AWS First Cloud AI Journey – **Project Plan**

[Mambo] – [FPT University] – [Mini-Market]
[September 8, 2025 – December 12, 2025]

Table of Contents

1 BACKGROUND AND MOTIVATION

1.1 EXECUTIVE SUMMARY

Mini-Market is an academic project simulating the digital transformation of a traditional mini-market model in Vietnam. The hypothetical client is a small mini-market chain currently operating primarily through manual processes (ledgers, Excel, offline sales).

Business objectives:

- *Reduce inventory loss and errors in warehouse management (due to manual inventory).*
- *Open additional online revenue channels through an e-commerce platform.*
- *Shorten payment time and improve customer experience at the counter.*
- *Establish a foundation for future expansion of mini-market branches.*

Technical objectives:

- *Build a modern, easy-to-maintain **3-layer (.NET 3-tier)** architecture, applying **the Repository Pattern & Unit of Work Pattern**.*
- *Apply **AWS Managed Services** (Elastic Beanstalk, RDS, S3, CloudFront, WAF, ElastiCache) according to **the AWS Well-Architected Framework**.*
- *Automate **CI/CD** with AWS CodePipeline & CodeBuild integrated with GitHub.*
- *Optimize costs by taking advantage of **the AWS Free Tier** in the first year.*

Mini-market e-commerce website with key features: product management, catalog, shopping cart, orders, user accounts, and basic inventory management.

- *Design **solution architecture** on the AWS platform.*
- *Deploy necessary AWS services for Staging and Production environments.*
- *Design & develop an **ASP.NET Core MVC** application using the 3-tier model.*
- *Set up CI/CD pipeline with AWS CodePipeline / CodeBuild.*
- *Configure basic security services: VPC, Security Group, WAF, Private Subnet, NAT Gateway.*

1.2 PROJECT SUCCESS CRITERIA

The project is considered successful if the following quantitative criteria are met (in the demo/POC environment):

Business success criteria

- *Reduce inventory management discrepancies by* ***≥ 90%*** *compared to the manual/Excel model (measured by the difference between system inventory and actual inventory during test checks).*
- *Reduce checkout time (from product scanning to receipt printing)* ***by ≥ 50%*** *compared to the simulated manual process.*
- *Achieve* ***≥ 20%*** *simulated revenue from online channels (compared to total simulated revenue) within* ***6 months*** *if implemented in practice.*

Technical success criteria

- *Uptime (on the demo environment)* ***≥ 99.9%*** *during the testing period.*
- *Average homepage (product listing) load time* ***< 2 seconds*** *when using CloudFront and ElastiCache (tested with simulated load scenario).*
- *CI/CD pipeline successfully deploys* ***≥ 90%*** *of code pushes to the main branch without manual intervention.*
- *All secrets/connection strings are not hard-coded in the source code but are managed via Elastic Beanstalk* ***environment variables****.*

1.3 ASSUMPTIONS

Key assumptions for project execution:

- ***AWS Account****: The customer (or project team) provides an AWS Account with permissions to create the following services: VPC, EC2/Beanstalk, RDS, S3, CloudFront, WAF, ElastiCache, CodePipeline, CodeBuild.*
- ***Free Tier****: Infrastructure uses* ***the AWS Free Tier*** *for the first year, including EC2 t3.micro/t3a.small, RDS db.t3.micro (SQL Server Express), ElastiCache t4g.micro, S3, and CloudFront.*
- ***Traffic****: Low to moderate initial traffic (demo environment, not yet a production system with tens of thousands of users).*
- ***Data volume****: Moderate number of products and orders (a few thousand SKUs, a few thousand orders/month) – suitable for the proposed architecture.*
- ***Team availability****: The team has sufficient time to work within the timeframe [08/09/2025 – 12/12/2025].*
- ***External dependencies****:*
    - *No mainframe systems or legacy on-premises systems require integration.*
    - *Integration with actual payment gateways (VNPay, MoMo, etc.)* ***is not within the main scope*** *– only payment simulation (e.g., COD).*
- ***Risks & constraints****:*
    - *Time constraints (semester project).*
    - *Some AWS services may change their pricing or Free Tier policies in the future.*

