Advanced AWS Course Structure for Intermediate Students

Module 1: Advanced Networking in AWS

- Deep Dive into VPC Peering, VPN, and Direct Connect
- Implementing Advanced Routing Techniques
- Hybrid Cloud Architectures

Module 2: Deep Dive into AWS Storage Solutions

- Advanced S3 Features (Lifecycle Policies, Versioning, Cross-Region Replication)
- Amazon S3 Glacier for Data Archiving
- Data Lake Architecture with Amazon S3

Module 3: High Availability and Fault Tolerance

- Designing Highly Available Architectures
- Multi-AZ and Multi-Region Deployments
- Disaster Recovery Strategies

Module 4: Scalable and Serverless Architectures

- Scaling Applications with Amazon ECS and EKS
- Building Serverless Applications with AWS Lambda and Amazon API Gateway
- Serverless Application Model (SAM)

Module 5: Advanced Database Solutions

- Amazon RDS Performance Optimization
- DynamoDB Advanced Features (Global Tables, DAX)
- Amazon Aurora Deep Dive

Module 6: Security, Governance, and Compliance

- Advanced IAM Policies and Strategies
- AWS Shield and WAF for Advanced Threat Protection
- Compliance and Governance using AWS Config and AWS Organizations

Module 7: DevOps and Continuous Integration/Continuous Deployment

- Implementing Full CI/CD Pipelines with AWS Developer Tools
- Infrastructure as Code with AWS CloudFormation and AWS CDK
- Microservices Deployment with Docker and Kubernetes on AWS

Module 8: Monitoring, Logging, and Troubleshooting

- Advanced Monitoring with Amazon CloudWatch and AWS X-Ray
- Log Analysis with Amazon Elasticsearch Service
- Troubleshooting Common AWS Issues

Module 9: Cost Optimization and Management

- Cost Optimization Techniques and Tools
- Reserved Instances and Savings Plans
- Managing and Reporting AWS Costs

Module 10: Emerging Technologies and Trends

- Cost Optimization Techniques and Tools
- Reserved Instances and Savings Plans
- Managing and Reporting AWS Costs

Hands-on Projects

Networking Project: Hybrid Cloud Solution Design and Implementation

Project Overview:

A simulated enterprise seeks to extend its on-premises infrastructure to AWS to leverage cloud resources for certain workloads while keeping sensitive operations on-premises. This project involves creating a hybrid cloud architecture that ensures seamless connectivity, security, and scalability between the on-premises data center and AWS services.

Objective:

Design and implement a robust hybrid cloud solution using AWS Direct Connect and VPN, ensuring secure, low-latency connectivity between the on-premises environment and AWS.

Task 1: Assess Requirements and Plan the Hybrid Architecture

Objective: Identify the enterprise's specific needs for the hybrid setup, including which workloads to migrate and which to retain on-premises.

Activities:

Conduct meetings with simulated enterprise stakeholders to gather requirements. Document the network architecture, including on-premises resources and AWS resources to be connected. Choose between AWS Direct Connect, VPN, or a combination of both based on latency, throughput needs, and cost considerations.

Task 2: Set Up AWS Direct Connect

Objective: Establish a dedicated network connection between the on-premises data center and AWS.

Activities:

Work with AWS and a Direct Connect partner to provision a Direct Connect connection. Configure the Direct Connect link at the AWS Management Console, setting up a Virtual Interface (VIF) for private connectivity to AWS services. Ensure proper routing and BGP configurations are in place for reliable, secure communication.

Task 3: Configure a Site-to-Site VPN as a Failover

Objective: Implement a Site-to-Site VPN to serve as a redundant, secure connection if the Direct Connect link fails.

Activities:

Utilize the AWS Site-to-Site VPN service to establish VPN tunnels between the on-premises data center and the VPC. Configure VPN settings, including the customer gateway on the on-premises side and the virtual private gateway on the AWS side. Test VPN failover to ensure automatic switching in case of Direct Connect downtime.

