05 June 2025 21:25

Week 1:

1. What is Cloud Computing?
Definition
Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications, and services)
Key Characteristics
- Renting/sharing IT resources through Internet/Web
- Innovative pricing models for measuring and charging resources
- On-demand access to computing resources
- Shared pool of configurable resources

IT Resources Include:
- Networks
- Servers
- Storage
- Applications
- Services

2. Cloud Service Models
Infrastructure as a Service (IaaS)
- What it provides: Fundamental computing resources (processing power, storage, networking)
- Consumer controls: Operating system, storage, deployed applications, networking components (firewalls, load balancers)
- Consumer does NOT control: Underlying cloud infrastructure
- Examples: AWS EC2, virtual machines

Platform as a Service (PaaS)
- What it provides: Hosting environment for applications (application framework)
- Consumer controls: Applications running in the environment, some hosting environment settings
- Consumer does NOT control: Operating system, hardware, network infrastructure
- Examples: AWS Elastic MapReduce, Google App Engine (2008)

Software as a Service (SaaS)
- What it provides: Complete applications
- Consumer controls: Application usage only
- Consumer does NOT control: Operating system, hardware, network infrastructure, application infrastructure
- Examples: Gmail, Google Docs, Salesforce.com CRM

3. Major Cloud Providers & History
Timeline
- 1998: XML-based SOAP web services proposed
- 2002: Amazon published SOAP-based services
- 2006: AWS launched three core services: EC2, S3, SQS
- 2008: Google App Engine released (early PaaS)
- 2010: Microsoft Azure officially released
- 2011: Google Cloud Platform launched

Major Players
- Amazon Web Services (AWS) - 2006
- Microsoft Azure - 2010
- Google Cloud Platform - 2011

4. Service Specification and Pricing

SaaS Pricing

- Specification: Based on application features (user accounts, storage size)
- Pricing: Subscription-based (monthly/yearly)

IaaS Pricing

- Specification: Similar to computer specs (CPU speed, cores, memory)
- Pricing: Pay-as-you-go hourly rate, now often per-second billing (minimum 60 seconds)

PaaS Pricing

- Between SaaS and IaaS: Can be hourly rate or subscription-based
- Example: MapReduce clusters charged hourly based on node count and type

Other Services

- Storage services: Complex pricing with static (storage size) and dynamic (number of queries) components
- Quality requirements: Additional charges for consistency and other features

5. Amazon Web Services (AWS)

Definition

AWS is a platform of web services offering solutions for computing, storing, and networking at different abstraction layers, accessible via internet using standard web protocols.

Web Services

- Software available over internet
- Uses standardized formats (XML, JSON)
- API-based request/response interaction

6. AWS Global Infrastructure

AWS Regions

- Geographical areas containing AWS resources
- Data replication across regions controlled by user
- Communication between regions uses AWS backbone network
- Full redundancy and network connectivity
- Contains: Two or more Availability Zones

Availability Zones

- Multiple per region
- Fully isolated partitions of AWS infrastructure
- Discrete data centers designed for fault isolation
- Interconnected with high-speed private networking
- User choice: You select which Availability Zones to use
- AWS recommendation: Replicate data across multiple AZs for resiliency

7. AWS Sample Applications

E-commerce Site Architecture

Bare Minimum Setup

- Web server: Handles customer requests (AWS EC2)
- Database: Stores products and orders (AWS RDS)

Enhanced Setup

- Content Delivery Network (CDN): For static content delivery
- Load balancer: For high availability and reduced response time
- Multiple web servers: Across different virtual machines
- Managed database: Reduces maintenance costs

High Availability Setup

- Database replica: Across multiple data centers
- Multiple virtual machines: As web servers across data centers
- Load balancer: Distributes customer requests

Batch Processing Infrastructure

- Use case: Daily data processing (e.g., gas turbine maintenance reports)
- Pay-per-use model: AWS bills VMs per second (minimum 60 seconds)

- Cost optimization: AWS offers spare capacity at substantial discounts

8. AWS Cost Structure

Free Tier

- Available for first 12 months of new account
- Limited service types
- Education credits available

Billing Categories

1. Time-based: Based on usage duration
2. Traffic-based: Based on data transfer
3. Storage-based: Based on storage consumption
4. Quality-based: Additional features and guarantees

Cost Scaling

- Cloud costs typically increase linearly with usage
- Example: 5x increase in website visitors = proportional cost increase

9. Interacting with AWS Services

Four Main Methods

1. Management Console (Web GUI)
2. Command Line Interface (CLI)
3. Software Development Kits (SDKs)
4. Blueprints (Infrastructure as Code)

Management Console

- Best for: Simple infrastructure setup for development/testing
- Starting point for nearly all users
- Easy to use graphical interface

Command Line Interface

- Best for: Automating recurring tasks
- Use cases: Creating infrastructure, uploading files, inspecting services
- Access: Through terminal

SDKs (Software Development Kits)

- Language-specific wrappers for AWS APIs
- Integration: AWS services into applications
- Multiple languages supported

Blueprints

- Infrastructure as Code tools
- Description: System containing all resources and dependencies
- Tools: Amazon CloudFormation, Terraform
- Function: Compare blueprint with current system and calculate needed changes

10. AWS Access and Security (IAM)

Identity and Access Management (IAM)

Essential Components

- IAM User: Person or application that can authenticate with AWS account
- IAM Group: Collection of IAM users with identical authorization
- IAM Policy: Document defining resource access and access levels
- IAM Role: Mechanism to grant permissions for AWS service requests

Authentication Types

1. Programmatic Access:
    - Access Key ID and Secret Access Key
    - Provides AWS CLI and SDK access
2. AWS Management Console Access:
    - 12-digit Account ID or alias
    - IAM username and password
    - Optional Multi-Factor Authentication (MFA)

Authorization Principles

- Default: All permissions implicitly denied

- Explicit deny: Always takes precedence over allow
- Best practice: Follow principle of least privilege
- Scope: IAM configurations are global (apply across all AWS regions)

IAM Policies

- Two types: Identity-based and Resource-based
- Identity-based: Attached to IAM entities (users, groups, roles)
- Resource-based: Attached to resources (e.g., S3 buckets)
- Flexibility: Single policy can attach to multiple entities; single entity can have multiple policies

IAM Groups

- Purpose: Grant same permissions to multiple users
- No default group exists
- Cannot be nested
- Users can belong to multiple groups

Week 2:

1. Cloud Storage Types

Storage as SaaS (Software as a Service)

- Examples: Google Drive, OneDrive, Dropbox, iCloud
- Features:
  - Extension to local hard disk
  - File system-like interface
  - Cloud features: sharing, collaborative editing, versioning
- Target: End users

Storage as IaaS/PaaS (Infrastructure/Platform as a Service)

- Examples: AWS S3, Azure Storage, Google Cloud Storage
- Characteristics:
  - Less end-user UI features
  - More control over infrastructure aspects
  - More cost-effective
  - API-driven access

2. AWS Data Storage Services Comparison

Service	Access Method	Max Storage	Latency	Cost
S3	AWS API, CLI, third-party tools	Unlimited	High	Very low
EBS (SSD)	Attached to EC2 via network	16 TiB	Low	Low
EC2 Instance Store	Attached to EC2 directly	305 TB	Very low	Very low
EFS	NFSv4.1 from EC2/on-premises	Unlimited	Medium	Medium

3. File Store vs Object Store

File Store

- Hierarchical structure (folders and files)
- File is the representation of data

Object Store

- Flat structure with objects
- Each object contains:
  - Globally unique identifier (key)
  - Metadata (access control, size, etc.)
  - Content (the actual data)

4. Amazon S3 Architecture

Key Concepts

- S3 = Simple Storage Service (launched 2006)

- Distributed object store
- Data accessed via HTTPS APIs
- Bucket: Container for objects (globally unique name required)
- Object: Individual data items with unique keys

Data Organization

- Flat structure: Only buckets and objects (no sub-folders)
- Pseudo-directory structure: Uses object key prefixes
    - Example: media/welcome.mp4 (media appears as folder in console)

S3 Bucket URLs (Two Styles)

1. Path-style: https://s3.ap-northeast-1.amazonaws.com/bucket-name
2. Virtual hosted-style: https://bucket-name.s3-ap-northeast-1.amazonaws.com

Regional Storage

- Data redundantly stored across multiple facilities within a region
- Default region: us-east-1

5. S3 Common Use Cases

1. Media Hosting
    - Direct URL access to media files
    - Integration with CloudFront for global distribution
2. Static Website Hosting
    - URL format: http://<bucket-name>.s3-website.<Region>.amazonaws.com
    - Host HTML, CSS, JS, images directly from S3
3. Data Analytics Pipeline
    1. Store raw data in S3
    2. Spin up compute (EC2 Spot Fleet/EMR)
    3. Process and transform data
    4. Store processed data back to S3
    5. Terminate compute resources
    6. Analyze with QuickSight/Athena
4. Backup and Archive
    - Cross-region replication
    - Integration with on-premises systems

6. S3 Pricing Model

You Pay For:

- Storage: GBs per month
- Data Transfer OUT: To other regions
- API Requests: PUT, COPY, POST, LIST, GET

You DON'T Pay For:

- Data Transfer IN: To S3
- Data Transfer OUT: To CloudFront or EC2 in same region

Pricing Factors:

1. Storage class type
2. Amount of storage (number and size of objects)
3. Request types (GET vs PUT rates differ)
4. Data transfer out volume

7. S3 Storage Classes

Storage Class	Availability Zones	Min Object Size	Min Duration	Retrieval Charge	Use Case
S3 Standard	≥3	N/A	N/A	N/A	Frequently accessed
S3 Intelligent-Tiering	≥3	N/A	N/A	N/A	Unknown access patterns
S3 Standard-IA	≥3	128 KB	30 days	Per GB	Infrequently accessed

S3 One Zone-IA	1	128 KB	30 days	Per GB	Non-critical, infrequent
S3 Glacier Instant	≥3	128 KB	90 days	Per GB	Archive, instant access
S3 Glacier Flexible	≥3	N/A	90 days	Per GB	Archive, mins-hours
S3 Glacier Deep Archive	≥3	N/A	180 days	Per GB	Long-term archive

