PRACTICAL: -6

Aim:-Optimize and evaluate the application for better performance among aws,gcp,azure etc.

Chatting Application:-

1. Why Choose AWS?

- Global Infrastructure: Low latency, globally distributed regions.
- Native WebSocket Support: Application Load Balancer (ALB) supports WebSocket and session stickiness.
- Scalability: Supports container orchestration via ECS, EKS, and AWS Fargate.
- High Availability: Services like Elasticache, RDS, and S3 provide managed, reliable infrastructure.
- Monitoring Tools: CloudWatch provides detailed logs, metrics, and alerting.

2. Recommended Architecture:-

Components:

- Application Layer: Flask + Socket.IO (using Gunicorn with eventlet)
- Containerization: Docker
- Container Hosting: ECS with Fargate or EKS
- Message Broker: Elasticache Redis for pub/sub message distribution across instances
- Database: Amazon RDS (MySQL)
- Load Balancer: Application Load Balancer (ALB) for WebSocket support
- Monitoring & Logging: CloudWatch
- File Storage : Amazon S3

Benefits:

- Horizontal scaling support
- Secure, managed services
- Production-ready deployment capabilities

3. Optimization Techniques:-

A. Web Server Optimization:

- Use Gunicorn with eventlet for asynchronous support:

B. Redis for Socket.IO:

- Redis is used as a message queue to enable real-time broadcast across multiple instances:

C. WebSocket Load Balancing:

- Use ALB to distribute WebSocket traffic.
- Enable sticky sessions to maintain Socket.IO connections.
- Use HTTP/1.1 for compatibility.

D. Database Optimization:

- Optimize queries, indexes on sender_id, receiver_id, and timestamp.
- Archive old chat messages periodically.
- Use Amazon RDS with performance insights enabled.

E. Container & Scaling:

- Use ECS Fargate for serverless container hosting.
- Enable auto-scaling based on CPU and memory metrics.
- Use Elastic Load Balancing to distribute requests.

4. AWS Services Used:-

Functionality	AWS Service
Container Hosting	ECS Fargate / EKS
Load Balancing & WebSockets	Application Load Balancer
Message Broadcasting	Elasticache Redis
Database	Amazon RDS (MySQL)
File Storage	Amazon S3
Monitoring & Logging	Amazon CloudWatch
Secrets Management	AWS Secrets Manager

6. Cost & Performance Evaluation:-

Service	**Approximate Monthly Cost**	**Notes**
ECS Fargate	$10 – $30	Depends on app size and usage
Elasticache Redis	$15 – $40	Needed for Socket.IO message distribution
ALB	$15 – $25	Based on traffic and rules
RDS MySQL	$15 – $50	Scales with DB size and connections
S3 (optional)	Pay-as-you-go	For static file storage

PRACTICAL: -7

AIM: -Use & analyze tools like aws glue to extract, transform and load data & also mention steps to perform following modules.

What is AWS Glue?

AWS Glue is a fully managed **ETL service** that enables you to prepare and transform data for analytics, machine learning, and application development. It supports serverless data integration and can automatically discover, catalog, and clean data from multiple sources.

Why use AWS Glue?

AWS Glue is a serverless ETL service that helps you easily prepare and move data between sources for analytics. It automates data discovery, transformation, and job scheduling, making data integration fast, scalable, and cost-efficient.

Tools in AWS Glue for ETL:-

Tool / Module	Description
Glue Data Catalog	Central metadata repository to store table definitions and schema versions
Glue Crawlers	Automatically detect schema and create table metadata in the catalog
Glue Jobs	Execute ETL scripts (in Python or Scala)
Glue Studio	Visual interface to create, run, and monitor ETL jobs
Glue Triggers	Automate job runs on schedule or event
Glue Workflows	Manage complex ETL pipelines
Glue DynamicFrame	Enhanced version of DataFrames optimized for semi-structured data

ETL Phases Using AWS Glue:-

1. Extract:-

Objective: Connect to data sources and extract raw data.

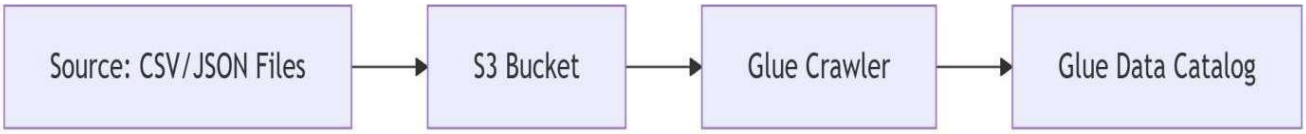
Steps:

1. Create a Crawler:

- Go to AWS Glue Console → Crawlers → Add Crawler
- Define the data source (e.g., S3, RDS, Redshift)
- Configure IAM roles and schedule
- Run the crawler to scan data and generate schema

2. Use Data Catalog:

- The crawler stores metadata (like table name, column types) in the Glue Data Catalog.
- You can query this using Amazon Athena.