Task 4: Implement Hybrid Cloud Networking

Objective: Configure networking to integrate on-premises and AWS resources seamlessly.

Activities:

Set up a Virtual Private Cloud (VPC) for AWS resources that need to communicate with on-premises systems. Design and implement VPC subnets, route tables, and NACLs to align with the hybrid architecture requirements. Establish private DNS to facilitate name resolution between on-premises and cloud environments.

Task 5: Security and Compliance

Objective: Ensure the hybrid cloud solution adheres to security best practices and compliance standards.

Activities:

Implement security groups and IAM policies to control access to AWS resources.Use AWS Shield and AWS WAF to protect cloud resources from DDoS attacks and web exploits. Conduct a security assessment to identify potential vulnerabilities and apply necessary patches or configuration changes.

Task 6: Monitoring and Optimization

Objective: Set up monitoring for the hybrid cloud environment and optimize for performance and cost.

Activities:

Utilize Amazon CloudWatch to monitor network and application performance across on-premises and AWS. Analyze traffic patterns to optimize bandwidth usage and reduce costs, considering options like AWS Direct Connect Gateway for multiple VPC connectivity. Prepare a report on the hybrid cloud setup's performance, offering recommendations for future scalability.

Deliverables:

A comprehensive design document outlining the hybrid cloud architecture. Configuration files and scripts used in setting up Direct Connect, VPN, and VPC components. A security assessment report with findings and remediation steps. A performance and cost optimization analysis with actionable insights.

==========

Storage Project: Serverless Migration and Data Lake Setup on Amazon S3

Project Overview :

A company is looking to modernize its on-premises application by migrating it to a serverless architecture on AWS, aiming to enhance scalability, reduce costs, and improve data analytics capabilities. As part of this transition, the company also plans to establish a data lake on Amazon S3 to consolidate various data sources for advanced analytics.

Objective:

Migrate the existing on-premises application to a serverless framework using AWS services, with a focus on utilizing Amazon S3 for storage and analytics. Set up a comprehensive data lake strategy that includes data ingestion, storage, and analysis.

Task 1: Assess and Plan the Serverless Migration

Objective: Evaluate the on-premises application components and data to plan their migration to AWS serverless architecture.

Activities:

Analyze the application's architecture, identifying components suitable for serverless deployment (e.g., web frontends, APIs). Design a migration strategy that outlines the transition to AWS Lambda, Amazon API Gateway, and other serverless services. Develop a data migration plan to move existing data to Amazon S3 securely.

Task 2: Implement Serverless Application Components

Objective: Migrate application components to AWS serverless services, ensuring scalability and performance.

Activities:

Configure Amazon S3 buckets with proper organization, naming conventions, and security policies for the data lake. Implement data ingestion pipelines using AWS Glue or AWS Data Pipeline for batch and real-time data. Set up data cataloging with AWS Glue Catalog to manage metadata and facilitate search and discovery.

Task 3: Establish the Data Lake on Amazon S3

Objective: Create a data lake on Amazon S3 to aggregate, store, and analyze data from various sources.

Activities:

Configure Amazon S3 buckets with proper organization, naming conventions, and security policies for the data lake. Implement data ingestion pipelines using AWS Glue or AWS Data Pipeline for batch and real-time data. Set up data cataloging with AWS Glue Catalog to manage metadata and facilitate search and discovery.

Task 4: Data Processing and Analytics

Objective: Enable advanced analytics on the data lake using AWS analytics services.

Activities:

Use AWS Glue for ETL operations to prepare data for analytics. Implement analytics solutions using Amazon Athena for querying and Amazon Redshift for complex analysis. Explore integrating Amazon SageMaker for machine learning models to derive insights from the data lake.

Task 5: Security and Compliance

Objective: Ensure the serverless architecture and data lake comply with security standards and regulations.