2 SOLUTION ARCHITECTURE / ARCHITECTURAL DIAGRAM

2.1 Technical Architecture Diagram

High-level architecture description

*The proposed solution uses a **3-tier (.NET)** architecture combined with AWS infrastructure according to the Well-Architected Framework:*

- ***Presentation Layer (WebShop)**:*
  - *ASP.NET Core MVC 9.0*
  - *Runs on AWS Elastic Beanstalk (Windows/Linux platform for .NET)*
  - *Communicates with the Application Layer through internal services within the same project/solution.*
- ***Application Layer (Services)**:*
  - *Contains business logic: ProductService, OrderService, UserService, InventoryService, etc.*
  - *Applies **the Repository Pattern & Unit of Work Pattern** to separate business logic and data access.*
- ***Persistence Layer (Data Access)**:*
  - *Uses **Entity Framework Core** to connect to **Amazon RDS for SQL Server**.*
  - *Repositories for each aggregate: ProductRepository, OrderRepository, etc.*

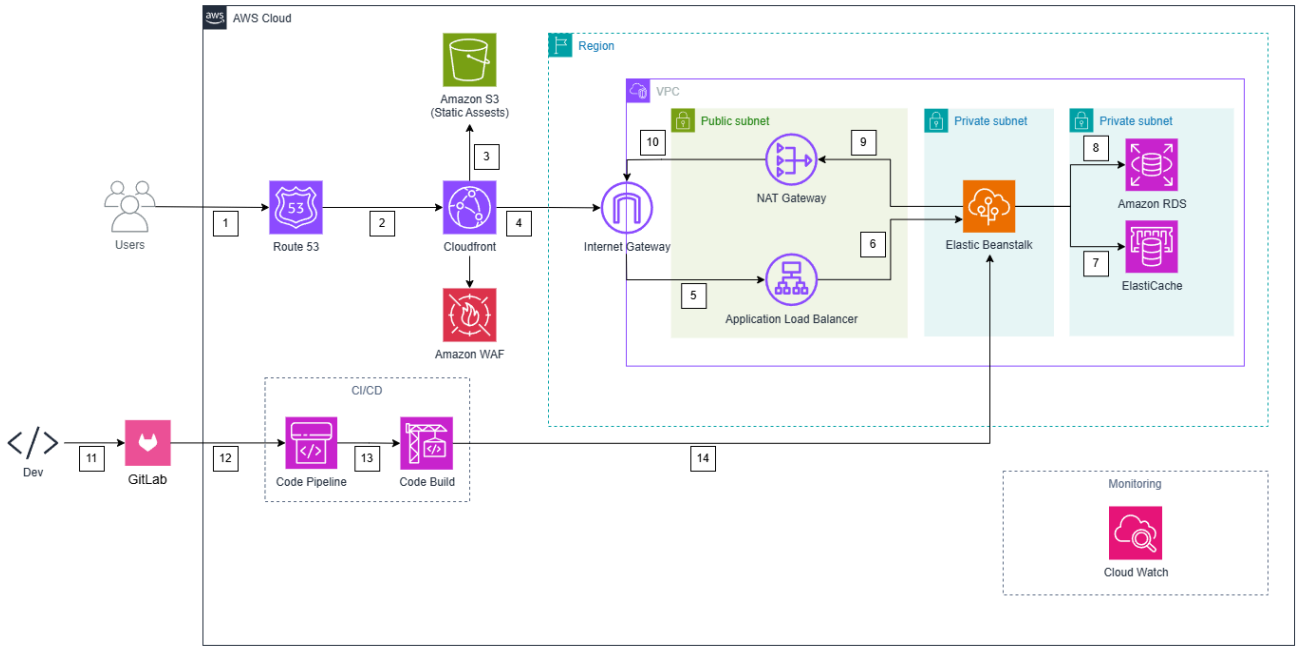
AWS Infrastructure components

- ***Amazon Route 53**: Manage DNS and domains for the mini-market application.*
- ***Amazon CloudFront**: CDN caches static content (product images, JS, CSS) from S3 to speed up page loading.*
- ***Amazon S3**: Store static assets (product images, CSS, JS).*
- ***Amazon Elastic Beanstalk**:*
  - *Host ASP.NET Core MVC applications.*
  - *Automatically create EC2, Auto Scaling Group, Application Load Balancer.*
  - *Place in **a Private Subnet** for increased security, accessible only through ALB.*
- ***Amazon RDS for SQL Server (Express)**:*
  - *Database for the entire system.*
  - *Placed in a Private Subnet, no public internet access.*
  - *Automatically back up, patch, supports Multi-AZ (can be enabled if needed).*
- ***Amazon ElastiCache (Redis)**:*
  - *Hot data cache: product lists, category information, general configuration.*
  - *Reduce repetitive queries to RDS.*

- ***AWS WAF**:*
    - *Integrated with CloudFront to block common attacks (SQL injection, XSS, bad bots).*
- ***AWS VPC + Subnets + NAT Gateway + Internet Gateway**:*
    - *Dedicated VPC for the system.*
    - *Public Subnet: ALB, NAT Gateway, CloudFront edge (outside).*
    - *Private Subnet: Elastic Beanstalk instances, RDS, ElastiCache.*
    - *NAT Gateway used for instances in the Private Subnet to access the Internet (updates, patches).*
- ***AWS CodePipeline + AWS CodeBuild**:*
    - *CI/CD from GitHub → Build → Deploy to Elastic Beanstalk.*

Proposed tools

- *Visual Studio*
- *Git & GitHub*
