Cloud-Based Learning Platform - Project Requirements Document

1. Executive Summary

1.1 Project Overview

This project involves building a comprehensive cloud-based learning platform that integrates AI-powered educational services with enterprise-grade cloud infrastructure. Students will gain hands-on experience with AWS services, event-driven architecture using Kafka, containerization, and microservices architecture.

1.2 Learning Objectives

- Master AWS core services (IAM, EC2, EBS, S3, VPC, Lambda, ELB, RDS)
- Implement event-driven architecture using Apache Kafka
- Design and deploy **containerized** microservices
- Integrate API Gateway for service orchestration
- Implement isolated storage architecture for each service
- Apply security best practices and access control

PHASE 1: AWS INFRASTRUCTURE LAYER (7 Marks)

Deadline: Thursday, 20/11/2025

2. AWS Services Requirements

2.1 Identity and Access Management (IAM)

Purpose: Secure access control for all AWS resources

Implementation Requirements:

- Create separate IAM roles for each service with least privilege principle
- Define IAM policies for EC2 instances to access S3, RDS, and other services
- Create IAM users for developers and administrators
- Implement MFA for administrative access
- Set up IAM roles for Lambda functions
- Create service accounts for containerized applications

Deliverables:

- IAM role definitions document
- Policy JSON files for each service
- Access control matrix

2.2 Elastic Compute Cloud (EC2)

Purpose: Host container orchestration platform and Kafka cluster

Implementation Requirements:

- Deploy EC2 instances for Docker/Kubernetes cluster (minimum 3 instances for HA)
- Deploy EC2 instances for Kafka brokers (minimum 3 brokers)
- Deploy EC2 instances for Zookeeper ensemble (3 nodes)
- Configure Auto Scaling Groups for container hosts
- Implement proper instance sizing (t3.medium minimum recommended)
- Set up CloudWatch monitoring for all instances

Deliverables:

- EC2 instance configuration templates
- Auto Scaling policies
- Launch templates for each instance type

2.3 Elastic Block Store (EBS)

Purpose: Persistent storage for EC2 instances and databases

Implementation Requirements:

- Attach EBS volumes to Kafka brokers for message persistence (GP3, minimum 100GB each)
- Attach EBS volumes to container hosts for Docker volumes (GP3, 50GB minimum)
- Configure EBS volumes for Zookeeper data directories (GP3, 20GB each)
- Implement automated EBS snapshots for backup
- Use encrypted EBS volumes for sensitive data

Deliverables:

- EBS volume configuration document
- Snapshot schedule and retention policy
- Disaster recovery procedures

2.4 Simple Storage Service (S3)

Purpose: Isolated object storage for each service

Implementation Requirements:

- Create separate S3 buckets for each service:
    - `tts-service-storage-{env}`: Store generated audio files
    - `stt-service-storage-{env}`: Store uploaded audio files for transcription
    - `chat-service-storage-{env}`: Store conversation histories and context
    - `document-reader-storage-{env}`: Store uploaded documents and generated notes
    - `quiz-service-storage-{env}`: Store generated quizzes and user responses
    - `shared-assets-{env}`: Static assets, container images backup
- Enable versioning on all buckets
- Configure lifecycle policies for cost optimization
- Implement server-side encryption (SSE-S3 or SSE-KMS)
- Set up bucket policies to restrict access per service
- Configure CORS for web access where needed

Deliverables:

- S3 bucket naming convention document
- Bucket policy JSON files
- Lifecycle policy configurations

2.5 Virtual Private Cloud (VPC)

Purpose: Network isolation and security

Implementation Requirements:

- Create VPC with CIDR block (e.g., 10.0.0.0/16)
- Design subnet architecture:
    - Public subnets (2 AZs): 10.0.1.0/24, 10.0.2.0/24 - for Load Balancers
    - Private subnets (2 AZs): 10.0.10.0/24, 10.0.11.0/24 - for containers
    - Data subnets (2 AZs): 10.0.20.0/24, 10.0.21.0/24 - for RDS
    - Kafka subnets (2 AZs): 10.0.30.0/24, 10.0.31.0/24 - for Kafka cluster
- Configure Internet Gateway for public subnets
- Configure NAT Gateways for private subnet internet access (one per AZ)
- Set up route tables for each subnet tier
- Implement Network ACLs for subnet-level security
- Create Security Groups for each service tier

Deliverables:

- VPC architecture diagram
- Subnet allocation table
- Security group definitions
- Network ACL rules

2.6 Lambda Functions

Purpose: Serverless processing for specific tasks

Implementation Requirements:

