Besteam - Servicio de gestión de partidos basado en microservicios

Motivación

Desarrollo de una aplicación en la nube para poder generar partidos con jugadores puntuables de manera balanceada.
Destinados a grupos de amigos que desean hacer un partido competitivo con aleatoriedad en los jugadores.

Objetivos

Sistema de microservicios independientes, reusables y desplegables en un cluster de k8s.
Sistema de mensajería para comunicación entre servicios.
Buenas prácticas en el diseño software como patrones como DTO, principios SOLID, etc.
Domain Driven Design para definir microservicios separando la lógica de negocio.
Test Driven Development para pruebas funcionales unitarias, de integración y e2e.
Trunk Based Development como metodología a seguir para el flujo de desarrollo.
Ejecución automática de procesos de integración y entrega continua en el cluster.
Despliegue de producción en la nube.

Contexto

Dos entornos usados en diferentes namespaces de Kubernetes:

- PRE→ Entorno de pruebas para validar funcionalidades.
  Cada vez que se hace push en la rama principal de los repositorios se despliega.
  Cluster en local con Minikube y contenedores con volúmenes para BBDD.
- PROD → Despliegue automático cuando se hace la validación en PRE.
  Recursos desplegados en AWS.

Flujo de CI/CD con GitHub Actions:

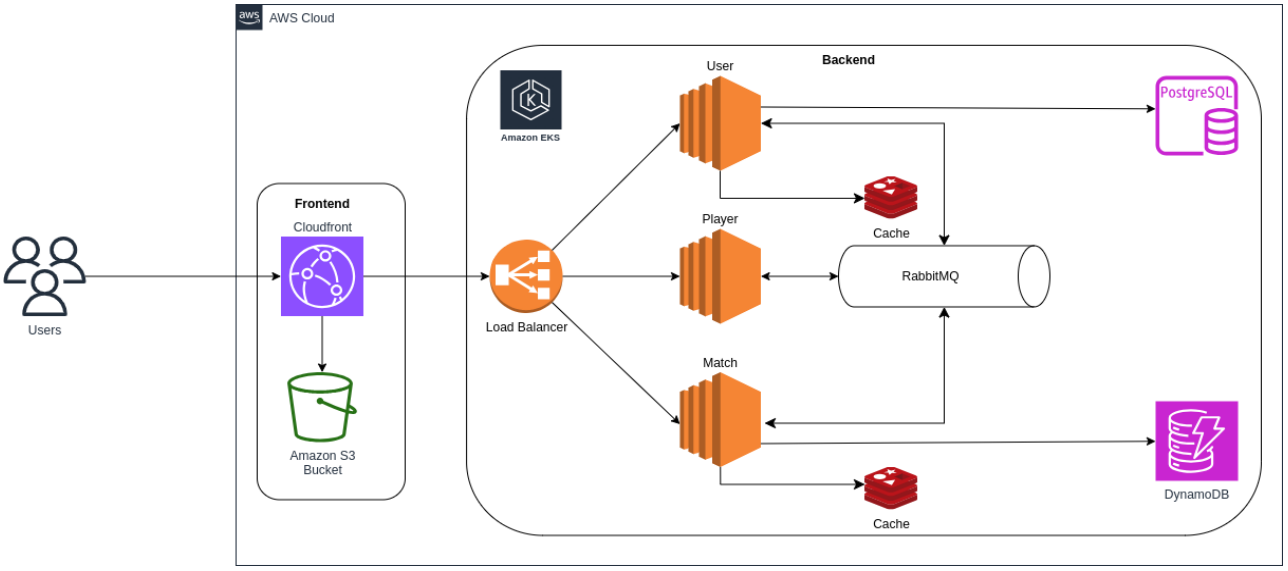
1. pre-commit
   Cada commit lanza una tarea para validar el formato del código y lanzar test unitarios. Si falla no permite realizar el commit.
2. pre-push
   Cada push lanza una tarea con tests de integración. Si los tests fallan no se sube el código al repositorio.
3. on-push
   Cada push exitoso en la rama main, dispara la tarea para generar la imagen docker con la etiqueta dev y se despliega en el cluster usando el namespace de PRE.
4. on-push-tag
   Si se cumple con la funcionalidad deseada se añade el tag y se hace push para generar la release. Se dispara la tarea para comprobar que el tag sea correcto, generar la imagen con la etiqueta de la release y latest, se publican los artefactos en DockerHub, y se despliega la imagen en el cluster EKS usando el namespace de PRO.