Durability and Availability

- S3 Standard: 11 9's durability (99.999999999%), 4 9's availability (99.99%)
- S3 Standard-IA: 11 9's durability, 3 9's availability (99.9%)

8. S3 Lifecycle Management

Lifecycle Rules Define:

- Transition actions: Move to different storage class
- Expiration actions: When objects expire/delete

Example Lifecycle:

1. S3 Standard (30 days) → S3 Standard-IA (1 year) → Glacier (7 years) → Deep Archive (10 years) → Delete

9. S3 Versioning

Key Behaviors:

Action	Versioning Enabled	Versioning Disabled
Upload same key	Creates new version with unique ID	Overwrites original
Delete object	Adds delete marker, object recoverable	Permanently deletes object

Versioning Operations:

- Upload: Creates new version ID, keeps all versions
- Delete: Adds delete marker (object still exists)
- GET: Returns most recent version (or 404 if delete marker)
- GET with version ID: Returns specific version
- DELETE with version ID: Permanently deletes that version

S3 Data Consistency:

- Read-after-write consistency for all operations
- Consistent across all regions
- Immediate consistency for GET, LIST, PUT, DELETE

10. S3 Redundancy vs Versioning

S3 Redundancy (Default)

- Purpose: High availability and durability
- Method: Automatic storage across multiple devices/AZs
- Benefit: Fault tolerance against hardware failures
- Cost: No extra cost (default feature)

S3 Versioning (Optional)

- Purpose: Protection against accidental deletions/overwrites
- Method: Maintains multiple versions in same bucket
- Benefit: Audit trail and easy recovery
- Cost: Extra storage costs for multiple versions

11. S3 Replication

Requirements:

- Versioning must be enabled on both source and destination buckets
- Objects automatically copied between buckets

Use Cases:

- Disaster recovery
- Compliance requirements

- Latency improvement (geographic distribution)
- Storage cost optimization

Delete Operation Behavior:

Default Delete (no version ID):

- Source: Adds delete marker
- Destination: No delete marker added (inconsistency issue)
- Solution: Enable delete marker replication

Version-Specific Delete:

- Source: Permanently deletes specified version
- Destination: Version remains (not replicated)
- Solution: Manual deletion required in destination

Replication Features:

- Cross-Region Replication (CRR)
- Same-Region Replication (SRR)
- Delete marker replication (optional)
- Replica modification sync

12. Important Exam Points

Remember These Key Differences:

1. File Store vs Object Store: Hierarchical vs flat structure
2. SaaS vs IaaS Storage: End-user features vs infrastructure control
3. Redundancy vs Versioning: Hardware failure protection vs accidental change protection
4. Storage Classes: Cost vs access speed tradeoffs
5. Replication Delete Behavior: Default inconsistency and solutions

Critical S3 Features:

- Global unique bucket names
- Pseudo-directory structure with prefixes
- Multiple URL styles for access
- Regional data storage with cross-AZ redundancy
- Lifecycle automation for cost optimization
- Versioning protection with delete markers
- Replication requirements and limitations

Cost Optimization Strategies:

1. Choose appropriate storage class
2. Use lifecycle policies for automatic transitions
3. Monitor data transfer costs
4. Understand request pricing differences
5. Leverage free transfers within same region

Week 3:

S3 Review Key Points

Pseudo Folder Structure

- S3 buckets use a flat structure - folders are conceptual, not physical
- "Folders" are actually 0-byte objects that simulate directory structure
- Object key = bucket_name + object_prefix + object_filename
- Object key determines the physical location of the object
- Objects in same "folder" don't necessarily reside on same physical machine

S3 Object Concepts

- Most objects are files uploaded by users or AWS services
- Special objects include:
    - Folder objects: 0-byte objects
    - Delete markers: Special objects created in versioned buckets when objects are deleted
- In versioned buckets, multiple objects can have same key but different version IDs

Virtualization Overview
Definition and Core Concepts
Narrow definition: "Virtualization lets you run multiple virtual machines on a single physical machine, with each VM sharing resources across multiple environments"
Broad definition: "Virtualization is the transparent emulation of an IT resource producing benefits that were unavailable in its physical form"
Common Features of Virtualization

1. Emulation: Pre-existing IT resources are emulated
2. Transparency: Consumers cannot distinguish between real and emulated resources
3. Benefits: Provides advantages unavailable in physical form (expansion, optimization, high availability)

Historical Examples

- Virtual Memory (address translation, data transfer, swap strategies)
- Mainframe Virtualization (IBM 1972, solved OS migration problems)
- Hot Standby Router Protocol (solved single point of failure using virtual IP)

Virtualization Technology Areas
Timeline of key developments:

- 1957: Stanford (early concepts)
- 1962: University of Manchester
- 1972: IBM mainframe virtualization
- 1999: Vmware

Server Virtualization
Key Definitions

- Virtual Machine: Software-defined computer running on physical computer with separate OS and resources
- Host Machine: The physical computer
- Guest Machines: The virtual machines
- Hypervisor: Software component managing multiple VMs, ensuring resource allocation and isolation

Hypervisors
Types of Hypervisors

1. Type 1 (Bare-metal): Runs directly on hardware
2. Type 2 (Hosted): Runs on top of host operating system

Hypervisor Roles

- Provide environment identical to physical environment
- Minimize performance cost
- Retain complete control of system resources
- Handle resource sharing, device management, virtual storage management

Example Hypervisors
Xen Hypervisor
Components:

- Xen Hypervisor: Thin layer managing CPU, memory, interrupts
- Control Domain (Domain 0): Specialized VM handling I/O and control interface
- Guest Domains: User-allocated VMs
- Toolstack and Console: Admin interface

Hyper-V
Components:

- Hypervisor: Software layer on hardware providing isolated partitions
- Root Partition: Manages I/O and communication (like Xen's Domain 0)
- Child Partitions: Guest VMs

KVM (Kernel-based Virtual Machine)

- Turns Linux kernel into hypervisor
- Uses modules: KVM, QEMU, libvert
- Requires hardware virtualization support (Intel VT-x, AMD AMD-V)
- Used by Google Cloud Platform and AWS

- QEMU: Generic emulator virtualizing I/O devices
- Libvert: Common management layer for VMs

VM Resource Management
CPU Management
- vCPU: Virtual CPUs that are time-shared on physical CPU cores
- Hypervisor manages dynamic scheduling using custom algorithms
- vCPU count: Maximum threads VM can run simultaneously
- VMs can run on any host CPUs over time
- VM with multiple vCPUs needs each vCPU scheduled on different physical cores
- Key insight: Most applications aren't designed for multiple threads, so too many vCPUs may not help

Memory Management
- Two-level mapping: Guest OS manages virtual-to-guest-physical, VMM manages guest-physical-to-host-physical
- Guest OS gets dedicated memory portion for isolation
- Guest OS doesn't know which physical memory it's using
- Only VMM has access to physical memory

I/O Virtualization
Disk:
- Partitioned device where each VM gets virtual disk partition
- External drives can be mounted easily

Network:
- Physical adapter is time-shared by VMs
- Each VM has virtual network adapter
- Requests passed through VMM

Amazon EC2 Overview
EC2 Basics
- Amazon EC2: Provides resizable compute capacity in the cloud
- Provides virtual machines with:
    - Different CPU and memory configurations
    - Various storage options (Instance store, EBS)
    - Network connectivity
- Benefits: Complete control, cost optimization options, runs any workload

EC2 Provisioning Steps
1. Amazon Machine Image (AMI)
- Template used to create EC2 instances
- Contains OS (Windows/Linux) and pre-installed software
- Types:
    - Quick Start (AWS-provided)
    - My AMIs (user-created)
    - AWS Marketplace (third-party)
    - Community AMIs
- Benefits: Repeatability, Reusability, Recoverability

2. Instance Type Selection
Naming Convention: Example t3.large
- t = family name
- 3 = generation number
- large = size

Categories:
- General Purpose (a1, m4, m5, t2, t3): Broad use cases
- Compute Optimized (c4, c5): High performance computing
- Memory Optimized (r4, r5, x1, z1): In-memory databases
- Accelerated Computing (f1, g3, g4, p2, p3): Machine learning
- Storage Optimized (d2, h1, i3): Distributed file systems

Networking Features:

- Network bandwidth varies by instance type
- Enhanced Networking:
    - Elastic Network Adapter (ENA): Up to 100 Gbps
    - Intel 82599 Virtual Function: Up to 10 Gbps

3. Key Pair
    - Public key: Stored by AWS
    - Private key: Stored by user
    - Windows AMIs: Use private key to get administrator password
    - Linux AMIs: Use private key for SSH access
4. Network Settings
    - Specify VPC and subnet
    - Configure public IP assignment for internet accessibility
5. Security Groups
    - Firewall rules controlling traffic to instance
    - Exists outside guest OS
    - Specify: port number, protocol (TCP/UDP/ICMP), source
6. Storage Configuration

Storage Types:

Storage Type	Persistent	Multi-Instance Access	Use Cases
Amazon EBS	Yes	Single instance	Database, general app data
Instance Store	No (ephemeral)	Single instance	Buffers, cache, scratch data
Amazon EFS (Linux)	Yes	Multiple Linux instances	Shared file systems
Amazon FSx (Windows)	Yes	Multiple Windows instances	Windows shared storage

EBS vs Instance Store:

- EBS: Network-attached, persistent, can attach to any instance in same AZ, supports encryption and snapshots
- Instance Store: Local storage, temporary, lost when instance stops, uses HDD or SSD

7. IAM Role (Optional)
    - Enables EC2 instance to interact with other AWS services
    - Kept in instance profile
    - Can be attached at launch or to existing instances
8. User Data Script (Optional)
    - Script runs on first instance start
    - Used to customize runtime environment
    - Reduces need for custom AMIs
9. Tags
    - Labels with key-value pairs
    - Benefits: Filtering, automation, cost allocation, access control

EC2 Instance Lifecycle

Key states: Pending → Running → Stopping → Stopped → Shutting down → Terminated

AWS Nitro System

Evolution

- Early AWS: Used Xen hypervisor
- 2017: Moved to Nitro system
- Architecture: Microservices-based virtualization

Nitro Components

1. Nitro Hypervisor: Thin HVM-based hypervisor for CPU and memory partitioning
2. Nitro Cards: Hardware support for:
    - Networking (VPC, ENA API)
    - Storage (EBS with NVMe controller, encryption)
    - Management and Monitoring
    - Security
3. Security Chips

