**Week 2: Cloud Computing**

1. **What is cloud computing?**

* On demand
* Delivery of computer power, applications, databases, IT resources
* Through Cloud Service platform via Internet
* Pay-as-you-go price

1. **Key technological innovations that enable the creation of the cloud**

* Hardware support for virtualization
* Linux
* The high cost of running an IT organization
* XEN hypervisor

1. **Main motivators of cloud computing?**

* Lower start-up and maintaining cost
* Scalability
* Reliability
* Accessibility
* Platform

1. **Main Cloud Provider?**

* Amazon 60%
* Microsoft 30%
* Google 10%

1. **Cloud computing must have (Essential characteristics of Cloud Computing)?**

* On demand
* Self-serve
* Broadband network
* Resource pooling
* Rapid elasticity(co dãn, đàn hồi) (Scalability)
* Metered Service Model

1. **Cloud Service Models?**

* **SaaS (Software-as-a-Service): End-User Applications**
  + CRM, Email, virtual desktop, communication, games, …
* **PaaS (Platform-as-a-Service)**
  + Execution runtime, database, web server, development tools, …
* **IaaS (Infrastructure-as-a-Service)**
  + Virtual machine, servers, storage, load balancers, network ,…

1. **Core Cloud Service?**

* Compute (VMs)
* Storage (Block, Object, Ephemeral)
* Databases (Relational and NoSQL)
* Networking (redundant, virtual)
* Message queues (Scalability, core building block)
* Load balancers (scalability, core building block)

1. **Delivery models (Type of cloud)?**

* Public cloud: services offered by public provider
* Private cloud (On-premises): services offered in-house by corporate data center
* Hybrid cloud: services offered both in-house and by cloud provider
* **Hybrid Could is the most popular model (94% of respondents using)**

1. **Cloud opportunities?**

* Easy to get started
* Low cost
* Move Capital cost to Operational cost
* Use what you need, pay as you go pricing
* Scale up and down quickly and easily
* Low need for IT staff
* Secure
* Efficient
* Performant
* Powerful
* Various kind of APIs
* Global deployment

1. **Your opportunity**

* New platform to develop for
* Grate employment opportunity
* Opportunity to differentiate yourself

1. **Cloud Computing use what languages?**

* C#
* JavaScript
* Python

1. **Useful skills?**

* Serverless
* Database as a Service
* C# / JS / Python
* Stream processing
* Configuration management (Dev-ops)
* Cloud architecture
* Core services

1. **Business concern?**

* Different cost model
* Different IT skill set
* Privacy issues
* Data location
* Liability(tin cậy, tin tưởng)

1. **Cloud growth areas?**

* Serverless
* Stream processing
* Machine learning
* Internet of Thing
* Container as a Service

1. **Capital cost vs Operational cost?**

* Capital cost: an expense a business incurs to create benefits in the future
* Operational cost: an expense required for the day-to-day functioning of a business (Pay as you go)

**==============================================================================**

**Week 3: AWS Platform**

1. **What is AWS?**

* **Global** Cloud Computing platform
* Currently the **largest** provider of cloud services
* Provides **IaaS** and **PaaS** services
* Customers host **SaaS** on its platform

1. **AWS pros**

* Pros:
  + Great platform
  + Mature (Professional)
  + Trustworthy components
  + Cheap
  + Scalability
  + Global deployment
  + Secure
  + Fault tolerance

1. **AWS Physical layer**

* Consists of series of **69** highly optimized **data centers** (Availability Zones or AZ) in **23** different **regions**
* AWS Regions: are separate geographic areas that AWS uses to house its infrastructure
* Not all regions have AWS Services
* Different regions have different costs
* **Choose regions close to your customers**

1. **What is AWS region?**

* Are separate geographic areas that AWS uses to house its infrastructure
* Current 23 of them
* Not all regions have all AWS services
* Different regions have different cost