- Create Lambda function for S3 event processing (e.g., document upload triggers)
- Implement Lambda for cleanup tasks (delete old files, archive data)
- Create Lambda for Kafka consumer monitoring
- Implement Lambda for automated scaling decisions
- Configure Lambda layers for common dependencies
- Set up CloudWatch Events for scheduled Lambda executions
- Configure Lambda with VPC access where needed

Deliverables:

- Lambda function code repository
- Deployment automation scripts
- Lambda execution role policies

2.7 Elastic Load Balancer (ELB)

Purpose: Distribute traffic and ensure high availability

Implementation Requirements:

- Deploy Application Load Balancer (ALB) for API Gateway (public-facing)
- Deploy Network Load Balancer (NLB) for Kafka cluster (internal)
- Configure target groups for each microservice
- Implement health checks for all targets
- Set up SSL/TLS certificates using ACM
- Configure sticky sessions where needed
- Implement connection draining

Deliverables:

- Load balancer configuration documents
- Target group definitions
- Health check specifications

2.8 Relational Database Service (RDS)

Purpose: Isolated database for each service requiring relational storage

Implementation Requirements:

- Deploy separate RDS instances for services:
    - **User Management DB**: PostgreSQL (stores user accounts, authentication)

    - **Chat Service DB**: PostgreSQL (stores conversation metadata, user preferences)
    - **Document Reader DB**: PostgreSQL (stores document metadata, note associations)
    - **Quiz Service DB**: PostgreSQL (stores quiz definitions, user responses, scores)
- Configure Multi-AZ deployment for production
- Implement automated backups (7-day retention minimum)
- Enable encryption at rest
- Configure appropriate instance sizes (db.t3.medium minimum)
- Set up read replicas for read-heavy services
- Implement database parameter groups for optimization
- Configure security groups to allow access only from service containers

Deliverables:

- RDS configuration templates
- Database schema designs for each service
- Backup and recovery procedures
- Database access credentials management plan

PHASE 2: MICROSERVICES & KAFKA LAYER (7 Marks)

Deadline: Thursday, 04/12/2025

3. Apache Kafka

3.1 Kafka Cluster Architecture

Purpose: Event streaming and asynchronous communication between services

Implementation Requirements:

- Deploy 3-node Kafka cluster for high availability
- Deploy 3-node Zookeeper ensemble
- Configure topics for each service interaction:
    - `document.uploaded`: Triggered when document is uploaded
    - `document.processed`: Document reading completed
    - `notes.generated`: Notes created from document
    - `quiz.requested`: User requests quiz generation
    - `quiz.generated`: Quiz creation completed
    - `audio.transcription.requested`: STT request
    - `audio.transcription.completed`: STT completed
    - `audio.generation.requested`: TTS request

    - `audio.generation.completed`: TTS completed
    - `chat.message`: Chat interactions
- Configure appropriate replication factor (minimum 2)
- Set up retention policies per topic
- Implement partitioning strategy for scalability
- Configure producer and consumer groups

Deliverables:

- Kafka cluster deployment scripts
- Topic configuration specifications
- Producer/consumer implementation guidelines
- Kafka monitoring dashboard

3.2 Kafka Integration Patterns

- **Event Sourcing**: Store state changes as events
- **CQRS**: Separate read and write operations
- **Saga Pattern**: Manage distributed transactions
- **Event Notification**: Notify services of important events

---

4. API Gateway Implementation

4.1 API Gateway Architecture

Purpose: Single entry point for all client requests

Implementation Requirements:

- Deploy containerized API Gateway (Kong, AWS API Gateway, or custom)
- Configure routes for each microservice:
    - `/api/tts/*` → TTS Service
    - `/api/stt/*` → STT Service
    - `/api/chat/*` → Chat Service
    - `/api/documents/*` → Document Reader Service
    - `/api/quiz/*` → Quiz Service
- Implement authentication and authorization (JWT-based)
- Configure rate limiting per endpoint
- Set up request/response transformation
- Implement API versioning
- Configure CORS policies
- Set up request logging and monitoring

Deliverables:

- API Gateway configuration files
- API documentation (OpenAPI/Swagger)
- Authentication flow diagrams
- Rate limiting policies

---

5. Microservices Specifications

5.1 Text-to-Speech (TTS) Service

Functionality:

- Convert text input to natural-sounding speech
- Support multiple languages and voices
- Provide audio format options (MP3, WAV, OGG)

Technical Requirements:

- **Container**: Docker image based on Python 3.11
- **Storage**: Dedicated S3 bucket (`tts-service-storage-{env}`)
- **Database**: Optional Redis cache for frequent requests
- **APIs**:
  - `POST /api/tts/synthesize`: Generate speech from text
  - `GET /api/tts/audio/{id}`: Retrieve generated audio
  - `DELETE /api/tts/audio/{id}`: Delete audio file
- **Kafka Integration**:
  - Produce: `audio.generation.completed`
  - Consume: `audio.generation.requested`
- **Libraries**: gTTS, Coqui TTS, or AWS Polly SDK
- **Resources**: 1 CPU, 2GB RAM minimum

