AWS ‘Architecting on AWS’ training course DAY TWO (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. https://online.vitalsource.com/reader/books/200-ARCHIT-74-EN-SG-E/pageid/0 (Lecture slides link - Theory)
2. https://us-east-1.student.classrooms.aws.training/class/oa6PnQtm5WkBM7Gv3XdVaA (Labs link - Practical labs)
3. https://explore.skillbuilder.aws/learn/course/external/view/elearning/8319/architecting-on-aws-online-course-supplement (Additional learning resources - optional)

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 5 (Storage)
   a. Block storage (EBS), File storage (EFS - Linux, FSx - Windows), Object storage (S3)
   b. Think of file storage & object storage as abstractions of the original raw block storage
   c. The lecturer also mentioned other types of storage used in the industry – specifically ‘Tape Libraries’ (Tape Storage). This is basically used to store data (such as transaction history) offline – usually such data has to be stored for X number of years due to regulatory/compliance laws. More info: https://www.oracle.com/sg/storage/tape-storage/
   d. S3 does not live in your VPC → S3 is outside your VPC
   e. By default, only the owner of the S3 bucket can access it (private access). But we can configure it to have public access (but this is dangerous; avoid) or controlled access (restricted access – some people can access, some cannot).
   f. We can encrypt objects in S3 using a variety of solutions – SSE-S3, SSE-KMS, SSE-C (Server Side Encryption using S3, KMS or Customer provide their own keys).
   g. If your multiple EC2 instances need to use the same storage:
      i. Can’t use EBS – since an EBS block is usually attached to one instance only
      ii. S3 not built for file systems
      iii. Can use EFS (linux systems) or FSx (window systems) instead
   h. To sync up your on-prem data centres with your AWS data centres, can consider usage of AWS DataSync
2. Module 6 (Database Services)
   a. AWS offers a lot of DB services – RDS, Aurora, DynamoDB, DocumentDB, ElasticCache and so on
   b. Using AWS database services, we can:
      i. Have a standby DB instance (cannot use it to read transactions, can be used to sync up its content/data with another primary DB instance. If the primary DB instance fails, the standby DB instance will be synchronized and promoted

to the primary DB, while the original & failed primary DB instance will become the new standby DB instance).

ii. Or we can also configure a Read Replica DB instance (from a primary DB instance, we can do 'asynchronous replication' to replicate a 'Read Replica' DB instance in one region/other regions, which your other EC2 instances can access to. We can also promote the 'Read Replica' DB to the primary DB if the original primary DB fails.)

c. DynamoDB:

i. By default, AWS DynamoDB writes your data in different tables across different AZs.

ii. 2 modes: Eventually consistent read (ECR) vs Strongly consistent read (SCR)

iii. The main difference between those 2 modes is essentially how fast AWS syncs up database tables

iv. AWS automatically handles the sync-ing up of data across the different tables in the different AZs.

v. Refer to slide 300 for more info

d. Database caching:

i. When to cache?

1. Data that is expensive and/or slow to query (Expensive operations)
2. Frequently accessed data
3. Static data

ii. Cache hit → fetch from cache OR else if Cache miss → fetch from DB instead

iii. From AWS documentation on database caching: "The cache itself can live in a number of areas including your database, application, or a standalone layer."

iv. Reference: https://aws.amazon.com/caching/database-caching/

v. Slide 304's architecture diagram shows a Cache Cluster (this is an example of a distributed caching system) → this is probably a standalone layer for this particular solution

vi. Various caching solutions such as:

1. AWS DAX
2. AWS ElastiCache (uses either Memcached or Redis under the hood)

vii. Question: On a side note, a question asked was that "How would on-prem databases be able to communicate with a cloud-hosted/AWS hosted database?"

viii. Answer: In summary, there would be a VPN connection between the on-prem DB and the AWS hosted DB. Whitelisting of both the on-prem & AWS DB IP addresses through the organization's network/firewall.

ix. To migrate from on-prem DBs to the cloud, we can use AWS DMS (AWS Database Migration Service).

e. Lab 3 – Create a database layer in your Amazon VPC infrastructure:

i. Main point of this lab is to learn how to create a AWS Aurora instance in your private subnet of your VPC, and link it to a EC2 instance (which is also in your private subnet and is the only resource that can access the DB), and the traffic is routed to the EC2 instance by an ALB (which we have to create and configure also).