1. **What is Availability Zone (AZ)?**

* Fault-independent data centers within a region
* Redundant everything (power, network, cooling, …)
* High-speed connectivity
* Most services support replication(nhân rộng) and fail-over(thất bại) other AZ
* Each region has **at least 2 AZ**

1. **Shared Responsibility?**

* Amazon responsible for
  + Compute
  + Storage
  + Database
  + Networking
  + AZ, regions, edge location
* Users responsible for
  + Customer Data
  + Applications
  + Platform
  + Operating System & Network Configuration
  + Data Encryption
  + Networking Security

1. **Amazon Services**

* Currently **160 services** across **24 different categories**
* **Not available everywhere**
* Foundational services
  + EC2 (Elastic Compute Cloud): Highly managed VM
  + S3 (Simple Storage Service): Provide storage
  + IAM (Identity and Access Management): Security configuration
  + Lambda: Part of AWS serverless structure
  + RDS (Relational Database Service): Provide database that AWS will manage for you
  + SQS (Simple Queue Service)
  + CloudWatch: Data collection, monitoring, management services
  + API Gateway: Let developer creates AWS-hosted API
  + ELB (Elastic Load Balancer): Distribute incoming requests
  + Route53: AWS’s DNS
  + VPC: Virtual networking environment
* Services accessible via **AWS** **Console**, Command Line and API
* Uses TCP/IP (**Rest**)
* All integrated(tích hợp) into a comprehensive(toàn diện) security, automation, management and billing framework

1. **What is Elastic Cloud Compute (EC2)?**

* Highly managed VM with many options
* Run in cloud
* Easily scale up and down
* Provision(hợp tác) with different software load
  + Windows
  + Amazon Linux or other Linux
  + Cent OS
  + Debian
  + Fedora
  + Gentoo
  + openSUSE
  + Red Hat
  + Ubuntu
* Different platform
  + 32-bit (x86)
  + 64-bit (x88)
  + 64-bit (Arm)

1. **EC2 key concepts**

* Amazon Machine Image (AMI)
* Instance Type (small to crazy large)
  + Smallest – t2.nano (1 CPU, 512MB RAM - $0.005/hour)
  + Midsize – x1e.32xlarge (4 CPU, 8GB RAM - $0.17/hour)
  + Largest – x1e.32xlarge (128 CPU, 3904GB RAM - $26.68/hour)
* Elastic Block Storage (EBS)
* Security Group
  + Little firewalls configured on a per-instance basis
  + Lock down SSH session
  + Allow instances to talk to each other
  + Allow access to Database
* Elasticity (Scale up and down easily)

1. **What is Virtual Private Cloud (VPC)?**

* A virtual networking environment
* Complete with subnets, local lps, route tables, NATs, IP Gateways, ACLs, …
* **Best way to secure a deployed enterprise application**

1. **What is Elastic Load Balancer (ELB)**

* Distribute incoming requests across multiple targets
* Perform health checks and intelligent routing
* 3 different types:
  + **Application** for intelligent HTTPs request
  + **Network** for high performance hon-http traffic
  + **Classic** for legacy applications

1. **What is Simple Storage Service (S3)?**

* Store’s (Objects) – all or nothing
* Max file size: 5TB
* Durable – never lost files
* Available
* **Objects** are stored in **Bucket**
* Options are set at **Bucket level**
* Weak consistency(chắc chắn)
* **Not block storage**

1. **S3 Buckets Options**

* Versioning
* Server access logging
* Static website hosting
* Object-level logging
* Default encryption

1. **S3 Pricing**

* Charge on:
  + Amount of data stored
  + Number of requests
  + Amount of data transferred

1. **What is Relational Database Service (RDS)**

* AWS manages software, updates and backups and infrastructure
* Support MySQL, PostgreSQL, SQL Server, Oracle, MariaDB, and Aurora
* Easy configuration
* Easy creation and read replicas
* Easy configuration of fail-over

1. **RDS pros**

* Amazon will manage things for you
* Easy to scale the size of instance
* Easy to take snapshot
* No need for a DBA
* Access control by security groups