4. No Domain 0 required (unlike Xen)

Nitro Cards Functionality
Networking Card:
- Provides VPC network interface
- Supports Elastic Network Adapter (ENA) API
- Hardware-independent drivers

Storage Card:
- EBS with NVMe controller and encryption
- Instance storage (local SSD/NVMe when I/O > network throughput)

Bare Metal Instances
- Don't use Nitro hypervisor but still use Nitro cards
- Access VPC, EBS, and other AWS services through Nitro cards
- Minimal performance difference vs hypervisor-supported instances
- Hypervisor-supported instances recommended for most use cases

Week 4

1. Amazon EBS (Elastic Block Store)

AWS Data Storage Services Comparison

Service	Access Method	Max Storage	Latency	Storage Cost	Level
S3	AWS API, CLI, tools	Unlimited	High	Very low	Object
EBS (SSD)	Attached to EC2 via network	16 TiB	Low	Low	Block
EC2 Instance Store	Attached directly	305 TB	Very low	Very low	Block
EFS	NFSv4.1	Unlimited	Medium	Medium	File

Block vs Object Storage
- Block Storage: Change one block containing the character (efficient for small changes)
- Object Storage: Must update entire file (inefficient for small changes)

EBS Key Features
- Persistent storage attached over network to EC2 instances
- Not part of EC2 instances - volumes persist after instance termination
- DeleteOnTermination attribute:
  - Root volume: default = true
  - Other volumes: default = false
- Single attachment - can only attach to one instance at a time
- Automatic replication within Availability Zone
- Snapshots to S3 for backup

EBS Volume Types

SSD Types

Type	General Purpose	Provisioned IOPS
Max Volume Size	16 TiB	16 TiB
Max IOPS/Volume	16,000	64,000
Max Throughput/Volume	250 MiB/s	1,000 MiB/s
Use Cases	Most workloads, boot volumes, virtual desktops, dev/test	Critical business apps, large databases, data warehouses

HDD Types

Type	Throughput-Optimized	Cold
Max Volume Size	16 TiB	16 TiB
Max IOPS/Volume	500	250
Max	500 MiB/s	250 MiB/s

Throughput/Volume

Use Cases	Streaming workloads, big data, log processing	Infrequently accessed data, lowest cost scenarios
Boot Volume	No	No

EBS Pricing Components

1. Volumes: Charged by GB provisioned per month
2. IOPS:
    - General Purpose SSD: Charged by GB provisioned
    - Magnetic: Charged by number of requests
    - Provisioned IOPS SSD: Charged by IOPS provisioned
3. Snapshots: Per GB-month of data stored in S3
4. Data Transfer: Inbound free, outbound across regions charged

2. Amazon VPC (Virtual Private Cloud)

IP Addressing Fundamentals

- IPv4: 32-bit addresses (0.0.0.0 – 255.255.255.255)
- IPv6: 128-bit addresses
- Private IP Ranges:
    - 10.0.0.0 – 10.255.255.255
    - 172.16.0.0 – 172.31.255.255
    - 192.168.0.0 – 192.168.255.255

CIDR (Classless Inter-Domain Routing)

- Format: <IP Address>/<number>
- Example: 192.0.2.0/24 represents 256 IP addresses
- Number indicates how many bits are fixed for network identifier

VPC Architecture

- Logically isolated section of AWS Cloud
- Single AWS Region, spans multiple Availability Zones
- Control over:
    - IP address range selection
    - Subnet creation
    - Route tables and network gateways
    - Multiple security layers

Subnets

- Range of IP addresses that divide a VPC
- Single Availability Zone each
- Public or Private classification
- CIDR blocks cannot overlap
- AWS reserves 5 IP addresses in each subnet

Public vs Private IP Addresses

- Private IP: Allocated within subnet's CIDR range
- Public IP assignment:
    - Automatic: Through subnet's auto IP assign property (changes on restart)
    - Manual: Through Elastic IP address (can be reassigned)

Security Groups

- Acts as firewall for virtual machines and services
- Stateful - return traffic automatically allowed
- Allow rules only (no deny rules)
- All rules evaluated before allowing traffic
- Default behavior:
    - Deny all inbound traffic
    - Allow all outbound traffic

3. Running Databases in the Cloud

Database Deployment Models

Cloud Hosted Databases

- Install existing database servers on VMs (e.g., MariaDB on EC2)
- AWS RDS - Fully managed database services
- MongoDB Atlas - Managed MongoDB

Cloud Native Databases

- Azure Cosmos DB
- Google Bigtable, Spanner
- Amazon DynamoDB, Aurora

Management Responsibility Matrix

Task	On-Premises	EC2 Hosted	Managed Service
Power, HVAC, Network	You	AWS	AWS
Server Maintenance	You	AWS	AWS
OS Installation	You	You	AWS
OS Patches	You	You	AWS
DB Software Install	You	You	AWS
DB Software Patches	You	You	AWS
Database Backups	You	You	AWS
High Availability	You	You	AWS
Scaling	You	You	AWS
App Optimization	You	You	You

Database Design Considerations

1. Scalability: Throughput requirements and scaling needs
2. Storage Requirements: GB, TB, or PB scale
3. Data Characteristics: Data model and access patterns
4. Latency Requirements: Low latency needs
5. Durability: Data durability, availability, and recovery requirements

Relational vs Non-Relational Databases

Feature	Relational	Non-Relational
Structure	Tabular (rows/columns)	Variety (key-value, document, graph)
Schema	Strict schema rules	Flexible schemas
Benefits	Ease of use, data integrity, SQL	Flexibility, scalability, performance
Use Cases	OLTP, migrations	Caching, JSON documents, millisecond retrieval
Optimization	Complex queries with joins	Fast access to various data types

4. AWS RDS (Relational Database Service)

Key Benefits

- Lower Administrative Burden: No infrastructure provisioning or software maintenance
- Highly Scalable: Scale compute and memory resources
- Available and Durable: Automated backups, snapshots, host replacement
- Secure and Compliant: VPC isolation, encryption

Database Engine Options (7 total)

1. Aurora with MySQL compatibility
2. Aurora with PostgreSQL compatibility
3. RDS for MySQL
4. RDS for PostgreSQL
5. RDS for MariaDB
6. RDS for Oracle
7. RDS for SQL Server

RDS Architecture

- RDS instances: Isolated database environments

- Multiple databases per instance possible
- EBS volumes for database and log storage
- Network connectivity through designated ports (e.g., 3306 for MySQL)

High Availability: Multi-AZ Deployment

- Primary instance in one AZ
- Standby instance in another AZ
- Synchronous replication between primary and standby
- Automatic failover if primary fails
- Focus: High availability and durability

Read Replicas

- Asynchronous replication
- Read scaling for read-heavy workloads
- Can be promoted to primary if needed
- Same or different AZ placement possible
- Focus: Performance improvement for reads

Instance Types and Sizing

- General Purpose: T4g, T3, M6g, M5
- Memory-Optimized: R6g, R5, X2g, X1E

Example scaling decisions:

- Need more CPU: m6g.large → m6g.xlarge
- Need more memory: m6g.large → r6g.large

Security Best Practices

1. Run DB instance in VPC
2. Use IAM policies for resource management
3. Use security groups for access control
4. Use SSL/TLS connections
5. Encrypt instances and snapshots with AWS KMS
6. Use database engine security features

5. Amazon Aurora

Overview

- Relational database built for the cloud
- MySQL and PostgreSQL compatibility
- Managed by Amazon RDS
- High performance and availability at 1/10th the cost
- Multi-AZ deployments with Aurora Replicas

Key Innovation: Disaggregated Architecture

- Separate DB engine layer from storage layer
- DB Layer: Primary (writer) + up to 15 readers
- Storage Layer: Distributed across multiple AZs

Aurora Cluster Architecture

- Database Cluster: 1+ database instances + cluster volume
- Cluster Volume: Virtual storage spanning multiple AZs
- Each AZ: Has copy of DB cluster data
- Capacity: Up to 64 TB (at publication time)

Storage Layer Details

- Storage Nodes: EC2 instances with local SSD
- Protection Groups (PGs): 6 replicas of 10GB segments
- Distribution: 2 segments per AZ across 3 AZs
- Durability: 6-way replication with quorum system

Durability at Scale

- Replication Factor: 6 copies across 3 AZs
- Write Quorum: 4 out of 6
- Read Quorum: 3 out of 6
- Fault Tolerance:
    - Can lose entire AZ + 1 node (read availability)
    - Can lose entire AZ (write ability maintained)

6. Aurora: Log as Database

Traditional Database Write Burden

Write operations typically require:

- Write-ahead log (WAL) append
- Data file updates
- Binary log writes
- Double-write operations
- Metadata file updates

Aurora's Innovation: "The Log is the Database"

Write Operation Process:

1. Primary Instance: Sends only redo logs across network
2. Replica Instances: Receive redo logs to update memory
3. Storage Nodes:
    - Receive redo logs
    - Materialize database pages from logs (asynchronously)
    - Primary waits for 4 out of 6 acknowledgments

Storage Node Processing (8 steps):

1. Receive log record → in-memory queue
2. Persist record on disk → acknowledge
3. Organize records and identify gaps
4. Gossip with peers to fill gaps
5. Coalesce log records into new data pages
6. Stage log and pages to S3 periodically
7. Garbage collect old versions periodically
8. Validate CRC codes on pages periodically

Log Sequence Numbers (LSN)

- LSN: Monotonically increasing value from primary instance
- VCL (Volume Complete LSN): Highest LSN with all prior log records available
- CPL (Consistency Point LSN): LSNs marking transaction boundaries
- VDL (Volume Durable LSN): Highest CPL $\leq$ VCL

Example LSN Scenario

If VCL = 1007 but CPLs are at 900, 1000, 1100:

- VDL = 1000 (must truncate at transaction boundary)
- LSN 1007 represents operation mid-transaction

7. Aurora Replication Example

Sample Configuration

- 15GB database
- 2 Protection Groups: PG1 and PG2
- 5 DB instances: 1 primary (AZ1) + 4 replicas (2 in AZ2, 2 in AZ3)
- 10 storage nodes: AZ1(3), AZ2(3), AZ3(4)
- Replication:
    - PG1: SN11, SN12, SN21, SN22, SN31, SN32
    - PG2: SN11, SN13, SN22, SN23, SN33, SN34

