Сервер для многопользовательской игры на .NET Core под Linux

Орлов Юрий

Разработчик C#

28.02.2018

Обо мне

- Разрабатываю серверные приложения на платформе .NET в CUSTIS
- Изучаю балансировку нагрузки с использованием методов для решения NP-трудных задач, пишу статьи в научных изданиях, являюсь инженером-исследователем в ФИЦ ИУ РАН
- Разрабатываю на .NET Core клиент-серверную игру

Почему именно .NET Core?

- Что важно для серверной части многопользовательской игры?
- Что важно при выборе платформы для indie игры?
- Что говорит Microsoft?
- В чем состоит задача?

Что особенно важно для серверной части многопользовательской игры?

- Высокая скорость работы
- Гибкая архитектура
  (легко поддается расширению, масштабированию, изменению архитектуры)
- Низкая стоимость содержания
  (разработка, поддержка, железо)

Что важно при выборе платформы для indie игры?

- Возможности платформы
- Простота использования
- Низкая стоимость разработки
- Низкий порог вхождения

Что нам говорит Microsoft об использовании .NET Core?

Используйте среду .NET Core для создания серверных приложений в следующих случаях:

- для создания кроссплатформенных решений;
- для создания решений, ориентированных на микрослужбы;
- при использовании контейнеров Docker;
- если нужны масштабируемые системы с высокой производительностью;
- для создания приложений с поддержкой разных версий .NET.

Задачи

- Основная логика должна быть реализована на сервере
- Клиентом может стать что угодно: настольное или мобильное приложение, Unity2D, браузер.
- Наличие различных способов аутентификации
- Мультиплеер реализован в виде пошаговых сражений 1 на 1 или 2 на 2, которые хостит сервер. Каждый шаг ограничен по времени
- Каждое игровое сражение имеет ограниченный срок жизни
- В БД хранятся только достижения игроков и некоторые настройки, процесс сражения выборочно логируется

Почему .NET Core?

- Высокая скорость работы и отклика
- Нацеленность платформы на масштабируемость
- Платформа не требует лишних финансовых затрат
- Клиент по своей сути – браузер. Сервисы ASP.NET Core отлично справятся с потребностями клиента
- Поддержка всех необходимых функций из коробки
- Платформа широкоизвестна и довольно проста в использовании – можно привлекать других разработчиков
- Это мой любимый C#

Производительность .NET Core

- Практически в 1,5–2 раза увеличивает скорость работы с коллекциями по сравнению с .NET Framework 4.7, в том числе Concurrent
- В 4–7 раз ускоряет работу LINQ-запросов, сокращая количество memory allocation в 2 раза
- Ускоряет некоторые математические операции
  и многое другое

Подробнее о преимуществах платформы – в .Net Blog

Трудности, с которыми можно столкнуться

- Нет полноценной Community среды разработки для Linux
- Docker работает с Windows, но только, начиная с Professional
- Некоторые классы, типичные для .NET Framework, были переименованы (например, reflection)
- Пока что нет некоторых полезных библиотек. Отдельные библиотеки только недавно оказались в alpha release

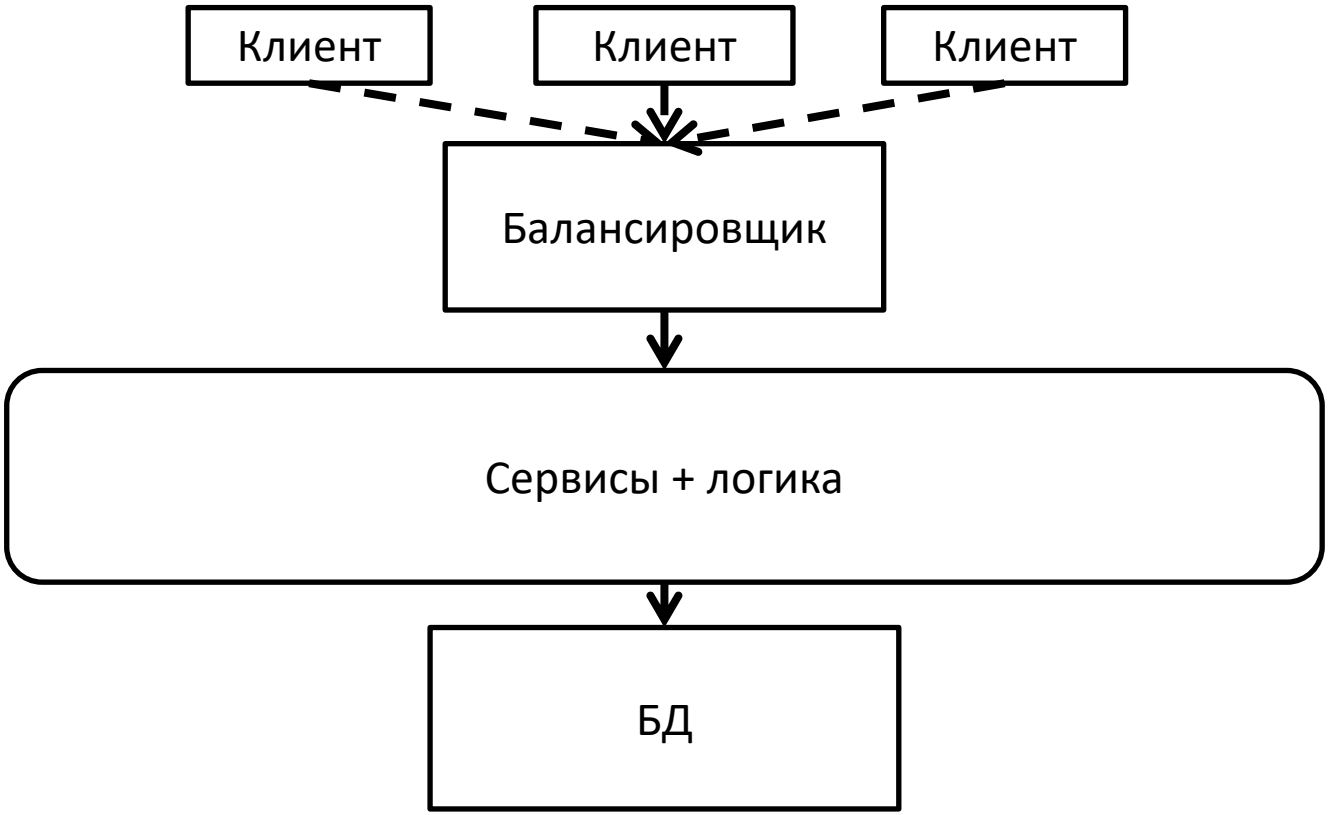
Что еще?