1. **RDS Pricing**

* Depends on Database and Instance Type running database

1. **What is CloudWatch?**

* Data Collection, Monitoring and Management Service
* Managed Service
* Used for logging, performance metrics and events
* Can set up Event handler
* Write custom event handler using AWS Lambda

1. **What is Lambda?**

* Part of AWS Serverless infrastructure
* Functions written in
  + Java
  + Go
  + PowerShell
  + Node JS
  + C#
  + Python
  + Ruby
* Call directly or invoked as part of an event handler
* Usually pair with **CloudWatch** or **API Gateways**
* Excellent for implementing REST endpoints
* Cheap
* Charged by GB/s (memory usage)

1. **What is API Gateway?**

* Let developer to create AWS-hosted APIs
* Route web service calls to a Lambda Function
* Fully managed
* Scales automatically
* Used hand-in-hand with Lambda

1. **What is DynamoDB?**

* Amazon NoSQL database
  + Base on tables and indexes
* Fast and fair price
* Great for sorted logs, look-up tables
* Not for relational data

1. **What is Route53?**

* Amazon’s DNS that integrates with AWS services
* Functions as light-weight Load Balancer (Health checks)
* Register domain, set up zones, route to AWS resources
* Reliable
* Pretty cheap
* Easiest way to get outside users to AWS application

1. **What is CloudFront?**

* Amazon’s Content Delivery Network (CDN)
* Enables fast delivery of graphics, data, video globally
* Integrates with other AWS services
* Easy way to support HTTPs over rest calls

1. **What is CloudTrail?**

* Logging service (log everything)
* Just need to turn it on

1. **Reliability (minh bạch)**

* A user may forget about the other issues of their data being secure and available in the cloud without having to be at risk of personal loss

1. **Durability**

* The tiny errors that occur in files that could make one or more individual bytes get corrupted or lost when write, read, and rewrite

1. **Availability**

* The percentage of time an application and its services are available

1. **Fail-over**

* A system usually used for mission-critical programs or reusable services that can introduce a **single point of failure** for multiple applications.
* Can span more than one geographical region

**==============================================================================**

**Week 4: Security Models**

1. **What is IAM?**

* Identity and Access Management
* Includes
  + Users
  + Groups
  + Policies

1. **What is the principle of least privilege of IAM?**

* Super important
* Allow users to do what they need to and no more
* Implication
  + More security
  + More stability
  + More thought required
* Best practice: Review users’ permissions to ensure this permission is adhere to

1. **What is Authentication?**

* **Establish who you are**
* **Verified by trusted provider**
  + Facebook
  + Google
  + IAM
* Sets principle of a request context

1. **What is Authorization?**

* **Establish if you are allowed to do an action**
* In AWS, enforced by AWS Policies

1. **What is policy?**

* **A rule** that is set to **an identity** to determine if that identify is **given access to or denied access** to a resource.
* Expressed as a **JSON document** with **Effect, Action** and **Resource** attributes
* Attached to **Users** and **Groups** or **Resources**
* **Immediately reflected**

1. **What is Policy Precedent?**

* Can be **Allow** or **Deny**
* Everyone denied by default

1. **What is Effect attribute?**

* Just **Allow** or **Deny**

1. **What is Action attribute?**

* What the principle wants to do
* Expressed as **service:action** (iam:DeleteUser)
* Can contains **wildcards** (iam:\* or iam:Delete\*)

1. **What is Condition attribute?**

* Limit access to an AWS resource by IP address
  + (“Condition”: {“aws:SourceIp”: “210.75..12.75/16”})

1. **What is Resource attribute?**

* The **target**(s) of the policy (what you want to operate on)
* Can be **single** resource or **array** of recourses
* **Wildcards** are ok
* Expressed as **Amazon Resource Name (ARN)**

1. **What is Amazon Resource Name (ARN)?**

* Used in policies to identify resources the policies apply to
* arn:partition:service:region:account:resource
  + arn:aws:lambda:us-east-1:424808903159:function:test1
  + arn:aws:iam::123456789:root
  + arn:aws:iam::123456789:user/quang

