COMP5349 Assignment 2 Report: AWS Project

Unikey- akur0860 Student ID: 550217239

Introduction

This report documents the deployment of an enhanced image annotation application that combines traditional web application architecture with serverless computing components. The solution implements a scalable, highly available system that automatically processes uploaded images using generative AI services while maintaining robust infrastructure through auto-scaling capabilities.

The architecture leverages AWS cloud services to create a two-tier system: a web application component hosted on EC2 instances with auto-scaling and load balancing capabilities, and a serverless component using AWS Lambda functions for automated image processing. This hybrid approach ensures both reliability for user-facing operations and cost-effective, event-driven processing for backend tasks.

Key architectural features include multi-availability zone deployment for high availability, private subnet hosting for enhanced security, automatic scaling based on traffic demands, and seamless integration between web and serverless components through S3 event triggers.

Architecture Diagrams

Web Application Architecture

The web application architecture consists of the following key components deployed across multiple availability zones:

Network Layer:

- Custom VPC with public and private subnets in two availability zones
- Internet Gateway for public subnet internet access
- NAT Gateway enabling private subnet internet connectivity
- Route tables configured for appropriate traffic routing

Compute Layer:

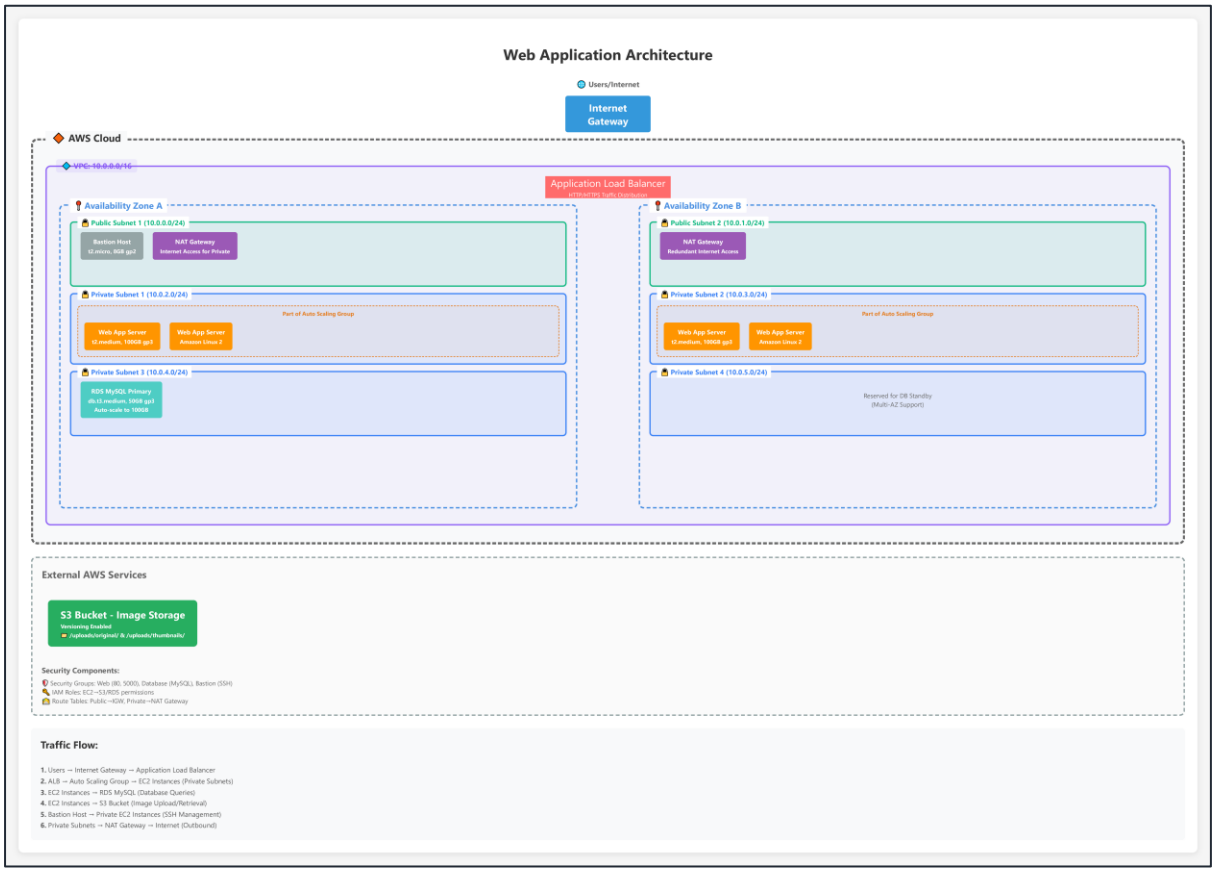
- EC2 instances in private subnets for enhanced security
- Application Load Balancer (ALB) in public subnets for traffic distribution
- Bastion host in public subnet for secure administrative access
- Auto Scaling Group managing EC2 instance lifecycle