- Возможно, вам пока будет не хватать WCF
- Захотите real time - используете
  1) WebSocket;
  2) SignalR - на свой страх и риск, он пока в alpha release и его очень сильно изменили;
  3) UDP.
- Привыкли работать с NHibernate – пока есть только версия, которая «компилится».

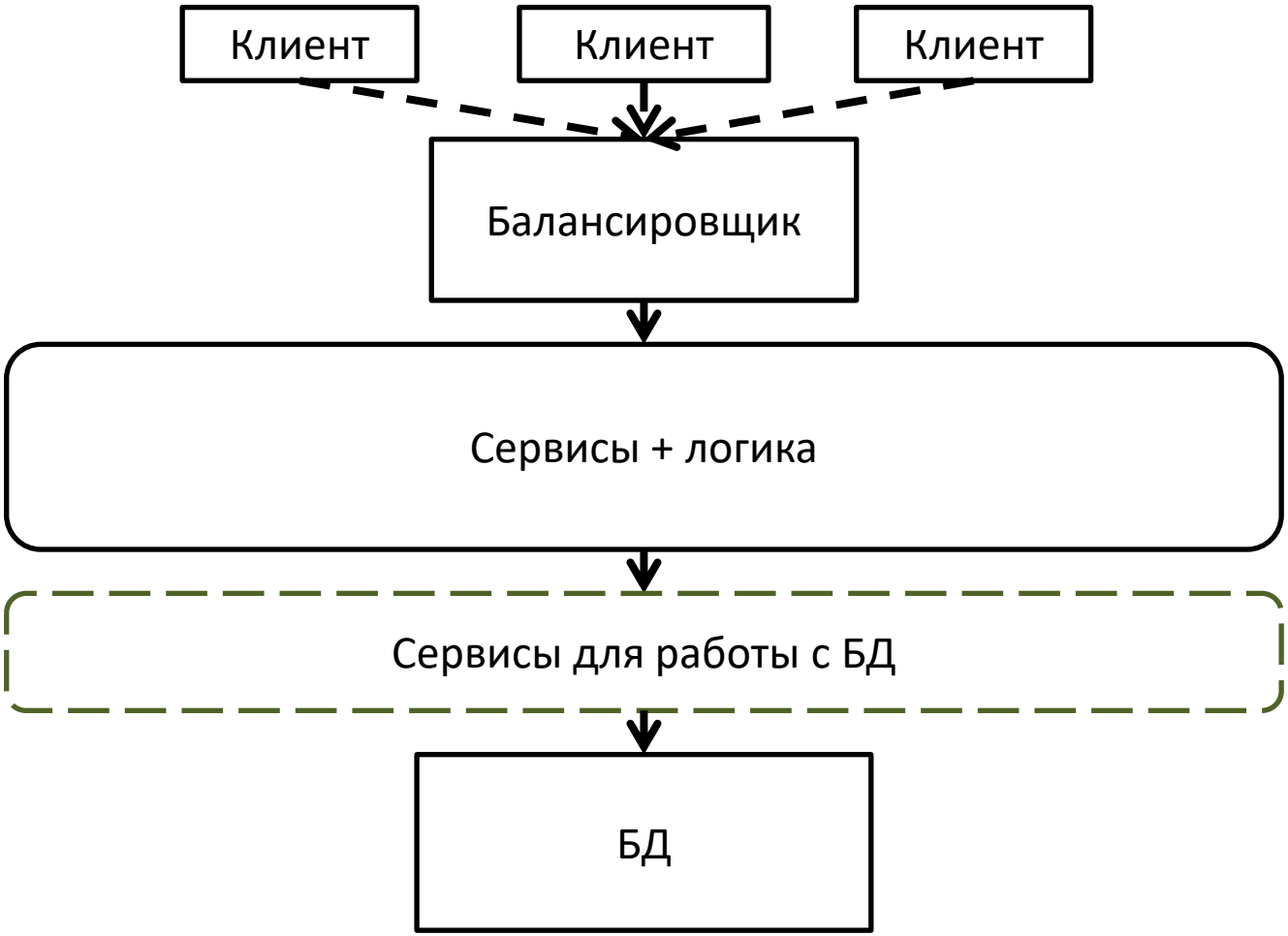
Опасные места

- Архитектура
- Инфраструктура
- Распределение обязанностей между клиентом и сервером