Transaction Processing Example

Transaction X (PG1 only):

- Redo logs: 1001, 1002, 1003

Transaction Y (PG1 and PG2):

- PG2 logs: 1004, 1005
- PG1 logs: 1006, 1007

Write Success Criteria

- Primary sends redo logs to all replicas and relevant storage nodes
- 4 out of 6 storage nodes must acknowledge for write success
- Read replicas use logs only for in-memory updates

Storage Node Processing Details

Example for SN11:

- Receives logs: 1002, 1003, 1001, 1004, 1005, 1006 (out of order)
- Backtrack links: Each log links to previous log of same PG
- Process: Insert in memory queue → persist → acknowledge → gossip for missing logs
- Page writing: Only for log sequences with no gaps

Key Exam Concepts Summary

Critical Differences to Remember

1. EBS vs Instance Store: Persistence, network attachment, cost
2. Multi-AZ vs Read Replicas: Availability vs scalability, synchronous vs asynchronous
3. RDS vs Aurora: Traditional vs cloud-native architecture
4. Block vs Object Storage: Granular vs complete file updates
5. Public vs Private Subnets: Internet accessibility and IP addressing

Important Numbers

- EBS: 16 TiB max, various IOPS limits
- Aurora: 64 TB capacity, 1 writer + 15 readers
- Aurora Replication: 6 copies, write quorum 4, read quorum 3
- VPC: 5 reserved IP addresses per subnet

Security Essentials

- Security groups are stateful firewalls
- Default: deny inbound, allow outbound
- VPC provides network isolation
- Multiple layers of security possible

Week 5:

1. AWS Physical Infrastructure

Key Infrastructure Components

- Data Centers: Contain thousands of servers in racks with network routers and switches
- Availability Zones (AZs): Groups of data centers with single-digit millisecond latency between them
- Regions: Collections of AZs with 10s of milliseconds latency between regions
- Resource Isolation: Each AWS account has isolated resources within VPCs

2. Amazon VPC Basic Concepts

VPC Fundamentals

- Virtual Private Cloud (VPC): Logically isolated network dedicated to your AWS account
- Region-specific: VPCs belong to single AWS Region but can span multiple AZs
- Default VPC: Automatically created in each region for your account

Subnets

- Definition: Range of IP addresses that divide a VPC
- AZ-specific: Each subnet belongs to a single Availability Zone
- Types: Public (internet access) or Private (no direct internet access)
- No Overlap: CIDR blocks of subnets cannot overlap

IP Addressing

- VPC CIDR Block: Range of private IPv4 addresses assigned to VPC
- Size Limits:
  - Largest: /16 (65,536 addresses)
  - Smallest: /28 (16 addresses)
- Immutable: Cannot change address range after VPC creation
- IPv6 Support: Available with different block size limits

Reserved IP Addresses (Per Subnet)

For any subnet (e.g., 10.0.0.0/24):

- 10.0.0.0: Network address
- 10.0.0.1: Internal communication
- 10.0.0.2: DNS resolution
- 10.0.0.3: Future use
- 10.0.0.255: Network broadcast address
- Available IPs: 251 out of 256 total addresses

Public IP Address Types

- Public IPv4:
  - Manually assigned (Elastic IP)
  - Auto-assigned at subnet level
- Elastic IP:
  - Associated with AWS account
  - Can be reallocated anytime
  - Additional costs may apply

Elastic Network Interface (ENI)

- Virtual network interface that can be attached/detached from instances
- Attributes follow when reattached to new instance
- Default ENI: Each instance has one assigned private IPv4 address
- Multiple ENIs: Instances can have multiple ENIs
- Processing: Network traffic handled by Nitro Card for VPC

3. VPC Routing

Route Tables Fundamentals

- Route Tables: Control network traffic flow with rules (routes)
- Route Components:
  - Destination: CIDR block specifying where traffic goes
  - Target: Next hop device for reaching destination

Main Route Table

- Default: Every VPC has automatic main route table
- Local Route: Allows communication within VPC (destination: VPC CIDR, target: local)
- Subnet Association: Subnets use main route table unless explicitly associated with custom table
- Best Practice: Create custom route tables, leave main as default

Public Subnets

- Internet Access: Associated with Internet Gateway
- Route Table: Contains local route + internet route (0.0.0.0/0 → IGW)
- IP Requirements: Instances need both private AND public IP addresses
- Internet Gateway (IGW):
  - Horizontally scaled, redundant, highly available
  - Provides NAT functionality
  - One-to-one relationship with VPC

Private Subnets

- No Direct Internet: Only local route in route table
- Internet via NAT: Use NAT Gateway for outbound internet access
- Route Configuration: 0.0.0.0/0 → NAT Gateway ID

NAT Gateway

- Two Types:
  - Public NAT Gateway: Private subnet → Internet (outbound only)
  - Private NAT Gateway: Private subnet → Other VPCs/on-premises
- Public NAT Gateway Features:
  - Created in public subnet
  - Assigned Elastic IP address
  - AZ redundancy (one per AZ recommended)
  - One-way traffic (outgoing only)

Gateway Comparison

Feature	Internet Gateway	NAT Gateway
Traffic Direction	Two-way	One-way (outgoing)
Quantity per VPC	One	One per AZ (recommended)
Public IP Required	Yes	No (for instances)
Use Case	Public subnet access	Private subnet internet access

4. VPC Security

Security Layers (Defense in Depth)

1. Route Table Layer: Controls traffic routing
2. Network ACL Layer: Subnet-level filtering
3. Security Group Layer: Instance-level filtering
4. Internet Layer: Protocol-based filtering

Security Groups

- Scope: Instance level (like apartment door key)
- Rules: Only ALLOW rules (no deny rules)
- Stateful: Return traffic automatically allowed
- Default New SG:
  - No inbound rules
  - All outbound traffic allowed (0.0.0.0/0)
- Rule Components: Source/Destination, Protocol, Port Range

Network ACLs

- Scope: Subnet level (like apartment doorman)
- Rules: Both ALLOW and DENY rules
- Stateless: Return traffic must be explicitly allowed
- Rule Evaluation: Processed in numerical order (lowest first)
- Default ACL: Allows all inbound and outbound traffic
- Custom ACL: Denies all traffic until rules added

Security Groups vs Network ACLs

Attribute	Security Groups	Network ACLs
Scope	Instance level	Subnet level
Rules	Allow only	Allow and Deny
State	Stateful	Stateless
Rule Order	All evaluated	Numerical order
Default	No inbound, all outbound	All allowed

Bastion Hosts

- Purpose: Secure administrative access to private instances
- Setup: Public subnet instance with restricted SSH access
- Security: Use security group chaining (SG A → SG B)

5. Connecting to Managed Services

Direct Internet Connection

- Public Subnet: Direct HTTPS to AWS services via internet
- Private Subnet: Via NAT Gateway → Internet Gateway → AWS services

VPC Endpoints (Recommended)

- Purpose: Private connection to AWS services without internet
- Benefits: Improved security, performance, reduced costs

Interface VPC Endpoints

- Technology: Powered by AWS PrivateLink
- Implementation: Uses Elastic Network Interfaces (ENIs)
- Services: Most AWS services
- Security: Secured with security groups and IAM policies
- Infrastructure: ENIs backed by Nitro hardware on AWS-managed instances

Gateway VPC Endpoints

- Services: S3 and DynamoDB only
- Implementation: Adds route to route table
- Configuration: Uses prefix lists in route table

Gateways vs VPC Endpoints

Feature	Gateway	VPC Endpoint
Purpose	Connect to external networks	Connect privately to AWS services

Traffic	Leaves AWS network	Stays in AWS network
Direction	Inbound/Outbound	Primarily outbound
Cost	NAT: $$, IGW: Free	Low cost (data processing fee)

6. AWS Internal Traffic Routing
VPC Internal Communication
- Local Routing: Instances in same VPC communicate via local route
- Implementation: Handled by virtual router and Nitro card
- Encapsulation: Traffic encapsulated for multi-tenant physical hosts
- IP Reuse: Different VPCs can have same private IP addresses

Key Components
- Virtual Router: Implemented by Nitro card for VPC
- Mapping Service: Handles traffic routing within AWS infrastructure
- Encapsulation: Enables secure multi-tenant networking

7. AWS External Traffic Routing
Internet Gateway Traffic
- Outgoing: NAT translation from private to public IP
- Incoming: NAT translation from public to private IP
- Stateful: Tracks connections for return traffic

Traffic Flow Examples
- Public Subnet: Instance ↔ IGW ↔ Internet
- Private Subnet: Instance → NAT Gateway → IGW → Internet

Key Exam Tips
Subnet Selection Guidelines
- Database instances: Private subnet
- Batch-processing instances: Private subnet
- Web application instances: Public or private subnet (depends on requirements)
- NAT gateway/instance: Public subnet

Important Numbers to Remember
- VPC CIDR: /16 (max) to /28 (min)
- Reserved IPs per subnet: 5 addresses
- Available IPs in /24 subnet: 251 addresses
- AZ latency: Single-digit milliseconds
- Region latency: 10s of milliseconds

Common Misconceptions
- Security groups are NOT firewalls (only allow rules)
- Network ACLs are stateless (need explicit return rules)
- Private subnets can access Internet via NAT (outbound only)
- VPC endpoints keep traffic within AWS network
- Each subnet can only be in one AZ
- Each VPC can only have one Internet Gateway

Week 6:

1. AWS Shared Responsibility Model
Core Concept
- AWS Responsibility: Security OF the cloud (infrastructure)
- Customer Responsibility: Security IN the cloud (data, applications, configurations)

AWS Responsibilities
- Physical security of data centers
- Hardware and software infrastructure
- Network infrastructure and intrusion detection
- Virtualization infrastructure and instance isolation
- Storage decommissioning, host OS access logging

- Controlled, need-based access

Customer Responsibilities

- EC2 instance operating systems (patching, maintenance)
- Applications and passwords
- Security group configurations
- OS/host-based firewalls and intrusion detection/prevention
- Network configurations
- Account management and user permissions
- Client-side data encryption and authentication
- Server-side encryption and network traffic protection