Storage and Database:

- RDS MySQL database in private subnets with multi-AZ deployment
- S3 bucket for image storage with organized folder structure
- EBS volumes providing persistent storage for EC2 instances

Security Components:

- Security groups controlling inbound/outbound traffic
- IAM roles and policies for service permissions
- VPC isolation providing network-level security



Serverless Architecture

The serverless architecture handles automated image processing through event-driven Lambda functions:

Event Sources:

- S3 bucket configured with event notifications
- EventBridge integration for function orchestration

Processing Functions:

- Image Annotation Function integrating with Gemini API
- Thumbnail Generator Function using Python Pillow library

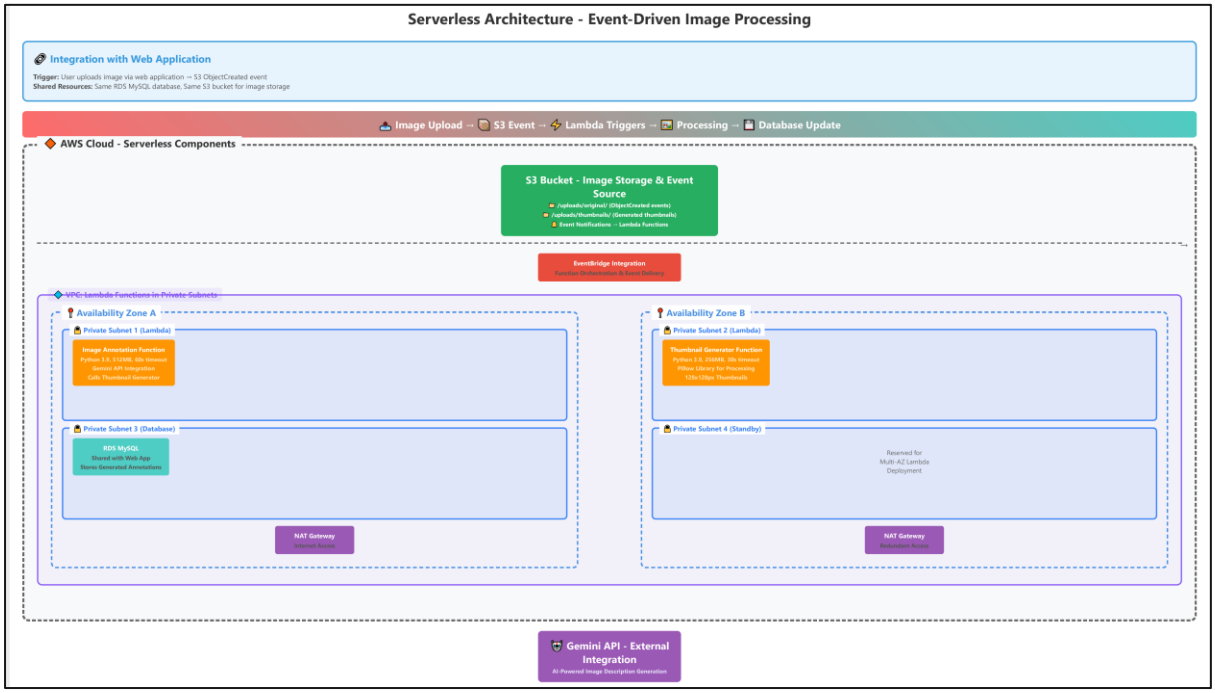
External Integrations:

- Gemini API for AI-powered image description generation
- RDS database for storing generated annotations

- S3 bucket for thumbnail storage

Supporting Infrastructure:

- VPC configuration for Lambda functions
- IAM roles enabling cross-service communication
- Environment variables for configuration management



Integration Between Components

The integration between web application and serverless components occurs through several key interaction points:

1. **S3 Event Triggers:** When users upload images through the web application to the S3 bucket, S3 ObjectCreated events automatically trigger the Lambda functions

2. **Shared Database Access:** Both web application and Lambda functions access the same RDS MySQL database for storing and retrieving image metadata and annotations
3. **Unified Storage:** The web application displays images and thumbnails stored in the same S3 bucket, with Lambda functions generating thumbnails in a separate folder structure

Web Application Deployment

Compute Environment

EC2 Configuration: The web application is deployed on EC2 instances using t2.medium instance types, specifically selected because they handle high-demand tasks efficiently with their balanced CPU and memory resources (2 vCPUs, 4 GiB RAM). This instance type provides sufficient computational power for web application processing while remaining cost-effective for scaling operations.

