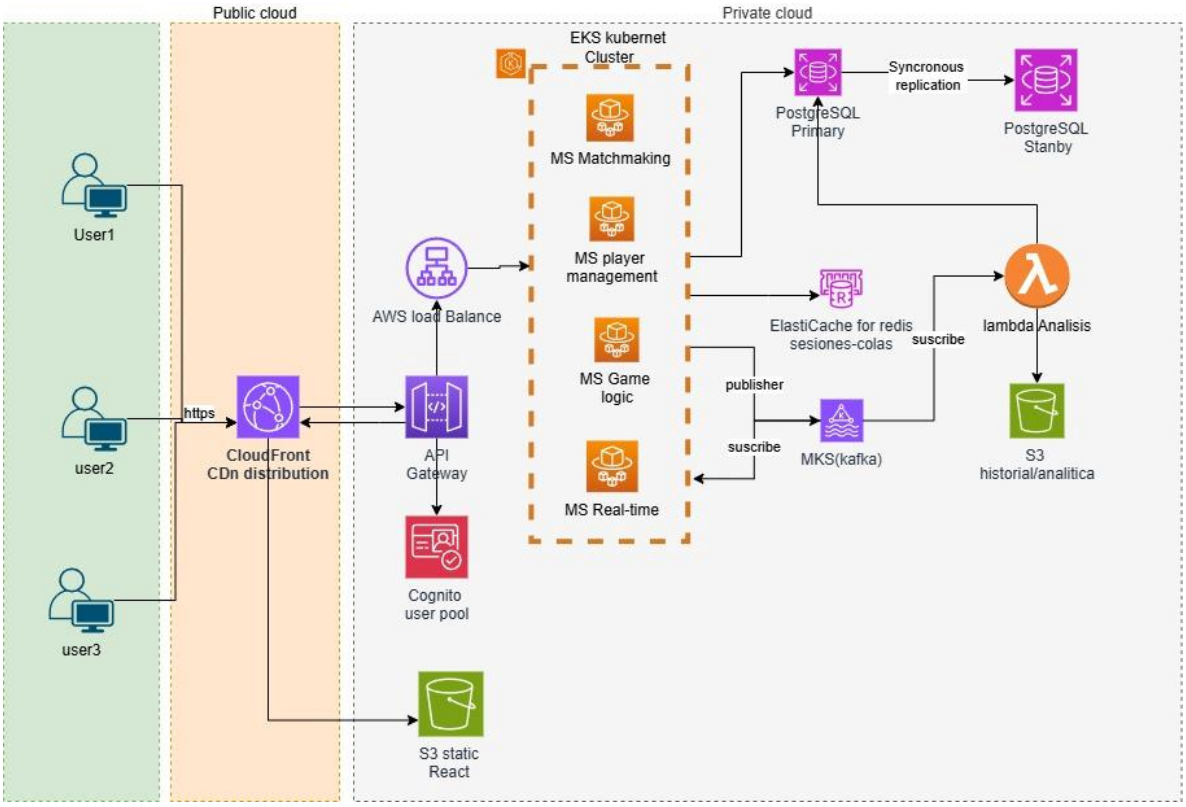
Diagrama arquitectónico



Diseño Técnico

Stack Tecnológico y Justificación

Backend

- **Lenguaje:** Node.js con WebSockets
    - Maneja eventos concurrentes eficientemente. WebSockets es ideal para la comunicación en tiempo real entre el frontend y el backend, ya que permite conexiones bidireccionales persistentes con baja latencia. Se recomienda para actualizaciones en vivo del estado del juego y comunicación entre jugadores.
- **Framework:** NestJS

    - Ofrece una estructura modular con inyección de dependencias, lo que facilita la escalabilidad del proyecto. También tiene soporte nativo para WebSockets con decoradores y módulos bien integrados.