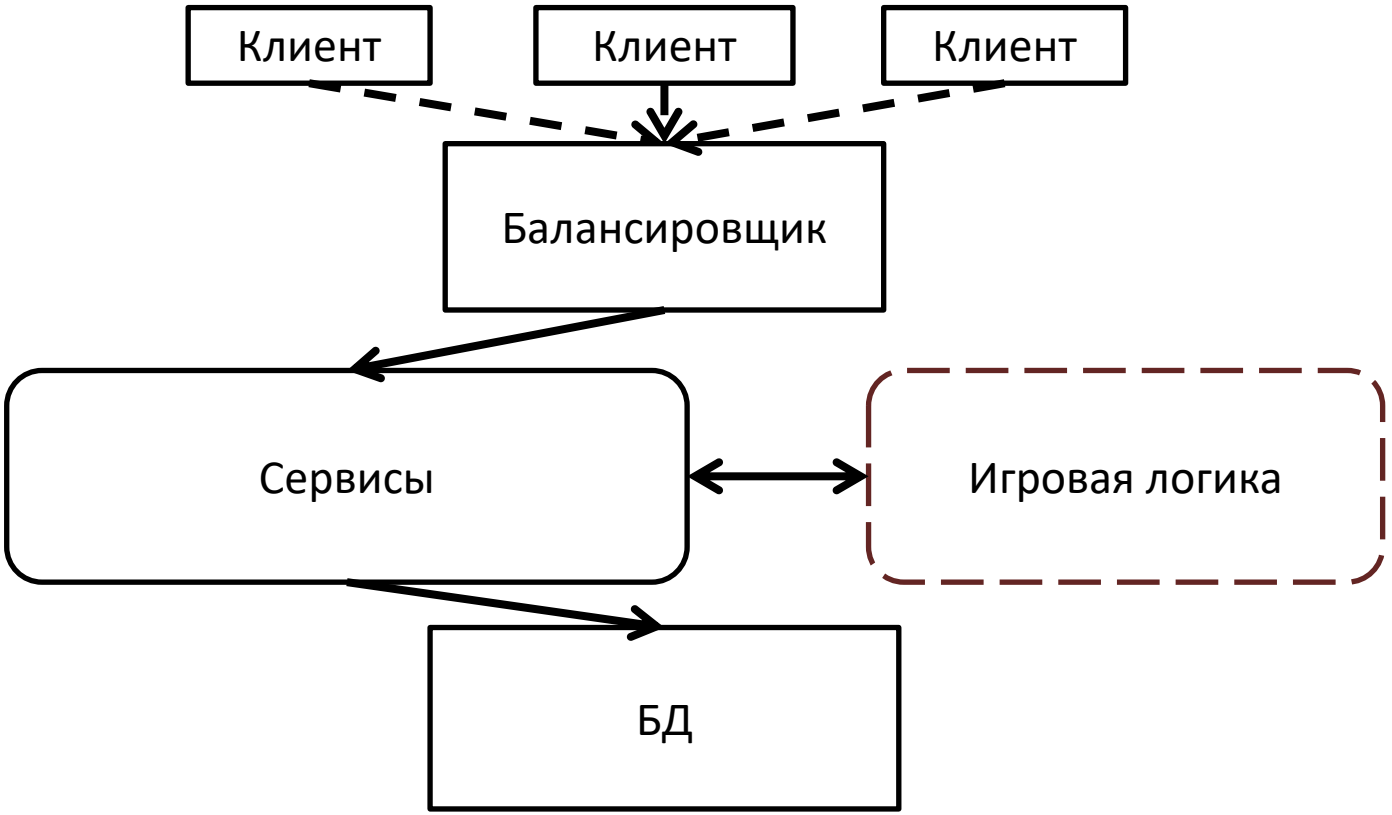
Общая архитектура



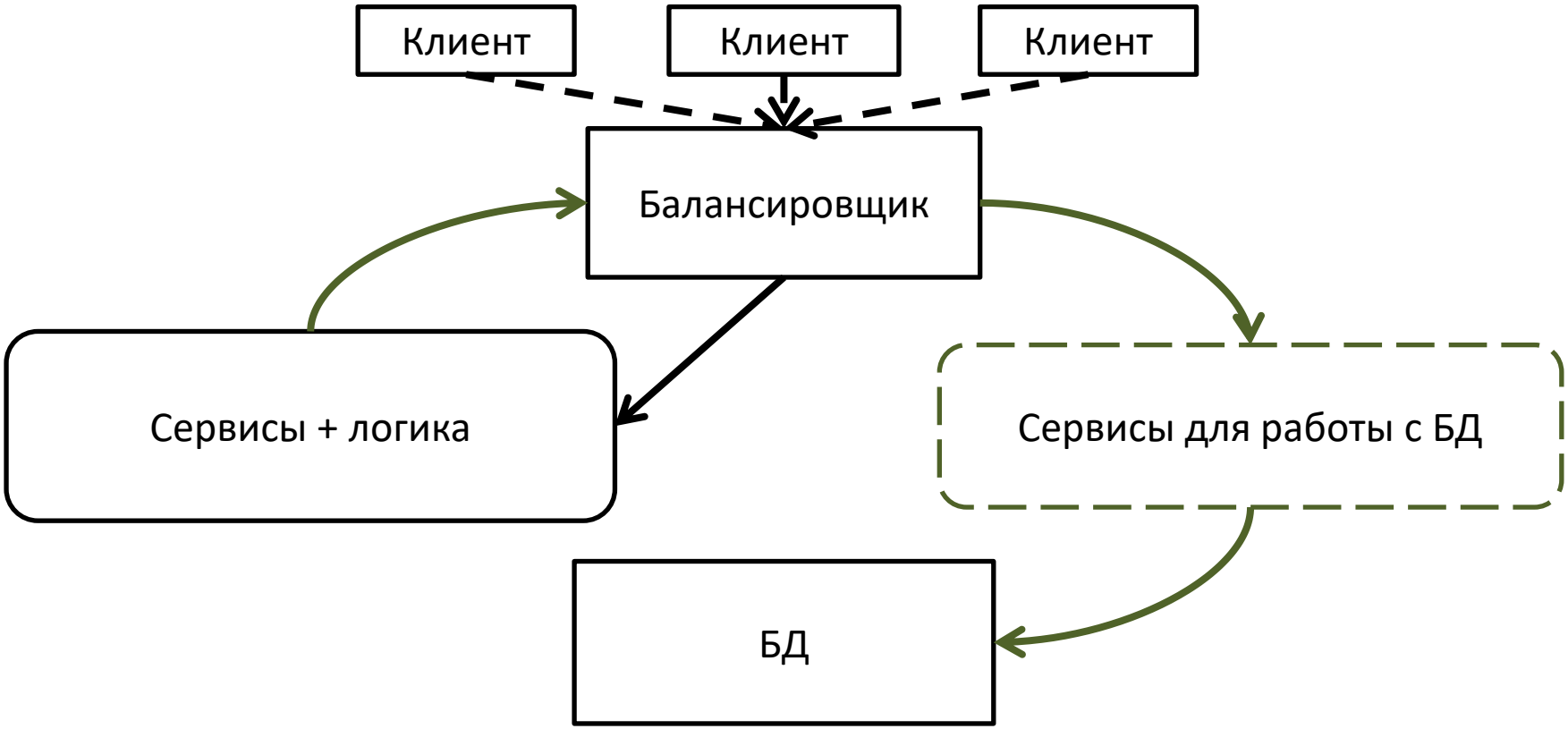
Если необходимо выделить сервисы данных



Если необходимо отделить логику от сервисов взаимодействия