Amazon Linux 2 was chosen as the operating system for several critical reasons:

- **Stability and Security**: Provides a stable, secure foundation with regular security updates
- **AWS Integration**: Pre-integrated AWS tools and CLI simplify service interactions and reduce configuration overhead
- **Performance Optimization**: Optimized for AWS infrastructure with better performance characteristics
- **Cost Efficiency**: No licensing fees compared to commercial operating systems

Auto Scaling Group Setup: A Launch Template was created to ensure consistent and repeatable instance launches across the Auto Scaling Group. The ASG is configured to automatically add or remove EC2 instances based on traffic demands, providing both cost optimization and performance scaling.

Storage Justification: EBS volumes are configured with gp3 storage type (100GB) for several strategic reasons:

- **Performance**: gp3 provides more consistent performance compared to gp2 with baseline performance of 3,000 IOPS
- **Flexibility**: Independent control over IOPS and throughput, allowing optimization without over-provisioning
- **Cost Efficiency**: Better price-performance ratio, especially for applications requiring consistent I/O
- **Scaling Consideration**: 100GB provides sufficient space for application files, logs, and temporary data while being cost-efficient across multiple scaled instances

Network Configuration: The deployment utilizes a custom VPC with strategically designed subnets:

- Private subnets hosting web application instances for enhanced security
- Public subnets containing the bastion host and load balancer
- Multi-availability zone deployment ensuring high availability
- NAT Gateway configuration enabling private subnet internet access for software updates and external API calls

Security Configuration: Multiple security groups control traffic flow:

- Web application security group allowing HTTP traffic on ports 80 and 5000
- Database security group restricting RDS access to EC2 instances only

- Bastion host security group permitting SSH access from specific IP ranges

IAM roles provide necessary permissions for EC2 instances to interact with S3, RDS, and other AWS services without embedded credentials.

Administrative Access: A bastion host deployed in the public subnet provides secure SSH access to private subnet instances. This architecture follows security best practices by eliminating direct internet connectivity to application servers while maintaining administrative capabilities.

Bastion Host Configuration Justification:

- **Instance Type**: t2.micro selected for cost efficiency as it only handles SSH traffic, not application workloads
- **Storage**: 8GB gp2 storage sufficient for basic OS and administrative tools
- **Placement**: Public subnet enables direct SSH access while serving as secure gateway to private resources
- **Security**: Dedicated security group restricts SSH access to specific IP ranges, preventing unauthorized access

Load Balancer Setup

Application Load Balancer Configuration: The ALB is deployed across public subnets in multiple availability zones, ensuring high availability and fault tolerance. Listener configuration is set to accept HTTP traffic on port 80 and forward requests to the target group containing EC2 instances.

Target Group Configuration: Target groups are configured to route traffic to web application instances running on port 80. Health checks are implemented to ensure only healthy instances receive traffic, with automatic removal of unhealthy instances from the load balancing rotation.

Traffic Distribution: The ALB uses round-robin distribution to balance incoming requests across multiple EC2 instances, ensuring optimal resource utilization and preventing any single instance from becoming a bottleneck.

Database Environment

RDS MySQL Configuration: The database is deployed using db.t3.medium instance class with 2 vCPUs to handle concurrent connections and query processing efficiently. This configuration was selected for several reasons:

- **Performance Requirements**: 2 vCPUs provide sufficient processing power for concurrent database operations from multiple EC2 instances
- **Memory Optimization**: 4 GiB RAM supports efficient query caching and connection handling
- **Cost-Performance Balance**: Medium instance size balances performance needs with cost efficiency
- **Scalability**: Can handle increased load from auto-scaled web application instances

Storage Configuration: gp3 storage is configured with 50GB initial capacity and auto-scaling enabled up to 100GB for several strategic reasons:

- **Performance Consistency**: gp3 provides baseline 3,000 IOPS ensuring consistent database performance

- **Cost Optimization**: More cost-effective than gp2 for databases requiring consistent I/O performance
- **Auto-scaling**: Prevents storage exhaustion as application usage grows, with 100GB maximum providing sufficient headroom
- **Backup Efficiency**: Smaller initial size reduces backup time and storage costs while allowing growth

Security and Access: The RDS instance is deployed in private subnets with security groups restricting access to EC2 instances only. Database credentials are managed securely, and connection pooling is implemented to optimize database performance.

