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

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 11 (Building a Modern Application):
    a. Monolith application – One single codebase, tightly coupled, while ok for smaller companies, this way of doing things is no longer practical in bigger companies since if there's an issue, it can be difficult to debug as well, technology stack tends to be restricted/limited (e.g. if original lead decides to use Java for this monolithic codebase, rest of developers have to follow – since ONE codebase only).
    b. Instead of using the "older" approach of monolith application, more modern businesses are moving towards microservices architectures
    c. Microservice architecture/application – Multiple codebases, loosely coupled, easier to debug if things go wrong as we can easily pinpoint and find out which microservice is causing the issue and fix from there (the comparison to monolithic is that in monolithic, need to search through the whole codebase to debug - troublesome). Microservices are more scalable as well, and with each microservice app we can choose and have MORE FREEDOM OF CHOICE to choose our own tech stack.
    d. AWS trainer recommends decoupling your monolithic application and break it up into smaller microservices, in a phased approach (strangler design pattern). Aka transit to microservices INCREMENTALLY…not one whole shot migrate from mono to microservices, as it can cause culture shock and *may* cause application breakages/outages (if not migrated properly).
    e. We will also learn the concepts of 'steps functions' in AWS.
    f. In summary, AWS 'Step Functions' provides a visual interface/service to manage your AWS workflows. You will use the AWS Console to use step functions in AWS.
    g. AWS 'Step functions' documentation: https://aws.amazon.com/step-functions/#:~:text=AWS%20Step%20Functions%20is%20a,machine%20learning%20(ML)%20pipelines.
2. Module 12 (Granting Access to your Application Users):
    a. Learn about AWS Cognito to explore the authentication/authorization process so as to grant access to users of your application
    b. Amazon Cognito provides an easy to use service for your users to sign up, sign in and manage access to your web/mobile application.
    c. In short AWS Cognito handles the authentication & authorization workflow for your application.
    d. Take note IAM and AWS Cognito are different, even both of them are used for authentication/authorization purposes:
        i. IAM is used for managing access to your AWS resources (Think of it more of Infrastructure Accessibility)
        ii. AWS Cognito is used for managing access to user sign up, user sign in, user data management (Think of it more of like Web/Mobile Application or Functionality Accessibility)
    e. AWS Cognito can be used in 2 ways:

i. User pool (Authentication - Verify the user identity directly in a list of users collected from Amazon Cognito or Social Identity Providers. Think of it like 'Sign up/Sign In' → Users information collected in a pool → which we will use to verify users sign in, in the future)
ii. Identity pool (Authorization – Does user have the proper permissions to access the AWS resources? The identity pool provides temporary IAM credentials to access the AWS resources. Think of it more of granting temporary, limited-privilege AWS credentials through the aid of external SSO providers -e.g Facebook, and if they verify OK, then we issue the necessary credentials to access our resources)
iii. Both User pool & Identity pool can be used together actually

f. How AWS Cognito works under the hood (User Pool):
i. User sign in
ii. AWS Cognito SDKs checks the User pool to authenticate/verify the user if he/she exists
iii. Once user is authenticated/verified, AWS Cognito issues the JWT token that can allow you to access your own server-side resources or AWS API Gateway endpoints.
iv. The JWT token can also be sent to the AWS (Identity pool) to exchange for temporary security AWS/IAM credentials
v. Those AWS IAM credentials will then be used to check & allow the necessary access to the required AWS resources
vi. So yes, AWS Cognito does use IAM under the hood.

g. How AWS Cognito works under the hood II (Identity Pool):
i. User authenticates through external identity providers (e.g. Facebook, Google, SAML SSO)
ii. External Identity provider provides the JWT
iii. App sends JWT to AWS Cognito (Identity Pool) to verify
iv. Once verified and all ok, AWS Cognito issues the temporary AWS credentials tied to the IAM role
v. User/App (client) sends and uses the AWS credentials to access your AWS resources.

h. While both approaches seem similar, the main difference between User Pool and Identity Pool is that:
i. User Pool: AWS STORES the user profiles into a collection/pool which we will verify against. Those profiles could be gathered from a) manual user sign ups in your web app OR b) social identity providers – e.g. Facebook SSO sign in.
ii. Identity Pool: Identity pools DO NOT STORE any user profiles. Authenticate users through external identity provider directly but no storage – e.g. social identity SSO (example – Facebook SSO).