Service Types and Responsibility Distribution

Infrastructure as a Service (IaaS)

- Examples: EC2, EBS, VPC
- Customer: More control, more security responsibility
- Customer manages: Networking, storage, access controls

Platform as a Service (PaaS)

- Examples: RDS, Elastic Beanstalk, Lambda
- AWS handles: OS, database patching, firewall config, disaster recovery
- Customer focuses on: Code and data management

Software as a Service (SaaS)

- Examples: AWS Trusted Advisor, AWS Shield, Amazon Chime
- Customer: Minimal infrastructure management
- Access via: Web browser, mobile app, or API

2. Identity and Access Management (IAM)

Core Concepts

- Identity: Each entity needs an identity (authentication)
- Access: Ensuring entities can only perform necessary tasks (authorization)

IAM Components

IAM Users

- Account root user (most powerful - should be secured)
- IAM users (for individuals/applications)
- Authentication methods:
    - Programmatic access: Access key ID + Secret access key (for CLI/SDK)
    - Console access: Account ID + username + password (+ optional MFA)

IAM Groups

- Collections of IAM users with identical permissions
- Users inherit group permissions
- Cannot be nested
- Best practice: Assign permissions to groups, not individual users

IAM Roles

- Provide temporary security credentials
- Not uniquely associated with one person
- Assumable by users, applications, or services
- Use cases:
    - AWS resources accessing AWS services
    - External user authentication
    - Third-party access
    - Cross-account access

IAM Policies

- Define allowed/denied actions on resources
- JSON documents specifying permissions
- Follow principle of least privilege

Policy Types

Identity-Based Policies

- Attached to users, groups, or roles
- Answer: "What can this identity access?"

Resource-Based Policies

- Attached to AWS resources
- Answer: "Who can access this resource?"

Permission Evaluation Process

1. Evaluate all applicable policies
2. Explicit deny? → DENY (explicit deny always wins)
3. Explicit allow? → ALLOW
4. No explicit allow → IMPLICIT DENY (default)

3. IAM Policy Structure

Policy Document Elements

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow|Deny",
  "Principal": "account/user/role", // (resource-based only)
  "Action": "service:action",
  "Resource": "arn:aws:service:region:account:resource",
  "Condition": {
   "condition_operator": {
    "key": "value"
   }
  }
 }]
}
```

Key Elements

- Version: Policy language version
- Effect: Allow or Deny
- Action: What operations are permitted/denied
- Resource: Which resources the action applies to
- Condition: When the policy applies
- Principal: Who (only in resource-based policies)

Amazon Resource Names (ARNs)

- Format: arn:partition:service:region:account:resource
- Example: arn:aws:iam::123456789012:user/mmajor
- Wildcards (*) supported for broader access

Policy Examples

Read-Only IAM Access

```
{
 "Effect": "Allow",
 "Action": ["iam:Get*", "iam:List*"],
 "Resource": "*"
}
```

Conditional EC2 Termination

```
{
 "Effect": "Allow",
 "Action": "ec2:TerminateInstances",
 "Resource": "*",
 "Condition": {
  "IpAddress": {
   "aws:SourceIp": ["192.0.2.0/24"]
  }
 }
}
```

4. Attribute-Based Access Control (ABAC)

Challenges of Role-Based Access Control (RBAC)

- Requires updating policies for each new resource
- Multiple policies need modification for new resources
- Doesn't scale well with growing infrastructure

ABAC Benefits

- More flexible than listing individual resources
- Granular permissions without constant policy updates
- Highly scalable approach
- Fully auditable
- Uses tags as attributes

Tag-Based Access Control

- Tags are key-value pairs attached to resources
- Enable dynamic access control based on attributes

Example: Resource Tag Condition

```
{
 "Effect": "Allow",
 "Action": "ec2:StopInstances",
 "Resource": "*",
 "Condition": {
  "StringEquals": {
   "aws:ResourceTag/Project": "DataAnalytics"
  }
 }
}
```

Example: Principal and Resource Tag Matching

```
{
 "Effect": "Allow",
 "Action": ["ec2:StartInstances", "ec2:StopInstances"],
 "Resource": "*",
 "Condition": {
  "StringEquals": {
   "aws:ResourceTag/CostCenter": "${aws:PrincipalTag/CostCenter}"
  }
 }
}
```

5. Identity Federation

Core Concept

System of trust between parties for authentication and authorization

Key Players

- Identity Provider (IdP): Handles user authentication
    - Examples: Active Directory, Google, Facebook, SAML providers
- Service Provider (SP): Controls access to resources
    - Examples: AWS services, applications

AWS Federation Services

- AWS IAM: Core identity management
- AWS IAM Identity Center: Successor to AWS SSO, centralized access management
- AWS STS: Provides temporary credentials
- Amazon Cognito: User management for web/mobile apps

Federation Patterns

Workforce Identity Federation

1. User authenticates with external directory
2. Identity broker requests temporary credentials from STS
3. STS returns temporary credentials
4. User accesses AWS resources with temporary credentials

SAML Federation

1. User navigates to portal
2. IdP authenticates user
3. IdP returns SAML assertion

4. Assertion posted to AWS SAML endpoint
5. User redirected to AWS console

Amazon Cognito

- User Pools: User directory with authentication
- Identity Pools: Unique identities with AWS permissions
- Features:
    - Authentication, authorization, user management
    - Social identity provider integration
    - SAML federation support
    - Token-based authentication

6. Security Best Practices

Root Account Security

1. Create IAM admin user
2. Lock away root user credentials
3. Use IAM users for daily tasks
4. Enable MFA on root account

IAM Best Practices

- Use groups for permission management
- Apply principle of least privilege
- Regularly review and rotate credentials
- Use roles for applications and cross-account access
- Enable MFA for sensitive operations
- Use policy conditions for additional security

Policy Management

- Combine user and group policies for fine-grained control
- User policies can override group policies (if more restrictive)
- Test policies thoroughly before implementation
- Document policy purposes and requirements

7. Key Exam Points

Remember These Distinctions

- Authentication vs Authorization: Who you are vs what you can do
- Identity-based vs Resource-based policies: Attached to identities vs resources
- IAM Groups vs Roles: Groups for users, roles for temporary access
- RBAC vs ABAC: Role-based vs attribute/tag-based access control

Common Scenarios

- Cross-account access requires roles
- EC2 instances need roles to access other AWS services
- External users require federation
- Temporary access uses STS tokens
- Web/mobile apps use Cognito

Policy Evaluation Priority

1. Explicit Deny (always wins)
2. Explicit Allow
3. Implicit Deny (default)

Service Responsibility Examples

- Customer: EC2 OS patches, security groups, application configs
- AWS: Physical security, hypervisor, managed service maintenance
- Shared: Depends on service type (IaaS, PaaS, SaaS)

Week 7:

1. AWS Academy Lab Session Management

Federated User Authentication

- Role: All federated users assume the "voclabs" role

- Session Policies: Advanced policies passed as parameters during role assumption
- Dynamic Permissions: Different labs have different permission sets associated with the same role

Session Management Process

1. Start Lab:
    - Calls AWS STS AssumeRole on preconfigured IAM role
    - Passes custom session name for tracking
    - Attaches lab-specific session policy
    - Returns temporary credentials (SessionToken, SecretAccessKey)
    - Generates federated user sign-in URL
    - Creates initial resources using CloudFormation template
2. AWS Console Access:
    - Federated sign-in URL opens AWS Console
    - AWS sets cookies: aws-creds, aws-sessionID, others
    - Session tracked via browser cookies
3. End Lab:
    - All lab resources automatically cleared
    - Session permissions revoked/disabled
    - AWS button disabled
    - Console may still show UI but actions fail

Session Types and Expiration

- Lab Session vs AWS Session: Independent lifecycles
- Session Expiration: Both expire after inactivity or set time
- Restart Lab: Extends current session timer, keeps resources intact

2. Scalability and Elasticity Concepts

Scalability

- Definition: Property of a system to handle growing amounts of work
- Measurement: How easy it is to add more resources to handle changes

Elasticity

- Definition: Fine-grained capacity expansion AND contraction
- Cloud vs Traditional: Traditional focuses on expansion; cloud includes contraction
- Examples:
    - Increasing web servers during traffic spikes
    - Decreasing database capacity when traffic drops
    - Handling day-to-day demand fluctuations

Scaling Types

1. Horizontal Scaling (Scale Out/In):
    - Launch/terminate instances
    - Uses Auto Scaling groups
    - Requires Elastic Load Balancing
2. Vertical Scaling (Scale Up/Down):
    - Increase/decrease instance size
    - Single instance modification

3. Requirements for Cloud Elasticity

1. Monitoring Capability:
    - CloudWatch for workload monitoring
    - User configuration required
2. Automatic Resource Management:
    - Start/stop resources with minimal latency
    - EC2 instances from AMI with small latency
3. System Recognition:
    - Running system must recognize resource changes
    - Elastic Load Balancing directs requests to new resources
    - Applications must handle resource addition/removal

4. Amazon CloudWatch

Monitoring Capabilities

- AWS Resources: EC2, RDS, etc.
- Applications: Running on AWS
- Metrics: Standard and custom metrics collection

CloudWatch Alarms

- Trigger Types:
    - Static threshold
    - Anomaly detection
    - Metric math expression
- Configuration:
    - Namespace, Metric, Statistic, Period, Conditions
- Actions:
    - SNS notifications
    - EC2 Auto Scaling actions
    - EC2 instance actions

CloudWatch Events

- Define rules matching AWS environment changes
- Route events to target functions/streams for processing

5. Amazon EC2 Auto Scaling

Core Function

- Manages logical collection of EC2 instances (Auto Scaling group)
- Operates across Availability Zones
- Launches/retires instances using launch templates
- Integrates with Elastic Load Balancing
- Free service

Auto Scaling Group Components

Capacity Settings

- Minimum instances: Lower bound
- Maximum instances: Upper bound
- Desired instances: Target number

Launch Template

- Instance Configuration: Type, AMI, Security Groups, User Data
- Important: Subnet defined in ASG, not launch template

Load Balancer Integration

- Receives health check notifications
- Registers new instances automatically

Scaling Mechanisms

- Schedule Actions: Time-based scaling
- Dynamic Scaling Policies: Metric-based scaling
- Predictive Scaling Policy: Machine learning predictions

Scaling Policies

1. Schedule-Based:
    - Example: Turn off dev/test instances at night
2. Policy-Based (CloudWatch alarms):
    - Constant number maintenance
    - Target tracking (e.g., maintain 60% CPU)
    - Custom metrics
3. Predictive Scaling:
    - Uses machine learning for forecasting

6. Elastic Load Balancing (ELB)

Purpose

- Distributes incoming traffic across multiple targets
- Targets: EC2 instances, containers, IP addresses, Lambda functions

- Can be external-facing or internal-facing
- Receives DNS name
- Recognizes and responds to unhealthy instances

Load Balancer Types

1. Application Load Balancer (ALB):
    - HTTP/HTTPS traffic
    - OSI Layer 7 (Application)
    - Application architectures
2. Network Load Balancer (NLB):
    - TLS offloading, UDP, static IPs
    - OSI Layer 4 (Transport)
    - Millions of requests/second, ultra-low latency
3. Gateway Load Balancer:
    - Third-party virtual appliances
    - OSI Layer 3 (Network)
    - GENEVE protocol
4. Classic Load Balancer:
    - Previous generation EC2-Classic
    - OSI Layers 3 and 7
    - Legacy support

Load Balancer Components

Core Components

- Request: Incoming traffic
- Listeners: Define port and protocol
- Rules: Routing logic
- Target Groups: Logical grouping of targets
- Targets: Actual compute resources
- ACM: TLS certificate management

ALB Specific Components

- Listeners:
    - Define port/protocol for load balancer
    - Each ALB needs ≥ 1 listener
    - Can have multiple listeners
    - Define routing rules
- Target Groups:
    - Logical grouping of AWS resources
    - Auto Scaling groups can be target groups
    - Load balancer performs health checks

Rules and Actions

- Conditions: When rule applies
- Actions: What to do when condition met
    - Forward to target group
    - Redirect to URL
    - Respond to client

Load Balancing Algorithms

1. Round-Robin (Default):
    - Cycles through targets sequentially
2. Least Outstanding Request (LOR):
    - Selects target with lowest pending requests
    - Better for varying response times

Health Checks

- Purpose: Test target status periodically
- Routing: Requests only sent to healthy targets
- Configuration: Protocol, Path, Timeout, Interval
- Integration: Works with Auto Scaling Groups

7. Health Check Integration
EC2 vs ELB Health Checks
- EC2 Health Check: Always enabled in ASG
  - Monitors running status
  - Detects hardware/software issues
- ELB Health Check: Optional ASG configuration
  - Application-level issue detection
  - Can detect app crashes while instance runs

Health Check Scenarios
Application Crash
1. App crashes, EC2 instance still running
2. Target group health check fails
3. ASG (if using ELB health checks) marks unhealthy
4. Instance terminated and replaced

Unresponsive Instance
1. EC2 health check fails
2. CloudWatch detects and reports to ASG
3. ASG terminates and replaces instance
4. Faster detection than target group health check

8. Database Scaling
Amazon RDS Scaling
Vertical Scaling
- Push-button scaling: Scale DB instances up/down
- Range: Micro to 24xlarge
- Downtime: Minimal downtime scaling

Horizontal Scaling (Read Replicas)
- Purpose: Scale read-heavy workloads
- Limits: Up to 5 RDS read replicas, 15 Aurora replicas
- Replication: Asynchronous
- Supported: MySQL, MariaDB, PostgreSQL, Oracle
- Usage: Primary for read/write, replicas for read-only

Amazon Aurora Scaling
- Cluster Architecture: Up to 15 Aurora replicas per cluster
- Distribution: Across multiple Availability Zones
- Storage: Shared cluster volume with multiple copies
- Performance: Better than standard RDS read replicas

Key Exam Points to Remember
1. Session Management: Understand lab vs AWS session differences
2. Elasticity Requirements: Monitoring, automation, system recognition
3. Auto Scaling Components: Capacity, launch templates, scaling policies
4. Load Balancer Types: Know when to use ALB vs NLB vs GLB
5. Health Check Integration: EC2 vs ELB health checks and their purposes
6. Database Scaling: Vertical vs horizontal, read replicas limitations
7. CloudWatch Integration: Alarms trigger scaling actions
8. Target Group Concepts: Health checks, algorithms, integration with ASG

Week 8:

1. AWS CLI Fundamentals
Ways to Interact with AWS
- Management Console: Web-based GUI (manual, not scalable)
- Command Line Interface (AWS CLI): Terminal-based interaction
- SDK: Programmatic access via software development kits
- Blueprints: Pre-configured templates

Key CLI Commands