Monitoring and Logging: Enhanced monitoring and logging are enabled to track database performance metrics and maintain audit trails for troubleshooting and optimization purposes.

Storage Environment

S3 Bucket Configuration: The S3 bucket is configured with versioning enabled to prevent data loss from accidental overwrites when users upload images with identical names. This ensures data integrity and provides rollback capabilities if needed.

Folder Structure: An organized folder structure is implemented:

- /uploads/original/ - storing original uploaded images
- /uploads/thumbnails/ - storing generated thumbnail images

Event Configuration: S3 bucket events are configured to trigger Lambda functions automatically when new objects are created in the uploads folder, enabling seamless integration between storage and processing components.

Access Policies: Bucket policies are configured to allow read and write access from EC2 instances and Lambda functions while maintaining security through IAM role-based permissions rather than public access.

Serverless Component Deployment

Event-Driven Architecture

The serverless architecture is built around S3 ObjectCreated events that automatically trigger Lambda functions when new images are uploaded. EventBridge integration provides additional orchestration capabilities and ensures reliable event delivery between components.

Lambda functions are deployed in VPC private subnets to maintain network security consistency with the web application tier while enabling access to RDS and other VPC resources.

Annotation Function

Packaging and Deployment: The Image Annotation Function is packaged as a deployment package containing the application code and necessary dependencies for Gemini API integration. The function is deployed through the AWS Management Console with the following specifications:

- **Deployment Method**: AWS Management Console for direct deployment and configuration
- **Runtime**: Python 3.9 for compatibility with required libraries
- **Memory Allocation**: 512 MB to handle image processing and API calls efficiently

- **Timeout**: 60 seconds to accommodate Gemini API response times
- **Packaging**: Deployment package includes application code and dependencies

Integration Settings and Justifications: The function is configured with specific permissions and settings:

- **S3 Permissions**: Read access to retrieve uploaded images from the uploads folder
- **RDS Access**: VPC configuration and security group membership enabling database connectivity
- **Internet Access**: NAT Gateway routing for external Gemini API calls
- **Environment Variables**: Database connection parameters, API credentials, and configuration settings stored securely
- **Trigger Configuration**: S3 ObjectCreated events filtered to uploads folder for automatic activation

Processing Logic: The function implements coordinated processing by calling the Thumbnail Generator Function directly, ensuring both annotation and thumbnail generation occur as a unified process. This design choice eliminates the need for separate event handling and reduces processing latency.

VPC Configuration: The function is deployed within the VPC private subnets with appropriate security group configurations enabling database connectivity while maintaining network isolation.

Thumbnail Generator Function

Packaging and Deployment: The Thumbnail Generator Function utilizes the Python Pillow library for image processing, requiring specific packaging and deployment considerations:

- **Deployment Method**: AWS Management Console for direct deployment and testing
- **Runtime**: Python 3.9 with Pillow library dependency
- **Memory Allocation**: 256 MB sufficient for image processing operations
- **Timeout**: 30 seconds adequate for thumbnail generation tasks
- **Packaging**: Deployment package includes Pillow library and image processing utilities

Image Processing Justification: The function implements specific image processing requirements:

- **Thumbnail Size**: 128x128 pixels chosen for optimal balance between file size and visual quality
- **Compression**: Maintains acceptable image quality while reducing file size for faster web loading
- **Format Consistency**: Standardizes thumbnail format regardless of original image type
- **Aspect Ratio**: Preserves original image proportions to prevent distortion

Performance Optimization: Configuration choices optimize processing efficiency:

- **Memory Allocation**: 256 MB provides sufficient memory for image processing without over-provisioning
- **Concurrent Execution**: Function can handle multiple simultaneous thumbnail generation requests
- **Error Handling**: Robust error handling ensures processing continues even with problematic images

Auto Scaling Test Observations

Test Methodology

Load testing was conducted using a Python-based concurrent request generator to simulate high traffic conditions on the image listing page. The test generated sustained traffic over a 10-minute period with varying request rates to trigger auto-scaling events.