Activities:

Implement encryption at rest using Amazon S3 server-side encryption and AWS KMS. Define IAM roles and policies for controlled access to the serverless components and data lake. Conduct regular security audits to ensure compliance with data protection regulations.

Task 6: Monitoring, Optimization, and Cost Management

Objective: Set up monitoring for the serverless components and data lake, optimize performance, and manage costs.

Activities:

Utilize Amazon CloudWatch for monitoring application performance and logging. Employ AWS Lambda cost optimization techniques, such as adjusting memory allocation and optimizing execution time. Analyze Amazon S3 usage and apply lifecycle policies to archive or delete old data to reduce costs.

Deliverables:

A detailed migration report outlining the transition to a serverless architecture, including architectural diagrams. A comprehensive data lake strategy document, covering data ingestion, storage, processing, and analytics. Security and compliance assessment report, including encryption and access control implementations. Performance and cost management analysis with recommendations for further optimizations.

High Availability Project: Multi-Region Web Application Deployment with Automated Failover

Project Overview :

A content delivery company wants to ensure their web application remains available and responsive to users worldwide, even in the event of a regional AWS service disruption. The project involves deploying the application across multiple AWS regions and implementing automated failover mechanisms.

Objective:

Create a resilient, multi-region deployment of a web application on AWS, incorporating Route 53 for DNS management and automated failover to enhance availability and disaster recovery capabilities.

Task 1: Design Multi-Region Architecture

Objective: Architect a deployment strategy that spans at least two AWS regions to support high availability.

Activities:

Identify suitable AWS regions based on the geographic distribution of the application's user base. Design an architecture that replicates the application's environment across these regions, including web servers, databases, and other critical components.

Task 2: Implement Application Deployment

Objective: Deploy the application in multiple regions, ensuring consistency and synchronization between regions.

Activities:

Use AWS CloudFormation or the AWS CDK to define and deploy infrastructure as code, ensuring environments are identical across regions. Set up Amazon RDS Multi-AZ deployments with read replicas in different regions for database high availability and cross-region read scalability. Configure Amazon S3 Cross-Region Replication for shared assets and user data.

Task 3: Configure Automated Failover with Route 53

Objective: Implement DNS-level failover using Amazon Route 53 to automatically reroute traffic in case of a regional outage.

Activities:

Set up health checks in Route 53 to monitor the health of the application endpoints in each region. Configure DNS failover policies in Route 53, ensuring automatic traffic rerouting to a healthy region if the primary region becomes unavailable. Test failover mechanisms to validate the setup.

Task 4: Monitoring and Alerts

Objective: Establish comprehensive monitoring across regions to detect and respond to issues quickly.

Activities:

Implement Amazon CloudWatch alarms and dashboards for real-time monitoring of application health and performance metrics across all regions. Set up Amazon SNS notifications to alert administrators of failover events and other critical metrics that indicate potential issues.

Deliverables:

A detailed design document outlining the multi-region architecture and deployment strategy. Infrastructure as code templates for deploying the multi-region environment. Configuration details and testing results for Route 53 automated failover. A monitoring and alerting strategy document, including CloudWatch dashboard configurations and SNS notification setups.

==========

Serverless Architecture Project: Serverless Backend for Web Application

Project Overview :

An online retail company seeks to modernize their e-commerce platform by moving to a serverless architecture for better scalability and cost efficiency. The focus is on developing a serverless backend for their web application using AWS services.

Objective:

Build a serverless backend for an ecommerce web application, leveraging AWS Lambda, API Gateway, and DynamoDB to create scalable, efficient, and cost-effective solutions.

Task 1: Design Serverless Backend Architecture

Objective: Create a scalable and maintainable serverless architecture suitable for an ecommerce platform.

Activities:

Define the application's core functionalities (e.g., user authentication, product catalog management, order processing) to be implemented as microservices. Design the serverless architecture, detailing the use of AWS Lambda functions, API Gateway for RESTful endpoints, and DynamoDB for data storage.

