# Virtual Fighting Champions

## Esports Companion App

By Ettione C. Stuckey, Jasmine Jiang, Alejandro Otaola, Finn Mikkola, and Lance Boza

# Table of contents

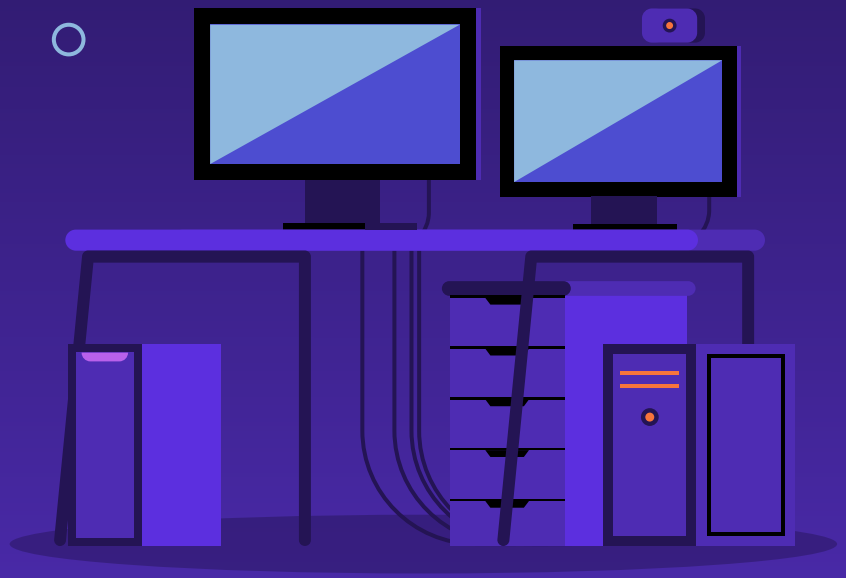
# 01

# About the project

## Virtual Fighting Champions


# What is it?

Virtual Fighting Champions (VFC) is an esports companion app that engages users with esports events by allowing the buying, selling, and exchanging of VFC cards representing real esports players.



# 02

## The Set Up

Let’s get started!

# The Building Blocks

## AWS IAM

Utilized for managing access permissions to DynamoDB data, Lambda functions, and any other resources.

## AWS EC2

A server used to host our API endpoints, as well as the front end to our application.

## AWS VPC

Controls traffic to our resources and protects them from unauthorized access.

# Configuration: AWS IAM

- When starting a project with multiple collaborators, IAM is usually the first resource that comes to mind.
- Here, the team has been given access to only the resources they needed for the duration of the project, enforcing the principle of least privilege.

* The majority of work was a group effort, and so all members typically had the same level of access.*

# Configuration: AWS EC2

```
[Unit]
Description=Gunicorn instance for our app
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/Baseball-Cards
ExecStart=gunicorn -b 0.0.0.0:8000 app:app
Restart=always

[Install]
WantedBy=multi-user.target
```

```
# Default server configuration
#
upstream flaskbaseballcards {
        server 127.0.0.1:8000;
}

server {
        listen 80 default_server;
        listen [::]:80 default_server;
```

Because Flask developmental servers are non-persisting, we used Gunicorn and nginx to host our API endpoints and make them resistant to outages.

# Configuration: AWS EC2 contd.

```
ubuntu@ip-172-31-95-114:~/Baseball-Cards/client$ pm2 start baseball-cards
[PM2] Applying action restartProcessId on app [baseball-cards](ids: [ 0 ])
[PM2] [baseball-cards](0) ✓
[PM2] Process successfully started
```

| id | name | mode | ↺ | status | cpu | memory |
|---|---|---|---|---|---|---|
| 0 | baseball-cards | fork | 2 | online | 0% | 11.8mb |

Next, we used pm2 to create a service to start and run our React frontend whenever the EC2 is booted up in order to ensure that our application is resistant to outages.

# Configuration: AWS VPC

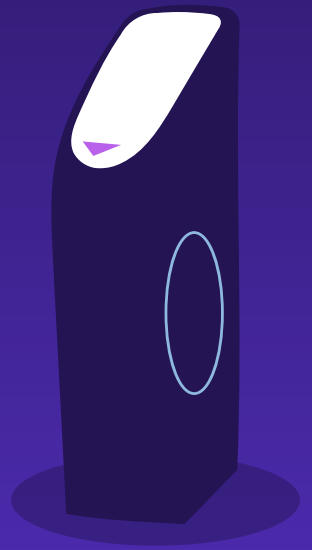
- When setting up our server, it is important to ensure devs and users have access to the necessary ports. This can be done using our VPC and Security Groups. Some are for development, while others are for user traffic. Here are some examples:

