Архитектура микросервисов для системы космических боев

Проанализировав задание, разработана микросервисная архитектура для системы космических боев с учетом всех требований. Необходимо разделить систему на логические компоненты и определим связи между ними.

Предлагаемая архитектура представляет собой систему взаимосвязанных микросервисов для организации космических боев. Основной принцип - каждый сервис ответственен за конкретную бизнес-функцию и имеет собственное хранилище данных.

Общий поток работы системы:

1. Пользователи регистрируются через User Service и получают доступ к системе.
2. Они могут создавать турниры через Tournament Service или подавать заявки на участие в существующих.
3. Для участия в боях пользователи загружают свои программы через Agent Service, который валидирует их и подготавливает к исполнению.
4. Battle Service организует бои, запуская программы игроков через Agent Service и фиксируя результаты.
5. По результатам боев Rating Service обновляет рейтинги пользователей и турниров.
6. Notification Service информирует пользователей о всех важных событиях.

Особенности обмена сообщениями:

Взаимодействие между сервисами организовано по событийной модели (Event-Driven Architecture) с использованием брокера сообщений (например, Apache Kafka или RabbitMQ). Это обеспечивает слабую связанность сервисов и позволяет асинхронно обрабатывать события.

Масштабируемость:

Для обеспечения высокой доступности и масштабируемости каждый сервис может быть развернут в нескольких экземплярах за балансировщиком нагрузки. Stateless-сервисы легко масштабируются горизонтально, а для stateful-компонентов применяется шардирование данных.

Отказоустойчивость:

В архитектуре предусмотрены механизмы отказоустойчивости:

- Шаблон Circuit Breaker для предотвращения каскадных отказов
- Повторные попытки выполнения операций
- Сохранение состояния в постоянных хранилищах
- Мониторинг и автоматическое восстановление сервисов

Безопасность:

Авторизация и аутентификация централизованно управляются через User Service с использованием JWT-токенов. API Gateway обеспечивает единую точку контроля доступа и защиты от типовых атак.

Данная архитектура обеспечивает возможность независимой разработки, тестирования и развертывания отдельных компонентов системы, что ускоряет выход новой функциональности и упрощает поддержку системы.

Определение набора микросервисов

Предлагается следующий набор микросервисов:

1. **User Service (Микросервис пользователей)**
    - Регистрация и аутентификация пользователей
    - Управление профилями и рейтингами пользователей
2. **Tournament Service (Микросервис турниров)**
    - Создание и управление турнирами
    - Обработка заявок на участие
    - Расчет рейтингов турниров
3. **Battle Service (Микросервис боев)**
    - Организация боев между игроками
    - Обработка результатов боев
    - Запись и воспроизведение истории боев
4. **Agent Service (Микросервис агентов)**
    - Загрузка и управление программами игроков
    - Обеспечение среды выполнения программ
5. **Notification Service (Микросервис уведомлений)**

- Оповещение игроков о различных событиях
- Управление подписками на уведомления

6. **Rating Service (Микросервис рейтингов)**
    - Расчет и обновление рейтинговых очков игроков
    - Агрегирование рейтингов для турниров
7. **API Gateway**
    - Единая точка входа для клиентских приложений
    - Маршрутизация запросов к соответствующим микросервисам
8. **Фронтенд-приложения**
    - Веб-интерфейс для пользователей
    - Приложение "Агент" для выполнения программ игроков

Направления обмена сообщениями и Endpoints

User Service (Микросервис пользователей)

Endpoints:

- `POST /api/users/register` - регистрация нового пользователя
- `POST /api/users/login` - аутентификация пользователя
- `GET /api/users/{id}` - получение данных пользователя
- `PUT /api/users/{id}` - обновление профиля пользователя
- `GET /api/users/{id}/rating` - получение рейтинга пользователя

Обмен сообщениями(топики):

- Публикует: `user.registered, user.updated`
- Потребляет: `rating.updated`

Tournament Service (Микросервис турниров)

Endpoints:

- `POST /api/tournaments` - создание нового турнира
- `GET /api/tournaments` - получение списка турниров
- `GET /api/tournaments/{id}` - детали турнира
- `POST /api/tournaments/{id}/applications` - подача заявки на участие
- `PUT /api/tournaments/{id}/applications/{appId}` - решение по заявке
- `GET /api/tournaments/{id}/results` - результаты турнира