Task 2: Implement RESTful APIs with API Gateway and Lambda

Objective: Develop and deploy the serverless functions that power the application's backend APIs.

Activities:

Implement Lambda functions for handling API requests such as user registration, product browsing, and order placement. Configure API Gateway to expose and manage these Lambda functions as RESTful endpoints. Secure API endpoints using API Gateway authorizers and AWS IAM roles.

Task 3: Set Up Data Storage with DynamoDB

Objective: Design and configure DynamoDB tables to store application data, ensuring scalability and performance.

Activities:

Define DynamoDB tables and indexes for efficient data access patterns, considering the application's requirements for user data, product information, and orders. Implement data access layer within Lambda functions to interact with DynamoDB. Use DynamoDB Streams to trigger other Lambda functions for asynchronous processing (e.g., inventory updates, recommendations).

Task 4: Integration and Testing

Objective: Integrate the serverless backend with the frontend application, ensuring seamless communication and functionality.

Activities:

Develop integration tests to validate the interaction between the frontend, the serverless backend, and the database. Perform end-to-end testing of the ecommerce platform to ensure that all components work together as expected. Optimize performance based on testing feedback, adjusting Lambda memory sizes and DynamoDB throughput settings as necessary.

Deliverables:

An architecture diagram and documentation detailing the serverless backend design. Source code for all AWS Lambda functions and the API Gateway configuration. DynamoDB table design and implementation details. A comprehensive testing report, including integration and performance testing results.

==========

Database Project: Amazon RDS Performance Optimization and Migration to Aurora

Project Overview :

A financial services company experiences performance bottlenecks with their existing Amazon RDS instance under high transaction loads. They aim to optimize the RDS instance for enhanced performance and migrate it to Amazon Aurora to leverage Aurora's high performance and scalability, all while ensuring minimal downtime during the migration process.

Objective:

Enhance the performance of an existing Amazon RDS database and seamlessly migrate it to Amazon Aurora, taking advantage of Aurora's advanced features for scalability and reliability without significant downtime.

Task 1: Assess and Optimize the RDS Instance

Objective: Identify performance issues and optimize the RDS instance settings for improved efficiency.

Activities:

Perform a comprehensive analysis of the current RDS instance to identify performance bottlenecks, using Amazon CloudWatch and RDS Performance Insights. Optimize parameters in the RDS instance parameter group for better performance, including memory allocation, query execution plans, and connection settings. Implement indexing strategies and query optimizations to reduce latency and improve throughput.

Task 2: Prepare for Migration to Amazon Aurora

Objective: Prepare the environment and the data for a smooth migration from Amazon RDS to Amazon Aurora.

Activities:

Evaluate the compatibility of the existing database schema and data with Amazon Aurora, making necessary adjustments. Create an Amazon Aurora cluster with an instance size and configuration that meets or exceeds the current RDS setup. Enable binary logging on the RDS instance to capture transaction changes during the migration process.

Task 3: Migrate to Amazon Aurora with Minimal Downtime

Objective: Seamlessly migrate the optimized RDS database to Amazon Aurora with minimal impact on application availability.

Activities:

Use the AWS Database Migration Service (DMS) to replicate the data from the RDS instance to the Aurora cluster, ensuring ongoing changes are captured. Monitor the replication process, addressing any errors or lags to keep the Aurora cluster in sync with the RDS instance. Perform a cut-over to the Aurora cluster during a scheduled maintenance window, updating application connection strings to point to the new Aurora endpoint.

Task 4: Validate and Optimize the Aurora Deployment

Objective: Ensure the Aurora database functions correctly post-migration and is fully optimized for the company's workloads.

Activities:

Conduct thorough testing to validate data integrity, application functionality, and performance on the Aurora cluster. Leverage Aurora-specific features such as Aurora Global Databases for cross-region disaster recovery, if applicable. Fine-tune Aurora performance settings based on observed workload patterns, utilizing Aurora's Performance Insights for guidance.