- *AWS Management Console*
- *(Optional) AWS CloudFormation or Elastic Beanstalk configuration templates*
- *Docker Desktop (for local testing)*



2.2 Technical Plan

Our group will:

- Set up a 3-layer .NET solution consisting of the following projects: Domain, Application, Persistence, WebShop.
- Build code that complies with Repository + Unit of Work to ensure easy testing and maintenance.
- Use GitHub as source control, with branches: main, develop, and feature branches.
- Set up a CI/CD Pipeline using AWS CodePipeline + CodeBuild:
    - Trigger: Push to the main branch on GitHub.
    - CodeBuild: Build the .NET solution, run unit tests.
    - Artifacts: .zip package for Elastic Beanstalk.
    - Deploy: CodePipeline automatically deploys to the Staging environment and (after approval) to Production.

Some configurations will be performed manually via the AWS Console (appropriate for the scope of the semester), including:

- Create VPC, Subnets, Security Groups.
- Create RDS, ElastiCache, S3, WAF, CloudFront.
- Connecting the domain from Route 53 to CloudFront/ALB.

All critical paths such as: Registration, Login, Add to Cart, Checkout, Create Order will be written as test cases and thoroughly tested on the Staging environment.

2.3 Project Plan

Our group will adopt an **Agile Scrum-like** model with short sprints (1–2 weeks) over a total of **12 weeks**, divided into 4 phases:

- **Phase 1 – Assessment & Foundation (Weeks 1–4)**
    - Analyze requirements and finalize use cases.
    - Design logical & physical architecture.
    - Set up .NET solution, GitHub repo, AWS VPC + Subnets.
- **Phase 2 – Core Feature Development (Weeks 5–8)**
    - Develop core features (Product, Cart, Order, Auth).
    - Build Application Services, Repositories, and Units of Work.
    - Write Unit Tests for business logic.
- **Phase 3 – Cloud Preparation & Refactoring (Weeks 9–11)**

- Refactor the source code for cloud compatibility.
- Convert hardcoded values to Environment Variables.
- Write buildspec.yml and prepare Dockerfile.

- **Phase 4 – Infrastructure Deployment & Go-live (Week 12)**
  - Provision all AWS resources (Beanstalk, RDS, ElastiCache, S3).
  - Configure Security (WAF, CloudFront).
  - Activate the CI/CD Pipeline.
  - UAT & Monitoring.

Communication & Governance

- Weekly meeting (review & planning) between Mambo members and instructors.
- Sprint Review at the end of each phase with feature demo.
- Any scope changes (if any) will be recorded in the backlog and prioritized in the next sprint.