2. Transform:-

Objective: Clean, format, join, filter, or enrich the extracted data.
Steps:
1. Create a Glue Job:

- Go to Glue → Jobs → Add Job
- Choose "Visual with Glue Studio" or "Script Editor" mode
- Set source from Data Catalog table
- Apply transformations using:
- Built-in transforms (map, join, filter, drop nulls, etc.)
- Custom logic via PySpark or Scala
- DynamicFrames for schema flexibility

2. Schema Mapping:

- Use the “Transform” node to map source schema to target schema.
- Optionally handle nested data (like JSON structures).

3. Load:-

Objective: Store the transformed data into target systems for further use.
Steps:
1. Select Target:

- Common targets: Amazon S3, Redshift, RDS, or other JDBC-compliant databases.

2. Write Data:

- Define the output format (Parquet, CSV, JSON, etc.)
- Choose write mode (overwrite or append)
- Define S3 path or connection for databases

3. Run the Job:

- Optionally set job triggers or schedule using **AWS Glue Triggers**
- Monitor execution using **Glue Console** or **CloudWatch Logs**

PRACTICAL: -8

AIM: -Describe security controls provided by AWS along with its specifications.

1. Identity and Access Management:-

AWS Identity and Access Management (IAM)

- **Function**: Controls user and application access to AWS resources.
- **Specifications**:
  - Fine-grained permissions using policies (JSON).
  - Supports multi-factor authentication (MFA).
  - Role-based access control and temporary credentials via IAM Roles.
  - Integration with AWS Organizations for centralized account control.

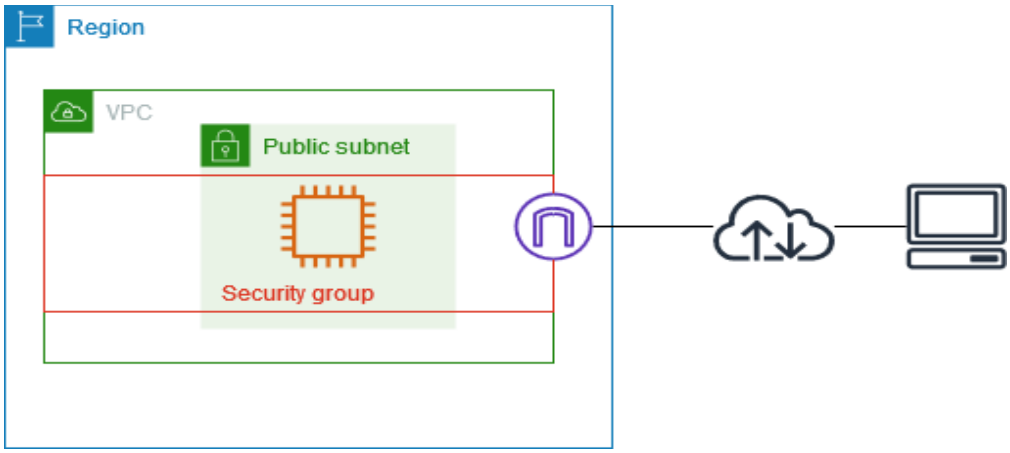
AWS Single Sign-On (SSO)

- **Function**: Enables centralized access management across AWS accounts and third-party apps.
- **Specifications**:
  - Supports SAML 2.0.
  - Integrates with Active Directory.
  - Provides user dashboards for access to assigned accounts and apps.