Task 5: Implement Monitoring and Failover Strategies

Objective: Set up comprehensive monitoring for the Aurora cluster and plan for automated failover to enhance availability.

Activities:

Configure Amazon CloudWatch alarms for critical metrics related to Aurora performance and availability. Test Aurora's automatic failover feature by simulating failure scenarios, ensuring the application seamlessly connects to a healthy Aurora replica without manual intervention. Document the failover process and any manual steps required for recovery.

Deliverables:

A report detailing the performance optimization steps taken for the original RDS instance and the rationale behind each change. Migration plan documentation, including schema modifications, compatibility checks, and the migration strategy used. Testing and validation reports for the Aurora deployment, highlighting performance improvements and any adjustments made post-migration. A failover and recovery strategy guide for the Aurora cluster, including monitoring setup and alerting policies.

DevOps Project: Microservices Deployment on Amazon EKS with CI/CD Pipeline

Project Overview :

A software development company is transitioning to a microservices architecture for its flagship application to improve scalability and accelerate feature deployment. The project involves deploying the microservices application on Amazon Elastic Kubernetes Service (EKS) and automating the deployment process with a Continuous Integration/Continuous Deployment (CI/CD) pipeline using AWS CodePipeline and CodeBuild.

Objective:

Implement a robust DevOps workflow that automates the build, test, and deployment of a microservices-based application on Amazon EKS, leveraging AWS CodePipeline and CodeBuild for CI/CD.

Task 1: Set Up Amazon EKS for Microservices Deployment

Objective: Configure Amazon EKS to host the microservices application, ensuring it is secure, scalable, and highly available.

Activities:

Create an EKS cluster with worker nodes spread across multiple availability zones to ensure high availability. Define Kubernetes namespaces for organizing microservices components based on their operational requirements and environments (e.g., development, staging, production). Implement network policies and IAM roles for Kubernetes service accounts to enforce security at the microservices level.

Task 2: Containerize Microservices and Manage Repositories

Objective: Containerize each microservice and push the Docker images to Amazon Elastic Container Registry (ECR).

Activities:

Write Dockerfiles for each microservice, ensuring they are optimized for size, security, and build speed. Build Docker images locally and push them to ECR repositories created for each microservice. Set up ECR image scanning to automatically scan images for vulnerabilities on push.

Task 3: Create a CI/CD Pipeline with AWS CodePipeline and CodeBuild

Objective: Automate the build, test, and deployment process for the microservices application using AWS CodePipeline and CodeBuild.

Activities: Define a source stage in CodePipeline to trigger the pipeline on code changes in a Git repository (e.g., AWS CodeCommit, GitHub). Set up CodeBuild projects to build Docker images from the source code, run unit tests, and push successful builds to ECR. Configure a deployment stage in CodePipeline to automatically deploy the Docker images to EKS using Kubernetes manifests or Helm charts.

Task 4: Implement Automated Testing and Rollback Mechanisms

Objective: Ensure code quality and reliability by integrating automated testing in the CI/CD pipeline and implementing rollback mechanisms for deployment failures.

Activities:

Integrate automated testing frameworks (e.g., Selenium for UI testing, JUnit for unit testing) in the build stage of the pipeline. Use CodePipeline's approval actions and manual gates to review and approve deployments to production environments. Implement Kubernetes deployment strategies (e.g., blue-green deployments, canary releases) to minimize downtime and facilitate rollbacks in case of errors.

Task 5: Monitoring, Logging, and Performance Tuning

Objective: Set up monitoring, logging, and alerting for the microservices application and the CI/CD pipeline to maintain operational excellence.