2.4 Security Considerations

1. **Access**
   - Enable mandatory **MFA** for AWS root accounts and IAM users.
   - Use **IAM Roles** for EC2/Beanstalk, CodeBuild, instead of storing access keys in code.
   - Apply the **Least Privilege** principle to all IAM policies.
2. **Infrastructure**
   - Separate **Public** and **Private Subnets** in VPC.
   - RDS, ElastiCache, and Beanstalk instances are located in the Private Subnet and do not have public IP addresses.
   - Only ALB and CloudFront are public.
   - Security Groups only open necessary ports (HTTP/HTTPS, DB port with source restrictions).
3. **Data**
   - Enable **Encryption at Rest** for RDS, S3, and ElastiCache (if applicable).
   - Encrypt sensitive data (user passwords using Hashing + Salt in the application).
   - Do not store plaintext passwords/secrets in code or repositories.
4. **Detection**
   - Enable **AWS CloudTrail** for the entire account to audit API calls.

- Use **AWS CloudWatch Logs & Metrics** for Beanstalk, RDS, and ALB to monitor health.
- (Optional) Configure basic rules to monitor misconfigurations.

5. **Incident Management**
   - Set up **CloudWatch Alarms** for CPU, response time, and error rate.
   - Define incident response procedures: scale up, rollback deploy, restore DB from backup (PITR).
   - Test the process of restoring RDS from a snapshot periodically (within the scope of demo/academic use).

3 Activities AND Deliverables

3.1 Activities and Deliverables

Project Phase	Timeline	Activities	Deliverables/Milestones	Total man-day
Assessment & Foundation	Weeks 1–4	• Requirements gathering • Architecture design • Set up VPC	• Solution Architecture Doc, AWS Network, GitHub Repo	32 Hours
Core Feature Development	Weeks 5-8	• Backend/Frontend Code • Unit Test • Business Logic	• Web app running locally with stable core features	72 Hours
Cloud Prep & Refactoring	Weeks 9-11	● Refactor code for Cloud ● Configure Environment Variables ● Dockerize ● Buildspec	● Dockerfile, Buildspec.yml, Cloud Native-compliant source code	54 Hours
Infrastructure & Go-live	Week 12	● Setup RDS/Redis/EB ● Configure Security ● CI/CD ● UAT	● Production URL, Handover Documentation	24 Hours

Project Phase: Assessment & Foundation

- **Timeline**: Weeks 1–4
- **Activities**:
    - Collect business and technical requirements.
    - Design logical architecture and infrastructure architecture on AWS.
    - Set up basic VPC, Subnets, and Security Groups.
    - Initialize GitHub repository, create a 3-layer .NET solution.

- **Deliverables / Milestones**:
    - Solution Architecture Document.
    - AWS Network (VPC + Subnets) created.
    - GitHub repo + base code structure.

Project Phase: Set up Base Infrastructure

- **Timeline**: Weeks 5–8
- **Activities**:
    - Develop business services: Product Management, Shopping Cart, Orders, User.
    - Integration of Entity Framework Core, Repository Pattern, Unit of Work.
    - Build the WebShop interface (Razor Views/Bootstrap). Write Unit Tests for critical logic.
- **Deliverables**:
    - A stable Mini-market Web application running on a Local environment (with full functional features).
    - Unit Test results meet coverage requirements.

Project Phase: Application Component Setup

- **Timeline**: Weeks 9–11
- **Activities**:
    - Refactoring: Convert hard-coded configurations to environment variables for Elastic Beanstalk compatibility.
    - Containerization: Write an optimized Dockerfile (Multi-stage build) and buildspec.yml.
    - S3 Integration: Write a service to upload images to S3 instead of storing them locally.
    - Redis Integration: Configure Distributed Cache using Redis for Session.
- **Deliverables**:
    - Source code optimized for the cloud (Cloud-Native ready).
    - Complete Docker configuration files and CI/CD (Buildspec).