1. **What are Policy Types?**

* AWS Managed Policies (507 policies)
* Customer Managed Policies (those customers create)
* IAM Policies (policies attached to users, groups and roles)
* Resource Policies (policies attached to resources)
* **Use AWS Managed Policies as much as possible**
* **Use AWS Policy Generator when you can’t**
* **Always apply policies to groups**

1. **What is AWS User?**

* IAM User consists of a name and a set of credentials
* Might be actual use or an application
* Not globally unique – needs an account number
* **Root user is all-powerful**

1. **What is AWS Root User?**

* All-powerful
* Can’t be restricted
* Can do anything it wants
* Can expose all your data
* Can delete all data and backups
* Can get you into a lawsuit

1. **What is Access Type when create user?**

* Must select **Management Console access** to log into Management Console
* Normally, users need **AWS Account ID**, **IAM** **username** and **password** to login

1. **What are Password Policies?**

* **First thing to do with new AWS Account**
* Common options
  + Minimum length
  + Multi-character requirement
  + Expiration policy
  + Admin reset requirement
  + Password reuse limit
  + **Affects all account**

1. **What is Groups in AWS?**

* A set of IAM Users
* Assigned a set of permissions
* Group member inherit group’s policies
* **No default groups**
* **Cannot be nested**
* **Users can be in more than one group**
* **No minimum membership sizes**

1. **What is programmatic Access?**

* VIA access keys
* Access Key ID / Secret Access Key pair
* Max 2 keys per account. Enables rotation
* Rotate every 90 days

1. **What is Multifactor Authentication (MFA)?**

* Like 2 steps verification
* Username & Password + Phone || Fingerprint = 2-steps verification (multifactor authentication)
* Managed under security credentials

1. **What is Credential report**

* Used to identify
  + Unused accounts
  + Unused access keys
  + Account without MFA

1. **What is Security best practice?**

* Turn on MFA
* **Disable root’s access key**
* Rotate credentials
* Don’t share credentials
* Create admin user and groups
* Uses groups as much as possible
* Don’t apply permission directly to users
* Audit who has access to what
* Practice of **Principle of Least Privilege**
* Use policy conditions – IP Address limits

1. **Developer Guidelines**

* Always
  + Implement Web Services over HTTPs
  + Authenticate and authorize web service calls
  + Design applications from the start with roles in mind
  + Never implement your own security.
  + Audit code for security issues
  + Insist(năn nỉ) people who work for/with you do the same

==============================================================================

**Week 5: Lambda**

1. **What is Serverless?**

* Servers I don’t have to manage

1. **Key feature of Serverless?**

* Event driven
* Code focused
* No infrastructure management
* **Use Serverless as much as possible**

1. **Benefits and drawbacks of Serverless?**

* Benefits
  + Low cost
  + Managed machines
  + Service integration(toàn vẹn)
  + Scaling
  + Can focus on building Apps
* Drawbacks
  + Debugging
  + Control
  + Bleeding Edge

1. **Serverless services in AWS?**

* Lambda
* API Gateway
* S3
* AuroraDB serverless
* SQS
* CloudWatch
* CodeCommit

1. **What is Event Driven Programming?**

* Code that handle an event of the application
* Event fires -> code executes

1. **What is Event Driven Programming in AWS?**

* Events are from AWS Services
* Event handlers are call **Lambda** function and can be written in
  + JavaScript (NodeJS)
  + Python
  + C# (.NET Core)
  + Java
  + Go
  + PowerShell
* Events in AWS can be
  + File uploads
  + Timers
  + Error conditions
  + Billing alarms
  + EC2 instances started
  + Failed logins
  + Build completion
  + REST APIT Calls
  + ….