- 5000: API Endpoints
- 3000: Front End App
- 3306: MySQL Database
- 80: Requests to API

| Security group rule ID | Port range | Protocol |
|---|---|---|
| sgr-006ec92f25346542f | 5000 | TCP |
| sgr-025e13e91da101b11 | 22 | TCP |
| sgr-089e5465a130e9768 | 443 | TCP |
| sgr-040a95b7937f1ff78 | 3000 | TCP |
| sgr-092d82c39cf4117cb | 8080 | TCP |
| sgr-06f19491b5844e8f6 | 80 | TCP |
| sgr-0d0d7dbfcac0ab994 | 3306 | TCP |

# 03

# Data Collection and Storage



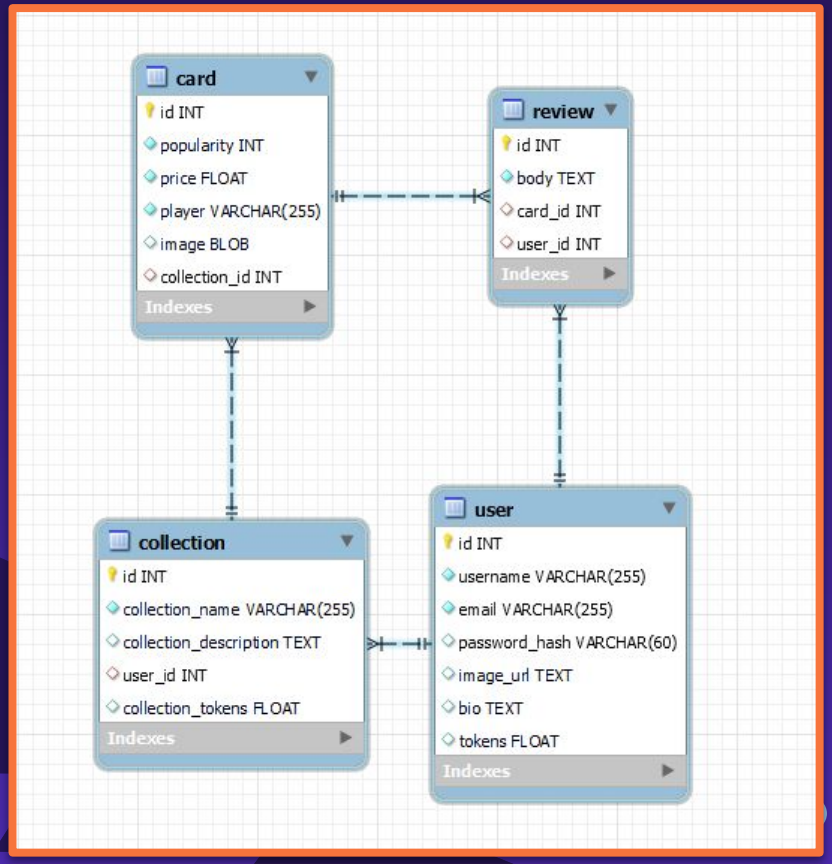
# Data Collection and Storage

## AWS RDS

- Stores data directly related to the application (i.e user data, card data, reviews, etc.)
- Used SQLAlchemy to create the schema and update the database programmatically.


## AWS DynamoDB

- For analytics, we used AWS DynamoDB.
- Stores metadata to track usage of our application.




# The Schema

**Nothing new here! Just an ER diagram to provide a better look at our schema.**

| Name | Status | Partition key |
| --- | --- | --- |
| Card_Statistics | Active | id (N) |
| Collection_statistics | Active | id (N) |
| Trade_Statistics | Active | id (N) |
| User_Statistics | Active | id (N) |

# DynamoDB Tables

**As our application grows, metadata is stored in these tables for future CloudWatch analysis.**

# 04

## Data Processing

## With AWS Lambda!

# Configuration: AWS Lambda

## Logging

Import logger onto the Lambda function to track functionality.

## Validation

Ensure that all fields are present and valid.

## Formatting

Ensure that data is in the correct format for analytics.

## Transfer

Send query results to DynamoDB tables.