Project Phase: AWS Integration & Optimization

- **Timeline**: Week 12
- **Activities**:
    - Create all resources: RDS (SQL Server), ElastiCache (Redis), Elastic Beanstalk.
    - Configure Security: CloudFront (HTTPS), WAF (Firewall).
    - Enable CI/CD Pipeline (CodePipeline) for automated deployment.
    - Perform UAT (User Acceptance Testing) and check CloudWatch Metrics.
- **Deliverables**:

- Production system running (Live Demo URL).
- Handover documentation and operational guidelines.

3.2 OUT OF SCOPE

Actual integration with Vietnamese payment gateways (VNPay, MoMo, ZaloPay, etc.) – simulation only.

3.3 PATH TO PRODUCTION

The POC/MVP will be built with the following key use cases: registration, login, product browsing, shopping cart, payment, order management.

The POC will run in a **Staging** environment **on Elastic Beanstalk** with full RDS, S3, and ElastiCache.

To move to Production, additional steps include:

- Tuning Auto Scaling rules, RDS instance size, backup & retention.
- Optimize WAF rule sets, rate-limiting, CORS, and HTTPS enforcement.
- Expand test cases (load testing, security testing).

The production setup within the project scope will be a **demo environment** with Route 53 domain, CloudFront, WAF, ready to scale up if deployed in production.

4 EXPECTED AWS COST BREAKDOWN BY SERVICES

AWS Pricing Calculator

Assumptions for cost estimation:

- Use small instances (t3a.small, t3.micro, t4g.micro) suitable for a mini-market business scale.
- Moderate traffic (demo project, not a system serving millions of users).
- Utilize **the AWS Free Tier** for the first 12 months for EC2, RDS, S3, CloudFront, and ElastiCache.
- Enterprise/Business support plan costs are not included (only Developer/Basic plans are used).

5 TEAM

Partner Project Team (Mambo – FPTU)

Name	Title	Role	Email / Contact Info
Dương Tuấn Kiệt		Team Lead	

	Backend Engineer (.NET)	Developing **.NET backend**, business logic, APIs	
Hồ Chí Kiệt	Backend Engineer (.NET)	Developing **.NET backend**, business logic, APIs	
Nguyễn Hoàng Gia Huy	Cloud & Backend Engineer	AWS architecture & overall system design	
Lâm Vĩnh Cường	Cloud Engineer	Design & deployment of **AWS infrastructure** (VPC, RDS, EB, CI/CD)	
Nguyễn Đăng Khôi	Frontend Engineer	Design & implementation of **UI/UX**, Razor Views, Bootstrap	

6 Resources & cost estimates

Resources	Responsibility	Rate (USD) / Hour
Solution Architects [1]	AWS Architecture Design	$30/hr
Engineers [3]	Development, testing, operations	$20/hr
Other (Project Manager) [1]	Progress management, reporting	$25/hr

Project Phase	Solution Architects	Engineers	Other (Project Manager)	Total Hours
Assessment & Foundation	8	20	4	32

Core Feature Development	6	60	6	72
Cloud Prep & Refactoring	10	40	4	54
Infrastructure & Go-live	4	16	4	24
Total Hours	28Hr	136 hours	18Hr	182Hr
Total Cost	840$	2720$	450$	4010$

Cost Contribution distribution between Partner, Customer, AWS:

Party	**Contribution (USD)**	**% Contribution of Total**
Customer	0	0
Partner	4,010$ (effort value)	100
AWS	200$ credits (Free Tier)	

7 ACCEPTANCE

Acceptance process (simulated according to SOW standards):

- *After completing each Phase, Mambo will deliver the corresponding Deliverables to the Customer (the simulated mini-market owner and instructor).*
- *The Customer will have **8 business days** to review and test the Deliverables against the defined acceptance criteria (success criteria & scope).*
- *If the Deliverable **meets the requirements**, the Customer will confirm acceptance via email/minutes.*
- *If the Deliverable **does not meet the requirements**, the Customer will respond with a detailed description of the errors or points that are not met. Mambo will:*
    - *Fix the errors/adjust the Deliverable within a reasonable timeframe.*
    - *Resubmit the revised version for Customer review, but the review will only focus on the previously identified errors/shortcomings.*
- *If the Customer does not respond within the Acceptance Period, the Deliverable is **deemed accepted**.*