Requirements Analysis — CI/CD Pipeline with GitHub Actions and Containerized Microservices

Project: CI/CD Pipeline for Microservices-Based Task Management System
Team: Culidiuc Maria, Grecu Narcis, Peteri Maria
Platform: AWS (ECR + EKS + DynamoDB)
Automation: GitHub Actions, Docker, Kubernetes

1. Major Components

The application includes three lightweight microservices:

Component	Description	System Role
Project Service	Manages projects; CRUD operations	Provides project data for tasks and reports.
Task Service	Manages tasks linked to projects	Links tasks to projects; serves as intermediary for reporting.
Report Service	Aggregates project/task data	Collects and summarizes data across services for analytics.

Infrastructure Components:

- **AWS ECR** — Docker image repository.
- **AWS EKS** — Kubernetes cluster for automated deployment and scaling.
- **AWS DynamoDB** — persistent storage for minimal state.
- **GitHub Actions** — orchestrates CI/CD: building, testing, pushing, deploying.

Focus: All components exist to showcase automated build, test, and deployment, not complex business logic.

2. Scope & Components to Implement by Semester End

By semester end, the team will deliver:

- All three microservices
    - Lightweight CRUD functionality sufficient to demonstrate pipeline functionality.
    - Dockerfiles prepared for each service.
- CI/CD Pipelines in GitHub Actions
    - CI: Run unit and integration tests on PRs.
    - Build: Build Docker images for each service.
    - Push: Upload images to AWS ECR automatically.
    - Deploy: Deploy to AWS EKS cluster.
    - Verify: Automated health checks and optional E2E tests.

- AWS Infrastructure Automation
    - Kubernetes deployments, scaling via replicas/HPA.
    - DynamoDB tables for each service.
- Testing Automation
    - Unit tests (JUnit) for service logic.
    - Integration tests for service-to-service calls.
    - Optional automated E2E tests to validate deployments.
- Documentation & Diagrams
    - Architecture diagram showing CI/CD workflow and services.
    - README explaining pipeline and deployment steps.

3. Actors

1. Human Actors
    - Development Team: Responsible for coding microservices, implementing tests.
    - End User / Client: Interacts with the deployed Task Management System (performing CRUD operations and viewing reports).
    - System Administrator / DevOps Engineer: Responsible for managing and maintaining the AWS infrastructure (EKS, ECR, DynamoDB) and the CI/CD environment, configuring Docker and setting up the CI/CD pipeline.

2. System / Technical Actors (Tools & Components)
    1. Application Core
        - Project Service: Manages project data; interacts with the database and other services.
        - Task Service: Manages tasks; links tasks to projects and serves data for reporting.
        - Report Service: Aggregates and summarizes data from Project and Task services.
        - AWS DynamoDB: Provides persistent storage for the microservices.

    2. CI/CD Pipeline
        - GitHub Actions: The central orchestrator that automatically triggers, manages, and executes the CI/CD workflow (tests, build, push, deploy).
        - Docker / Docker Engine: Builds the container images for each microservice.
        - Automated Tests (Unit, Integration, E2E): Validates code quality and deployment correctness during the pipeline run.

    3. Infrastructure & Runtime
        - AWS ECR: Stores the built Docker images.
        - AWS EKS: The cloud platform hosting the Kubernetes cluster.
        - Kubernetes (K8s): The container orchestration system that manages deployments, scaling, and service communication within EKS.

4.Use Cases

1. Application Functionality Use Cases
    - Manage Project (CRUD): Create, read, update, and delete basic project entities using the Project Service.
    - Manage Task (CRUD): Create, read, update, and delete task entities, ensuring they are linked to specific projects via the Task Service.
    - Generate Report: Collect and summarize aggregated data across the Project and Task services via the Report Service.

2. CI/CD Pipeline & Automation Use Cases
    - Run CI Tests: Automatically run Unit/Integration tests on code changes (e.g., PRs).
    - Build Docker Image: Create the Docker container image for each microservice.
    - Push Image to ECR: Upload built images to the AWS ECR registry.
    - Deploy to EKS: Use kubectl commands to apply updated Kubernetes manifests directly to the cluster, ensuring that the new Docker image is deployed across the services.
    - Verify Deployment: Run health checks/E2E tests to confirm operational status.
    - Automate Infrastructure Setup: Define and manage Kubernetes manifests and DynamoDB tables.
    - Scale Microservice: Kubernetes (HPA) manages the number of running service replicas.

