credentials to access your AWS resources. More info: AWS STS is a web service for requesting temporary, limited-privilege credentials for AWS Identity and Access Management users or for users that you authenticate (federated users).
    b. AWS presigned URLS – provide temporary access to your S3 bucket's objects
    c. Reference: https://wa.aws.amazon.com/wat.concept.sts.en.html#:~:text=AWS%20Security%20Token%20Service%20(STS,you%20authenticate%20(federated%20users
    d. https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html
8. AWS 'Developing on AWS' Day One Quiz:
    a. Got **first place** in my class for the quiz 😊