Storage Isolation:

- All generated audio files stored in dedicated S3 bucket
- Temporary files stored in container ephemeral storage
- No shared storage with other services

5.2 Speech-to-Text (STT) Service

Functionality:

- Transcribe audio files to text
- Support multiple audio formats
- Provide confidence scores
- Support multiple languages

Technical Requirements:

- **Container**: Docker image based on Python 3.11
- **Storage**: Dedicated S3 bucket (`stt-service-storage-{env}`)
- **Database**: PostgreSQL for transcription metadata and history
- **APIs**:
  - `POST /api/stt/transcribe`: Upload audio and transcribe
  - `GET /api/stt/transcription/{id}`: Get transcription result
  - `GET /api/stt/transcriptions`: List user transcriptions
- **Kafka Integration**:
  - Produce: `audio.transcription.completed`
  - Consume: `audio.transcription.requested`
- **Libraries**: Whisper, SpeechRecognition, or AWS Transcribe SDK
- **Resources**: 2 CPU, 4GB RAM minimum

Storage Isolation:

- Uploaded audio files stored in dedicated S3 bucket
- Transcription results stored in dedicated PostgreSQL database
- Audio files deleted after configurable retention period

5.3 Chat Completion Service

Functionality:

- Provide conversational AI capabilities
- Maintain conversation context
- Support multi-turn conversations
- Integrate with document knowledge base

Technical Requirements:

- **Container**: Docker image based on Python 3.11
- **Storage**:
  - S3 bucket (`chat-service-storage-{env}`) for conversation archives
  - PostgreSQL database for conversation metadata
- **APIs**:
  - `POST /api/chat/message`: Send message and get response
  - `GET /api/chat/conversations`: List user conversations

    - `GET /api/chat/conversations/{id}`: Get conversation history
    - `DELETE /api/chat/conversations/{id}`: Delete conversation
- **Kafka Integration**:
    - Produce: `chat.message`
    - Consume: `document.processed` (to access document knowledge)
- **Libraries**: OpenAI SDK, LangChain, or Hugging Face Transformers
- **Resources**: 2 CPU, 4GB RAM minimum

Storage Isolation:

- Conversation histories stored in dedicated S3 bucket and PostgreSQL
- Session data stored in Redis cache (optional)
- No direct access to other services' storage

5.4 Document Reader Service

Functionality:

- Upload and process various document formats (PDF, DOCX, TXT)
- Extract text and structure from documents
- Generate summarized notes using AI
- Store documents and notes for future reference

Technical Requirements:

- **Container**: Docker image based on Python 3.11
- **Storage**:
    - S3 bucket (`document-reader-storage-{env}`) for documents and notes
    - PostgreSQL database for document metadata and relationships
- **APIs**:
    - `POST /api/documents/upload`: Upload document
    - `GET /api/documents/{id}`: Get document details
    - `GET /api/documents/{id}/notes`: Get generated notes
    - `POST /api/documents/{id}/regenerate-notes`: Regenerate notes
    - `GET /api/documents`: List user documents
    - `DELETE /api/documents/{id}`: Delete document
- **Kafka Integration**:
    - Produce: `document.uploaded`, `document.processed`, `notes.generated`
    - Consume: None
- **Libraries**: PyPDF2, python-docx, spaCy, OpenAI SDK
- **Resources**: 2 CPU, 4GB RAM minimum

Storage Isolation:

- Original documents stored in dedicated S3 bucket
- Generated notes stored in same bucket with different prefix

- Document metadata in dedicated PostgreSQL database

5.5 Quiz and Exercise Service

Functionality:

- Generate quizzes from previously uploaded documents
- Create multiple question types (multiple choice, true/false, short answer)
- Store user responses and calculate scores
- Provide detailed feedback and explanations

Technical Requirements:

- **Container**: Docker image based on Python 3.11
- **Storage**:
  - S3 bucket (`quiz-service-storage-{env}`) for quiz templates
  - PostgreSQL database for quiz definitions, user responses, scores
- **APIs**:
  - `POST /api/quiz/generate`: Generate quiz from document
  - `GET /api/quiz/{id}`: Get quiz questions
  - `POST /api/quiz/{id}/submit`: Submit quiz answers
  - `GET /api/quiz/{id}/results`: Get quiz results and feedback
  - `GET /api/quiz/history`: Get user's quiz history
  - `DELETE /api/quiz/{id}`: Delete quiz
- **Kafka Integration**:
  - Produce: `quiz.generated`
  - Consume: `quiz.requested`, `notes.generated`
- **Libraries**: OpenAI SDK, LangChain for question generation
- **Resources**: 2 CPU, 4GB RAM minimum

