RocketGames Account Management System

Author	Lakshmikanth Reddy
Solution Name	Account Management System
Date	26 Jan 2021

Project Summary

RocketGames is looking to become a publisher of third party games. A number of studios have contacted them with potential games, and therefore RocketGames wants produce a cutting edge publishing platform to manage this venture.

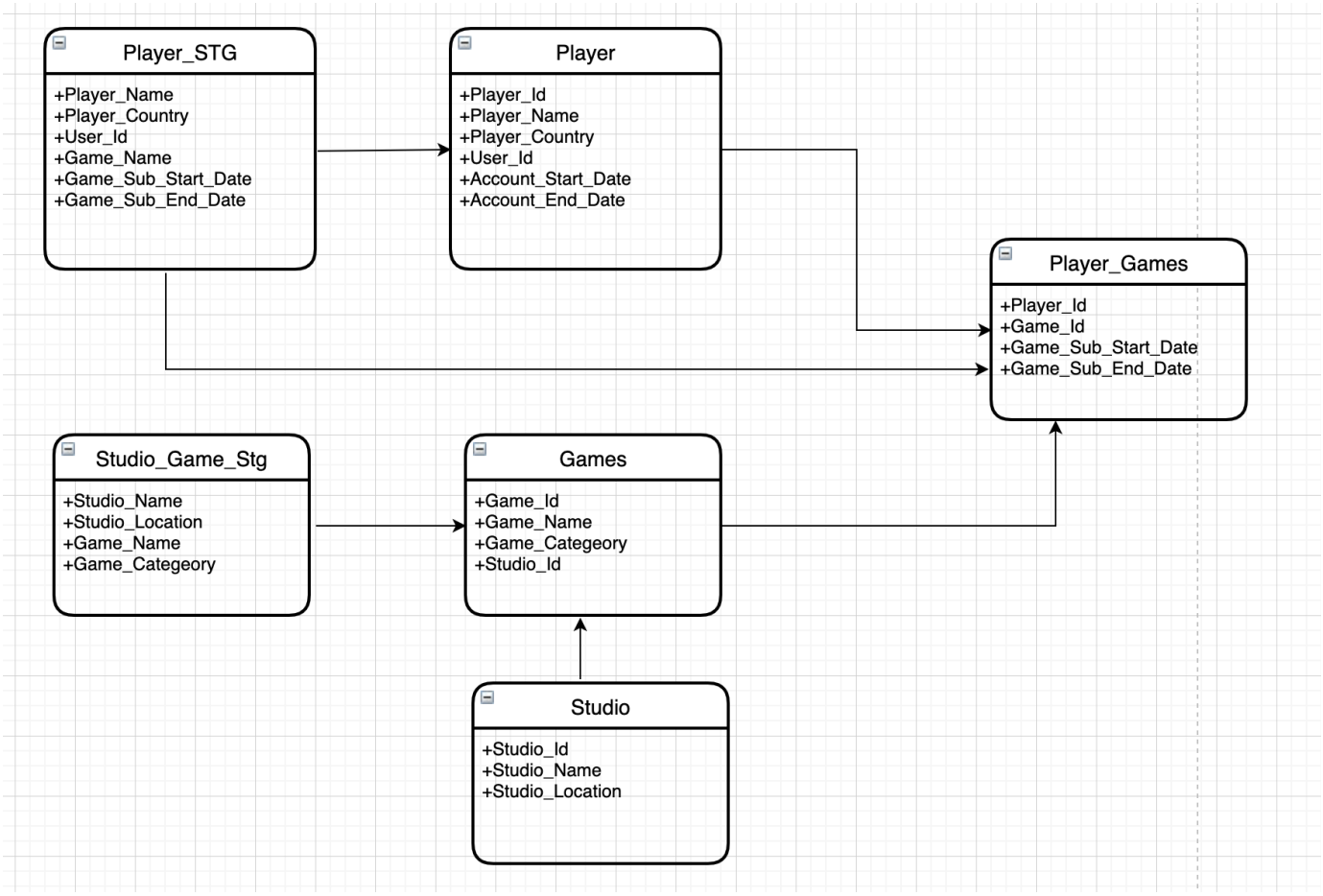
Objective

As part of this project below design components to be produced.

- Design and implement the data models for an account management system
  Design and implement a basic API interface for populating and querying such systems.