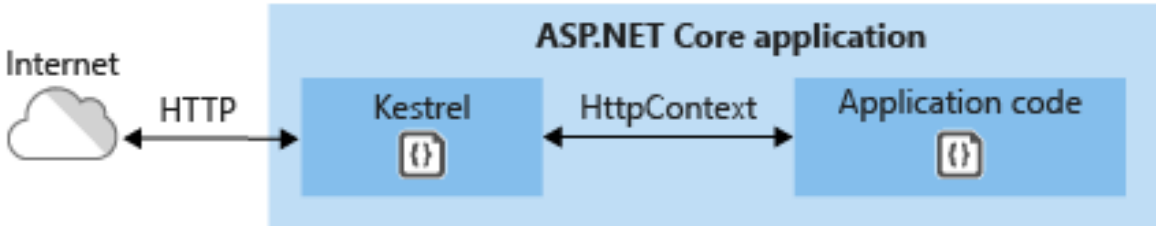
Если необходимы частые обращения к БД

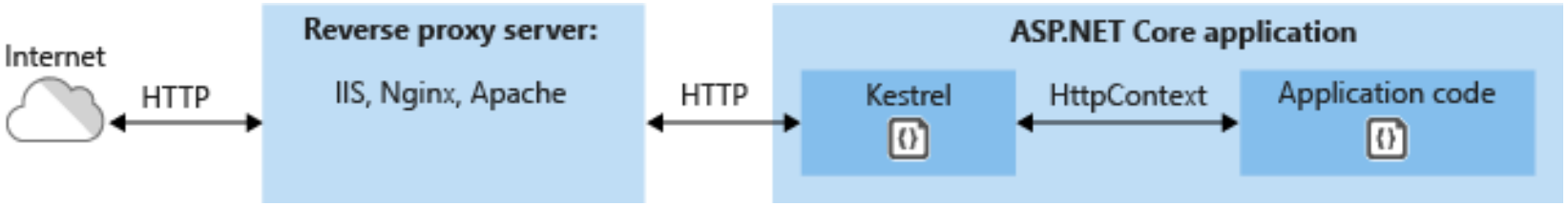


На что нужно обратить внимание при проектировании?