3. Module 7 (Monitoring and Scaling)

a. Reasons for monitoring infrastructure:

i. Get operational visibility and insight (operational health)
ii. Understand application performance
iii. Improve resource utilization (e.g. CPU utilization too high or too low for an EC2 instance)
iv. Security purposes

b. AWS Monitoring tools: AWS CloudWatch, AWS CloudTrail
c. AWS Scaling tools: AWS Load balancers
d. Different types of AWS Load balancers:
   i. Application load balancer (load balancing of **HTTP/HTTPS** traffic)
   ii. Network load balancer (load balancing of **TCP/UDP** traffic)
   iii. Gateway load balancer (load balancing of traffic based on **IP**)
e. Components of a Load Balancer:
   i. Listener 1 (e.g. listens to traffic – e.g. TCP port 80) → route to Target Group (contain various AWS Resources – e.g. your EC2 instances)
   ii. Listener 2 (listens to traffic – e.g. TCP port 3306) → route to another Target Group (containing other AWS resources)
f. AWS EC2 does auto scaling using Auto Scaling Groups → higher traffic, spin up more EC2 resources to handle the increased traffic → vice versa lower traffic, spin down EC2 instance to minimize resource utilization
g. We need to specify minimum number of EC2 instances to spin up and maximum number of EC2 instances to spin up in the Auto Scaling Group policy so that AWS knows how to auto scale up or down for you. Recall platform definitions.
h. Lab 4 - Configure high availability in your Amazon VPC:
   i. Deploying resources in a VPC such as NAT gateway, VPC routing, EC2 auto scaling groups, and Amazon Aurora DB clusters
   ii. Fault Tolerance: A system's ability to remain in operation even if some components used to build the system fails
   iii. Highly Available: The ability of a system to quickly recover from failures.
   iv. We must design systems that are both HA and FT

4. Module 8 (Automation)
   a. AWS CloudFormation:
      i. Write a CloudFormation template (Infrastructure as Code – IaC)
      ii. Then AWS automatically generates your AWS infrastructure/architecture based on your template
      iii. Benefits: Reusability – one template to generate your AWS infrastructure
   b. How AWS CloudFormation works under the hood:
      i. Create the CF template (JSON or YAML file)
      ii. Upload the template to AWS CloudFormation
      iii. Through a bunch of API calls done by AWS CF,
      iv. It transforms your CF template into a bunch of AWS resources (your architecture)
   c. Other options to automatically help you manage your AWS infrastructure:
      i. AWS Elastic Beanstalk
5. Module 9 (Containers)
   a. Containers are self-contained environments → contain your code, dependencies, and configurations into a single object.
   b. Slide 432 explains the differences between Bare Metal Servers, Virtual Machines, and Containers.

c. Main difference between the 3 is that:
    i. BMS has server hardware, server OS, on top of it, your system libraries and various apps
    ii. VM is basically server hardware, server OS, there's a hypervisor layer (this is the software/firmware that creates your individual VM). Each VM can have **their own guest OS**, libraries, app.
    iii. Containers is essentially sitting on top of server hardware, server OS, then we have the Container engine (e.g. Docker), which creates various containers – each container has the necessary libraries & app – **does not have their own OS (unlike VMs – instead using the underlying server infrastructure's OS instead).**

d. E.g. of containers: Docker, Kubernetes

e. Solutions provided by AWS:
    i. AWS ECS (Docker based) → operates in clusters
    ii. AWS EKS (K8s based) → operates in pods → can use together with AWS FarGate to help manage your infrastructure