Data Model

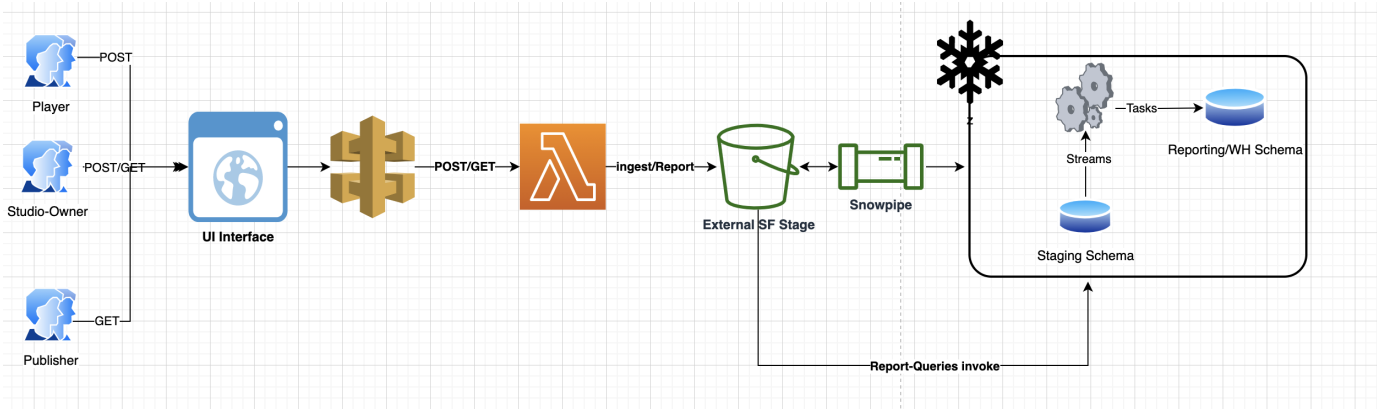
Below is the data model for account management system. This data model is implemented in snowflake cloud data warehouse platform.



High-Level Design Patterns

Considering this project will implement in cloud platform, hence provided various options to achieve this API design to integrate with data model.

Option1:



Above simple design approach implemented using AWS cloud and Snowflake Technical stack. Each design has Pros and Cons.

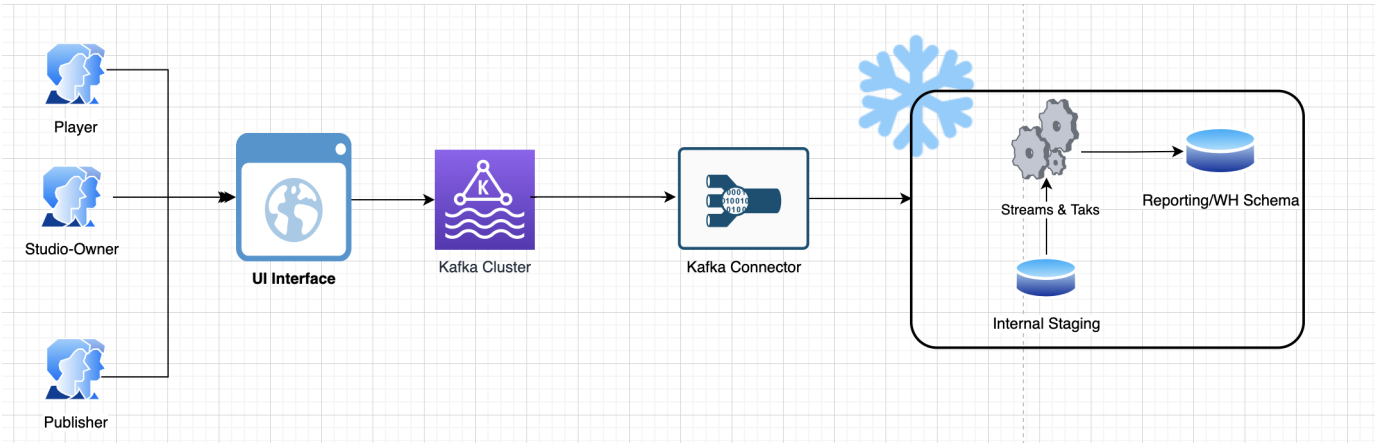
Pros:

- For Account management business case this design is simple and effective w.r.t implementation.
- This is a API and event based architecture architecture and decoupled the AWS and Snowflake architecture.
- As API is designed in server-less pattern there no need hire a server and host the application infrastructure will be taken care by AWS
- As snowflake SaaS model Data Warehouse and very easy to maintain the Data warehouse.