- Действовать из принципов необходимости и достаточности
- Быть готовым к изменениям:
  1) не переусердствовать с «заглушками» для будущих возможных изменений
  2) закладывать гибкость системы в целом
- Избегать узких мест

Способы публикации .NET Core-приложения





NGINX

Есть статистика использования Web серверов

- NGINX занимает 2-е место по популярности в тройке Apache, NGINX, IIS
- Прост в конфигурации, подробная документация
- Достаточно гибкий
- Обладает высокой производительностью

Что нужно знать о балансировке нагрузки

```
upstream backend {
    server backend1.somesite.com;
    server backend2.somesite.com;
    server backend3.somesite.com;
 }
```

NGINX – способы балансировки

- Round-Robin (по умолчанию)
- least_conn (наименьшее количество подключений)
- Ip_hash (зависит от хэша, вычисляемого из IP)
- hash (то же самое, только хэш берется от любого значения: текста, переменной, комбинации)
- least_time (зависит от задержки)
  Директивы header и last_byte определяют, исходить из первого или последнего отклика от сервера

NGINX – веса

Веса полезны, когда машины, включенные в балансировку, отличаются по мощности. Например:

```
upstream backend {
    server backend1.example.com weight=5;
    server backend2.example.com;
    server 192.0.0.1 backup; }
```

Здесь каждые 6 запросов backend1 обрабатывает 5, backend2 – 1, а последний будет использован, только если первые два недоступны.

Какой способ балансировки лучше?

- Исходить лучше из того, какой способ какие плюсы и минусы даст для того или иного типа игры
- Много серверов – не всегда хорошо. Может получиться так, что на 50 серверах сидят по 1 человеку и ждут соединения с 19 другими игроками
- Всегда делайте аналитику по количеству активных пользователей
- Автоматизируйте мониторинг состояния машин. Старайтесь предвидеть большие проблемы

Kestrel

Kestrel – с англ. пустельга

Зачем так сложно: Kestrel и NGINX?

- Kestrel включен в ASP.NET Core по умолчанию.
  Можно и без него – например, под Windows: WebListener, HTTP.sys и т.д. Но тогда «все в ваших руках»
- Kestrel может предоставлять доступ максимум к одному приложению на одной конкретной паре IP:Port
- У Kestrel более скудные возможности. Нет кэширования запросов, возможности работать со статическим контентом, сжимать запросы и т.д.

Что Kestrel умеет

Можно регулировать следующие параметры:

- максимальное число клиентских подключений;
- максимальный размер текста запроса;
- минимальная скорость передачи данных в тексте запроса.

Также доступны настройки Endpoint

Как распределить ответственность между Kestrel и NGINX?

- На сайте NGINX можно увидеть следующую настройку Kestrel: app.UseKestrel();
- По сути NGINX умеет больше, чем Kestrel
- Постарайтесь максимально правильно настроить NGINX

Как распределить ответственность между Kestrel и NGINX?

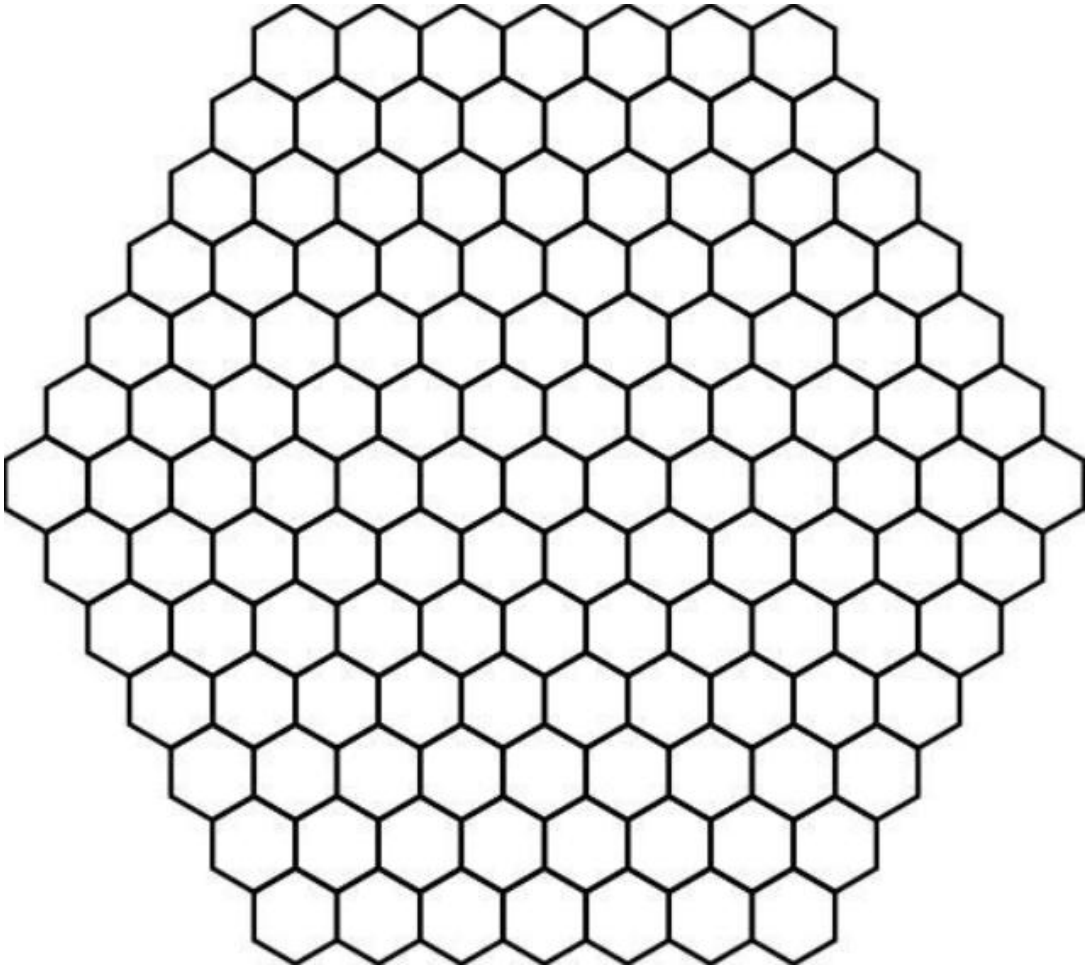
- Настройте Kestrel максимально там, где у запроса есть возможность достичь сервиса, минуя NGINX. Например:
  1) когда NGINX работает только как первичный балансировщик
  2) в местах, где есть взаимодействие между сервисами или между двумя машинами по внутренней сети
  3) иные случаи обращения напрямую к Kestrel
- Настраивайте Kestrel так, чтобы не ухудшать, а улучшать жизнь NGINX

