# Deployment of a Scalable Simple Learning Management System (LMS) Using AWS Cloud Services

**Team**

- **Name 1 (Leader):** Ibrahim Sattar — Registration #23004
- **Name 2:** Cheema Amjad Abdullah — Registration #23091
- **Repository:** https://github.com/frappe/lms (Frappe Learning)

## 1. Introduction / Background

With the growing need for online education, institutions require platforms where students can access courses, track progress, and interact with educational content reliably. Many traditional LMS deployments struggle with server overload, downtime during peak usage, and complex maintenance needs.

This project proposes deploying the open-source **Frappe Learning (frappe/lms)** project on AWS cloud services. Frappe Learning is a modern, easy-to-use learning system built on the **Frappe Framework** with a Vue-based UI library (Frappe UI). The project will be adapted for a cloud-native deployment that prioritizes scalability, high availability, security, and operational best practices.

**High-level tech stack (as adapted from the repository):**

- **Application framework / Backend:** Frappe Framework (Python)
- **Frontend UI:** Frappe UI (Vue-based)
- **Container / Image:** Official image available (can be deployed via Docker / container runtime)
- **Database:** MariaDB / MySQL-compatible (Frappe uses MariaDB/MySQL by default)

## 2. Problem Statement

Traditional single-server LMS deployments commonly fail to meet the following operational and functional requirements:

- Fast response times during periods of high traffic
- Automatic scaling when the number of concurrent users increases
- Protection against data loss and accidental deletion
- Clear separation of concerns between frontend, backend, and database
- Ease of maintenance and automated, repeatable updates via CI/CD

A cloud-based, distributed architecture is required to address these challenges.

## 3. Project Goal

Design and deploy a scalable, secure, and highly available Frappe Learning instance on AWS. The deployment will follow cloud-native principles and use the following AWS services as appropriate:

- Amazon EC2 (or ECS / EKS as an alternative)
- Auto Scaling Groups (EC2) or ECS Service Autoscaling
- Application Load Balancer (ALB)
- Amazon RDS (MySQL / MariaDB)
- Amazon S3 + CloudFront (for static assets and possible CDN of public assets)
- AWS Secrets Manager (for database and other secrets)
- IAM Roles and Policies
- VPC with Public and Private Subnets, NAT, and Route Tables
- CloudWatch for logs and metrics

## 4. Scope of the Project

**Frontend Hosting (S3 + CloudFront)**

- Host static assets (images, uploaded course previews, and public assets) in S3 and accelerate with CloudFront for global low-latency delivery.
- Configure HTTPS (via ACM) at the CloudFront distribution.

**Backend Hosting (EC2 + Auto Scaling + ALB)**

- Run the Frappe application on EC2 instances using Docker (official image ghcr.io/frappe/lms) or using a Python/bench-based deployment.
- Use Launch Templates and Auto Scaling Groups to scale based on CPU / request load.
- Route external HTTP(S) traffic through an Application Load Balancer which performs health checks and SSL termination.

**Database Layer (RDS MySQL / MariaDB)**

- Deploy Amazon RDS for MySQL (or Amazon Aurora MySQL compatible) in private subnets with no public access.
- Enable Multi-AZ for high availability and automated failover.
- Use AWS Secrets Manager to store DB credentials and rotate them if needed.

**Network Architecture (VPC, Subnets, Gateways)**

- Public subnets for ALB and NAT Gateway.
- Private subnets for application EC2 instances and RDS database (or keep application instances in public if required by design but prefer private for security).
- NAT Gateway for controlled outbound internet access for instances in private subnets.
- Proper route tables and security groups to restrict traffic (least-privilege).