Cons:

- This is a light weight design and unable to scale when request rate is high.
- By default, API Gateway support 10,000 request per second hence will downgrade performance of down stream components such as lambda and DB writing.
- If you receive a large burst of traffic, both API Gateway and Lambda will scale in response to the traffic. However, relational databases typically have limited memory/cpu capacity and will quickly exhaust the total number of connections.

Option2:



In above design option, Kafka cluster and Kafka connector is considered for implementing the account management system. As Kafka cluster is a streaming platform and Kafka connector is integration between Snowflake and able support high volumes of data ingestion.

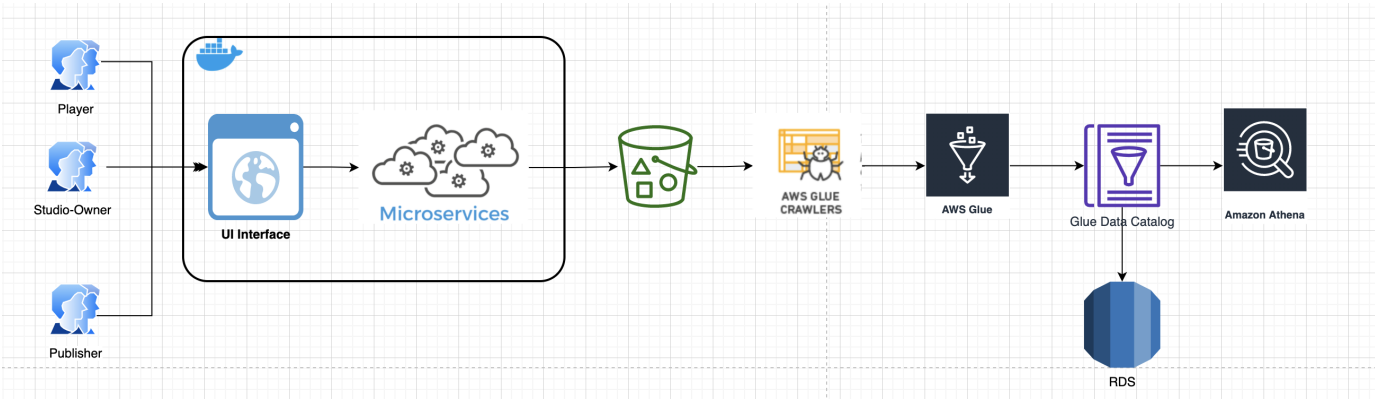
Pros:

- This design support high volumes of request data from incoming user data.
- As Kafka has 7 days of retention policy by default, if there is any down time with down stream application messages will retained and consumer applications can consume once they are up.
- Kafka Cluster can be deployed on any host and available as managed service in AWS and third party support also available like confluent.

Cons:

- Kafka requires external support to deploy, maintain and scale. As this follows cluster based infrastructure need to maintain external servers on cloud or on-premise.
- There will be an additional cost associated to Kafka cluster if we take an external license and also for servers as well.

Option3:



The above design is a container based micro service application. It is responsible to unload the Account management data to S3 bucket. From S3 bucket AWS Glue crawlers are configure to look for schema changes. From there a traditional ETL AWS Glue job extracts the data and store in Glue Data catalog. Form AWS glue can load the data RDS or Redshift based on use case. Analysts also can query the data from Glue Catalog using AWS Athena service. The data and ETL services are hosted on AWS services and a fully managed service.

Pros:

- This use case is designed for high available and scalable purposes.

- Transformations for high volume data can be easily managed in AWS glue using Pyspark or Python in build capabilites.
- Analysts can easily query the data from Data Catalog

Cons:

- AWS glue is server-less and writing the code in for heavy data transformations logics is complex.
- AWS glue service is expensive to run and only able to process batch/micro batch based loads.
- Glue is designed for typical warehouse loads which happen overnight or intraday.

Design Approach

According to requirement, this application maintains Player, Studio and Game details of the Players and studio-owners data. Based on this design has separated into two parts.

1. Storing the Data in DB/warehouse according to data model.
2. An api to get the data from UI/Web and ingest that data into a storage location.

Considering above, to implement this as a light-weight application **OPTION 1** is considered for development/implementation. However, this can implemented in other two options as well based on use case if requirement demands.