PlayStudios Leaderboard Solution – Documentation

# Author

Pham Vu Phuong

# Overview

This solution implements a backend service and frontend client for a Leaderboard system. It demonstrates clean architecture with separation of concerns, automated tests, containerization (Docker), and deployment readiness (AWS ECS/Fargate).

# Live Demo

Front end:

<http://playstudios-lbs-1451608053.ap-southeast-2.elb.amazonaws.com:8080/>

Back end with Swagger:

<http://playstudios-lbs-1451608053.ap-southeast-2.elb.amazonaws.com/swagger/index.html>

# Run Locally:

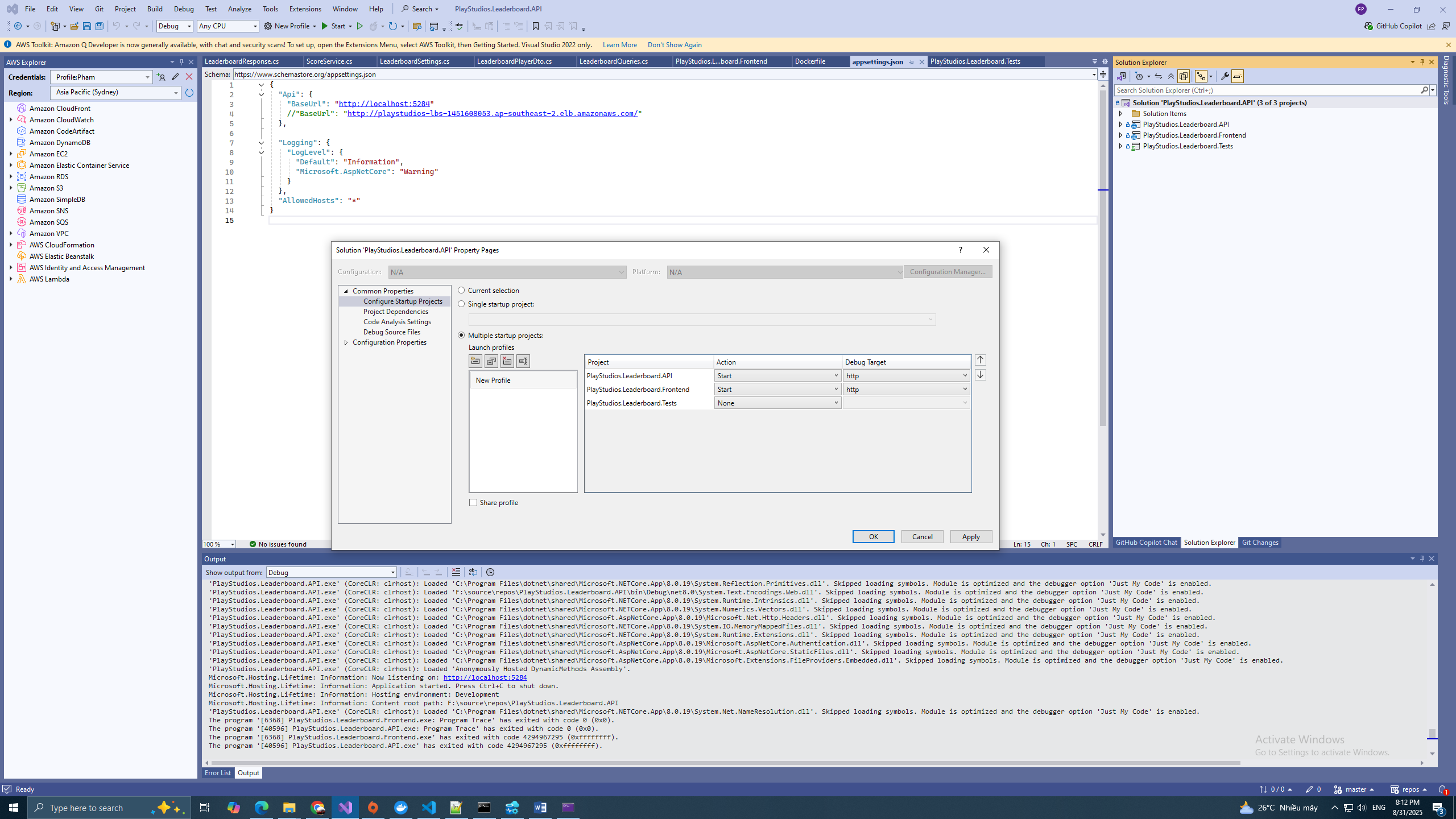
Clone the repo:

<https://github.com/phamvuphuong-bot/PlayStudios.Leaderboard.API>

This solution used:

* ASP.NET Core 8 backend Web API
* Blazor frontend WebApp for interacting with the API.
* xUnit automated test project with Moq + EF Core SQLite in-memory.
* Docker containerization and AWS deployment readiness.