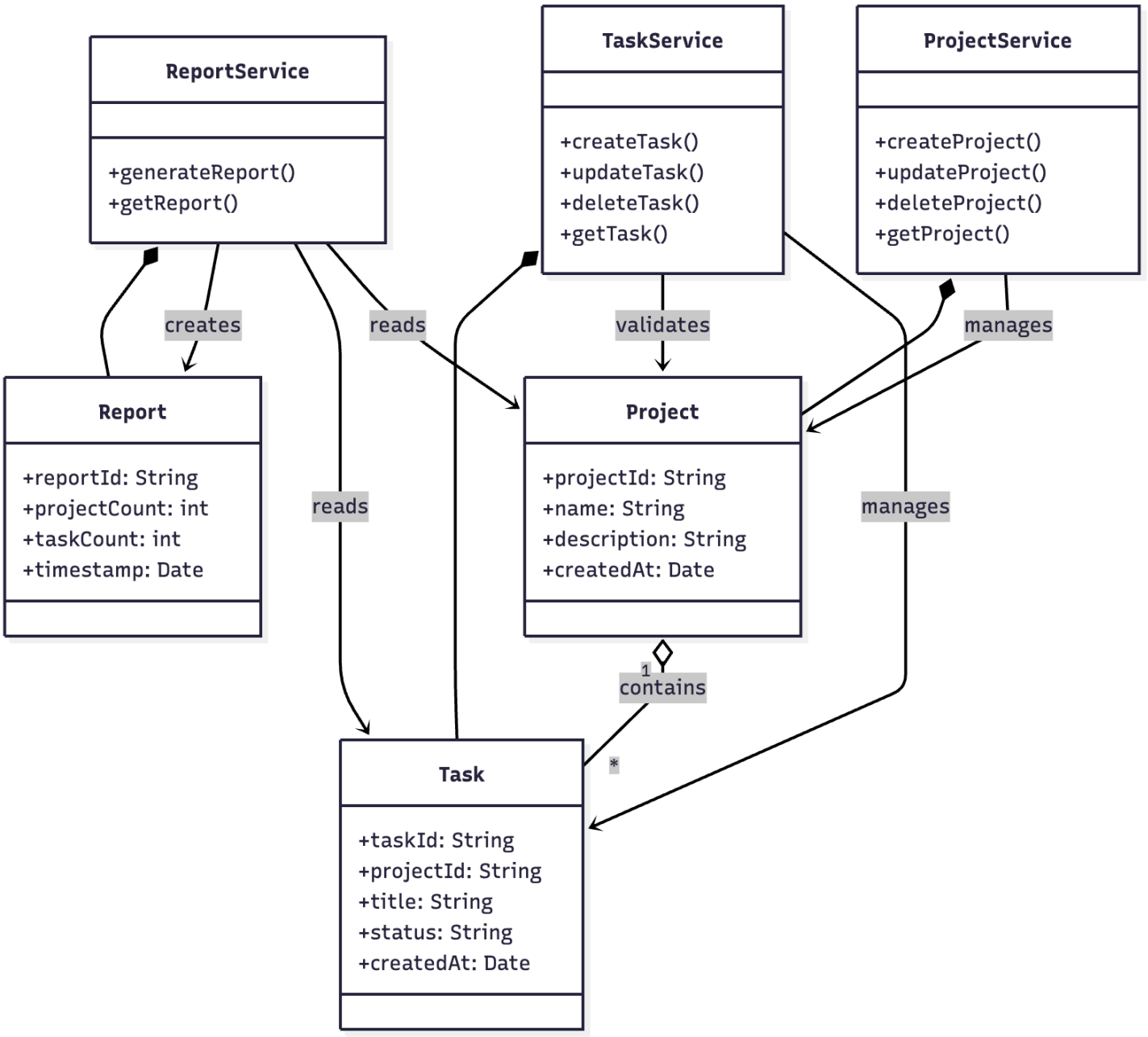
5. Success Criteria (CI/CD Focus)

