Draw.IO
Back end structure

by K.Kostenkov for Voodoo

Table of contents

Assignment

Define a basic back-office structure (how player data will be saved / synced with Unity)

Imagine that you have a functional backend running on AWS/Azure/GCloud servers with database of your choice

No budget restrictions

How player data will be saved\synched

Each player data is stored by unique id given to it by database.

Separate table should be used to describe relations of device unique Id and\or andy login provider's Id with player data unique Id.

Binding a device to a new login provider adds a new entry with "login provider id-player data id" relation.

During app launch and login back end should check if it has more fresh save.

Logging in on a new device should forcefully terminate sessions on already online devices to prevent save overwrites.

All communication with the back end is done by REST API calls from the client.

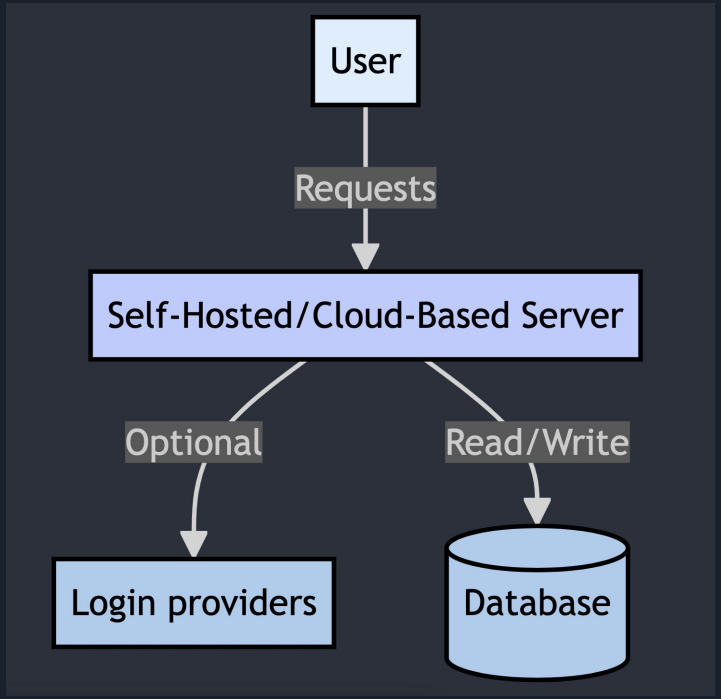
Back end architecture level: Basic

In the very beginning to store players' progress a simple .Net self-hosted application + database would be sufficient.

So the main challenge here goes to choosing the database and making the environment replicable on-demand for different environments.

Custom login providers could be polled to validate user tokens if the functionality is present on the client.

A basic back office could be implemented to download\upload saves manually.



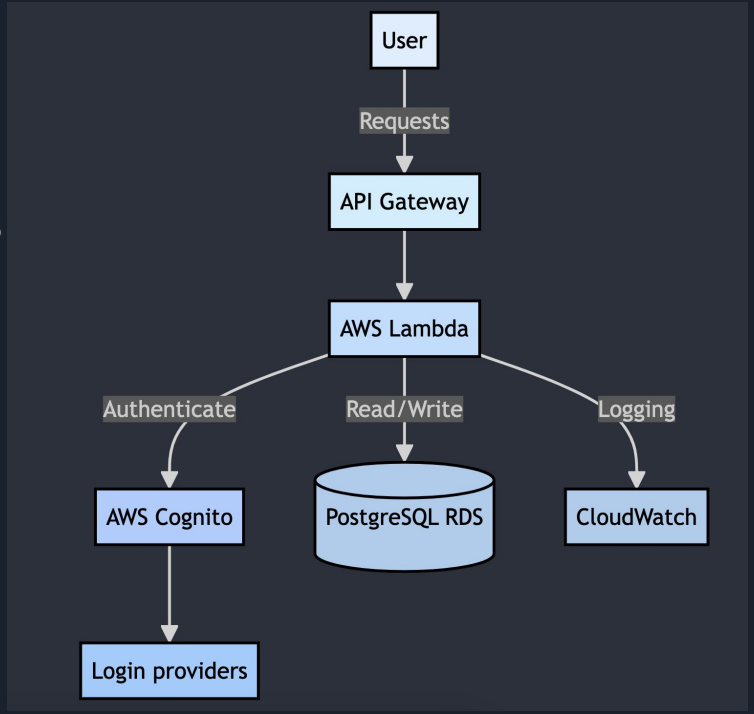
Database selection

Due to specifics of game developments document databases are preferred to handle rapidly changing schemas of player profiles.

Postgress is a relational database that supports working with documents, indexing, could provide JSON validation if needed. It’s proven to be fast and reliable. However some engineers prefer to work with MongoDB.

Both Postgress and MongoDB (variation of if) could be taken from AWS provider and that is important for scaling purposes and multi-stand deployment.

Back end architecture level: Medium



At this moment we start to case about scalability and stability.

Gateway is required to balance workload between scalable AWS Lambdas. Usage of AWS Shield is recommended to be protected from DDoS attacks.

AWS Cognito will make the authentication implementation easier.

Getting logs from multiple instances should not be painful. CloudWatch or DataDog could help with that.

At this stage the project should start considering using infrastructure as code approach to be able to track and easily orchestrate the setup and make the creation of test environments easier (Terraform)

Back end architecture level: Advanced

When the load could be predicted and calculated AWS Lambda should be considered to be changed with self-cloud hosted server instances.

This change will also support the project that could already be considering implementing more complex social features like leader boards, clan wars, chat that most likely will require some caching (ElastiCashe, Redis)

And going into live operations phase with remotely created events, downloadable content will require CDN addition.