Configure the Startup Projects as follow:



Run Project as normal.

# SQL Server:

I used SQL Server as database. I also created the database on Amazon RDS already. If you need to connect to the database - here are the parameters for the connection:

**SQL Server Management Studio:**

Server: database-1.c1mykki2s2zs.ap-southeast-2.rds.amazonaws.com,1433

User: admin

Password: IX1jfpyATXEAU6mFpWRU

**Azure Data Studio**

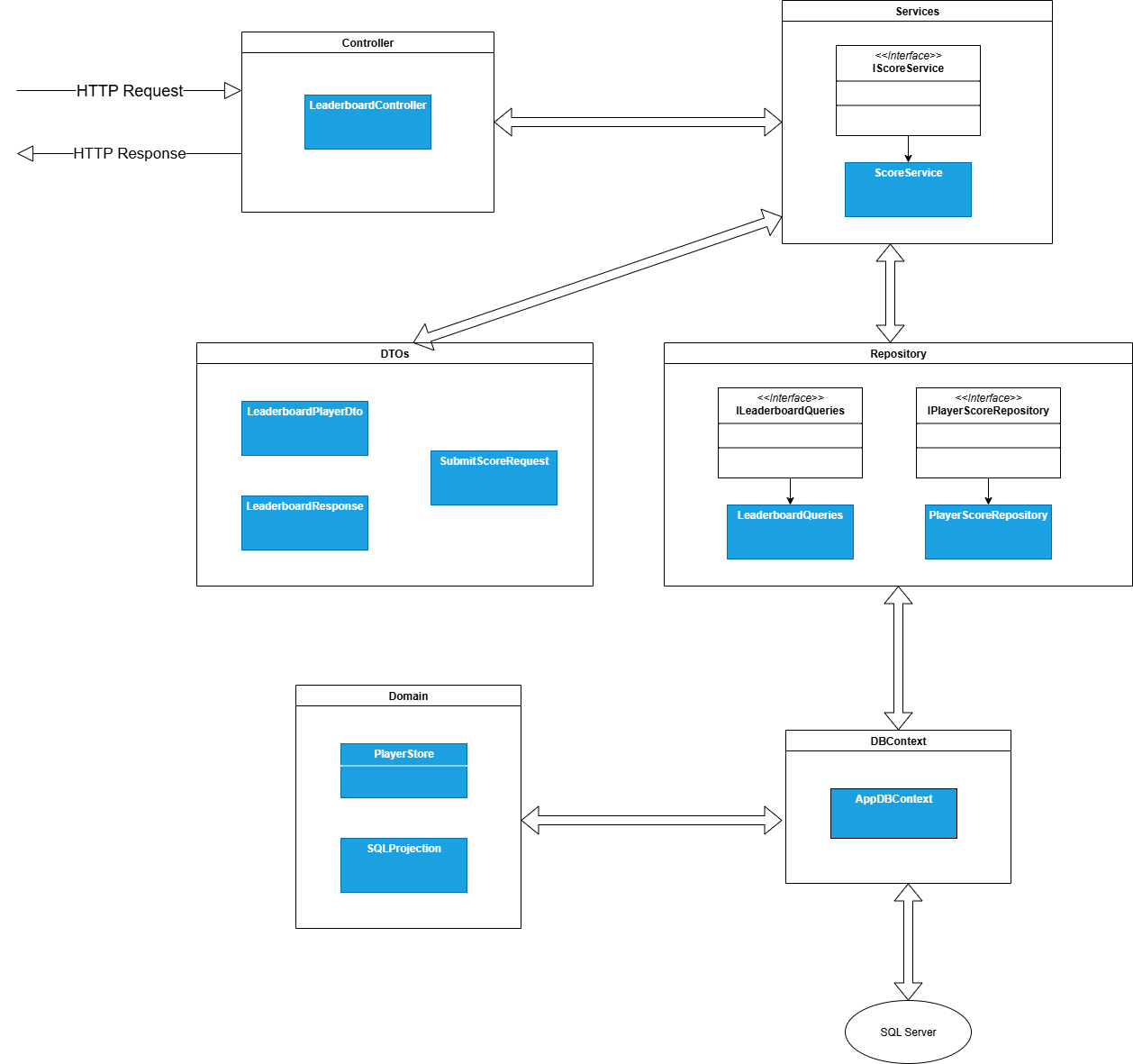
Server: database-1.c1mykki2s2zs.ap-southeast-2.rds.amazonaws.com