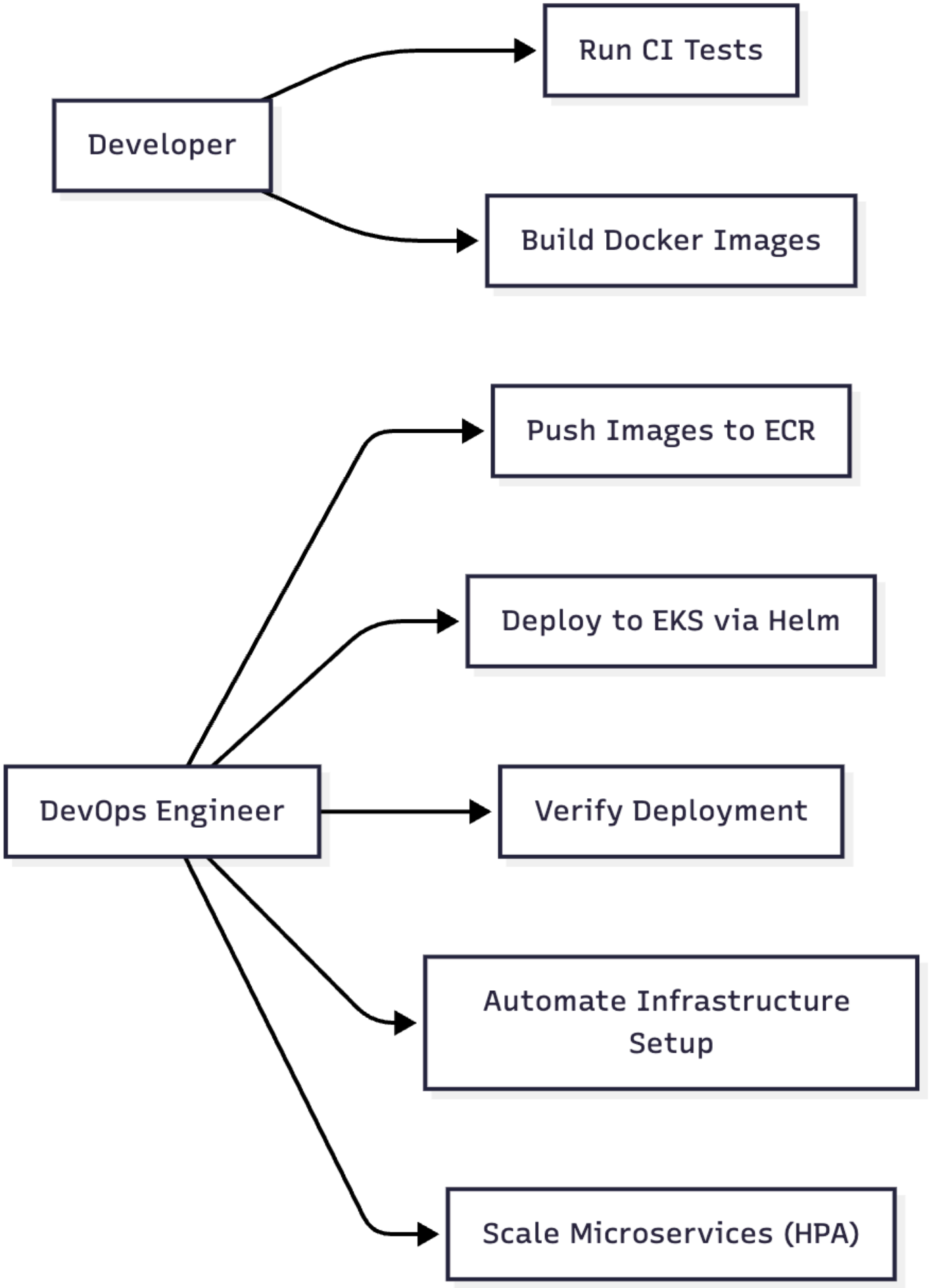
- All three microservices deployed to AWS EKS via GitHub Actions without manual intervention.
- Automated tests executed successfully before deployment.
- Docker images built and pushed automatically to AWS ECR.
- Health checks confirm all services running and communicating.
- Architecture diagram and documentation complete for submission.