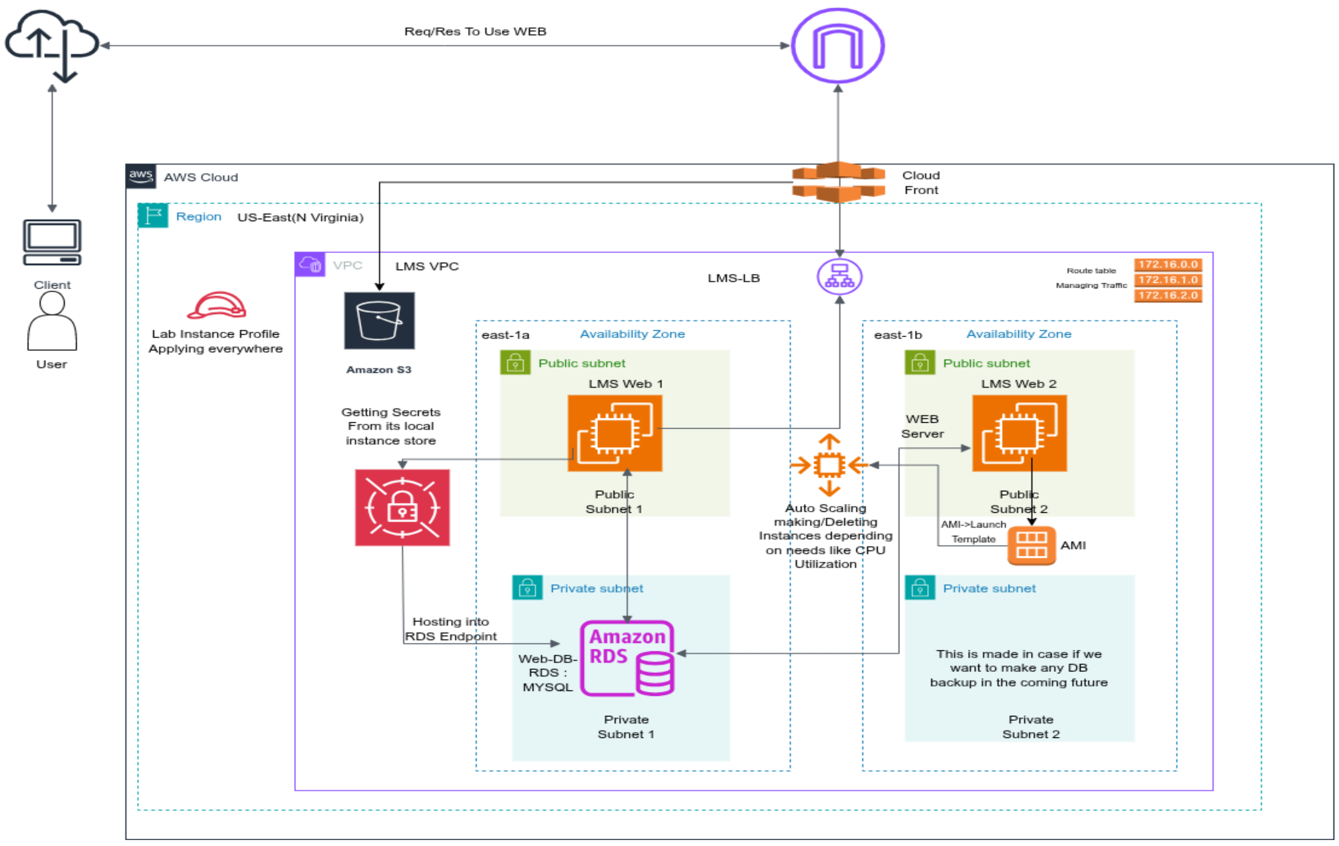
**Monitoring & Logging**

- Send application logs and metrics to CloudWatch (use CloudWatch Agent or container logging driver).
- Configure alarms for key metrics (CPU, disk, DB connections, HTTP 5xx rate).

**Optional Enhancements**

- Use AWS Lambda for lightweight background tasks or cron jobs (e.g., scheduled notifications, certificate generation).
- Integrate SNS for important alerts and notifications.
- Implement a CI/CD pipeline (CodePipeline / CodeBuild / GitHub Actions) to automate building and deploying container images and running migrations.

## 5. Proposed AWS Architecture



**Key features:**

- ALB handles SSL termination and load balancing.
- Auto Scaling ensures application capacity adjusts to load.
- RDS Multi-AZ protects against AZ failure.
- Secrets Manager stores DB credentials securely.