Cinco repositorios:

- Servicio de usuario → Permite autenticarse para poder crear/borrar plantillas.
- Servicio de jugadores → Permite añadir/borrar jugadores a las plantillas.
- Servicio de partido → Permite puntuar jugadores y crear equipos.
- Infraestructura → Ficheros CloudFormation para despliegue de recursos AWS.
- Frontend → Ficheros estáticos.

Bases de datos:

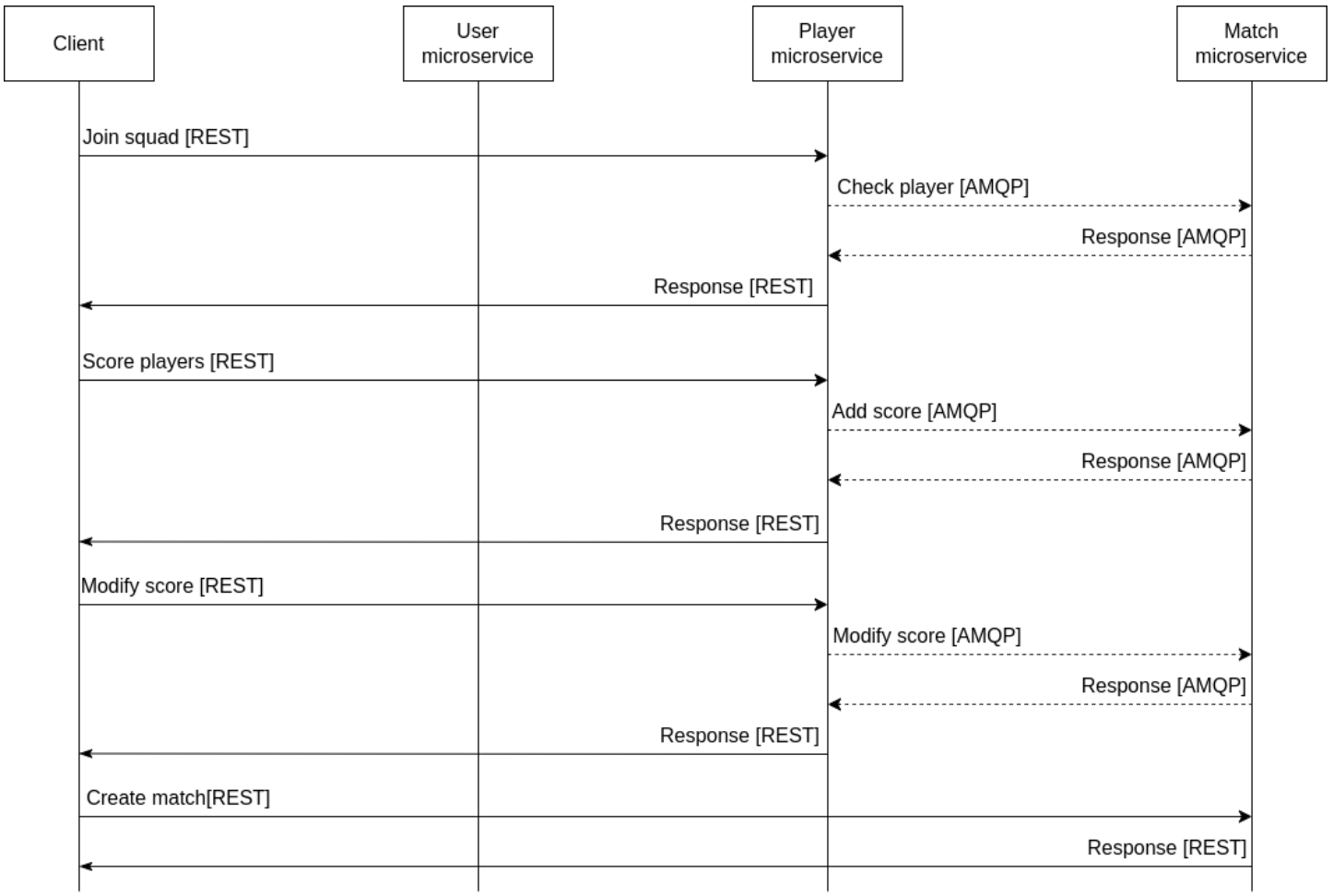
- Una BBDD relacional para usuarios y otra no relacional para la información de plantillas.
- Redis como caché para reducir la latencia de las consultas.