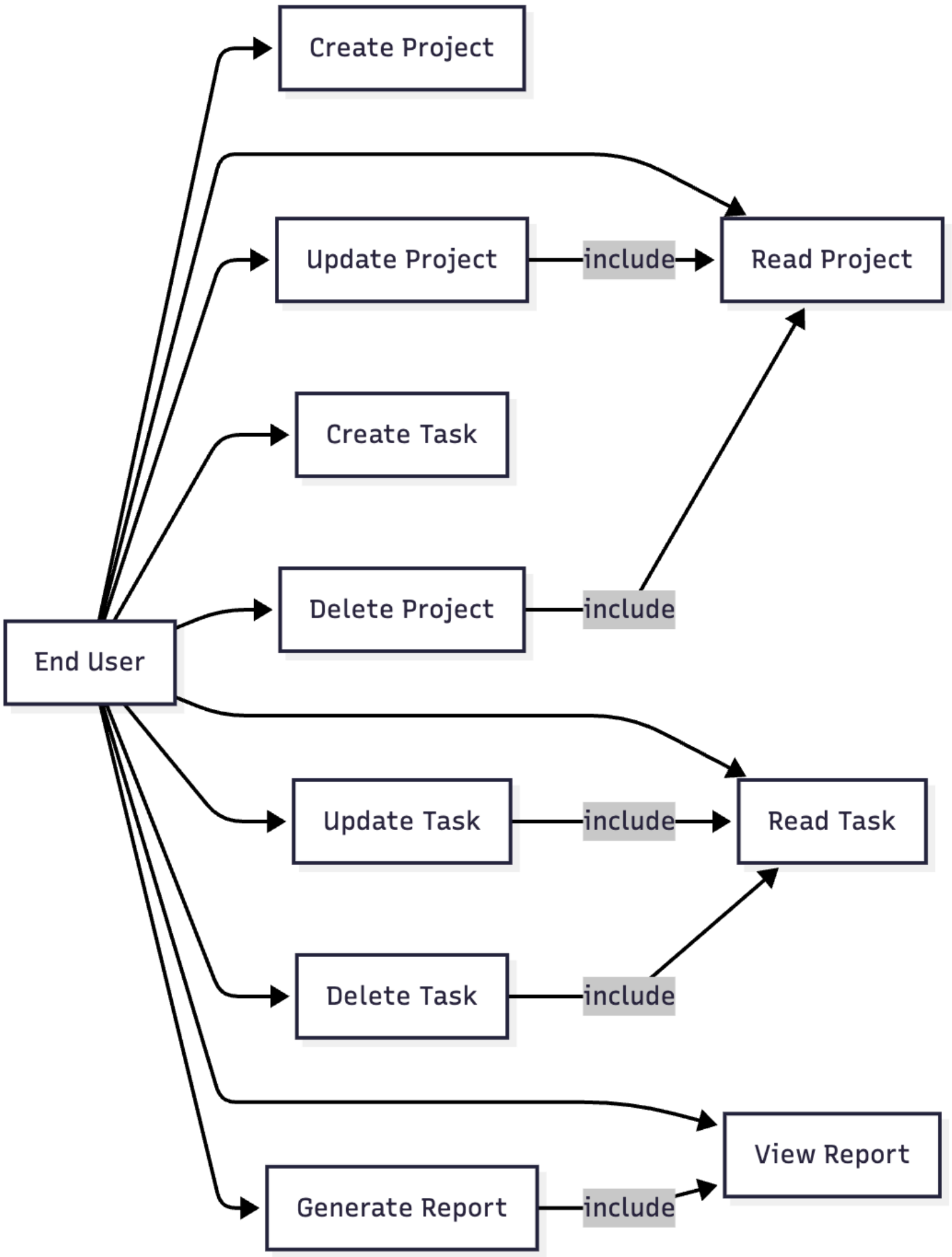
6. Use of AI Tools