Activities:

Configure Amazon CloudWatch for monitoring the EKS cluster, microservices performance, and logging container logs. Set up CloudWatch alarms and SNS notifications to alert the DevOps team about critical issues, such as deployment failures, resource utilization spikes, or security vulnerabilities. Analyze performance metrics to identify bottlenecks and optimize the microservices configuration and resource allocation for improved efficiency.

Deliverables:

An Amazon EKS cluster configured for microservices deployment, complete with security and network configurations. Dockerfiles and ECR repositories for each microservice, along with a process for automated image scanning.

A fully automated CI/CD pipeline in AWS CodePipeline and CodeBuild, documented with stages, actions, and integration points. Automated testing and rollback strategy documentation, including test frameworks used and deployment strategies implemented.

Monitoring and alerting setup documentation, including CloudWatch dashboards, alarms, and notification workflows. This project challenges students to apply DevOps principles to deploy a microservices architecture on AWS, automating the software development lifecycle with CI/CD best practices, and ensuring operational monitoring and security.

Monitoring and Logging Project: Centralized Logging with Amazon Elasticsearch Service and Kibana

Project Overview :

An online retail company is looking to improve its operational efficiency and troubleshooting capabilities by implementing a centralized logging solution. They aim to aggregate logs from various applications and AWS services into a single repository for analysis and visualization. The project involves using Amazon Elasticsearch Service (Amazon ES) for log storage and analysis, and Kibana for data visualization.

Objective:

Establish a scalable and centralized logging solution using Amazon Elasticsearch Service, facilitating log aggregation, search, and analysis. Utilize Kibana for creating insightful visualizations and dashboards for real-time monitoring.

Task 1: Design the Centralized Logging Architecture

Objective: Architect a logging solution that collects, stores, and analyzes logs from different sources within the AWS environment.

Activities:

Identify the sources of logs, including application logs, AWS CloudTrail logs, Amazon VPC Flow Logs, and logs from various AWS services. Plan the log ingestion pipeline using AWS services such as Amazon CloudWatch Logs, AWS Lambda, and Logstash (for non-AWS sources) to route logs to Amazon ES.

Task 2: Set Up Amazon Elasticsearch Service

Objective: Deploy and configure an Amazon Elasticsearch Service cluster to serve as the centralized logging platform.

Activities:

Create an Amazon ES domain, configuring the cluster size, instance types, and storage options based on anticipated log volume and retention requirements. Implement access policies to secure the Amazon ES domain, allowing authorized access from log sources and Kibana. Configure Amazon ES indices and index templates to efficiently organize logs by source and type.

Task 3: Log Ingestion and Aggregation

Objective: Implement log ingestion mechanisms to collect logs from various sources and aggregate them into the Amazon ES cluster.

Activities: Use AWS Lambda functions to process and transform logs from CloudWatch Logs and other AWS services, sending them to Amazon ES. Configure Logstash or AWS Data Firehose (for supported log sources) to ingest logs into Amazon ES, applying necessary transformations and enrichments. Ensure reliable and secure log transmission, implementing retry mechanisms and encryption in transit.

Task 4: Visualization with Kibana

Objective: Utilize Kibana to create visualizations and dashboards that provide insights into application and infrastructure performance and issues.

Activities:

Set up Kibana with the Amazon ES domain, configuring access controls to ensure secure dashboard access. Create Kibana visualizations for key metrics such as error rates, response times, and resource utilization, using logs aggregated in Amazon ES. Build comprehensive dashboards that aggregate multiple visualizations, providing a holistic view of system health and performance trends.

Task 5: Monitoring and Alerts

Objective: Leverage the centralized logging solution for operational monitoring and setting up alerts for anomaly detection.

Activities:

Use Amazon ES's built-in alerting features or integrate with Amazon CloudWatch for monitoring log data for specific patterns or thresholds. Configure alerts to notify the operations team via Amazon SNS or email for critical issues identified through log analysis. Document the process for responding to alerts, including initial diagnosis, escalation procedures, and resolution steps.