```
aws configure                # Set up CLI credentials
aws s3 ls                    # List S3 buckets
aws ec2 describe-instances   # List EC2 instances
aws cloudformation create-stack # Create CloudFormation stack
# Example: Create EC2 instance
aws ec2 run-instances \
  --image-id ami-0ccedee93274bbb8d \
  --instance-type t2.micro \
  --count 1
# Example: Create S3 bucket
aws s3api create-bucket --bucket abcd1234 --region us-east-1
```

Problems with Manual Processes

- No repeatability at scale: Hard to replicate across regions
- No version control: Can't roll back changes
- Lack of audit trails: No compliance tracking
- Inconsistent data management: Configuration drift across resources

2. Infrastructure as Code (IaC)

Core Concept

- Process of writing templates that provision and manage cloud resources
- Templates are both human readable and machine consumable
- Can replicate, redeploy, and repurpose infrastructure

Key Benefits

- Reusability: Same template creates multiple environments
- Repeatability: Consistent deployments every time
- Maintainability: Easy to update and manage
- Rapid deployment: Complex environments deployed quickly
- Clean up: Delete stack removes all resources

3. CloudFormation Overview

What is CloudFormation?

- AWS service for modeling, creating, and managing collections of AWS resources
- Collection of resources = CloudFormation Stack
- No extra charge (pay only for resources created)
- Enables orderly and predictable provisioning
- Supports version control of deployments

How CloudFormation Works

1. Define resources in a template (or use prebuilt template)
2. Upload template to CloudFormation or point to S3-stored template
3. Run create stack action - resources created across multiple services
4. Stack retains control - can update, detect drift, or delete stack

Stacks

- Collection of AWS resources managed as a single unit
- Created based on CloudFormation templates
- Think: Template = Class, Stack = Object instance
- Common grouping approaches:
    - Cohesion: Related resources together
    - Lifecycle: Resources with same lifespan
    - Security: Resources with similar security requirements
    - Layer approach: Network layer, Application layer, Database layer

4. CloudFormation Templates

Template Formats

YAML Advantages

- Optimized for readability
- Less verbose than JSON
- No brackets needed

- Quotes can be omitted
- Supports embedded comments

JSON Advantages

- More widely used by other systems
- Less complex to generate and parse

Template Anatomy

```
---
AWSTemplateFormatVersion: "2010-09-09"  # Optional
Description: String                     # Optional
Metadata: template metadata             # Optional
Parameters: set of parameters           # Optional
Rules: set of rules                     # Optional
Mappings: set of mappings               # Optional
Conditions: set of conditions           # Optional
Transform: set of transforms            # Optional
Resources: set of resources             # REQUIRED (only required section)
Outputs: set of outputs                 # Optional
```

5. Template Sections Deep Dive

Resources Section (REQUIRED)

```
Resources:
  resourceName1:
    Type: AWS::ServiceName::ResourceType
    Properties:
      PropertyName1: PropertyValue1
      PropertyName2: PropertyValue2
  resourceName2:
    Type: AWS::ServiceName::ResourceType2
    Properties:
      PropertyName1: PropertyValue1
```

Key Points:

- Logical ID (resourceName) chosen by user
- Resource type follows AWS::ServiceName::ResourceType format
- Properties depend on resource type

EC2 Instance Example:

```
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-9d23aeea
      InstanceType: m3.medium
      KeyName: !Ref KeyPair
```

Parameters Section

- Optional but powerful for customization
- Like parameterized constructors for different stack instances
- Specify property values of stack resources

```
Parameters:
  ParameterLogicalID:
    Description: Information about the parameter
    Type: DataType              # String, Number, or AWS-specific
    Default: value              # Optional
    AllowedValues:              # Optional
      - value1
      - value2
```

Example:

```
Parameters:
  KeyPair:
```

Description: SSH Key Pair
Type: String
Could also use AWS::EC2::KeyPair::KeyName as type
Outputs Section
- Optional section to declare output values for the stack
- Define what data to pass as output

Outputs:
OutputLogicalID:
Description: Description of output
Value: Value to return
Export: # Optional
Name: Name of resource to export
Example:
Outputs:
ID:
Value: !Ref Ec2Instance
Description: ID of the EC2 instance created
PublicName:
Value: !GetAtt EC2Instance.PublicDnsName
Description: Public DNS of the EC2

6. Intrinsic Functions
Definition
Built-in functions to assign values to properties not available until runtime
Basic Syntax
- Full form: Fn::function_name inputs
- YAML short form: !function_name input(s)

Key Functions
Ref Function
- Syntax: !Ref LogicalID or Ref: LogicalID
- Returns value of specified parameter, resource, or another function
- Most commonly used to create references between resources

Fn::GetAtt Function
- Syntax: !GetAtt ResourceLogicalID.AttributeName
- Returns value of an attribute from a resource
- Example: !GetAtt EC2Instance.PublicDnsName

Fn::Sub Function
- Syntax: !Sub "string with ${variables}"
- Substitutes variables in a string
- Useful for dynamic string creation

7. CLI Stack Operations
Creating a Stack
```
aws cloudformation create-stack \
 --stack-name mybucket \
 --template-body file://my_bucket.yaml
```
Updating a Stack
```
aws cloudformation update-stack \
 --stack-name mybucket \
 --template-body file://my_bucket.yaml
```
Supplying Parameters
```
aws cloudformation create-stack \
 --stack-name my-s3-bucket-stack \
 --template-body file://s3-bucket.yaml \
 --parameters ParameterKey=BucketName,ParameterValue=my-unique-bucket-name
```
8. Important Template Examples
Simple S3 Bucket