Обмен сообщениями(топики):

- Публикует: `tournament.created`, `tournament.application.received`, `tournament.application.processed`
- Потребляет: `battle.completed`, `rating.updated`

Battle Service (Микросервис боев)

Endpoints:

- `POST /api/battles` - создание боя
- `GET /api/battles/{id}` - получение информации о бое
- `POST /api/battles/{id}/start` - запуск боя
- `GET /api/battles/{id}/recording` - получение записи боя
- `GET /api/users/{id}/battles` - получение истории боев пользователя

Обмен сообщениями(топики):

- Публикует: `battle.scheduled`, `battle.started`, `battle.completed`
- Потребляет: `agent.validated`

Agent Service (Микросервис агентов)

Endpoints:

- `POST /api/agents` - загрузка программы
- `GET /api/agents/{id}` - получение информации о программе
- `PUT /api/agents/{id}` - обновление программы
- `POST /api/agents/{id}/validate` - валидация программы

Обмен сообщениями(топики):

- Публикует: `agent.uploaded`, `agent.validated`
- Потребляет: `battle.started`

Notification Service (Микросервис уведомлений)

Endpoints:

- `GET /api/notifications` - получение уведомлений пользователя
- `PUT /api/notifications/{id}/read` - пометка уведомления как прочитанного
- `POST /api/notifications/settings` - настройка подписок на уведомления

Обмен сообщениями(топики):

- Публикует: `notification.sent`
- Потребляет: `tournament.created`, `tournament.application.processed`, `battle.scheduled`, `battle.completed`

Rating Service (Микросервис рейтингов)

Endpoints:

- `GET /api/ratings/users` - получение рейтинга пользователей
- `GET /api/ratings/tournaments` - получение рейтинга турниров

Обмен сообщениями(топики):

- Публикует: `rating.updated`
- Потребляет: `battle.completed`, `tournament.completed`

Узкие места и проблемы масштабирования

Battle Service (Микросервис боев)

Проблема: Высокая нагрузка при одновременном проведении множества боев

Решение:

- Горизонтальное масштабирование сервиса
- Использование очередей для распределения нагрузки
- Применение шардирования базы данных по турнирам

Agent Service (Микросервис агентов)

Проблема: Ресурсоемкое выполнение программ игроков

Решение:

- Контейнеризация каждой программы (Docker/Kubernetes)
- Ограничение ресурсов для каждого контейнера
- Горизонтальное масштабирование среды выполнения

Tournament Service (Микросервис турниров)

Проблема: Пиковые нагрузки во время популярных турниров

Решение:

- Кэширование данных популярных турниров
- Балансировка нагрузки между экземплярами сервиса
- Асинхронная обработка заявок на участие

Notification Service (Микросервис уведомлений)

Проблема: Массовая рассылка уведомлений участникам больших турниров

Решение:

- Пакетная обработка уведомлений
- Приоритизация уведомлений
- Использование WebSocket для снижения нагрузки на HTTP-запросы

Общая проблема: Задержки в обмене данными между сервисами

Решение:

- Оптимизация схемы обмена сообщениями
- Использование распределенного кэша (Redis)
- Реализация механизма повторных попыток при сбоях

Компоненты с вероятными изменениями требований и OCP

Battle Service (Сервис боев)

Изменения: Механика проведения боев, алгоритмы, правил

OCP решение:

- Стратегии проведения боев через паттерн "Стратегия"
- Плагинная система для правил боя
- Версионирование API для совместимости со старыми агентами

Tournament Service (Сервис турниров)

Изменения: Типы турниров, форматы проведения

OCP решение:

- Абстрактная фабрика для создания различных типов турниров
- Инверсия зависимостей для плагинов форматов
- Системы настраиваемых правил турниров

Rating Service (Сервис рейтингов)

Изменения: Алгоритмы расчета рейтингов

OCP решение:

- Инкапсуляция алгоритма рейтинга в отдельные классы
- Интерфейс для различных стратегий расчета

- Конфигурируемые параметры рейтинговой системы

Agent Service (Сервис агентов)

Изменения: Поддержка новых языков программирования, среды выполнения

OCP решение:

- Абстракция над средой выполнения
- Система подключаемых интерпретаторов/компиляторов
- Версионирование API интеграции