User: admin

Password: IX1jfpyATXEAU6mFpWRU

# Projects

## PlayStudios.Leaderboard.Api



The core backend Web API project, built with micro-service architecture. There are 6 layers: DBContext, Repository, Domain, Services, DTOs and Controller. With this structure, we can change database, change business logic, extend everything with minimum change in code

Key Components:

- **Program.cs**: Configures services (DI, EF Core DbContext, repositories, AutoMapper, health checks) and middleware.  
- **Controllers:**   
 • LeaderboardController: Submit score, get leaderboard, reset.  
 • HealthController: Health check endpoints (liveness/readiness).  
- **Infrastructure:**  
 • AppDbContext: EF Core database context (PlayerScores, projections RankedPlayer/RankAndScore).   
- **Repository:** an abstraction layer centralizes data operations (like CRUD) into a single class, making the code cleaner, easier to maintain, and more testable  
 • LeaderboardQueries: Raw SQL query implementation for rank, top N, nearby window.  
 • PlayerScoreRepository: Write-side persistence logic.  
- **Domain:** It defines the entities, rules, and behaviors that model the real-world problem the software is solving  
 • Entities: PlayerScore, RankedPlayer, RankAndScore. .  
 • Config: LeaderboardSettings (TopLimit, NearbyRange, UpdateMode).  
- **DTOs:** **(Data Transfer Object)** is a simple object used to carry data between layers of an application. It helps shape or limit the data exposed to clients without exposing full domain models.  
 • LeaderboardResponse, LeaderboardPlayerDto, SubmitScoreRequest  
- **Services:** contains reusable business logic that coordinates between controllers, repositories, and other components. It helps keep controllers thin and maintains separation of concerns.  
 • IScoreService / ScoreService: Orchestrates repository and queries, applies business rules.

## 2. PlayStudios.Leaderboard.FrontEnd

Frontend Blazor Web App for interacting with the Leaderboard API. Provides forms for submitting scores, viewing leaderboard (top and nearby players), resetting data, and seeding random test data.

Key Components:

- Pages/Home.razor: Main UI with form inputs, buttons, and leaderboard tables.  
- Services/LeaderboardApiClient: Typed HttpClient wrapper for calling API endpoints.  
- Models: SubmitScoreRequest, LeaderboardResponse, LeaderboardPlayerDto.  
- Program.cs: Configures Blazor server interactivity and DI (registers LeaderboardApiClient).

## 3. PlayStudios.Leaderboard.Tests

Automated test project using xUnit and Moq. Ensures reliability and correctness of business logic and repository operations.

Key Components:

- ScoreServiceTests: Unit tests for ScoreService (Replace/Accumulate, Nearby behavior).  
- RepositoryTests: Integration tests with EF Core SQLite in-memory DB for PlayerScoreRepository.  
- Edge case tests (invalid input, ties, reset).

# Configuration JSON – Properties and Purpose

\*\*`appsettings.json` (example)\*\*

{

"ConnectionStrings": {

"SqlServer": "Server=localhost,1433;Database=LeaderboardDb;User Id=sa;Password=<...>;TrustServerCertificate=true"

}

\*\*Explanation\*\*

- `ConnectionStrings:SqlServer` – ADO.NET connection string used by EF Core `AppDbContext`.

All the settings are in LeaderboardSettings.cs

- `LeaderboardSetting.TopLimit` – Max number of entries returned in the Top list.

- ` LeaderboardSetting.NearbyRange` – Number of ranks above/below the user to include in Nearby.

- ` LeaderboardSetting.UpdateMode` – Business rule for score submission:

- `"Replace"`: new score overwrites existing score.

- `"Accumulate"`: new score is added to existing score.

## Data Schema – SQL Server

### Core Table

CREATE TABLE dbo.PlayerScores (

PlayerId NVARCHAR(128) NOT NULL CONSTRAINT PK\_PlayerScores PRIMARY KEY,   
 Score BIGINT NOT NULL CONSTRAINT DF\_PlayerScores\_Score DEFAULT (0),  
 UpdatedAt DATETIME2(3) NOT NULL CONSTRAINT DF\_PlayerScores\_UpdatedAt DEFAULT (SYSUTCDATETIME())