Storage Isolation:

- Quiz templates and generated quizzes in dedicated S3 bucket
- Quiz responses and scores in dedicated PostgreSQL database
- Access document content through Kafka events, not direct storage access

---

6. Containerization Requirements

6.1 Docker Implementation

Requirements for All Services:

- Create Dockerfile for each service
- Use multi-stage builds to minimize image size
- Implement health check endpoints

- Use environment variables for configuration
- Never hardcode credentials
- Implement graceful shutdown handling
- Use official base images (e.g., python:3.11-slim)
- Tag images with version numbers
- Scan images for vulnerabilities

6.2 Container Orchestration

Options:

- **Docker Swarm** (simpler, good for learning)
- **Kubernetes** (industry standard, more complex)

Requirements:

- Deploy all services as containers
- Implement service discovery
- Configure resource limits (CPU, memory)
- Set up persistent volume claims for databases
- Implement rolling updates
- Configure horizontal pod autoscaling
- Set up health checks and readiness probes

6.3 Container Registry

- Use Amazon ECR (Elastic Container Registry)
- Create repository for each service
- Implement automated image builds and pushes
- Configure image scanning
- Set up lifecycle policies for old images

Deliverables:

- Dockerfile for each service
- Docker Compose file for local development
- Kubernetes manifests or Docker Swarm configurations
- CI/CD pipeline for container builds

PHASE 3 — SECURITY, STORAGE & CI/CD (6 Marks)

Deadline: Thursday, 18/12/2025

7. Storage Isolation Architecture

7.1 Storage Isolation Principles

Each service must have completely isolated storage with no shared access:

S3 Isolation:

- Separate S3 bucket per service
- Bucket policies enforce service-specific IAM role access only
- No cross-bucket access permissions
- Separate encryption keys per bucket (KMS)

Database Isolation:

- Separate RDS instance or database per service
- Unique database credentials per service
- Security groups allow access only from specific service containers
- No shared schemas or tables

Volume Isolation:

- Container-specific volumes for temporary storage
- No volume sharing between containers
- Data persistence through S3 or RDS only

7.2 Data Sharing Mechanism

Services share data through events, not direct storage access:

- Service A processes data and publishes event to Kafka
- Service B consumes event and receives necessary data payload
- Service B stores data in its own storage if needed
- No direct database queries or S3 access between services

Example Flow:

1. User uploads document to Document Reader Service
2. Document Reader stores document in its S3 bucket
3. Document Reader publishes `document.processed` event with document ID and extracted text
4. Quiz Service consumes event and stores relevant data in its own database
5. Quiz Service generates quiz and stores in its own S3 bucket

---

8. Security Requirements

8.1 Network Security

- All services deployed in private subnets
- Only ALB exposed to internet
- Implement security groups with least privilege
- Use VPC Flow Logs
- Enable AWS WAF on ALB
- Implement DDoS protection with AWS Shield

8.2 Data Security

- Encrypt all data at rest (S3, EBS, RDS)
- Encrypt all data in transit (TLS 1.3)
- Use AWS KMS for key management
- Separate encryption keys per service
- Implement secrets management (AWS Secrets Manager)
- Regular key rotation

8.3 Access Control

- Implement JWT-based authentication
- Use IAM roles for service-to-AWS communication
- Never use long-term credentials in containers
- Implement service-to-service authentication
- Use API Gateway for authorization
- Implement audit logging

9. Deployment and CI/CD

9.1 Infrastructure as Code

- Implement modular infrastructure components
- Separate environments (dev, staging, production)

9.2 CI/CD Pipeline

Requirements:

- Automated testing (unit, integration, e2e)
- Automated container builds
- Automated vulnerability scanning
- Automated deployment to container platform
- Blue-green deployment strategy

- Automated rollback capability

Tools:

- GitHub Actions, GitLab CI, or AWS CodePipeline
- Docker for containerization

---

10. Documentation Deliverables

10.1 Technical Documentation

- Architecture diagrams
- API documentation (OpenAPI or Swagger)
- Database schemas
- Infrastructure documentation
- Deployment procedures
- Troubleshooting guide

10.2 User Documentation

- User guide for each service
- API usage examples
- Integration guide

10.3 Operations Documentation

- Monitoring and alerting guide
- Incident response procedures
- Backup and recovery procedures

Final Grading Summary

Phase	Description	Marks	Deadline
Phase 1	AWS Infrastructure Layer	**7**	20/11/2025
Phase 2	Microservices Development & Event Integration	**7**	04/12/2025
Phase 3	Containerization, CI/CD & Observability	**6**	18/12/2025
Total		**20 Marks**	