1. **What is Lambda?**

* Amazon’s event-driven serverless **framework**
* Integrated(tích hợp) to AWS CloudWatch and API Framework
* Easy to understand
  + Event fires
  + Lambda run-time starts
  + Event handler executes
  + Event result returns to caller
* **Best way to learn: Start writing code**

1. **What is Lambda Event Handler?**

* Event structure varies, depends on the source
* Context (body of function) contains information related to the request
  + Runtime information
  + Request source

1. **Where to write Lambda function?**

* AWS management Console (sometimes useful)
* Usually written in VSCode
* Can use most NodeJS compatible packages
* Can be **synchronous** or **asynchronous**
* **Each event handler is executed in a brand new environment**

1. **What is Lambda function package?**

* Write handler using VSCode
* Zip up directory structure
* Upload to AWS Lambda using Console
* Wire up to event

1. **How to test Lambda functions?**

* Use console.log()
* Invoke handlers using test events

1. **What is Lambda Constraints?**

* 1000 concurrent functions
* 250MB uncompressed max package size
* 15 minutes max execution time
* 3GB RAM (configurable) – **charged by GB/min**
* 6MB invocation payload (smaller if **async**)
* Can usually work arounds limitation by **breaking up** large function (Horizontal scaling)

1. **What is Lambda use cases?**

* REST API publishing
* Photo processing (S3 buckets)
* Stream processing

1. **What is REST Endpoint?**

* A web service
* Implemented on top of HTTP
* Takes a **URL**, **query string parameters** (optional), a **method type**, **request body** (optional) then **executes** an **operation** and **returns** a **response** code and a **result**

1. **REST Endpoint code languages and frameworks?**

* Python: Flask, Django
* C++: ASP.NET, Casablanca
* C#: NancyFX, WCF, WebAPI
* Java: JavaEE, Spring
* JavaScript: NodeJS (Express)

1. **Traditional way to approaches REST Endpoints**

* Using Swagger
* Implement atop REST framework
* Host on web server using NodeJS, IIS, Apache
* Do ad hoc
  + Monitoring
  + Scaling
  + Fail-over

1. **Lambda and REST Endpoints?**

* Very good way to integrate **different flatforms**
* **Scaling, monitoring, fail-over, security** is all integrated
* No infrastructure to manage
* API cache included
* API versioning
* Canary releases
* Faster time to market

1. **Process**

* Create Lambda functions
* Create API in API Gateway consisting of
  + Resources
  + Methods
* Attach methods to Lambda functions
* **Enable “Use Lambda Proxy Integration” to forwards to Lambda functions *entire* REST request context (body)**

1. **What is API Gateway?**

* The core of an API management solution
* Takes all the request from clients then determines which services are needed then combine them to a synchronous experience to user

1. **API Gateway options?**

* Options for
  + Entire REST API
  + Individual methods
* Common options
  + Authentication and Authorization
  + Throttling
  + Logging
  + API certificates
  + Web Application Firewalls
  + Caching

==============================================================================

**Week 6: Simple Storage Service (S3)**

1. **What is Storage?**

* A few types of volumes that can be mounted with EC2
  + Instance Store (Ephemeral)
  + Elastic Block Storage (EBS)
  + Elastic File System (EFS)

1. **What is Instance Store?**

* Mounted on an EC2 instances
* Stores temporary info
* Disappear when EC2 terminated
* Survives reboots
* May require initialization start up
* Selected along with EC2 Instance type

1. **What is Elastic Block Store?**

* Mounted on EC2 instances
* Can be backed up and stored in S3
* Can be mounted by multiple EC2 instances
* Can be encrypted
* Available in different configurations
* Produces CloudWatch events

1. **What is Elastic File Systems (EFS)?**

* Shared file system similar to NFS
* Network attached
* Easily scales up and down
* Multiple storage classes
* Highly available and very durable
* Cheap

1. **What is Simple Storage Service (S3)?**

* Oldest Storage Service in AWS
* Infinite Storage
* Incredibly durable
* Very fast
* Reasonable priced
* Simple

1. **Key features of S3**

* Scalable
* 11.9’s durability – stores data in multiple data center so nearly never lost data
* 4 9’s availability – stores data in data center with redundant network so nearly never lost connection
* 5TB Max “object” size
* Global reach
* Hosted regionally with optional cross region
* Not for databases or OS’s
* Web or API or CLI interfaces
* Object versioning
* Object Replication
* Encryption
* Timed URLs
* Server access logging