2. Network Security:-

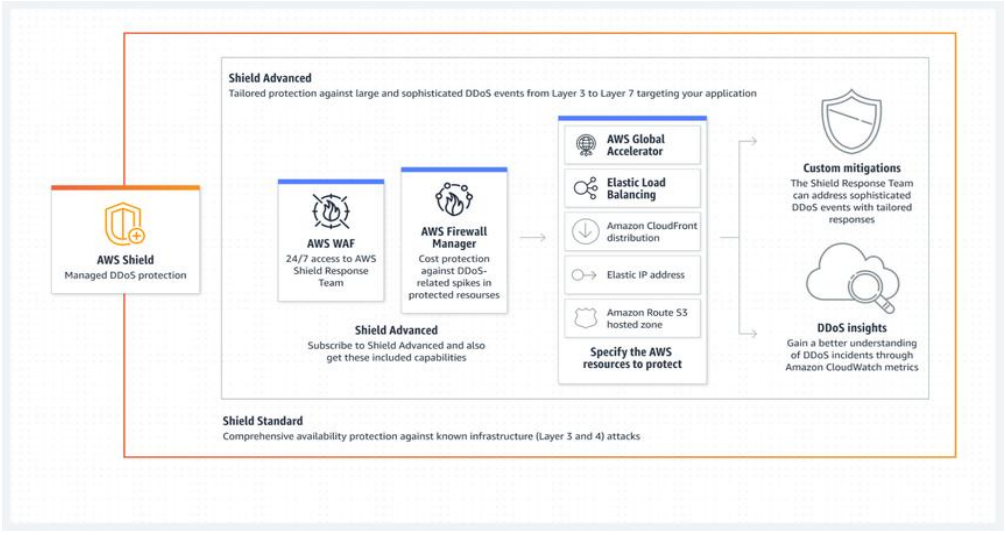
Amazon Virtual Private Cloud (VPC)

- **Function**: Provides logically isolated cloud networks.
- **Specifications**:
  - Customizable IP address ranges, subnets, route tables.
  - Security groups (stateful firewalls) and Network ACLs (stateless).
  - VPC Flow Logs for traffic monitoring.



AWS Shield

- **Function**: Protects against Distributed Denial of Service (DDoS) attacks.
- **Specifications**:
  - **Shield Standard**: Free, automatic protection against common attacks.
  - **Shield Advanced**: Paid service with advanced threat mitigation, 24/7 DDoS response team, cost protection.



AWS WAF (Web Application Firewall)

- **Function**: Protects web applications from exploits like SQL injection, XSS, etc.
- **Specifications**:
  - Custom rules or pre-configured managed rule sets.
  - Real-time metrics and logging via Amazon CloudWatch.

3. Data Protection:-

AWS Key Management Service (KMS)

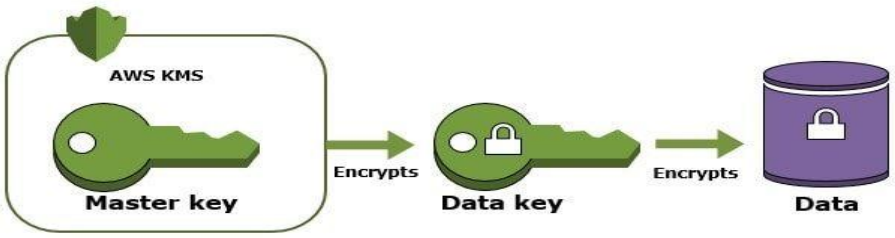
- **Function**: Manages cryptographic keys for data encryption.
- **Specifications**:
  - FIPS 140-2 validated hardware security modules (HSMs).
  - Integrated with many AWS services (S3, EBS, RDS, etc.).
  - Automatic key rotation and policy-based access control.

AWS Secrets Manager

- **Function**: Manages and rotates credentials, API keys, and other secrets.
- **Specifications**:
  - Automatic secret rotation using Lambda functions.
  - Fine-grained access control using IAM.
  - Secure and auditable storage of secrets.

Encryption (At-Rest & In-Transit)

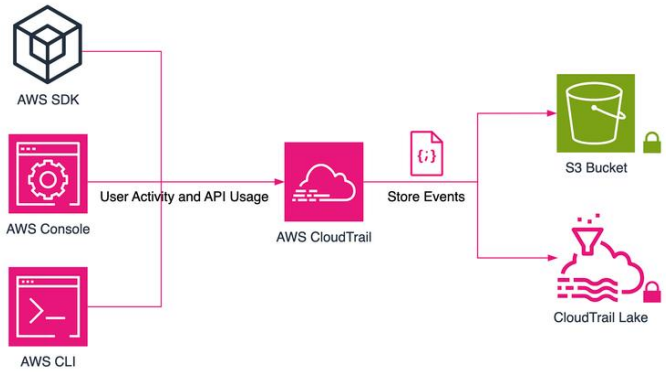
- **Specifications**:
  - Data at rest: Encrypted using AWS KMS or customer-managed keys.
  - Data in transit: SSL/TLS for secure communications between services.