- **Base de datos:** PostgreSQL + Redis
    - PostgreSQL para datos persistentes de jugadores e inventario.
    - Redis para almacenamiento en memoria de sesiones y estado del juego en tiempo real.
- **Mensajería:** Apache Kafka
    - Para distribuir eventos del juego y sincronizar servidores de juego en tiempo real, ejemplo movimientos del juego. Kafka maneja miles de eventos sin sobrecargar los servidores. Si un servicio falla, los eventos quedan en Kafka y se procesan cuando el servicio vuelva a estar disponible. Reduce la cantidad de conexiones directas entre servicios.
- **Infraestructura:** Kubernetes (EKS) + Docker
    - Permite la escalabilidad automática y gestión eficiente de contenedores, distribuye tráfico entre microservicios del juego.

Frontend

- **Framework:** React
    - es eficiente para aplicaciones web, por ejemplo utiliza un Virtual DOM, lo que optimiza la actualización de la interfaz de usuario y mejora la velocidad de renderizado. Compatible con WebGL y bibliotecas como PixiJS para gráficos en juegos en tiempo real.
- **Comunicación:** WebSockets y API REST.
    - WebSockets para interacción en tiempo real, entre el frontend y el backend
    - API REST para comunicación entre microservicios.

Infraestructura en AWS

- **Computación:** Amazon EKS (Kubernetes) para cluster de microservicios y autoescalamiento.

- **Eventos**: AWS Lambda, por ejemplo cuando finaliza una partida, Lambda se activa para actualizar estadísticas del jugador en la base de datos sin afectar el rendimiento del servidor principal.
- **Base de datos:** Amazon RDS (PostgreSQL) con replicas en standby. Si la instancia principal de la base de datos falla, Amazon RDS con Múltiples zonas de disponibilidad automáticamente conmuta a la réplica en standby sin intervención manual. Minimiza el tiempo de inactividad y garantiza la continuidad del servicio.
- **Almacenamiento en memoria:** Amazon ElastiCache (Redis), para estado del juego en tiempo real.
- **Balanceo de carga y API Gateway:** AWS Application Load Balancer + Amazon API Gateway.
- **Bucket** : en Amazon S3 para desplegar el frontend. Se configura un bucket para almacenar los archivos estáticosde la aplicacion (React), se habilita el acceso público restringido.
- **Mensajería en tiempo real:** Amazon MSK (Kafka) para manejar los eventos de juego y distribuirlos entre servicios.
- **CDN:** Amazon CloudFront para distribuir el contenido estático de S3 con baja latencia. Se configuran reglas de caché y distribución geográfica para optimizar el rendimiento.
- **Autenticación:** Amazon Cognito con OAuth 2.0 y JWT.
- **Seguridad:** AWS AWS Web Aplication Firewall (WAF), cifrado con Key Management Service (KMS).

3. Estrategias de Almacenamiento y Gestión de Datos

- **Datos de sesión y estado del juego:** Redis (Alto rendimiento y acceso rápido).
- **Datos persistentes:** PostgreSQL en Amazon RDS con replicación automática.
- **Eventos en tiempo real:** Kafka para distribuir eventos de juego.
- **Historial de partidas y logs:** Amazon S3 con lifecycle management.

4. Mecanismos de Escalabilidad

- **Balanceo de carga:** AWS ALB con Auto Scaling en EKS.

- **Microservicios desacoplados:** Servicios independientes como por ejemplo para gestión de jugadores, lógica de juego, emparejamiento de jugadores y movimientos del juego en tiempo real.
- **CDN:** Amazon CloudFront para servir assets rápidamente.
- **Auto Scaling:** Kubernetes Horizontal Pod Autoscaler (HPA) y AWS Auto Scaling Groups.

5. Tolerancia a Fallos

- **Replicación de bases de datos:** Amazon RDS Multi-AZ con standby replicas.
- **Fallback y reintentos:** Estrategias de reintento en peticiones críticas.
- **Despliegue sin tiempo de inactividad:** Blue/Green deployments en EKS.

6. Medidas de Seguridad

- **Autenticación y autorización:** OAuth 2.0 + JWT con Amazon Cognito.
- **Encriptación de datos:** TLS en tránsito y AES-256 en reposo con AWS Key Management Service (KMS).
- **Protección contra ataques DDoS:** AWS Web Aplication Firewall(WAF).
- **Validaciones y sanitización de datos:** Prevención contra inyecciones SQL, XSS y CSRF.