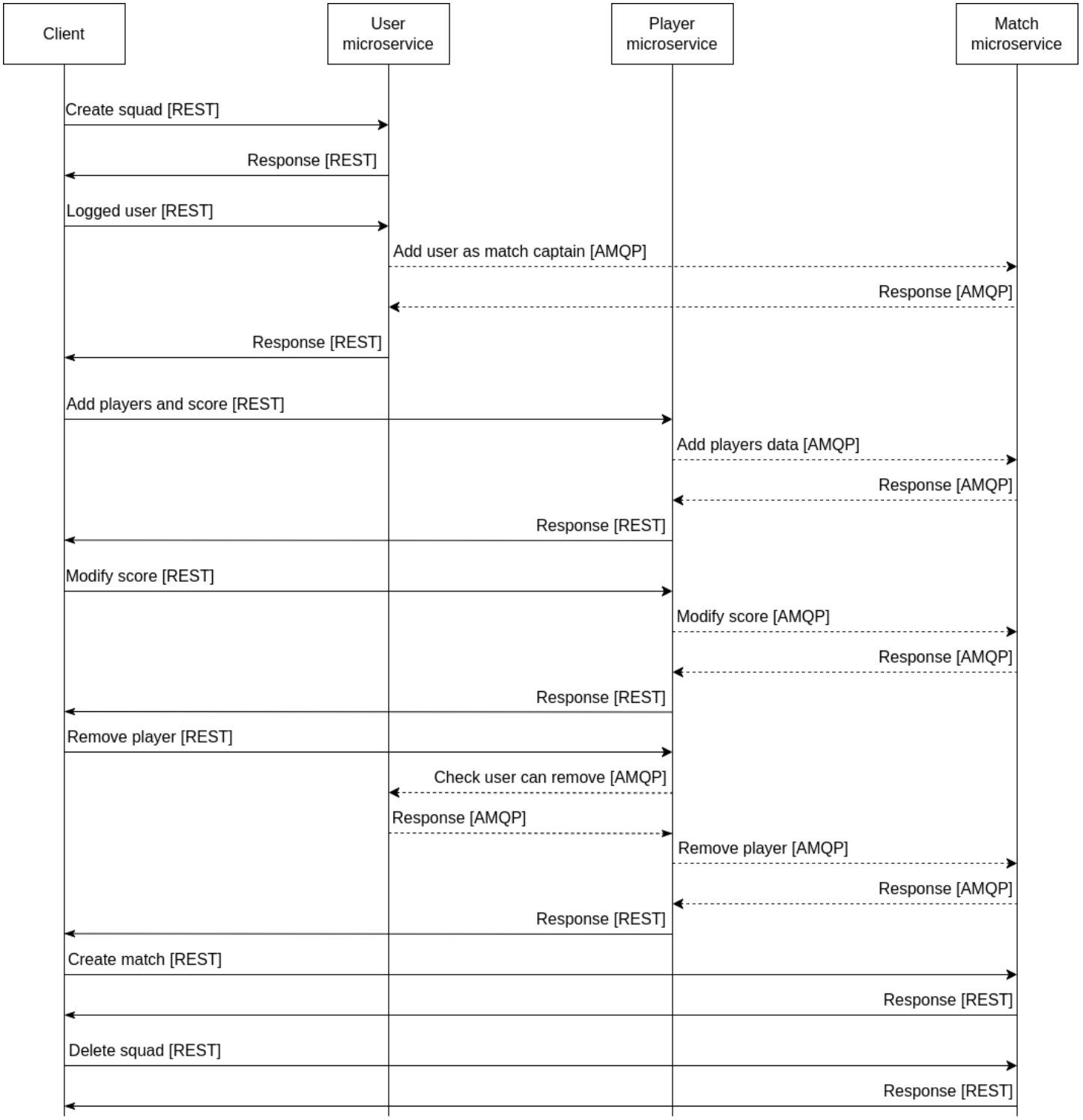
Funcionamiento de la aplicación:

1. Cuando un usuario accede a la aplicación puede elegir entre crear una plantilla debiéndose autenticar o unirse a una ya existente sin necesidad de autenticarse.
2. Un usuario no autenticado solo puede unirse a una plantilla ya existente para puntuar al resto de jugadores y/o formar equipos para un partido.
3. Solo un usuario autenticado puede crear/borrar plantillas y crear/borrar jugadores.
4. Cualquier usuario puede editar la puntuación de su plantilla tantas veces como quiera.
5. Cualquier usuario podrá generar equipos balanceados de su propia plantilla en base a las puntuaciones usando la media ponderada como métrica.

Ejemplo para un usuario no autenticado que se une a un partido:



Ejemplo para un usuario autenticado que puede crear/borrar plantillas y añadir/borrar jugadores.



Pautas de diseño:

- Se utilizará el framework FastAPI de Python para el desarrollo de los servicios.
- Se usará RabbitMQ como servicio de mensajería entre microservicios.
- Se hará uso de distintos motores de BBDD teniendo en cuenta aspectos como la concurrencia.
- Se hará uso de Redis como sistema de cacheo.
- Todos los micros dispondrán de un certificado autofirmado y trabajarán con https.
- Se realizará un enfoque API first para definir el contrato antes de comenzar el desarrollo.
- Todos los micros cumplirán la pirámide de tests, con sus tests unitarios y de integración. Las pruebas se lanzarán de forma automática durante el flujo de CI/CD.
- Los Dockerfile se realizarán haciendo uso de las buenas prácticas. Las aplicaciones implementarán espera de dependencias de otro servicio usando scripts como wait-for-it y otras estrategias.

- Todos los micros cuentan con un script para facilitar el desarrollo en local (dockerize.sh), que permite dockerizar y levantar la aplicación como un contenedor docker con sus dependencias necesarias (docker-compose).
- Se hace uso de charts de Helm para reutilizarlos en el despliegue de los micros en los diferentes clusters de cada entorno.
- Se hará uso de Github como repositorio.
- Se implementará el pipeline de CI/CD con Github actions.
- Se utilizará AWS como proveedor de servicios en la nube.
- Se dispondrá de dos entornos: PRE (pruebas integradas) y PRO (entorno productivo). Para ello se usa Kubernetes como cluster de k8s en la nube con diferentes namespaces. En el entorno de PRE todo serán contenedores con persistent volumes. En PRO se usan instancias en EKS.
- Se harán pruebas de carga con la herramienta Artillery.
- Todos los servicios contienen la documentación necesaria en su repositorio (README) que explica en detalle su estructura y uso.

Planificación

- ☑ ~~W40 Estudio previo~~
- ☑ ~~W41 Estudio previo~~
- ☐ W42 Diseño arquitectura
- ☐ W43 Desarrollo repositorio de infraestructura
- ☐ W44 Desarrollo del microservicio de usuarios
- ☐ W45 Desarrollo del microservicio de jugadores
- ☐ W46 Desarrollo del microservicio de partidos
- ☐ W47 Documentación y entrega
- ☐ W48 Defensa

Estudio previo

- ☑ Análisis despliegue de EoloPlanner en EKS y cluster local.
- ☑ Análisis TFM similar https://github.com/MasterCloudApps-Projects/shopping-infrastructure
- ☐ Desarrollo prueba de concepto con FastAPI https://github.com/manulorente/mca-tfm-pco
- ☐ Migrar EoloPlanner a arquitectura del TFM
- ☐ Desarrollo prueba de concepto serverless
    - EKS para backend y s3 + cloudfront para frontend
    - CloudFormation para desplegar infraestructura
    - Porter para despliegue en AWS?
    - Chart de Helm para despliegue en Kubernetes