Распределение ответственности между клиентом и сервером при использовании тонкого клиента

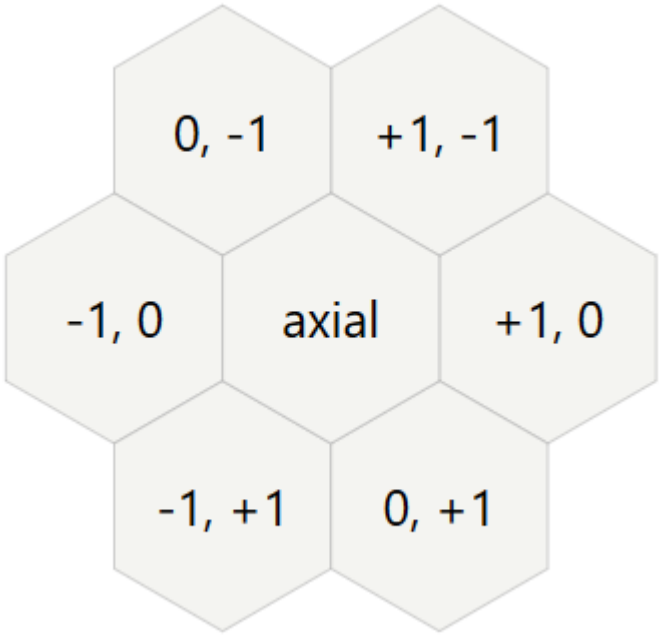
Особенности сервера

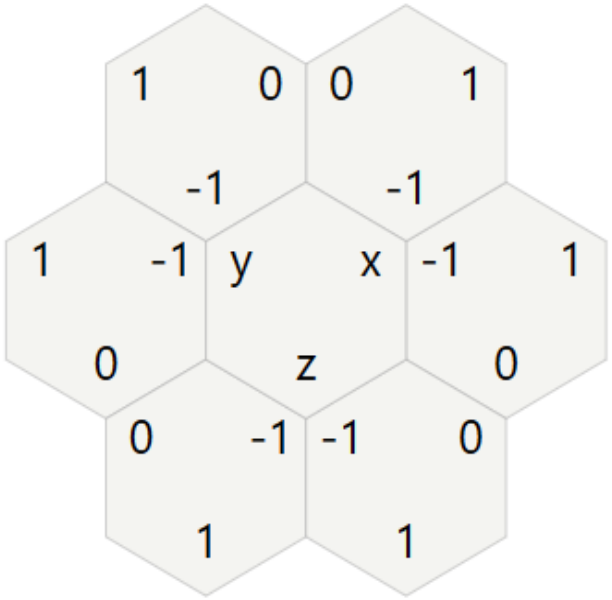
- Содержит всю логику, кроме перерисовки
- Хранит в памяти активные инстансы игр
- Изменяет положение и состояние объектов на игровой карте каждого инстанса