Deliverables:

A documented architecture design for the centralized logging solution, including data flow diagrams and component configurations. Amazon Elasticsearch Service domain setup and configuration files, along with access policies and index management strategies. A collection of Kibana dashboards and visualizations tailored to the company's operational monitoring needs.

An alerting and notification setup guide, detailing the configuration of alerts based on log data and integration with notification services. A best practices guide for log management, including retention policies, data privacy considerations, and optimization tips for cost management.

This project equips students with the skills to implement an advanced centralized logging system using AWS technologies, enhancing operational visibility and proactive issue resolution capabilities in a cloud environment.

Cost Management Project: AWS Cost Optimization with Cost Explorer and Trusted Advisor

Project Overview :

A growing tech startup, utilizing a wide range of AWS services, has observed a significant increase in their AWS bills over the past few months. The company aims to optimize its cloud spending without compromising on performance or scalability. The project involves analyzing the AWS environment's current cost structure using AWS Cost Explorer and identifying optimization opportunities with the help of AWS Trusted Advisor.

Objective:

Conduct a comprehensive cost analysis of the startup's AWS usage, leveraging AWS Cost Explorer and Trusted Advisor to uncover cost-saving opportunities and implement best practices for cloud cost management.

Task 1: AWS Cost Analysis with Cost Explorer

Objective: Utilize AWS Cost Explorer to analyze the company's AWS spending patterns and identify high-cost services and resources.

Activities:

Enable and configure AWS Cost Explorer to access detailed reports on the company's AWS spending and usage. Analyze spending trends over time, identifying services with the highest costs and any unusual spikes in spending. Break down costs by service, region, and resource tags to pinpoint areas where cost optimizations can be made.

Task 2: Recommendations with AWS Trusted Advisor

Objective: Use AWS Trusted Advisor to identify specific cost optimization opportunities and best practices not currently being followed.

Activities:

Enable Trusted Advisor and review the Cost Optimization checks, focusing on underutilized resources such as EC2 instances, EBS volumes, and RDS instances. Analyze Trusted Advisor recommendations for Reserved Instances and Savings Plans based on the company's usage patterns. Summarize findings in key areas where the company can reduce costs, such as deleting unused resources, resizing underutilized resources, and purchasing Reserved Instances or Savings Plans.

Task 3: Implement Cost Optimization Strategies

Objective: Execute a series of cost optimization measures based on the analysis and recommendations from Cost Explorer and Trusted Advisor.

Activities: Implement tagging strategies to improve visibility into costs and usage across different projects, departments, and environments. Resize, consolidate, or shut down underutilized resources as identified in the analysis phase. Purchase Reserved Instances or commit to Savings Plans for consistently used services to take advantage of discounted rates.

Task 4: Monitor and Adjust Cost-Saving Measures

Objective: Establish ongoing monitoring to track the effectiveness of implemented cost-saving measures and adjust strategies as needed.

Activities:

Set up budget alerts in AWS Budgets to monitor monthly spending against set thresholds, ensuring early detection of overruns. Regularly review AWS Cost Explorer and Trusted Advisor reports to monitor the impact of implemented cost-saving measures and identify new optimization opportunities. Conduct quarterly reviews to adjust the cost optimization strategy based on changing usage patterns and new AWS pricing models or services.

Deliverables:

A comprehensive cost analysis report detailing the findings from AWS Cost Explorer, including spending trends, high-cost services, and anomalies.

A list of cost optimization recommendations derived from AWS Trusted Advisor, categorized by potential impact and ease of implementation. Documentation of implemented cost optimization strategies, including details of resized or terminated resources, Reserved Instances, and Savings Plans purchases. A cost management best practices guide, including instructions for setting up budget alerts, resource tagging, and conducting regular cost reviews.

This project provides students with practical experience in cloud cost management, teaching them how to analyze AWS spending, identify cost-saving opportunities, and implement effective cost optimization strategies.