# AWS Microservices Migration Plan

**Date:** 2025-05-18
**Author:** Thanusha Reddy Gaddam

## 1. Executive Summary

A high-level overview of the goals:

- Decompose the existing C# monolith into loosely coupled microservices.
- Deploy each service on managed AWS services.
- Improve scalability, reliability, and deployment velocity.

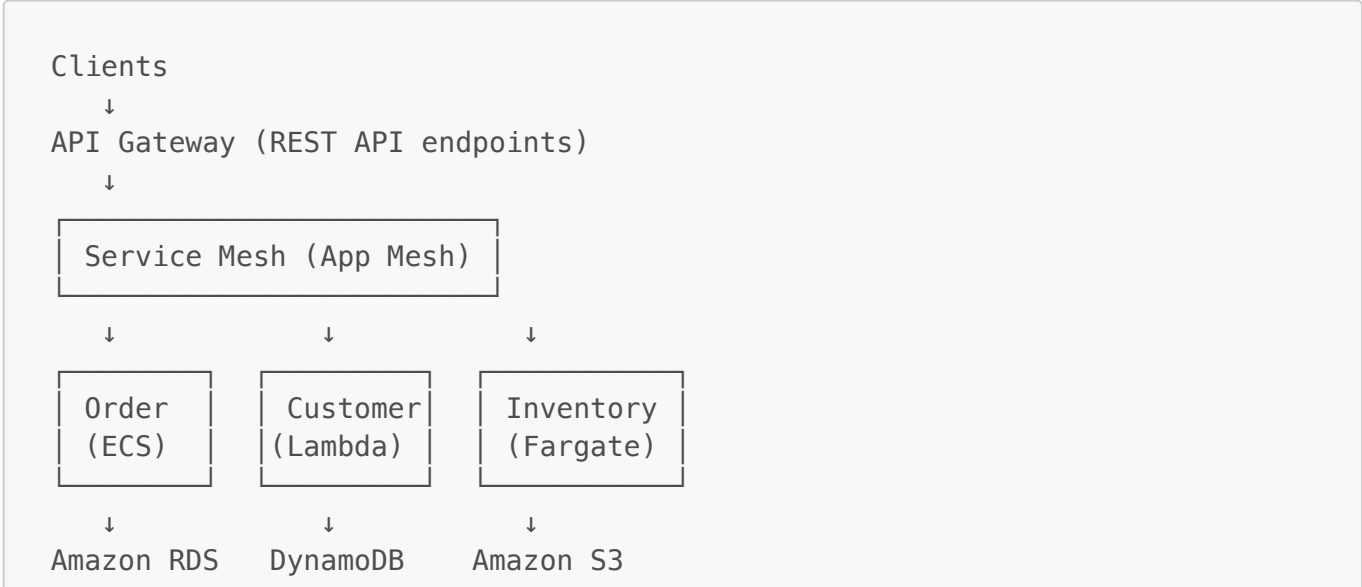
## 2. Current Monolithic Architecture

- Single ASP.NET app hosting business logic, data access, UI
- Single SQL Server database
- Synchronous request handling
- Deployment: on-prem Windows IIS VM

## 3. Migration Strategy

1. **Strangler Pattern**:
   - Introduce new microservices alongside the monolith behind an API Gateway.
   - Gradually redirect specific functionality.
2. **Service Boundaries**:
   - Identify Bounded Contexts (e.g. `Order`, `Customer`, `Inventory`, `Billing`).
3. **Data Migration**:
   - For each service, migrate its subset of tables to its own database (e.g. Amazon RDS / Aurora).
   - Use change-data-capture or dual-writes during cut-over.

## 4. Target AWS Architecture



```
   ↓            ↓           ↓
CloudWatch & X-Ray for observability




AWS Services
    •   API Gateway for external/facet-agnostic routing
    •   Amazon ECS/Fargate or AWS Lambda for containerized or serverless
services
    •   Amazon RDS/Aurora per-service relational DB
    •   DynamoDB for high-volume key-value use cases
    •   Amazon S3 for shared artifacts/uploads
    •   Amazon SQS/SNS for async messaging between services
    •   AWS App Mesh for service-to-service traffic (optional)
    •   CloudWatch & X-Ray for logging, metrics, tracing

5. CI/CD Pipeline
    •   CodeCommit/GitHub as source
    •   CodeBuild for build & unit tests
    •   CodeDeploy or CodePipeline for blue/green deployments to ECS/EKS
    •   Lambda functions deployed via SAM/Serverless Framework

6. Security & IAM
    •   Cognito / OIDC for end-user AuthN
    •   IAM Roles with least-privilege for ECS tasks, Lambdas, RDS access
    •   AWS WAF at API Gateway

7. Roll-out Plan
    1.  Deploy API Gateway + one microservice (e.g. Customer).
    2.  Verify in staging; switch live traffic for /customer/*.
    3.  Repeat for Order, Inventory, etc.
    4.  Decommission monolith module by module.

8. Monitoring & Roll-back
    •   Use CloudWatch Alarms on error rate, latency
    •   X-Ray service maps to detect performance regressions
    •   Keep old monolith available in a separate target-group for quick
rollback.
```