1. **Primary Applications**

* Object Storage
* Backups
* Website hosting
* Analytics
* Document Sharing
* Content Distribution

1. **What are buckets?**

* Contain objects
* Region affinity
* 100 buckets max/account
* No limit on objects/bucket
* Name must be DNS compatible and globally unique
* Access Control Lists applied to buckets
* Referred to with an ARN

1. **What are objects?**

* Store in buckets
* Name of objects called Key
* Contents of objects called Value
* Max key size: 1024 bytes
* Accessible with a URL
  + <https://bucketname.s3.amazonaws.com/objectid>
* Can be referred to with an ARN
* Can attach meta data

1. **Eventual consistency**

* Can read new object straight away
* Updates and deletes
* No object locking

1. **Reliability**

* S3 objects replicated 3 times
* Uses checksums to fix bad data
* Region replication
* Versioning – can keep every version ever created

1. **Type of Storage**

* Standard: default, frequently used
* Standard – IA: fast, cost less but charged more for retrieval
* Glacier: rarely used
* Deep archived: extremely cheap
* Intelligent tiering: move between Standard and Standard IA

1. **What is Access Control?**

* Works with Access Control List (ACL) or through Policies
* **ACLs are course grained** and high level read or write access
* **Policies are fined** grained and allow for many different scenarios
* Access provided to owner only by default
* Policies and ACLs applied to bucket and object level

1. **What are Encryption Terms?**

* Plaintext: info in unencrypted form
* Ciphertext: output of an encryption algorithm
* Encryption algorithm: step to take plaintext to ciphertext
* Symmetric and public key encryption

1. **Encryption locations?**

* In transit: over the network
* At rest: from anyone with physical access to disks

1. **S3 encryption options**

* You encrypt it when you upload the cyphertext to S3
* S3 encrypt it and store cyphertext on behalf
* SSE-S3: easiest
  + Each object is uploaded, a data encryption key is generated, and the object is encrypted using AES256 block cipher
  + The data encryption key is maintained with a master key maintained by Amazon
  + The encrypted object and the encrypted data encryption key are stored together on S3, master key is stored separately by Amazon
  + Amazon regularly rotate the master key for additional security
* SSE-KMS: you create and control keys
  + AWS operators don’t have access to all the keys
  + Auditable cloud trail
  + Each object is uploaded, a data encryption key is generated, and the object is encrypted using AES256 block cipher
  + Data encryption key is encrypted with a KMS CMK that is managed by you via KMS
  + The encrypted object and encrypted data encryption key are stored together
  + You will responsible for maintaining your CMK life cycle and key rotation
* SSE-C:
  + Upload data along with encryption key
  + S3 discards the key
  + Send key to get object
  + Must use https to ensure key not intercepted
* Client Side Encryption:
  + Library function provided as helper
  + Encryption is on client side

1. **Cross Origin Resource Sharing (CORS)**

* Prevent rogue scripts from accessing web resources
* Enforced by the browser
* Set up on a bucket
* Important if web apps are hosted in S3

**Essay example**

1. **Describe the 3 cloud delivery models. Describe what they are, examples of each, and when you would use each one**

There are 3 delivery models in AWS: public cloud, private cloud, and hybrid cloud. Public cloud model provides free cloud services that available for everybody to use. For example, Heroku is one of many public cloud services that anybody, who has an Heroku account, can use to host their website or server for free. The private cloud model provides cloud services that is owned by the users and only available within an organization or for a limit number of users. For example, when you have your own server and infrastructure that host your applications and data. Finally, the Hybrid cloud model contains both Public cloud and Private cloud. For example, a company can use private cloud to store its customer data while hosting its website on a public cloud.