4. Monitoring and Logging:-

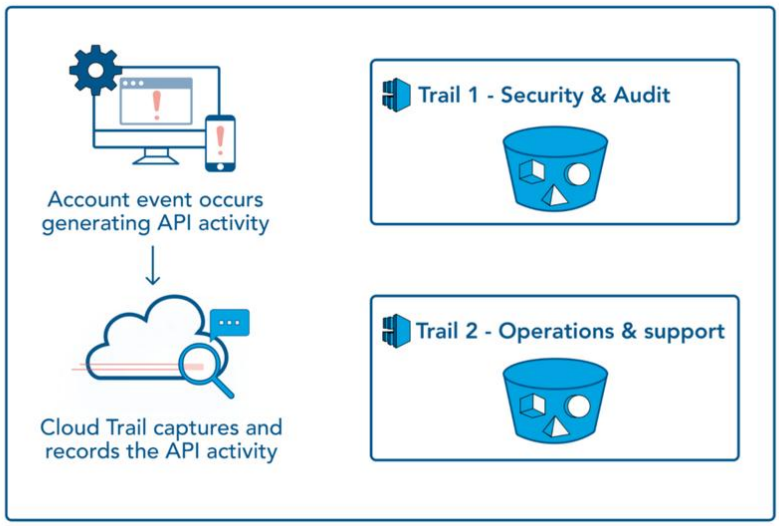
AWS CloudTrail

- **Function**: Records all API calls across AWS services.
- **Specifications**:
  - Event history for auditing and governance.
  - Real-time delivery of logs to S3 or CloudWatch.
  - Supports multi-account logging via AWS Organizations.



Amazon CloudWatch

- **Function**: Monitoring and observability of AWS resources and applications.
- **Specifications**:
  - Metrics collection, custom dashboards, log aggregation.
  - Alarms and automatic remediation using Lambda.



AWS Config

- **Function**: Tracks configuration changes and compliance.
- **Specifications**:
  - Maintains a history of AWS resource configurations.
  - Can trigger rules and remediation when non-compliant.

5. Compliance and Governance:-

AWS Artifact

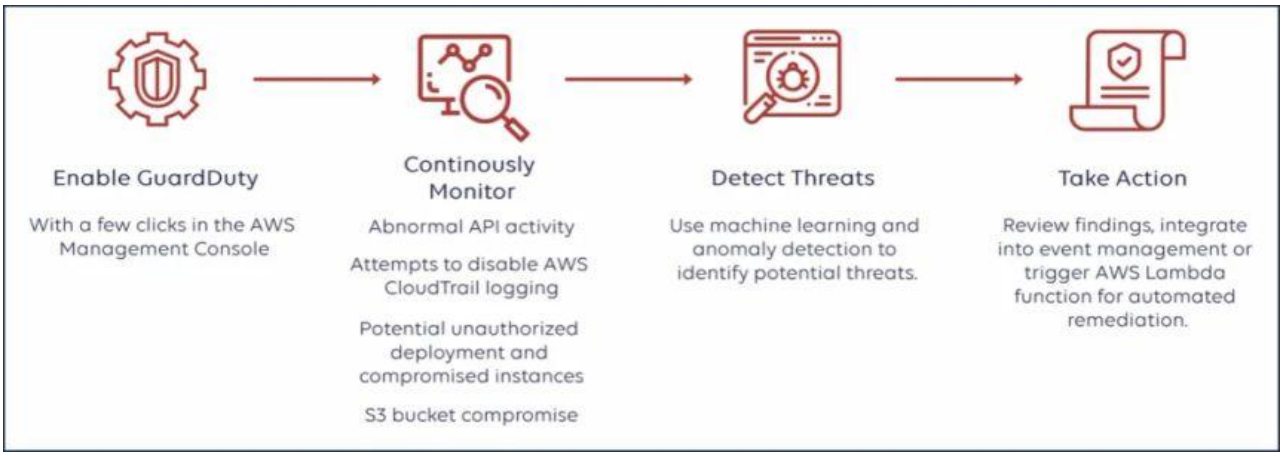
- **Function**: Provides on-demand access to AWS compliance reports and agreements.
- **Specifications**:
  - Reports include SOC 1/2/3, ISO 27001, GDPR, HIPAA, etc.
  - Helps meet audit requirements by supplying official documentation.

AWS Organizations

- **Function**: Enables centralized management of multiple AWS accounts.
- **Specifications**:
  - Service control policies (SCPs) for restricting actions.
  - Consolidated billing and account structure.