## 6. Component Details

1. **S3 + CloudFront**
    - Store public assets and static files (course images, static downloads).
    - Use CloudFront for caching, HTTPS, and faster global delivery.
2. **EC2 + Auto Scaling + Launch Template**
    - EC2 instances will run the Frappe application inside Docker (or native bench deployment if preferred).
    - Create a Launch Template which contains user-data to bootstrap the container and register with the ALB target group.
    - Auto Scaling policies will be based on CPU, memory, or custom ALB request metrics.
3. **Application Load Balancer**
    - Routes incoming HTTPS traffic to healthy EC2 targets.
    - Uses health checks (HTTP) pointing to an application health endpoint.
4. **RDS MySQL / MariaDB**
    - Production-grade RDS with Multi-AZ, automated backups, and snapshots.
    - Keep RDS inside private subnets with security group restrictions.
5. **Secrets Manager & IAM Roles**
    - Store DB credentials and other sensitive information in Secrets Manager.
    - Assign an instance profile (IAM Role) to EC2 instances so they can securely retrieve secrets and push logs to CloudWatch.
6. **VPC and Networking**
    - Design VPC with at least two AZs for resilience.
    - Public subnets for ALB and NAT Gateway; private subnets for application and DB.
    - Security groups enforce least privilege network access.

## 7. Why This Architecture Works for an LMS

- **Highly Available:** ALB + Auto Scaling + RDS Multi-AZ provides resilience against failures.
- **Scalable:** Auto Scaling reacts to user demand; CloudFront reduces load on origin.
- **Secure:** Database in private subnet, secrets stored in Secrets Manager, and controlled IAM roles.
- **Fast:** CloudFront CDN and caching reduce global latency.
- **Maintainable:** Containerization and CI/CD promote repeatable, quick deployments and rollbacks.

### Example user-data script (EC2) — bootstrap (illustrative)

```
#!/bin/bash
# Install Docker and start the frappe/lms container (illustrative)
apt-get update -y
apt-get install -y docker.io
systemctl enable --now docker
# Pull and run official image (replace tags & env vars appropriately)
docker run -d \
  --name frappe-lms \
  --restart unless-stopped \
  -e SITE_NAME=your.domain.tld \
  -e DB_HOST=<rds-endpoint> \
  -e DB_ROOT_USER=<username> \
  -e DB_ROOT_PASSWORD=<password-from-secrets-manager> \
  -p 8000:8000 \
  ghcr.io/frappe/lms:stable

# Note: In production, use ECS/EKS, or a process manager, and retrieve secrets from AWS Secrets Manager
```

**Security note:** We will use IAM roles + AWS CLI or SDK to fetch secrets at runtime.

## 8. Deployment Plan (high-level steps)

1. **Prepare AWS account & VPC**: Create VPC, subnets (public/private across 2 AZs), NAT Gateway, and route tables.
2. **Provision RDS**: Launch Amazon RDS for MySQL/MariaDB in private subnets with Multi-AZ.
3. **Create S3 Bucket & CloudFront**: Upload public assets to S3 and configure CloudFront + ACM certificate.
4. **Create Launch Template & AMI**: Prepare an AMI or use user-data to bootstrap containers on EC2.
5. **Create ALB & Target Group**: Configure ALB listeners (HTTPS) and target groups for EC2 instances.
6. **Create Auto Scaling Group**: Use Launch Template and configure scaling policies.

7. **Secrets & IAM**: Store DB credentials in Secrets Manager and give EC2 role permission to read secrets.
8. **CI/CD Integration**: Create a pipeline (GitHub Actions or CodePipeline) to build, test and deploy container images.
9. **Monitoring & Alerts**: Configure CloudWatch dashboards and alarms for key metrics.
10. **Testing & Handover**: Perform load testing and document the deployment with screenshots.

## 9. Risks & Mitigations

- **Single AZ dependency**: Use Multi-AZ for RDS and place EC2 across multiple AZs.
- **Misconfigured security groups**: Apply least-privilege rules and peer-review network configs.
- **Secrets leakage**: Use Secrets Manager, avoid embedding secrets in scripts.
- **Cost overruns**: Use cost estimates, turn off non-production resources when idle.

## 10. References

- Frappe Learning GitHub repository: https://github.com/frappe/lms
- Frappe Framework documentation: https://frappeframework.com

**Prepared by:** Ibrahim Sattar (Leader) & Cheema Amjad Abdullah

**Title:** Deployment of a Scalable Simple Learning Management System (LMS) Using AWS Cloud Services

*End of document.*