# Validation

```python
def transform_card_data(row):
    #ensures data is in the correct format for analytics
    card_id, popularity, price, player = row
    return {
        'id': int(card_id),
        'popularity': int(popularity),
        'price': Decimal(str(round(float(price), 2))),
        'player': str(player)
    }

def transform_user_data(cursor):
    query = """
    SELECT cu.user_id, cu.user_name, COUNT(r.review_id) AS total_reviews, AVG(r.rating) AS average_rating
    FROM card_user cu
    LEFT JOIN reviews r ON cu.user_id = r.reviewee_id
    GROUP BY cu.user_id, cu.user_name;
    """
    cursor.execute(query)
    rows = cursor.fetchall()
    return [{
        'id': row[0],
        'user_name': row[1],
        'total_reviews': int(row[2]),
        'average_rating': Decimal(str(row[3])) if row[3] is not None else None  # Handle None case if no reviews
    } for row in rows]

def transform_collection_data(cursor):
    query = """
    SELECT
        c.collection_id,
        c.user_id,
        cu.user_name,
        COALESCE(SUM(ca.price), 0) AS total_price,
        COALESCE(AVG(ca.popularity), 0) AS avg_popularity
    FROM collection c
    JOIN card_user cu ON c.user_id = cu.user_id
```

# Transfer

```python
def lambda_handler(event, context):
    # TODO implement

    #dynamoDB setup
    dynamodb  = boto3.resource('dynamodb')
    dynamo_table = dynamodb.Table('Card_Statistics')
    users_table = dynamodb.Table('User_Statistics')
    collection_table = dynamodb.Table('Collection_statistics')

    #Connect to the RDS database
    cnx = mysql.connector.connect(
        host='baseball-card-backend-db.c98ugqo62jg6.us-east-1.rds.amazonaws.com',
        user='admin',
        password='password',
        database='baseball_cards'

    )

    #will place a cards id, popularity, price into dynamodb table from rds
    try:
        cursor = cnx.cursor()
        logger.info("Connected to RDS")
        select_query = "SELECT card_id, popularity, price, player FROM cards WHERE popularity >= 90"
        cursor.execute(select_query)

        rows = cursor.fetchall()
        logger.info(f"Fetched {len(rows)} rows")

        for row in rows:
            transformed_card_data = transform_card_data(row)
            if validate_data(transformed_card_data):
                logger.info(f"inserting: {transformed_card_data}")
                response = dynamo_table.put_item(Item=transformed_card_data)
                logger.info(f"DynamoDB Response: {response}")
            else:
                logger.error("Data validation failed")
```

# 05 Analytics

## With CloudWatch!

# Configuration: CloudWatch

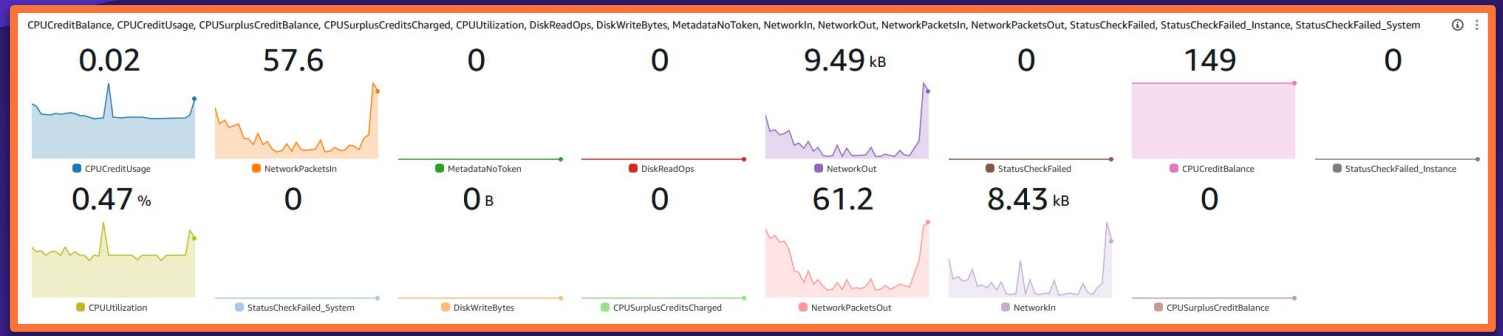
AWS CloudWatch was configured to provide basic monitoring to resources like RDS, EC2, and DynamoDB.

The DynamoDB dashboard will be able to provide more insights as the application grows. But for now, let's take a look at some metrics for our EC2 server and RDS database.