AWS GuardDuty

- **Function**: Threat detection service that monitors for malicious activity.
- **Specifications**:
  - Uses machine learning and threat intelligence feeds.
  - Detects anomalies, credential compromise, and malicious IPs.



PRACTICAL: -9

Aim: - Use different cloud management tools provided by AWS

1. AWS CloudFormation – *Infrastructure Automation Tool:-*

- **Function**: Defines infrastructure as code using YAML/JSON templates.
- **Use Case**: Automatically deploy a full web application stack (EC2, RDS, S3, IAM roles).
- **Key Features:**
  - Reproducible deployments across environments.
  - Dependency handling between resources.
  - Change sets and rollback support.

2. AWS CloudTrail – *Governance & Auditing Tool:-*

- **Function**: Logs all API calls made in your AWS environment.
- **Use Case:** Investigating unauthorized access attempts or changes.
- **Key Features:**
  - Full visibility of user/API activities.
  - Integration with S3 and CloudWatch.
  - Supports multi-region logging.

3. AWS Systems Manager – *Unified Operations Center:-*

- **Function**: Centralized management of EC2 and other AWS resources.
- **Use Case:** Automate patching and configuration of EC2 instances.
- **Key Features:**
  - Run Command, Session Manager, Patch Manager.
  - Works across hybrid environments (on-premise + AWS).
  - Parameter Store for secret/configuration management.

4. AWS OpsWorks – *Configuration Management (CM) Tool:-*

- **Function**: Leverages Chef/Puppet to manage server configurations.
- **Use Case**: Automating software deployments across fleets.
- **Key Features**:
  - State-driven configuration enforcement.
  - Chef cookbooks or Puppet modules integration.
  - Suitable for complex, legacy deployments.

5. AWS CloudWatch – *Monitoring & Observability Platform:-*

- **Function**: Monitors AWS services and applications in real time.
- **Use Case:** Auto-restart an EC2 instance if CPU spikes > 80%.
- **Key Features:**
  - Metrics, dashboards, alarms, and logs.
  - Anomaly detection and custom events.
  - Integrated with Lambda and Auto Scaling.

6. AWS Trusted Advisor – *Resource Optimization Tool:-*

- **Function**: Provides best-practice recommendations across five categories.
- **Use Case:** Identify idle resources or security misconfigurations.
- **Key Features**:
  - Cost optimization insights.
  - Fault tolerance checks.
  - Performance and security recommendations.

7. AWS Service Catalog – *Controlled Provisioning for IT Services:-*

- **Function**: Allows organizations to manage approved resource templates.
- **Use Case**: Enforce organizational standards in large teams.
- **Key Features:**
  - Restricts users to pre-approved configurations.
  - Tagging and access control policies.
  - Supports version control of services.

8. AWS Organizations – *Multi-Account Management Tool:-*

- **Function:** Manage multiple AWS accounts centrally.
- **Use Case:** Apply centralized policies and consolidated billing for teams.
- **Key Features:**
  - Apply Service Control Policies (SCPs).
  - Organize accounts into Organizational Units (OUs).
  - Linked billing and security enforcement.

9. AWS Config – *Resource Compliance Tool:-*

- **Function**: Tracks and evaluates configurations of AWS resources over time.
- **Use Case**: Ensure S3 buckets are not public.
- **Key Features:**
  - Detects drift from desired configurations.
  - Automatically triggers remediation actions.
  - Useful for compliance audits (HIPAA, PCI-DSS).

10. AWS Cost Explorer & Budgets – *Billing & Cost Management Tools:-*

- **Function**: Monitor usage patterns and control cloud expenses.
- **Use Case**: Set a monthly budget limit with email alerts.
- **Key Features:**
  - Visual cost and usage reports.
  - Forecasting and anomaly detection.
  - API access for external integrations.

PRACTICAL: -10

Aim:- Choose a business process and identify its dependency and requirements. Use the 7-step model of migration into cloud to plan and execute a migration to the cloud. Evaluate risk and assess impact of migration.

Business Process Selection: Student Notes Uploading & Access System:-

Purpose:

The core business process of the application is the uploading, storing, managing, and accessing of educational notes by students and admins.

Objectives:

- Allow students to upload notes by branch/subject.
- Enable admins to approve, reject, or manage those uploads.
- Provide users with search, preview, and download access.
- Ensure secure access based on user roles (user/admin).
- Allow scalability as user base and uploaded content grows.