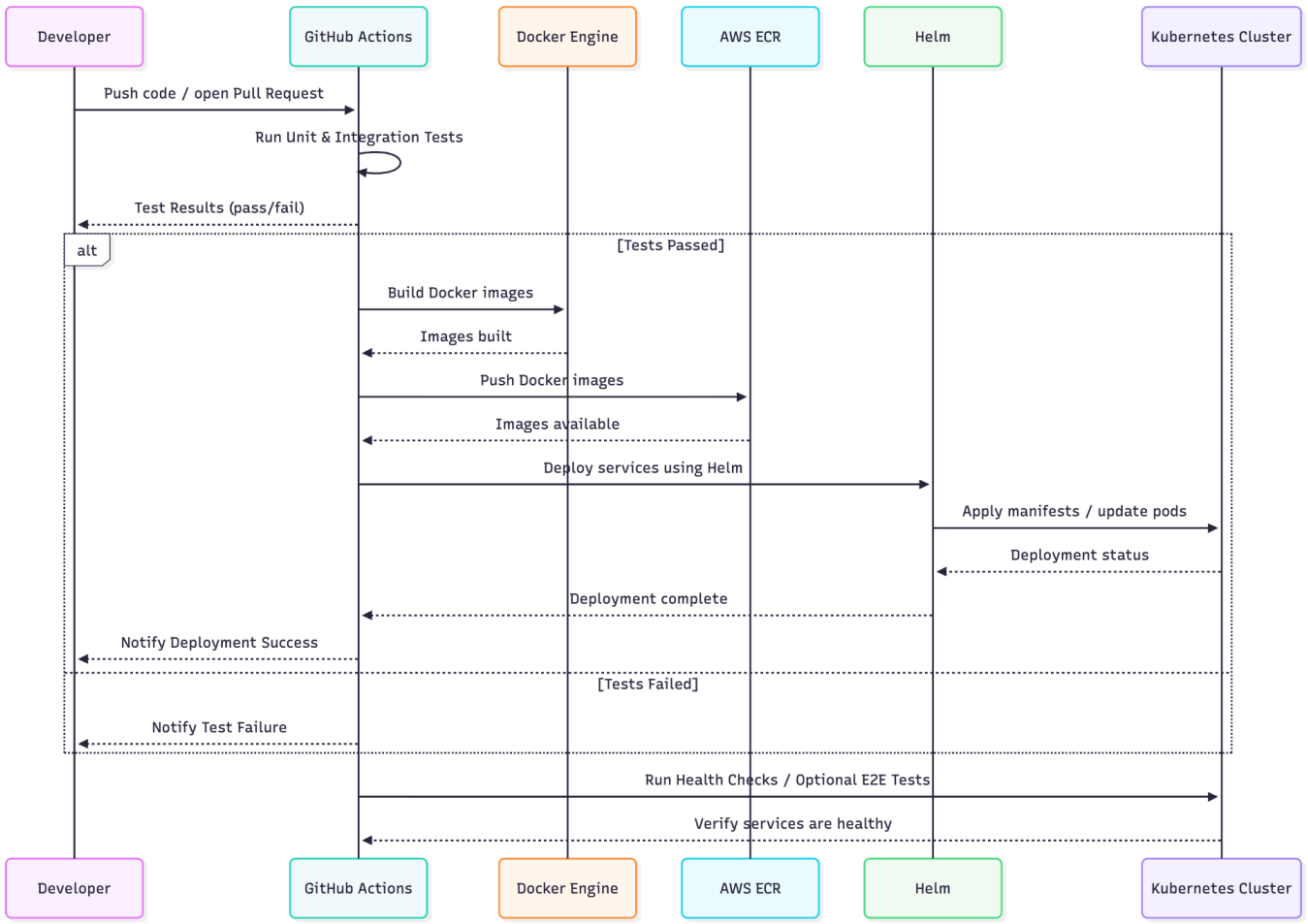
During this project, AI (OpenAI's ChatGPT) was used to:

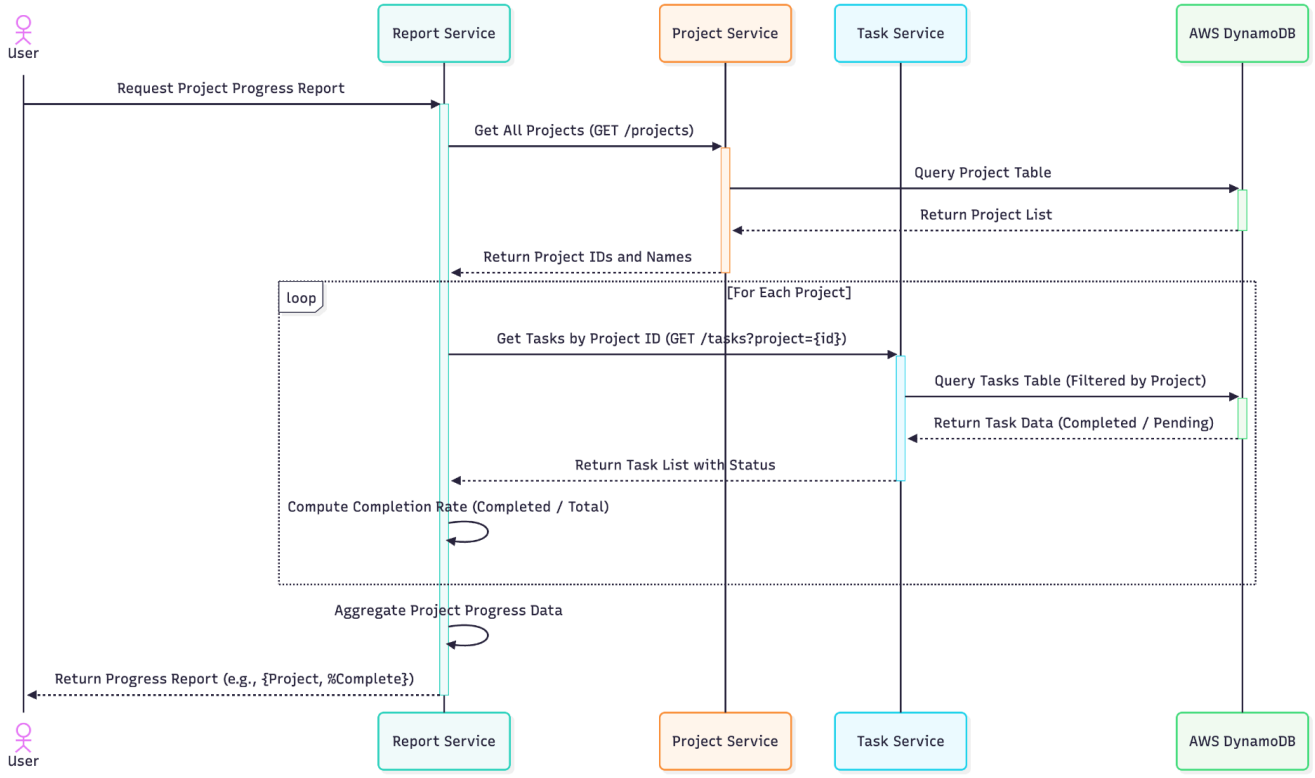
- Explore **deployment options** and selected suitable tools for the CI/CD pipeline.
- Refine **design and documentation**, improving clarity and structure.
- Compare **AWS services, CI/CD strategies, and container orchestration approaches**.