3. Module 13 (Deploying your application):
a. AWS SAM is an open-source framework that is used to deploy serverless applications.
b. SAM stands for Serverless Application Model.
c. AWS SAM generates a CloudFormation template that handles the creation, updating and deletion of your application.
d. When to use AWS SAM vs CloudFormation?
i. CloudFormation – for more complex apps

    ii. AWS SAM – for more simple apps; more for serverless apps usually
e. Format of AWS SAM: YAML (Yet another markup language) format
f. This concept here in this module is essentially teaching how to use AWS SAM:
    i. AWS SAM has a .**yaml template file** which is a CloudFormation template containing information about the application. This is actually very similar to the ecs.yaml (that config file at work) – which is essentially is a CloudFormation template that helps to build & deploy your apps. Or for clearer reference, that template file for deployment in SAM is similar to the 'serverless.yml' file I have in my personal twitter coding quotes project.
    ii. After making sure you have the template file, we can then install & use the **aws sam cli** to deploy your applications subsequently.
    iii. For info on how to use AWS SAM, pls refer to the AWS docs for the various steps: https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorial-lambda-sam-template.html
g. Lab 6 (Capstone - Complete the Application Build):
    i. In this lab, we will learn how to set up Amazon Cognito to authenticate & authorize users to use your APIs.
    ii. We will be using AWS Console, aws cli, aws cognito SDK to learn about Cognito.
    iii. In summary:
        1. Set up Amazon Cognito User Pool (this is the authentication * authorizer mechanism we are using)
        2. Before users can access your APIs which was setup in AWS API Gateway, they need to authenticate/authorize themselves first.
        3. In short, we will be creating an Amazon Cognito authorizer (Go to API Gateway > Authorizers)
        4. Once that is done, secure your existing API endpoints with that authorizer. To do this, go to API Gateway, click on your particular endpoint – e.g. /notes (GET), go to 'Method Execution', go to 'Settings' then edit the 'Authorization' field.
        5. Also need to edit your endpoint's 'Integration Request' > 'Mapping Templates' > Edit the template to something like { "UserId": "$context.authorizer.claims['cognito:username']" }. This probably means the AWS Cognito needs to forward a request object that looks something like this - { "UserId": "someValueFromAuthorizer" }
        6. In short, that AWS Cognito Authorizer will validate the user's identity first before allowing them to access our API endpoints/backend services hosted via API Gateway layer.
        7. Additional steps: see next point
    iv. Summarized steps:
        1. Step 1 (Configure AWS Cognito) > Step 2 (Configure API Gateway to use Amazon Cognito as an authorizer) > Step 3 (Create remaining API resources/endpoints using swagger file) > Step 4 (Configure frontend web app) > Step 5 (Test web app functionality).
    v. Notes app:
        1. http://labstack-0c817d3c-971f-4116-be2d-c3-pollynotesweb-ahwogr9mn8vk.s3-website-us-east-1.amazonaws.com/Login (Login

page – but after lab ends, won't be able to login to access the notes app)

vi. Reference: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-getting-started.html

vii. Summarized Architecture diagram of how the entire app works: https://online.vitalsource.com/reader/books/200-DODEVA-43-EN-SG-E/pageid/7 (You need authorized access provided by the AWS trainer before you can access the student e-book)