# Configuration: CloudWatch cont.

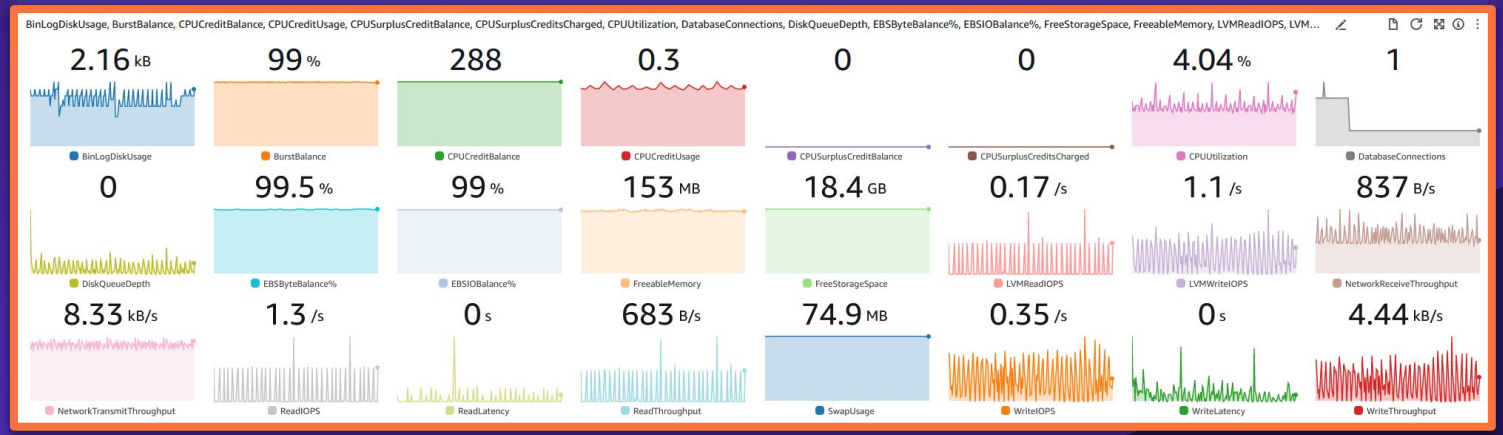
**EC2:**

**ECS:**




**RDS:**

**KDS:**



# 06

## Challenges

We had a lot of them!

# Challenges

01

## Changing Team

Over the course of the project, we lost 2 great members and gained another.

02

## Endpoints

Our API endpoints worked for ‘GET’ requests, but not ‘POST’ requests.

03

## Backend

Originally made by Lance, we eventually refactored the code so that we could work on it without him.

04

## Database Migration

Desyncing of SQLAlchemy with our actually database was caused a lot of confusion.

05

## Deployment

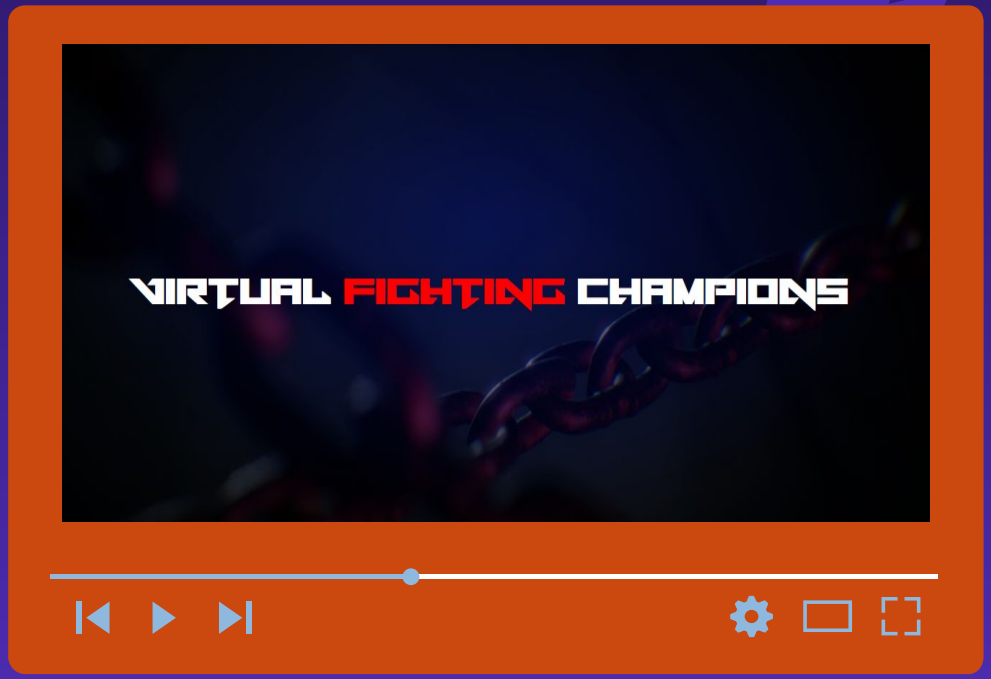
Ensuring the deployed version of the application was the most updated version.

06

## Goal Interpretation

Because we had so many changes in code bases and team members, it was very hard to stay on the same page.

# Questions?




# In the Arms of the Angel


**For real though, thank you Cognixia. I understand the market is tough.**
**Thank you to everyone who helped me learn.**

**To the Instructors and my classmates/coworkers.**
**Wish you all the best <3**

**- Finn**


Celtic Angels


Cognixia
I am inevitable.