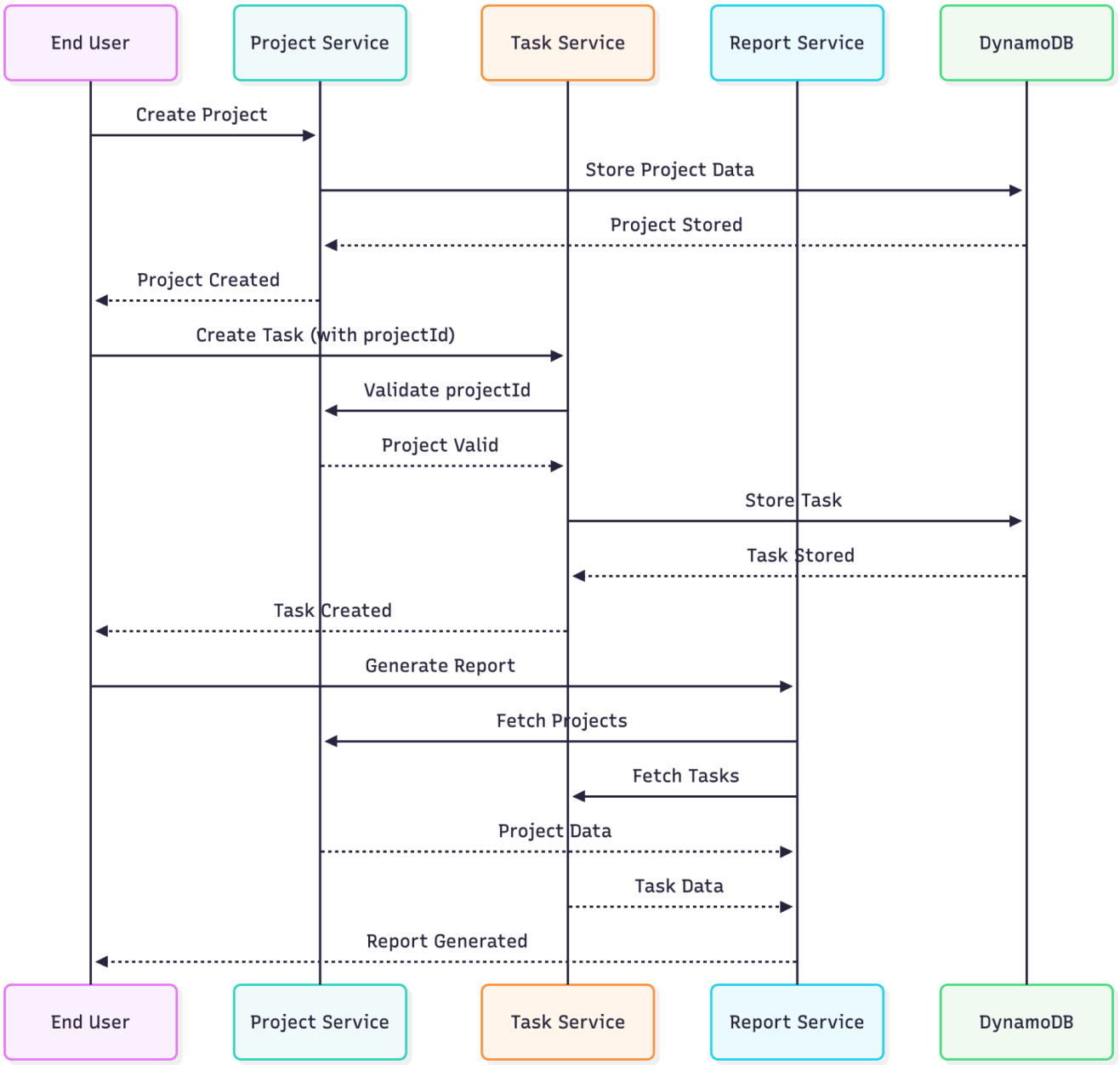

ReportService
+generateReport()
+getReport()
TaskService
+createTask()
+updateTask()
+deleteTask()
+getTask()
ProjectService
+createProject()
+updateProject()
+deleteProject()
+getProject()
creates
reads
validates
manages
Report
+reportId: String
+projectCount: int
+taskCount: int
+timestamp: Date
reads
Project
+projectId: String
+name: String
+description: String
+createdAt: Date
manages
1
contains
*
Task
+taskId: String
+projectId: String
+title: String
+status: String
+createdAt: Date


Developer
Run CI Tests
Build Docker Images
DevOps Engineer
Push Images to ECR
Deploy to EKS via Helm
Verify Deployment
Automate Infrastructure Setup
Scale Microservices (HPA)


Create Project
Update Project
include
Read Project
Create Task
Delete Project
include
End User
Update Task
include
Read Task
Delete Task
include
View Report
Generate Report
include


Developer
GitHub Actions
Docker Engine
AWS ECR
Helm
Kubernetes Cluster
Push code / open Pull Request
Run Unit & Integration Tests
Test Results (pass/fail)
alt
[Tests Passed]
Build Docker images
Images built
Push Docker images
Images available
Deploy services using Helm
Apply manifests / update pods
Deployment status
Deployment complete
Notify Deployment Success
[Tests Failed]
Notify Test Failure
Run Health Checks / Optional E2E Tests
Verify services are healthy
Developer
GitHub Actions
Docker Engine
AWS ECR
Helm
Kubernetes Cluster


User
Report Service
Project Service
Task Service
AWS DynamoDB
Request Project Progress Report
Get All Projects (GET /projects)
Query Project Table
Return Project List
Return Project IDs and Names
loop
[For Each Project]
Get Tasks by Project ID (GET /tasks?project={id})
Query Tasks Table (Filtered by Project)
Return Task Data (Completed / Pending)
Return Task List with Status
Compute Completion Rate (Completed / Total)
Aggregate Project Progress Data
Return Progress Report (e.g., {Project, %Complete})
User
Report Service
Project Service
Task Service
AWS DynamoDB


End User
Project Service
Task Service
Report Service
DynamoDB
Create Project
Store Project Data
Project Stored
Project Created
Create Task (with projectId)
Validate projectId
Project Valid
Store Task
Task Stored
Task Created
Generate Report
Fetch Projects
Fetch Tasks
Project Data
Task Data
Report Generated
End User
Project Service
Task Service
Report Service
DynamoDB