Technical Documentation
Anime Trivia Discord Bot

Naveen Prabakar

August 3, 2025

1 Overview

This document describes the architecture, functionalities, and design of the **Anime Trivia Discord Bot**. This bot provides interactive multiple-choice anime quizzes to Discord servers, supports multi-user participation, tracks scores and histories, and allows for language translation of questions.

2 Technology Stack

- **Discord API Wrapper:** discord.py with `discord.ext.commands` and UI components
- **Language Translation:** Custom `Translate` module
- **Question Management:** Custom `questionBank` module
- **Language:** Python
- **Cloud Compute:** AWS EC2
- **Image Storage:** AWS S3
- **Database:** AWS RDS (MySQL)

3 Cloud Deployment Architecture

The Anime Trivia Discord Bot is deployed using Amazon Web Services (AWS) services for scalability, reliability, and managed resource allocation:

- **AWS EC2 (Elastic Compute Cloud):** Hosts the main Python application, running the Discord bot logic and handling real-time interactions and commands.
- **AWS S3 (Simple Storage Service):** All static images (including anime quiz images) are uploaded, retrieved, and stored securely in S3 buckets. This enables efficient, scalable static asset management and offloads binary storage from compute instances.
- **AWS RDS (Relational Database Service):** Core game/user data, scores, and question banks are managed in a scalable, managed database.

The deployment pipeline allows for smooth updates, monitoring, and scaling as user demand grows, while AWS services ensure redundancy, data durability, and security.

4 Core Features

4.1 Interactive Quiz Game

- **Anime Selection:** Users pick an anime from a preset list via the `!anime [name]` command.
- **Joining the Game:** Users can join the game session in their server using interactive Discord buttons.
- **Question Delivery:** Presents questions with multiple-choice options and interactive answer buttons (A/B/C).
- **Score Tracking:** Scores are tracked per-user per-session, with a command to view the current score (`!current`).
- **Game Hosting:** Only the host (the user who started the session) can control the game flow (e.g., advance to the next question, change language, end game).
- **Endgame Summary:** Leaderboard is shown at the end of each round.
- **History Command:** Users can view their cumulative historical scores for each anime (`!history [anime]`).

4.2 Language Support

Game supports dynamic translation of questions and choices to a selected language via the `!language [lang]` command (host only).

4.3 User Commands

- `!hello`: Greets the user and offers introductory information.
- `!options`: Lists all available anime quizzes.
- `!more`: Details game instructions and available features.
- `!anime [anime]`: Starts a quiz session for the selected anime.
- `!language [lang]`: Changes language for the session (host only).
- `!current`: Shows the user's score for the ongoing session.
- `!history [anime]`: Displays the user's history for a particular anime.
- `!end`: Host ends the session, showing a leaderboard and saving scores.
- Answering: Users answer questions via buttons (A/B/C).

4.4 Multi-Server and Multi-User Support

Server-specific data is isolated, allowing multiple games to run in different servers or channels without interference. All participating users' scores in a server are tracked individually.

5 Architecture Description

5.1 Data Structures

- **server_data (dict):** Maintains per-server session state such as participants, current question, scores, language, and permissions.
- **option (dict):** Static dictionary mapping each anime to the number of available questions.

5.2 Game Flow

1. User invokes `!anime [anime]` to start a session. Host is set as the command invoker.
2. Bot posts instructions and session image (from S3), initializes server/player data.
3. Users click the Join button to enter the game.
4. Host clicks Next to begin. Bot asks a randomly-sequenced question, posting answer buttons.
5. All users select an answer via buttons. Only their first answer is counted for that question.
6. Bot updates scores; users are notified if correct/incorrect.
7. Host progresses through questions with Next.
8. Session can be ended anytime by the host with `!end`, which publishes a leaderboard and resets state.

6 Modularity and Extensibility

- **QuestionBank Module:** Responsible for fetching, randomizing, and storing questions/results; easily extensible for more content or new data backends.
- **Translate Module:** Abstracts translation logic, facilitating integration of external APIs or more languages.
- **Button-based UI:** Interaction logic can be extended for more types of questions or answer formats.
- **Permissions:** Designed with host/moderator control for game integrity.

7 Security and Error Handling

- **Session Isolation:** Server-specific `server_data` prevents cross-server data leakage.
- **Host Verification:** Only hosts can progress the session or change settings/language.
- **Input Validation:** Checks for game state and user permissions before taking actions.
- **Ephemeral Messages:** Private feedback for some responses via ephemeral flag.
- **No Token Exposure:** Discord bot token, database credentials, and S3 keys are managed securely (environment variables/IAM).

8 Example Usage

```
!anime blackclover
# Users react to join. Host presses "Next".
# Question appears, users pick A/B/C.
!current
# Displays your current score in the session.
!end
# Host ends the session, leaderboard shown.

!history bleach
# Returns your historical score for 'bleach' quiz.
```

9 Deployment and Configuration

- Application code is run on an AWS EC2 instance with suitable compute and memory resources.
- All images are managed through AWS S3, minimizing storage requirements and bandwidth on the EC2 instance.
- Core bot data, such as scores and question banks, are persisted in AWS RDS for reliability and low-latency querying.
- Environment variables or AWS IAM roles are used for secure credentials management (Discord token, RDS endpoint, S3 credentials).
- The bot requires network access to Discord's API, S3, and RDS endpoints.
- Regular AMI snapshots and/or database backups are recommended for disaster recovery.

10 Further Improvements

Possible future extensions include:

- Per-user stats dashboard
- Timed questions and speed-based scoring
- Automated retrieval and integration of new anime/questions
- Admin moderation tools and anti-cheating logic
- Persistent user preferences (saved language, color themes)