Functional Dependencies:

Feature	Dependency
User authentication	Login module, MySQL users table
File uploads	Web server, file system, or S3
Notes categorization	Branch/subject metadata in DB
Admin approval	Notes status tracking in MySQL
Search functionality	Full-text search across notes table
Chat support	Real-time messaging with Socket.IO & Redis

Infrastructure Requirements:-

Component	AWS Service
Web app hosting	Amazon ECS / EC2 / Elastic Beanstalk
File storage	Amazon S3
Database	Amazon RDS (MySQL)
Realtime messaging	Amazon Elasticache (Redis)
Load balancing	Application Load Balancer (ALB)
Logging & Monitoring	Amazon CloudWatch

Cloud Migration Strategy Using AWS 7-Step Model:-

Step 1: Assess (Discovery & Planning):-

Goal: Understand current architecture, workloads, and readiness for cloud.

Actions:

- Inventory tech stack: Flask, MySQL, Redis, HTML/CSS, file system.
- Identify bottlenecks: file upload latency, manual DB scaling, etc.
- Define key goals: scalability, high availability, cost control.

Output:

- Migration Readiness Assessment (MRA)
- Technical requirements document

Step 2: Mobilize (Build Foundation):-

Goal: Prepare AWS foundation and security environment.

Actions:

- Set up IAM roles, VPC, and subnets.
- Create S3 buckets for notes storage.
- Provision test RDS instance for MySQL migration.
- Prepare CloudWatch for monitoring.

Output:

- Configured cloud environment with initial service provisioning

Step 3: Plan (Migration Strategy & Architecture Design):-

Goal: Design the target cloud architecture.

Actions:

- Choose ECS with Fargate for container-based deployment.
- Define RDS instance configuration (Multi-AZ, backups).
- Configure S3 bucket structure with folder hierarchy (e.g., /notes/branch/subject/).
- Set up ALB for load balancing traffic.
- Plan migration of users, notes, and metadata from local MySQL to RDS.

Output:

- Architecture diagram and finalized migration blueprint

Step 4: Pilot (Proof of Concept):-

Goal: Validate migration with non-production workload.

Actions:

- Deploy dev version of Flask app to ECS.
- Connect to test RDS DB and S3 bucket.
- Perform test uploads, downloads, and approval workflows.
- Simulate 10–20 concurrent users to validate performance.

Output:

- Confidence in AWS setup and app functionality in cloud

Step 5: Migrate (Production Deployment):-

Goal: Perform actual production migration.

Actions:

- Backup local MySQL and import into RDS using mysqldump or AWS DMS.
- Upload existing files to S3.
- Deploy production containers on ECS.
- Switch DNS routing via Route 53 to AWS-hosted app.
- Notify users of new deployment.

Output:

- Live application fully running on AWS cloud infrastructure

Step 6: Modernize (Optimization & Refactoring):-

Goal: Optimize the app using cloud-native capabilities.

Actions:

- Add autoscaling for ECS tasks based on traffic.
- Use CloudFront CDN to serve S3-hosted files quickly.
- Enable S3 lifecycle policies to move old files to Glacier.
- Add Elasticache Redis for faster chat messaging.

Output:

- Enhanced, scalable, cloud-optimized app

Step 7: Operate (Monitoring, Support & Optimization):-

Goal: Manage and monitor app for performance, cost, and security.

Actions:

- Monitor app with CloudWatch Logs and Metrics.
- Set up alarms for DB CPU, memory, and latency thresholds.
- Enable RDS auto backups and cross-region replication (optional).
- Review AWS billing with Cost Explorer and use Savings Plans.

Output:

- Continuously optimized application with full visibility

Risk Evaluation:-

Risk	Description	Mitigation Strategy
Data loss during migration	Risk of losing notes or user data	Backup all data before migration, validate RDS dump
Downtime during switchover	Application unavailability during DNS update	Perform migration during off-peak hours
Misconfigured permissions	Improper IAM roles leading to data leaks	Implement least-privilege IAM policies
Performance issues	Latency in S3, RDS under load	Enable autoscaling and provisioned IOPS for RDS
Cost overrun	Overuse of AWS services	Set up budgets and usage alerts

Impact Assessment:-

Positive Impacts

- Scalability: Auto-scaling ECS and RDS for growing student base.
- High Availability: Multi-AZ RDS, S3 durability.
- Security: IAM roles, S3 access control, SSL/TLS.
- Performance: Faster access through ALB, S3 + CloudFront.
- Maintainability: Centralized logging, containerized deployments.

Challenges

- AWS learning curve for developers/admins.
- Managing billing across multiple services.
- Requires a CI/CD pipeline for efficient updates.