Scaling Out Behavior

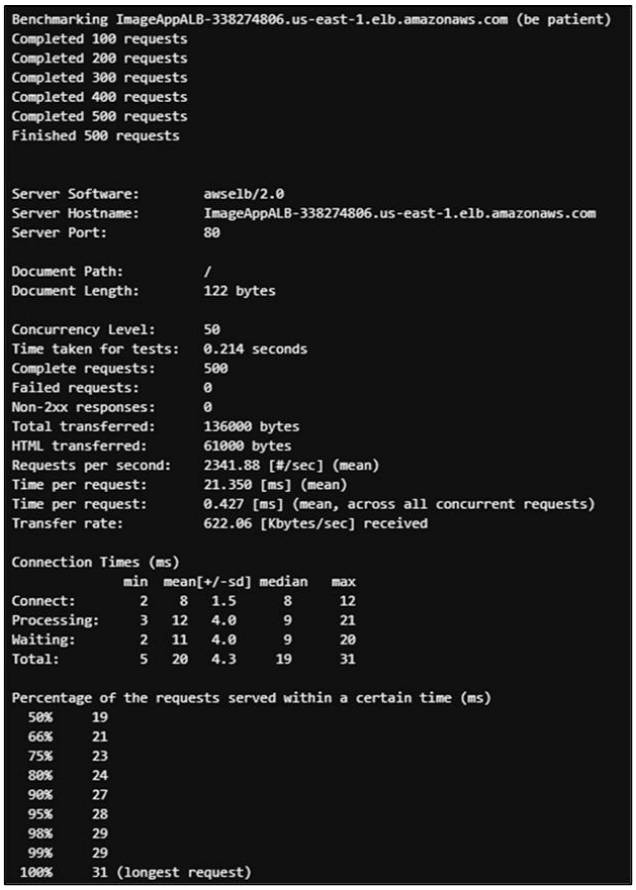
During peak load periods, the Auto Scaling Group successfully detected increased CPU utilization and launched additional EC2 instances. CloudWatch metrics showed CPU utilization exceeding the configured threshold, triggering the scale-out policy. New instances were automatically registered with the target group and began receiving traffic distribution from the ALB.

Scaling In Behavior

Following load reduction, the Auto Scaling Group demonstrated proper scaling-in behavior by terminating excess instances when CPU utilization dropped below the configured threshold. This automatic resource optimization ensures cost efficiency during low-traffic periods.

Load Distribution Evidence

ALB monitoring confirmed successful traffic distribution across multiple EC2 instances, with request counts balanced appropriately between healthy targets. Health check monitoring showed consistent instance health status throughout the scaling events.



```
Benchmarking ImageAppALB-338274806.us-east-1.elb.amazonaws.com (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        awselb/2.0
Server Hostname:        ImageAppALB-338274806.us-east-1.elb.amazonaws.com
Server Port:            80

Document Path:          /
Document Length:        122 bytes

Concurrency Level:      50
Time taken for tests:   0.214 seconds
Complete requests:      500
Failed requests:        0
Non-2xx responses:      0
Total transferred:      136000 bytes
HTML transferred:       61000 bytes
Requests per second:    2341.88 [#/sec] (mean)
Time per request:       21.350 [ms] (mean)
Time per request:       0.427 [ms] (mean, across all concurrent requests)
Transfer rate:          622.06 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2    8   1.5      8      12
Processing:     3   12   4.0      9      21
Waiting:        2   11   4.0      9      20
Total:          5   20   4.3     19      31

Percentage of the requests served within a certain time (ms)
  50%     19
  66%     21
  75%     23
  80%     24
  90%     27
  95%     28
  98%     29
  99%     29
 100%     31 (longest request)
```

Summary and Lessons Learned

Key Findings

The deployment successfully demonstrates a robust, scalable architecture combining traditional web application hosting with modern serverless computing. The integration between EC2-based web services and Lambda-based processing creates an efficient system that balances performance, cost, and reliability.

The multi-tier architecture with private subnet deployment significantly enhances security while maintaining operational efficiency through proper NAT Gateway and bastion host configuration.

Challenges and Solutions

Challenge: Initial Lambda function connectivity issues to RDS from VPC subnets. **Solution:** Proper security group configuration and NAT Gateway setup resolved connectivity problems while maintaining security isolation.

Challenge: S3 event trigger configuration for reliable Lambda function execution.

Solution: Careful event filter configuration and proper IAM permissions ensured consistent function triggers without excessive invocations.

Challenge: Auto Scaling Group optimization for responsive scaling without over-provisioning.

Solution: Fine-tuning CloudWatch metric thresholds and scaling policies achieved optimal balance between responsiveness and cost efficiency.

Technical Achievements

- Successful implementation of hybrid architecture combining traditional and serverless components
- Effective auto-scaling configuration providing both performance and cost optimization
- Secure network architecture with proper subnet isolation and access controls
- Reliable event-driven processing with robust error handling and monitoring
- Seamless integration between multiple AWS services creating a cohesive application experience