Особенности клиента

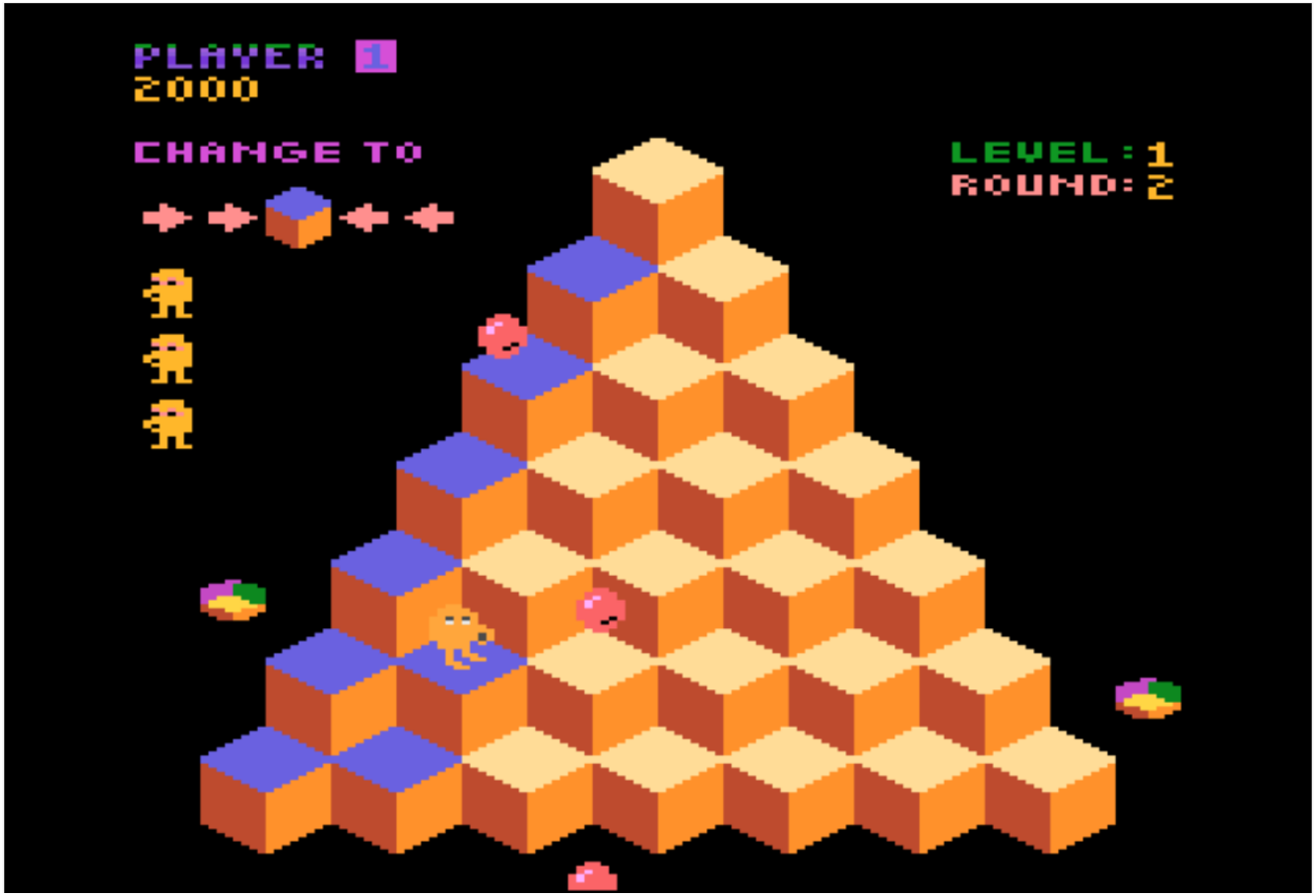


Способы вычисления координат





Откуда пошло кубическое представление

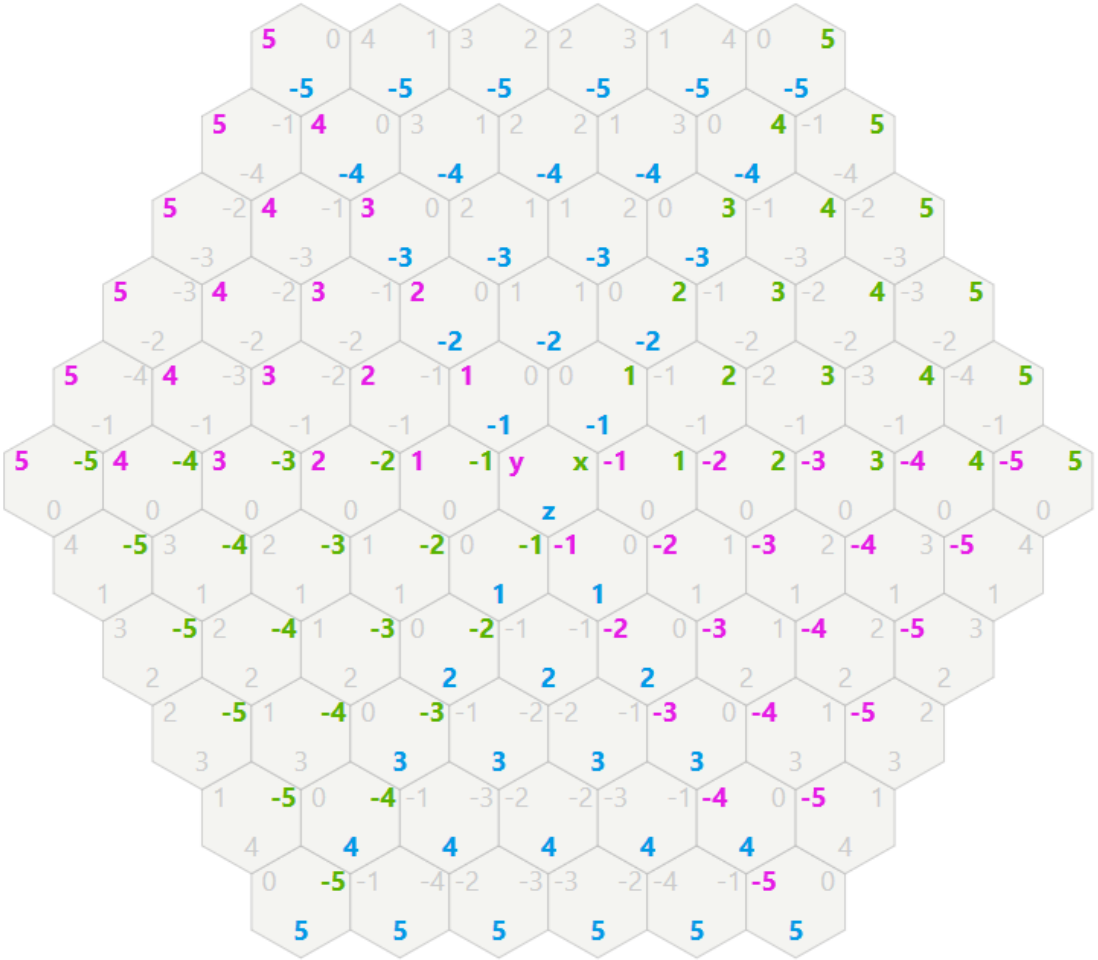


Способы вычисления координат

- Преимущество Axial – можно быстро вычислять адрес нужной ячейки
- Преимущество кубических координат – легко вычислять расстояние:

$$D = (|x_1-x_2| + |y_1-y_2| + |z_1-z_2|) / 2$$

Наиболее удобная нумерация

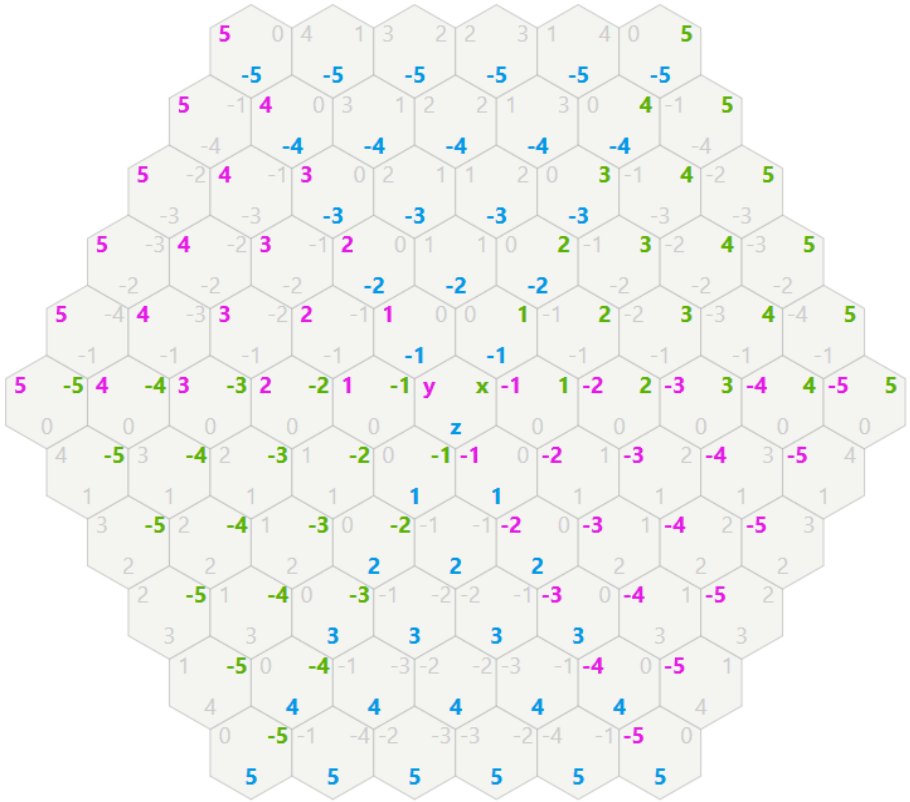


Способы вычисления расстояния

$$D = MAX(|x_1-x_2|, |y_1-y_2|, |z_1-z_2|)$$

Принцип работы: обновление, движение, атака и т.д.


```
{
    "activePlayer": 1,
    "fields": [
        {
            "fieldType": 1,
            "x": -5,
            "y": 5,
            "z": 0,
            "building": {
                "health": 200,
                "buildingType": 3,
                "owner": {
                    "playerId": 1,
```

Резюме

- Выбирайте правильные средства разработки
- При разработке мультиплеерных игр старайтесь придерживаться принципа необходимости и достаточности во всем
- Старайтесь быть гибкими, но не «костыльными» в плане архитектуры
- Каждая игра индивидуальна – максимально подстраивайтесь под ее требования и возможности
- Избегайте узких мест и тщательно следите за загрузкой и безопасностью серверов. Главное в данном ремесле – скорость и надежность.

Спасибо!

Вопросы?

Орлов Юрий

justice1786@gmail.com