```
AWSTemplateFormatVersion: "2010-09-09"
Description: This is my first bucket
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
```

Parameterized S3 Bucket

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Creates an S3 bucket with a user-provided name.
Parameters:
  BucketName:
    Type: String
    Description: The name of the S3 bucket to create.
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref BucketName
```

Complete EC2 Example with Parameter and Output

```
Parameters:
  KeyPair:
    Description: SSH Key Pair
    Type: String
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-9d23aeea
      InstanceType: m3.medium
      KeyName: !Ref KeyPair
Outputs:
  ID:
    Value: !Ref Ec2Instance
    Description: ID of the EC2 instance created
  PublicName:
    Value: !GetAtt EC2Instance.PublicDnsName
    Description: Public DNS of the EC2
```

9. Advanced Concepts from Lab Example

DependsOn Attribute

- Specifies creation order of resources
- Ensures one resource is created before another

```
VPCtoIGWConnection:
  Type: AWS::EC2::VPCGatewayAttachment
  DependsOn: VPCGateway
```

Multi-line Strings in YAML

- Use | for multi-line strings
- Useful for user data scripts

Variable Substitution

- !Sub function for dynamic string creation
- Can reference parameters and resource attributes

Resource Types to Remember

- AWS::EC2::VPC: Virtual Private Cloud
- AWS::EC2::InternetGateway: Internet access
- AWS::EC2::Subnet: Network subnet
- AWS::EC2::RouteTable: Network routing
- AWS::EC2::SecurityGroup: Firewall rules
- AWS::EC2::Instance: Virtual machine
- AWS::RDS::DBSubnetGroup: Database subnet group

- AWS::S3::Bucket: Object storage

Week 9:

1. DevOps and CI/CD
DevOps Overview
- Definition: Set of practices, cultural philosophies, and tools that integrate and automate processes between software development (Dev) and IT operations (Ops)
- Goal: Enable faster and more reliable software delivery with improved collaboration

DevOps Culture Key Elements
- Increased transparency, communication and collaboration between teams
- Shared responsibilities
- Autonomous teams
- Fast feedback
- Automation

DevOps Best Practices
- Agile Project Management
- Shift left with CI/CD
- Build with the right tools
- DevOps lifecycle: discover, plan, build, test, monitor, operate, continuous feedback
- Implement automation
- Monitor the DevOps pipeline and applications
- Observability: Three pillars are logs, traces, and metrics
- Change the culture ("You build it you run it")

AWS DevOps Practices
- Microservice architecture
- Continuous integration and continuous delivery (CI/CD)
- Continuous monitoring and improvement
- Automation focused
- Infrastructure as code

DevOps Tools on AWS
- CI/CD: CodePipeline, CodeBuild, CodeDeploy, CodeConnections
- Microservices: ECS, Lambda, Fargate
- Platform as a Service: Elastic Beanstalk
- Infrastructure as Code: CloudFormation, OpsWorks, Systems Manager
- Monitoring and Logging: CloudWatch, CloudTrail, X-Ray, Config

CI/CD Concepts
Continuous Integration (CI)
- Software development practice where team members frequently integrate work to main branch
- Each change is built and verified to detect integration errors quickly
- Requires: Version control system + CI server for automated building/testing

Continuous Delivery vs Continuous Deployment
- Continuous Delivery: Extends CI by automatically releasing software to repository
- Continuous Deployment: Extends further by automatically deploying to production

CI/CD Pipeline Flow
Code → Build → Test → Deploy
↑_____CI_____↑
↑_____CD_____↑ (with manual intervention)
↑___CD (full)___↑ (automated deployment)

2. AWS CodePipeline
CodePipeline Overview
- Service that automates steps required to release/deploy software on AWS
- CI/CD pictured as pipeline with each stage as logical unit

AWS CI/CD Components

- AWS CodeCommit: Source control
- AWS CodeBuild: Build service
- AWS CodePipeline: Pipeline orchestration
- AWS CodeDeploy: Deployment service

Pipeline Structure

- Pipeline: Workflow construct describing how software changes go through release process
- Stages: Logical units representing independent steps (build, test, etc.)
- Pipeline Execution: Traversing stages in order, can end in success or failure
- Triggers: Can be configured to start automatically (e.g., commit events)

3. Decoupled Architecture

Tight Coupling Problems

- Three-tier tight coupling:
  - All communication is synchronous
  - Web server communicates directly with app server
  - Scaling requires code updates
  - Single point of failure
- Scaling complexity: Each new server requires multiple connections updated in code

Loose Coupling Solutions

Between Tiers

- Use Application Load Balancers (ALB) between tiers
- Adding new servers requires minimal connections
- Automatic workload management and failover routing

Within Applications - Microservices

- Break monolithic applications into microservices
- Each service handles specific function (finance, transcoding, calculations)
- Failure of one function doesn't bring down entire application
- Services can be scaled independently

Request Offloading

- Use Amazon SQS (queues) and Amazon SNS (topics)
- Asynchronous processing
- Decouples producers from consumers

4. Amazon SQS (Simple Queue Service)

Point-to-Point Messaging

- Producer: Sending application
- Consumer: Receiving application
- Message Queue: Decouples applications
- Pull mechanism: Consumer pulls messages from queue

SQS Overview

- Fully managed message queuing service
- Integrates and decouples distributed software systems
- Highly available, secure, and durable
- Provides AWS Console interface and web services API

SQS Components

Message

- Up to 256 KB in size
- Remains in queue until explicitly deleted or retention period expires

Queue Types

- Standard Queue:
  - At-least-once delivery
  - Best-effort ordering
  - Nearly unlimited throughput
  - Messages may arrive out of order or duplicated
- FIFO Queue:
  - First-in-first-out delivery
  - Exactly-once processing

    - High throughput
    - Maintains strict order

Queue Configuration

- Message retention period
- Visibility timeout: Time message is invisible after being received
- Receive message wait time:
    - Short polling (wait time = 0)
    - Long polling (wait time > 0) - more efficient

Dead Letter Queue (DLQ)

- Stores messages that cannot be consumed successfully
- Can be associated with any queue

SQS Message Lifecycle

1. Create: Producer sends message to queue, distributed redundantly
2. Process: Consumer picks up message, visibility timeout starts
3. Delete: Consumer deletes message after successful processing

SQS Example Use Case

- Order capture applications (producers) send orders to queue
- Order fulfillment applications (consumers) process orders
- Message deleted only if processed successfully
- Failed messages can go to dead letter queue

5. Amazon SNS (Simple Notification Service)

Publish/Subscribe Messaging

- Publisher: Sending application (has little knowledge of receivers)
- Subscribers: Receiving applications
- Topic: Decouples applications
- Push mechanism: Topic pushes messages to subscribers

SNS Overview

- Fully managed pub/sub messaging service
- Decouples applications through notifications
- Highly scalable, secure, and cost-effective
- Provides AWS Console interface and web services API

Types of Subscribers

- Email destination
- Mobile text messaging
- Mobile push notifications
- HTTP/HTTPS endpoints
- AWS Lambda functions
- SQS queues
- Amazon Kinesis Data Firehose delivery streams

SNS Use Cases

- Application and system alert notifications
- Email and text message notifications
- Mobile push notifications

SNS Considerations

- Message Publishing: Single published message, no recall options
- Message Delivery:
    - Standard topic: Order doesn't matter
    - FIFO topic: Exact message delivery order required
    - Customizable delivery policy for HTTP/HTTPS endpoints

SNS + SQS Integration Example

1. Image uploaded to S3 bucket
2. S3 triggers SNS notification
3. SNS topic distributes to multiple SQS queues (mobile size, web size, thumbnail)
4. Different Auto Scaling groups process each queue
5. Results stored in converted images S3 bucket

6. SNS vs SQS Comparison

Feature	Amazon SNS	Amazon SQS
Messaging Model	Publisher-Subscriber	Producer-Consumer
Distribution Model	One to many	One to one
Delivery Mechanism	Push (passive)	Pull (active)
Message Persistence	No	Yes

Week 10:

1. Containerization Fundamentals

What is Containerization?

- Definition: Operating-system-level virtualization where the kernel allows multiple isolated user-space instances
- Containers: Isolated instances that look like real computers to programs running inside them
- Also called: partitions, virtualization engines (VEs), or jails

Linux Kernel Components

- Four Basic Jobs:
  - Memory Management
  - Process Management
  - Device Drivers
  - System Calls and Security
- Linux System = Kernel + system libraries/tools (e.g., GNU tools like gcc)
- Linux Distribution = Pre-packaged Linux system + applications (e.g., Redhat, Debian, Amazon Linux)

Key Linux Kernel Features for Containers

Namespaces

Purpose: Provide containers their own view of the system to avoid conflicts

Types of Namespaces:

- Mount (mnt): File system isolation
  - Each container has its own rootfs
  - Independent mount points
- Process ID (pid): Process isolation
  - Each container has numbering starting at 1
  - When PID 1 exits, all other processes exit immediately
  - Nested namespaces (same process can have different PIDs)
- User ID: User segregation and privilege isolation
  - Mapping between container UID and host OS UID
  - UID 0 (root) in container may have different UID on host

Control Groups (Cgroups)

Purpose: Control kernel resources allocated to each container/process

- Functions: Metering and limiting resources
- Resources Controlled:
  - Memory
  - CPU
  - I/O (File and Network)

Container Runtimes

Definition: Enables containers to run by setting up namespaces and cgroups (like hypervisor counterpart)

Types:

- Low-level: Focus on just running containers (LXC, Systemd-nspawn, OpenVZ)
- High-level: Include image formats, image management, etc. (Docker)

2. Container vs Virtual Machine

Key Differences

Aspect	Containers	Virtual Machines
OS	Share host kernel	Each VM has full OS
Resources	Lightweight, no OS overhead	Full OS resource consumption
Startup Time	Fast (like starting application)	Slower (OS boot time)
Isolation	Reasonable (kernel namespaces/cgroups)	Excellent (hardware support)
Security	Good	Very good (mature technology)

Container Lifecycle

- Container exits when internal process finishes
- Unless specified as interactive or running a server
- Much faster startup than VMs

3. Docker Overview

What is Docker?