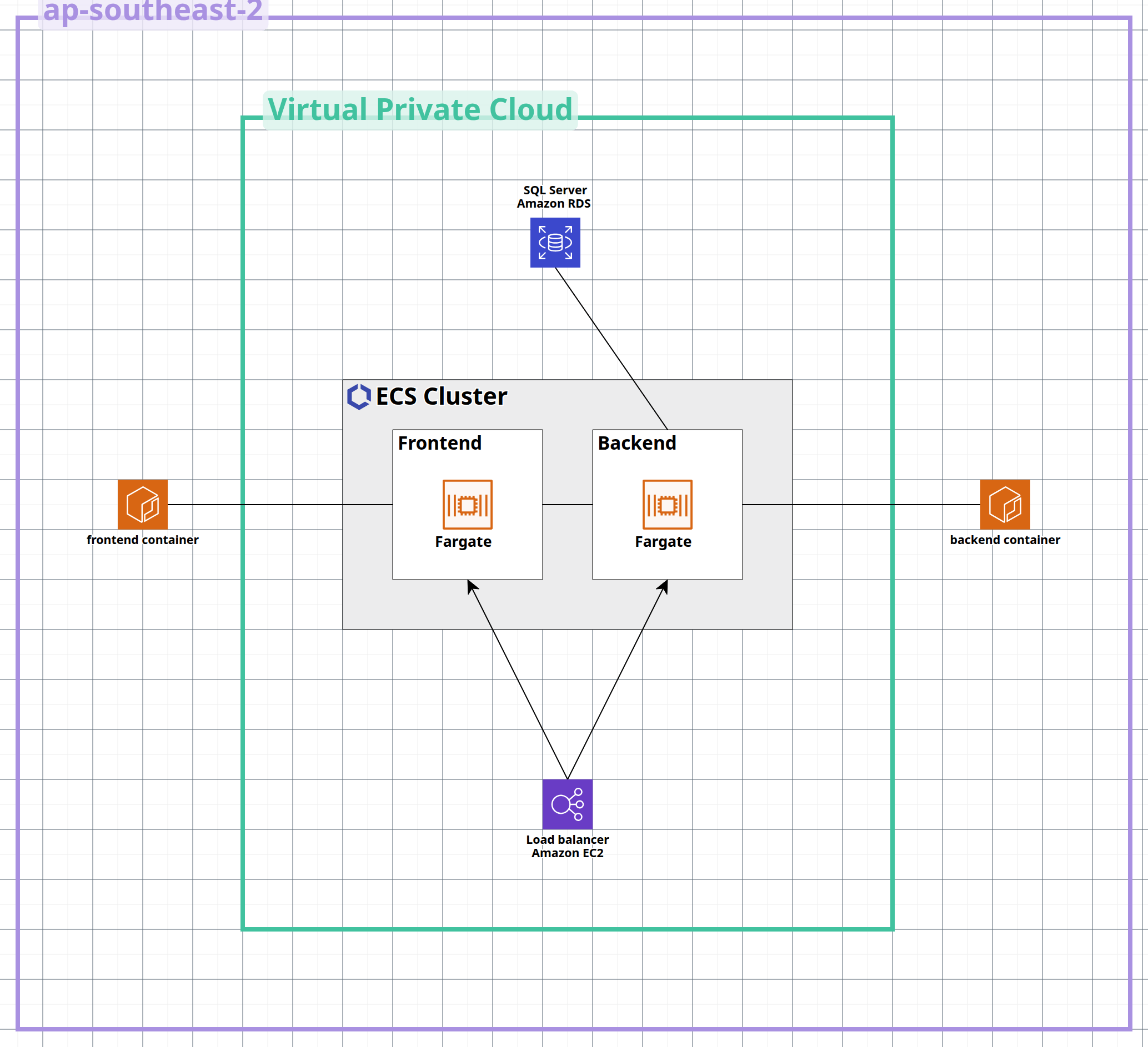
);

- \*\*Primary Key\*\*: `PlayerId` (string id chosen by client).

- \*\*Score\*\*: `BIGINT` (supports large totals).

- \*\*UpdatedAt\*\*: Last modification timestamp in UTC.

# AWS Deployment



You can access the live demo here:   
Front end:

<http://playstudios-lbs-1451608053.ap-southeast-2.elb.amazonaws.com:8080/>

Back end with Swagger:

<http://playstudios-lbs-1451608053.ap-southeast-2.elb.amazonaws.com/swagger/index.html>

Each project includes Dockerfile for building the container. I published the container to ECR, then used ECS Cluster and Fargate to host my projects. I also built the Load Balancer for the future use.

# Summary

This solution demonstrates a cleanly-architected leaderboard system with a well-documented API, frontend Blazor client, and robust test coverage. It is container-ready and deployable on AWS.