1. **Describe 5 core AWS services. Be sure to state their proper names, what they used for and why you think each is a core service**

The 5 AWS services that I think most important are Elastic Cloud Compute (EC2), Simple Storage Service (S3), Identity and Access Management (IAM), Lambda Function, and API Gateway. EC2 is the service that provides highly managed virtual machine for users to use it without doing too much configuration. Simple Storage Service helps users to manage and store data on AWS cloud. IAM helps admin users to create users, groups, policies that manage users on what they can do and which resources they can access. and set up many security configurations on AWS cloud. The Lambda Function help users to handle event on different services automatically, which will save a lot of time and effort on manage and maintain the cloud. The API Gateway connect the user requests to AWS services and Lambda function

1. **Discuss security best practices. List at least 5 and for each discuss why they are important and the implications of not following them**

The 5 security best practices are: turn off Root access, never share credential, never implement your own security, use groups as much as possible, audit who has access to what

==============================================================================

**Week 7: Introduce to RDS and DynamoDB**

1. **What is Relational Database Service (RDS)?**

* RDS is an AWS service that manages a Relational Database Instance
* Does a lot of **heavy works**
  + Installation
  + Monitoring
  + Backup/Restore
  + Clustering
  + Failover
  + Replication
* Follows familiar pattern – takes what is hard and commoditizes

1. **RDS support**

* RDS will manage
  + SQL server
  + Oracle
  + PostgreSQL
  + MariaDB
  + MySQL
  + Aurora

1. **Pros and cons of RDS**

* **Cons**
  + Need hardware installation
  + Configure and harden Operating System
  + Design and implement OS patching strategy
  + Install and configure DBMS software
  + Set up clustering
  + Design and implement backup strategy
  + Design and implement database patching strategy
  + Create database instance
  + Load data into instance
  + Tune queries for performance
  + Upgrade hardware as needed
  + Monitor ad deal with failures
* **Pros**
  + Create database instance
  + Load data into instance
  + Tune queries for performance

1. **RDS Provisioning**

* RDS is backed by EC2 instances (a lot like provisioning EC2 instances)
* Select
  + Database Engine
  + Instance Type
  + Failover Replication, Backups, Monitoring

1. **Instance type**

* Can’t SSH to them
* Easy to scale up and back
* Pair with multi-AZ for failover

1. **What is Multi-AZ?**

* **Cons**
  + Requires 2 EC2 instances in 2 AZs
  + Cost twice as much
  + Not faster (a little bit slower)
* **Pros**
  + More reliable
  + Failovers handled by Route53 DNS
  + Minimizes downtime during maintenance

1. **Backups and Encryption**

* Always need
* Backups are incremental and are stored in S3
* Backups happen during backup window
* Snapshots are stored until deleted
* Automatically backing up can restore to any instant in time

1. **Scaling**

* Common to scale RDS **vertically**
* Can’t scale automatically
* Harder to scale horizontally
  + Read replicas
  + AWS Aurora

1. **What is Aurora**

* Amazon’s own
* Compatible with PostgreSQL and MySQL
* Automatically scales
* Grows to 64TB
* 5-10x faster on equivalent hardware
* Enhanced monitoring
* Continuous backups
* Self healing
* Compelling use cases

1. **What is NoSQL?**

* New class of database
* Many competing databases
  + MongoDB
  + Cassandra
  + ElasticSearch
  + DynamoDB
  + Azure Tables
* DBA’s hate them – cause of many religious wars
* Recently rebranded as Not Only SQL
* Fail into
  + Key-value pairs
  + Document storage
* DynamoDB is both

1. **Key idea**

* RDBs are overkill for many applications
* RDBs are do not scale horizontally well
* DBAs can be difficult to work with
* No all data is relational

1. **Common features**

* Scaling
* Clustering
* Fault tolerance
* Speed
* Flexible schemas
* Eventual consistency
* Simple Query operation – NoSQL
* Stores simple fields or complex JSON

1. **What is DynamoDB?**

* AWS’s NoSQL
* Stores Key-Value pairs and documents
* Scales as large as you need
* Serverless
* Single digit millisecond access
* Fully managed
* Durable
* Eventually consistent