- Most famous container runtime
- More than runtime: Packaging and deployment system built on container technology
- Large ecosystem of components
- Key features: dependency management and "deploy everywhere"

Main Docker Concepts

Images

- Package containing application and its environment
- Defined in Dockerfile (text document with build instructions)
- Similar to makefiles, ant build files, Maven POM files
- Composed of read-only layers
- Base layers shared between images

Registries

- Repository storing Docker images
- Facilitates easy sharing between people and computers

Containers

- Regular Linux container created from Docker image
- Running container = process on Docker host

Docker Image Layers

- Image layers: Read-only templates
- Container layer: Thin writable layer added on top when running
- Benefits:
    - Space efficiency: Shared base images
    - Fast startup: Only add writable layer
- Data persistence: Writable layer deleted when container exits

Docker Storage Options

Data Volumes

- Bind mount: Mount host file/directory into container
- Volumes: Docker-managed designated location on host
- tmpfs: Uses host memory (rarely used)

Docker Networking Options

Default: Containers have networking enabled for outgoing connections

Network Drivers:

- Host: Removes isolation, container connects directly to host NIC
- Bridge (default): Docker manages private network with NAT translation
- None: No networking
- Overlay: For multi-host networking

Example Commands

- Host: docker run -d --name nginx-1 --net=host nginx
- Bridge: docker run -d --name nginx-1 -p 10000:80 nginx

4. Microservices Architecture

Traditional MVC vs Microservices

- MVC: Components strongly connected (monolithic)
- Microservices: Decouple services into individual deployable services
- Communication: RESTful API over HTTP with stable interfaces

Benefits of Microservices

- Independent deployment: Services deployed separately on different hosts
- Scalability: Scale individual services based on demand
- Technology diversity: Different services can use different technologies

Container + Microservices Integration

- One service per container principle
- Simplified deployment: All dependencies included
- Resource efficient: More efficient than VMs
- Orchestration needed: For multi-container distributed applications

Scaling Approaches

- Less desirable: Instance-level scaling (whole EC2 instances)
- More desirable: Container-level scaling and scheduling

5. AWS Elastic Container Service (ECS)

AWS Container Services Ecosystem

- Registry: Amazon ECR (Elastic Container Registry)
- Orchestration: Amazon ECS, Amazon EKS (Kubernetes)
- Compute: Amazon EC2, AWS Fargate, AWS Lambda

Decomposing Monoliths with ECS

Step 1: Create Container Images

- Break monolithic app into service-specific images
- Push images to Amazon ECR
- Example: Users service, Topics service, Messages service

Step 2: Create Service Task Definitions

Service Task Definition includes:

- Launch type (EC2 or Fargate)
- Service name
- Image ECR repo URL and version
- CPU allocation (e.g., 256)
- Memory allocation (e.g., 256)
- Container and host ports

Target Groups:

- Service name
- Protocol (HTTP)
- Port (80)
- VPC configuration

Step 3: Connect Load Balancer

- Application Load Balancer with listeners
- Listener Rules: Route based on path (e.g., /api/users → Users service)
- Target Groups: Connect to specific services

ECS Auto Scaling

- Cluster Auto Scaling: Automatically adjust EC2 instances
- Service Auto Scaling: Adjust number of tasks per service

Inter-Service Communication Options

1. Application Load Balancer
   - For clear directional communication
   - Path-based routing
2. Message Queues
   - Handle asynchronous communication
   - Decouple services
3. ECS Service Discovery
   - Services auto-register with DNS name in CloudMap Namespace
   - Route 53 resolves service locations

- Services communicate using DNS names

4. ECS Service Connect
   - Services register with logical names in CloudMap
   - ECS handles routing automatically
5. AWS App Mesh
   - Advanced approach
   - Dedicated infrastructure layer for service-to-service communication

Week 11:

1. Function as a Service (FaaS) Fundamentals

Evolution of Cloud Computing

- Berkeley View (2009): Identified XaaS models and two virtualization approaches
  - EC2 Spectrum: Full control over software stack from kernel upward
  - App Engine Spectrum: Application domain-specific platforms with clean separation between stateless computation and stateful storage
- Market Decision: IaaS won the first battle due to AWS's market dominance

Why IaaS Initially Won

- Early adopters wanted familiar environments: Recreate local computing environment in cloud
- Benefits: Experienced users, low migration cost, low risk
- Downsides:
  - Developers must manage VMs and environment setup
  - Requires significant administrative knowledge
- Evolution: Success of cloud computing → users willing to give up control for simpler operation

Serverless Computing Origins

- Original Purpose: Handle low duty-cycle workloads (infrequent file processing)
- Example Use Case: Phone app uploads images → cloud creates thumbnails → places on web
- Problem: Simple code requiring entire web application stack for infrequent requests is cost-ineffective
- AWS Lambda (2015): User writes function with simple configuration, Amazon provisions environment on invocation, charged by execution time

Critical Distinctions of Serverless

1. Decoupled computation and storage
   - Computation and storage scale independently
   - Computation is stateless, state saved elsewhere (S3, Database)
2. Execute code without managing resource allocation
   - User provides code, cloud automatically provisions resources
3. Pay for resources used, not allocated

2. Amazon Serverless Computing

AWS Lambda Basics

- Fully managed compute service
- Triggers: Runs on schedule or in response to events (S3 bucket changes, DynamoDB table changes)
- Supported Languages: Java, Go, PowerShell, Node.js, C#, Python, Ruby, Runtime API
- Edge Capability: Can run at edge locations closer to users

Key Concepts

- Function: Code that Lambda can run
- Trigger: Resource/configuration that invokes Lambda function
- Event: JSON document passed to Lambda for processing
- Execution Environment: Container/VM that runs the function
- Runtime: Language-specific environment within execution environment
- Deployment Package: Prebuilt package with code and dependencies (ZIP or container image)

Lambda Function Anatomy

```
import json
```

```
def lambda_handler(event, context):
  # Handler: Function run upon invocation
  # Event object: Data sent during invocation
  # Context object: Methods for runtime interaction
  return {
    'statusCode': 200,
    'body': json.dumps('Hello World')
  }
```

Lambda Features

- Layers: Share code across multiple functions
- Local Storage: Temporary /tmp storage (ephemeral, exists only during invocation)
- Common Use Cases: Download S3 files for processing, store intermediate results

Invocation Methods

1. Lambda console (testing)
2. AWS SDK (programmatically)
3. AWS CLI: aws lambda invoke --function-name my-function
4. Other services via triggers

Invocation Types

- Synchronous: External API requests (API Gateway, Lambda function URL)
- Asynchronous: AWS service triggers, scheduled events

3. Lambda Execution Environment

Multitenancy and Isolation

- Multitenancy Benefits: Elasticity, improved server utilization
- Isolation Challenges: Security and performance concerns

Trusted vs Untrusted Workloads

- Trusted Workloads: Internal applications, lower isolation needs, performance optimized (Kubernetes, ACS)
- Untrusted Workloads: Different clients, potentially malicious, require strong isolation

Lambda Execution Evolution

- Early Version:
  - Container technology for function isolation
  - Virtualization for user account isolation
  - Each function runs in own container
  - Functions from same account may share VM
  - Different accounts always use different VMs
  - Problem: Inefficient resource usage (at least one VM per account)

MicroVM Solution

- Purpose: Platform for untrusted workloads sharing resources efficiently
- Benefits: VM-level isolation with better utilization and performance
- Characteristics: Minimum VM offering high isolation with relatively low resource consumption

Comparison Table

Feature	Container	MicroVM	Traditional VM
Isolation	Medium	High	High
Resource Consumption	Low	Medium	High
Start Latency	Low	Medium	High
Use Case	Microservices	Serverless	General Purpose

Firecracker

- Lambda's MicroVM Implementation: Lightweight virtualization technology
- Benefits: High isolation with lower resource consumption than traditional VMs

Lambda Function Location Options

Default Configuration

- Runs in Lambda-managed VPC
- Has public internet access

- Can connect to external services
- Cannot be configured or seen by user
- Access to AWS services (S3, DynamoDB) through preconfigured endpoints
- Cannot access your own VPC resources (like RDS)

VPC Configuration

- Place Lambda function inside your VPC (public/private subnet)
- Configure VPC endpoints for services not in VPC
- Trade-off: Access to your VPC resources but more complex networking

4. Lambda Execution Anti-Patterns

General Design Principles

- Prefer many short functions over fewer large ones
- Each function should handle only the event passed to it
- Functions shouldn't have knowledge of overall workflow or other functions

1. The Lambda Monolith

Problems:

- All application logic in single function triggered for all events
- Hard to upgrade, maintain, reuse, test
- Package size too big
- Hard to enforce least privilege

Solution: Break into multiple focused functions

2. Recursive Patterns

Problem: S3 upload triggers Lambda → Lambda puts object back in same bucket → infinite loop

Solution: Carefully configure trigger events, use different input/output services

3. Lambda Functions Calling Lambda Functions

Problems:

- Deep call stacks don't work well in serverless
- Multiple concurrent functions with some just waiting (unnecessary cost)
- Complex error handling
- Workflow limited by slowest function
- Scaling complexity

Solutions:

- Use message queues
- AWS Step Functions for orchestration

4. Synchronous Waiting Within Single Function

Problem: Function waits for external process to complete Solution: Split into separate functions with event-driven communication

5. Amazon API Gateway

Overview

- Purpose: Create, publish, maintain, monitor, secure APIs as entry points to backend resources
- Capacity: Handles hundreds of thousands of concurrent API calls
- Backend Support: EC2, Lambda, any web application, real-time communication apps
- Versioning: Host multiple versions and stages of APIs

Security Features

1. DDoS Protection: Protection from distributed denial of service and injection attacks
2. Resource Policies: Apply fine-grained access control
3. Authorization: Require proper authorization
4. Throttling: Control request rates

Common Architecture Patterns

RESTful Microservices

- API Gateway → Lambda → DynamoDB
- Clean separation of concerns
- Scalable and maintainable

Serverless Mobile Backend

- Mobile apps → API Gateway → Lambda functions
- Integration with:

    - DynamoDB (NoSQL database)
    - S3 (object storage)
    - RDS (relational database)
    - ElastiCache (caching)
    - Cognito (identity management)

Integration Points

- CloudFront: CDN integration
- CloudWatch: Monitoring and logging
- VPC: Private network integration
- Multiple AWS Services: Comprehensive ecosystem integration