4. Module 14 (Observing your application):
    a. Observability is important in modern software development.
    b. AWS provides various tools for us to observe our apps – i.e. collect, assess, correlate data so that we can quickly understand & resolve issues from a system wide point of view.
    c. Examples of AWS resources for observability:
        i. AWS CloudWatch
        ii. AWS X-ray
    d. We will learn how to use AWS Python SDKs to use CW and X-ray for our labs (Boto3 SDK for CW, X-Ray SDK for AWS X-ray)
    e. Lab 7 (Python - Observe the Application Using AWS X-Ray):
        i. In this lab, we will use AWS X-Ray to observe the operational state of the application.
5. Module 15 (Course Wrap-Up – Things an AWS Developer should know):
    a. IDE: Cloud9, VSCode
    b. SDK (methods to interact with AWS resources), CDK (methods to create AWS resources - Infrastructure as Code)
    c. API – low level APIs vs high level APIs
    d. IAM:
        i. Principles of IAM: Users, Roles, Services, IdP (Identity Providers)
        ii. Policies have two types: Identity based policy (EAR) vs Resource based policy (EAR + P)
        iii. Policies are in JSON format.
        iv. We have IAM Groups as well
    e. Storage:
        i. Block (EBS, instance, SSD/HDD etc)
        ii. Object: S3 bucket (Access points, Encryption – 3 types SSE-S3/KMS/C, Private or Public, AC, Lifecycle management, Versioning, Multi-part upload, Different classes – e.g. Std/IA/One-zone IA/Glacier etc)
        iii. FS (EFS, FSx)
        iv. Databases (Relational vs Non-relational – e.g. RDS vs DynamoDB)
    f. AWS Lambdas:
        i. Serverless (AWS manages the server for you 😊)
        ii. There are other types of computing resources by AWS (but those you need to manage the servers yourselves – e.g. AWS EC2, ECS/EKS etc)
        iii. How to optimize your lambdas:
            1. Warm restart
            2. Memory allocation
            3. Code size (smaller)

iv. Lambdas have a handler function (e.g. handler() in Nodejs & handler_function() in Python)

g. AWS API Gateway:
    i. Creating and managing APIs for you in AWS
    ii. Three types:
        1. HTTP APIs
        2. REST APIs
        3. Websocket APIs
    iii. Also does authentication & authorization (with AWS Cognito as authorizer) & also can do throttling (have an API limit – reduce API access) & can link up to your other backend AWS services (e.g. can connect to AWS Lambda)

h. DevOps:
    i. AWS have a bunch of other devops or CI/CD tools to make the software development cycle much easier (automation)
    ii. E.g. CodeCommit, CodeDeploy, CodeBuild, CodePipeline etc

i. Application Deployment:
    i. Use AWS SAM
    ii. SAM templates we can either use:
        1. YAML template (easier)
        2. JSON template

j. Authentication & Authorization:
    i. Use AWS Cognito
    ii. Can use this service by AWS to sign up, sign in or register to your app
    iii. Two pools:
        1. User pools – have a directory of users which you can verify, once verified, will give us a JWT token. The JWT token can be passed to the identity pool to get temporary AWS security credentials to allow users access to your AWS resources
        2. Identity pool – as mentioned above & in notes

k. AWS Step Functions:
    i. This is essentially a visual flowchart or service provided by AWS to show you how your AWS workflow works

l. Observability:
    i. Observability is important because we need it to quickly detect & resolve issues in our apps, in the event an error occurs
    ii. Two ways: AWS CloudWatch & X-ray

6. Tips for the 'AWS Certified Developer Associate' Exam:
    a. For non-native English speakers, you can get extra 30 mins if you click on 'Request Exam Accommodations'.
    b. Can practice various mock papers/questions online:
        i. https://www.examtopics.com/exams/amazon/aws-certified-developer-associate/
        ii. https://www.examtopics.com/exams/amazon/aws-certified-cloud-practitioner/
    c. For more information: https://aws.amazon.com/certification/