1. **DynamoDB Object Model**

* Tables
* Partition key
* Sort keys
* Items
* Attributes
* Secondary indexes

1. **Where is primary key?**

* Primary key = partition key + sort key
* CRUD operations are WRT primary key
* Partition key sort key are required
* Can query partition key and optionally sort key
* Can scan any field

1. **Scan vs Find**

* Scan: more general and very flexible but expensive and slow
* Find: less general but very fast

1. **What is DynamoDB Indexes?**

* Can define other indices to enable different queries
* 2 types
  + Local secondary:
    - Have same partition keys and different sort keys
    - Create at table create time only
  + Global secondary:
    - Have different partition keys
    - Create any time

1. **What is TTL?**

* Data has a useful life span
* Super useful to manage database size
* Set TTL data deleted sometime afterward

1. **What is Caching?**

* DynamoDB is quick
* When need even quicker use DAX
* Use REDIS behind the scenes to cache results

1. **Cost**

* Different price models on demand
* $1.25/million write requests
* $0.25/million read requests
* $0.25/GB-month
* First 25GB free

==============================================================================

**Week 8: Serverless architecture**

1. **More than Lambda**

* Can’t see the servers in the service 🡺 Serverless
* Serverless cost nothing if no one is using it
* Event driven utility code execution environment in which you write code and consume services
* Managed services that scale to zero
* Serverless services
  + Aurora
  + S3
  + EC2
  + DynamoDB
  + API Gateway
  + Simple Notification Service
  + Simple Queuing Service

1. **Serverless advantages**

* Focus on **value** not **infrastructure**
* Patch only application
* Small amount of code
* **Glue code** to integrate with other managed services
* **Pay for consumption**
* **Highly available**
* Low maintenance
* **Easier deployment**
* **Scale infinitely**
* Scale down to zero cost

1. **Serverless runtime environment**

* Completely managed, elastic infrastructure
* Runtime environment
  + NodeJS
  + NET
  + Python
  + Java
  + Go
  + Ruby
  + Executable
* Little control over underlying hardware
* Stateless

1. **Typically**

* Create Lambda function
* Enable **Lambda Proxy Integration**
* Login
* Set session id as Cookie
* Use session id to load session data from cache or NoSQL database
* **Or say NO to state**

1. **Architecture comparison**

|  |  |
| --- | --- |
| **Server-centric** | **Serverless** |
| Programs run continuously | Program run on an event |
| State stored in memory | State stored external if at all |
| Large memory space | 3GB max |
| Can run for hours | Limited to 15 mins |
| Can choose OS | Can choose runtime |
| Scale configured by user | Scale configured by provider |

1. **Typical Serverless Toolbox**

* Use Lambda for compute layer
* Use S3, Aurora, DynamoDB for data layer
* Use API Gateway for integration
* Use SNS for messaging
* Use IAM and Cognito for Auth
* Use CloudFront for CDN

1. **Design pattern**

* The re-usable form of a solution to design a problem

1. **What you DON’T do**

* Load balancers to configure
* Message queues to set up
* EC2 instances to manage
* Scaling policies to establish
* Network to manage
* Cost when not running

1. **Serverless Challenges**

* Deep call stacks can be expensive
* Doesn’t integrate with non-serverless well
* New architecture patterns to learn
* Cold-start-syndrome – slow initial call
* Externally stored state
* Short function life-span
* Limited memory size
* Vendor lock-in
* Code-sharing

1. **Serverless Guidance**

* Write thicker, more powerful client
* Keep Lambda functions small, simple purpose and very fast
* Write glue code to integrate other serverless services
* **Don’t try and integrate non-serverless services**
* **Don’t recursively call yourself**
* Be careful with failure. Ensure you can recover
* Use logging to spot and fix failures
* Limit external dependencies
* Go stateless
* If not stateless, be careful with state
* Use API for code that can’t be trusted in browser
* Never authenticate in browser
* Use automated deployment solutions
* Become a student of serverless – Fluency is everything