AWS 'Architecting on AWS' training course DAY THREE (24/05/2023 – 26/05/2023)

Course objectives:

1. Identify AWS services, compare features, and explore best practices to architect resilient, secure, and highly available IT solutions on AWS
2. The end goal is to be able to design & create an AWS architecture diagram/solution after finishing the course – can refer to slide 19 of the lecture notes for the full diagram

Course layout:

1. This course is split into 2 parts – theory (lecture slides) & practical (labs)
2. Additional learning resources: https://explore.skillbuilder.aws/learn/course/external/view/elearning/8319/architecting-on-aws-online-course-supplement

Content:

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

1. Module 10 (Networking 2)
    a. VPC endpoints:
        i. Allow access to AWS services without an internet gateway, NAT gateway or public IP address
        ii. E.g. Your private EC2 instance can access your Amazon DynamoDB (positioned outside your VPC) via the VPC endpoint
        iii. Helps to ensure privacy, don't need to connect to public internet to access AWS resources
        iv. 2 main types:
            1. Gateway VPC endpoint: Specifically for Amazon DynamoDB and/or Amazon S3
            2. Interface VPC endpoint: Supports more services than gateway endpoints (DynamoDB, S3 and more services – e.g. AWS Systems Manager); More flexibility but higher costs as well – additional feature, we can even connect our on-prem servers to connect to this interface endpoint to connect to our AWS resources as well (the same can't be done for gateway endpoint)
            3. In practice, usually both types of VPC endpoints are used at the same time.
            4. Take note if your AWS resources live within your VPC, there is NO NEED for these VPC endpoints
            5. Only need to use this solutioning if your AWS resources live outside your VPC!
    b. How to establish private connection between 2 VPCs:
        i. Introduction to VPC Peering
            1. VPC peering establishes network connection between 2 VPCs
            2. Intra-region and inter-region and cross-account
            3. No transitive peering relationship (e.g VPC A ←→ VPC B ←→ VPC C). Just because A is connected to B, and B is connected to C… this setup DOES NOT mean A can directly communicate with C.
    c. Hybrid networking (How to connect on-prem network to the AWS cloud):

i. AWS Site-to-Site VPN (VPN connection)
ii. AWS Direct Connect (Optical Fiber connection/Fiber link)
iii. AWS Transit Gateway: Provides a gateway that allows us to directly connect to multiple VPCs (up to 5000 VPCs – slide 478 to 480)
iv. Basically, we can connect our on-prem data centers to the cloud using either a) VPN connection or b) Optical Fiber connection or c) AWS Transit Gateway

2. Module 11 (Serverless)
    a. Serverless: No infrastructure to provision/manage, Scales automatically, Pay per use, Security + Highly Available computing power
    b. A lot of serverless solutions:
        i. Amazon API Gateway (provide a gateway to all your public available endpoints),
        ii. AWS Lambda (can be triggered when there are events happening in another AWS resource – e.g. a new object is added in SQS queue → can trigger Lambda event),
        iii. AWS Fargate (manages containers that contain your microservices/logic),
        iv. AWS SQS (create a message queue for computing resources/different services to communicate with one another),
        v. AWS SNS (push notifications to users – e.g. send SMS or emails),
        vi. S3,
        vii. DynamoDB,
        viii. Aurora Serverless,
        ix. AWS Cognito (authentication of users)
        x. and many more…
    c. An example of a serverless architecture can be found in slide 504:
        i. In summary: POST request → API Gateway → Amazon SQS (request goes to a message queue which awaits to be processed by your worker service) → Worker containers in AWS FarGate (manages your containers – that contain your microservices) → writes the DB (e.g. DynamoDB) → response sent back to your container/microservices → once get response, worker service prompts/calls AWS SNS (push notifications – e.g. send SMS to client) → SMS sent to client
        ii. This example serverless architecture can be for a real-life scenario – e.g. new credit card promotion where user signs up
    d. Additional notes:
        i. AWS SQS – we can also have a **Dead-letter queue** (basically those queue messages that are not processed or have some errors/issues – they will go here for our analysis/debugging in future). Uses **POLLING**.
        ii. More info on AWS SQS - https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-queue-types.html
        iii. AWS SNS:
            1. In summary is made up of Publisher → Topic → Message → Subscriber
            2. PUSH is the keyword! Uses **PUSHING**.
            3. The subscribers are NOT PULLING for the message. They are being PUSHED the message (e.g. PUSH notifications).

iv. AWS Kinesis is a data-stream manager. In short, data produced by producers can be supplied into AWS Kinesis (which manages those data produced and creates a data stream), and the data stream created can be consumed by your apps/AWS resources
v. AWS Step Functions – provides a user-friendly GUI interface to develop your AWS workflow. You can easily drag and drop various AWS resources to create your own AWS workflow. An AWS workflow
vi. In summary, AWS Step Functions is a visual workflow service that helps developers use AWS services to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning (ML) pipelines.

e. Lab 5: In short, we will be building a serverless architecture like this:
   i. User (add object) → S3 bucket → trigger event notification to SNS topic, which generates a message → SQS (poll our SNS to detect new message) create a queue message → that our Lambda fns POLL for and then invoke our Lambda functions logic → resize that object & add in another S3 bucket + monitor logs via CloudWatch
   ii. Slide 541 for the full architecture diagram

3. Module 12 (Edge Services)
   a. Amazon Route 53 → Handles the DNS resolution (translates domain names/URLs into IP addresses) → It's called "53" because DNS resolution uses UDP protocol which uses port 53 (FYI) → Amazon Route 53 can do Geolocation Routing also, which means we route the user's request to the relevant EC2 instance in a particular AZ based on the user's geolocation (e.g. user from USA → Route 53 → route to us-west-2 AZ EC2 instance VERSUS user from asia → Route 53 → route to ap-southeast-1 AZ EC2)
   b. Amazon CloudFront (CDN) → Provides CDN caching services, Deliver content faster to consumers based on geographical location, Protect against DDoS (Security)
   c. Lab 6 – Configure an Amazon CloudFront distribution with an Amazon S3 origin
4. Module 13 (Backup and Disaster recovery)
   a. Backup: Ensures that your data is recoverable → In short, take backups/snapshots of your database tables & ensure your AMI/Container images are available in other regions should a particular region fail
   b. Disaster recovery: After a major disaster, ability to get your applications & data back
   c. Lab 7 – Capstone Lab (Build an AWS multi-tier architecture): Utilize what was learnt over the past 3 days to design & architecture an AWS multi-tier system.
   d. For additional information, please refer to this AWS docs link:
      i. Talks about the 4 various disaster recovery options in AWS (e.g. Backup & Restore, Pilot Light, Warm standby, Multi-site active-active strategies)
      ii. https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html