DOMAIN 2: DESIGN FOR NEW SOLUTIONS

- Weight: 29%
- Core Competency
  - Designing greenfield AWS architectures that are highly available, scalable, secure, and cost-optimized, while fulfilling specific business requirements.
- 2.1 Design Scalable and Highly Available Solutions
  - Core Design Principles
    - Fault tolerance: Design systems to operate even when components fail
    - High availability: Maximize uptime via redundancy
    - Elasticity: Auto-adjust resources with load
    - Resiliency: Recover from infrastructure and service-level failures
  - Key AWS Services
    - ELB (ALB/NLB): Distribute traffic across AZs
    - ASG (Auto Scaling Group): Automatically add/remove EC2 instances
    - S3: Regional storage across 3+ AZs
    - Aurora: Multi-AZ and up to 15 read replicas
    - Route 53: Global DNS with latency and health-based routing
    - Lambda: Serverless scaling per request
    - Global Accelerator: Low-latency routing across AWS edge locations
- 2.2 Design Loosely Coupled and Event-Driven Architectures
  - Why Decouple?
    - Prevent cascading failures
    - Isolate components for easier scaling
    - Increase maintainability and flexibility
  - Event-Driven Patterns
    - SQS: Message queue for asynchronous processing
    - SNS: Pub/sub for parallel delivery
    - EventBridge: Event routing and filtering
    - Lambda: Serverless event consumers
    - Step Functions: Orchestration with retries, error handling, branching
- 2.3 Design Serverless and Container-Based Solutions
  - Serverless Design
    - Lambda: Stateless compute
    - API Gateway: HTTPS REST or WebSocket APIs
    - DynamoDB: NoSQL, serverless with on-demand scaling
    - Step Functions: Workflow orchestration
    - S3 + EventBridge: Trigger events on object uploads
    - Aurora Serverless v2: Scalable relational database
    - SageMaker: Serverless ML training & inference
  - Patterns
    - Use Provisioned Concurrency for Lambda cold start mitigation
    - Use DLQ for failed events (Lambda, SQS)
  - Containerized Design
    - ECS: Fully managed container orchestration
    - Fargate: Serverless compute for containers
    - EKS: Kubernetes orchestration
    - App Mesh: Service mesh for microservices communication
    - Service Discovery: Via Cloud Map or Route 53 for ECS tasks
- 2.4 Design for Performance Efficiency
  - Storage
    - Object storage: S3 + Lifecycle policies
    - File storage: EFS (Linux), FSx (Windows, Lustre)
    - Block storage: EBS gp3/io2
    - Archive: S3 Glacier, Deep Archive
  - Compute
    - Use Compute Optimizer to identify instance sizing
    - Use Spot Instances for batch/fault-tolerant workloads
    - Use ARM-based Graviton2/3 for cost and performance
  - Database Selection
    - Relational, high performance: Aurora, RDS
    - Scalable NoSQL: DynamoDB (with DAX/streams)
    - Caching: ElastiCache (Redis/Memcached)
    - Time-series: Amazon Timestream
    - Analytics: Redshift, Athena, QuickSight
- 2.5 Design for Security and Compliance
  - Identity & Access
    - Fine-grained access: IAM policies, Resource policies
    - Application auth: Cognito, IAM Identity Center
    - Federation: SAML, OIDC + AssumeRole
  - Encryption
    - At rest: KMS, SSE-S3, SSE-KMS, EBS encryption
    - In transit: TLS 1.2+, ELB SSL termination, HTTPS
    - Use PrivateLink, VPC Endpoints to avoid Internet exposure
    - Enable GuardDuty, Security Hub, Macie for monitoring
- 2.6 Design Cost-Optimized Solutions
  - Cost-Saving Strategies
    - Compute: Spot, Fargate, Lambda
    - DB: Aurora Serverless, DynamoDB on-demand
    - Storage: S3 Intelligent-Tiering, Lifecycle rules
    - Data Transfer: Use VPC endpoints, CloudFront, NAT instances
    - Observability: CloudWatch with custom retention
    - Use Budgets, Cost Explorer, Trusted Advisor
- Common SAP-C02 Scenarios (Domain 2)
  - Ingest billions of events per hour and process them serverlessly: Use Kinesis + Lambda + S3
  - Run a global web app with <100ms latency worldwide: Use CloudFront + Global Accelerator + Multi-region deployment
  - Low-maintenance architecture for a retail app with unpredictable traffic: Use Lambda, API Gateway, DynamoDB, S3
  - High-throughput file system shared across AZs for a media firm: Use Amazon FSx for Lustre
- Summary Cheat Sheet
  - Task
    - Scale compute: ASG, Lambda, Fargate
    - Orchestrate workflows: Step Functions
    - Event-based design: SQS, SNS, EventBridge
    - Secure workloads: KMS, IAM, PrivateLink
    - Optimize performance: S3 + Glacier, DynamoDB + DAX, EBS gp3
    - Monitor apps: CloudWatch, X-Ray, Config
    - Optimize cost: Spot, Tiered storage, Budgets
- Recommended Resources
  - AWS Well-Architected Framework
  - Serverless Land Patterns
  - AWS Lambda Best Practices
  - DynamoDB Design Patterns
  - Event-Driven Architecture on AWS

Presented with xmind AI