Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Работа допущена к защите
Директор ВШПИ
П. Д. Дробинцев
« » 2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
магистерская диссертация

РАЗРАБОТКА И АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ POLYMAP: ПЛАТФОРМА ДЛЯ ПРОЕКТИРОВАНИЯ И РАЗМЕЩЕНИЯ ИНТЕРАКТИВНЫХ КАРТ ПОМЕЩЕНИЙ И ТЕРРИТОРИЙ. ПРИМЕНЕНИЕ SERVERLESS-ПОДХОДА.

по направлению подготовки (специальности)
09.04.04 Программная инженерия

Направленность (профиль)
09.04.04_01 Технология разработки и сопровождения качественного программного продукта

Выполнил студент гр. 5140904/20102 / Сопрачев А. К. /

Руководитель доцент, к.т.н / Дробинцев П. Д. /

Консультант по нормоконтролю / Локшина Е . Г. /

Санкт-Петербург
2025

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

УТВЕРЖДАЮ
Директор ВШПИ
П. Д. Дробинцев
« » 2025 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы
студенту Сопрачеву Андрею Константиновичу, группа 5140904/20102

1. **Тема работы:** Разработка и архитектурные особенности PolyMap: платформа для проектирования и размещения интерактивных карт помещений и территорий. Применение Serverless-подхода.
2. **Срок сдачи студентом законченной работы:** 15.05.2025
3. **Исходные данные по работе:** Исходный код iOS приложения PolyMap
4. **Содержание работы (перечень подлежащих разработке вопросов):**
   - Актуальность исследования
   - Анализ существующих решений
   - Требования к разработке
   - Архитектура сервиса
   - Реализация сервиса
   - Анализ результата
5. **Перечень графического материала (с указанием обязательных чертежей):** -
6. **Перечень используемых информационных технологий, в том числе программное обеспечение, облачные сервисы, базы данных и прочие сквозные цифровые технологии (при наличии):** VSCode, Yandex Cloud, Cloudflare
7. **Консультанты по работе:** -
8. **Дата выдачи задания:** 01.04.2025

Руководитель ВКР / Дробинцев П. Д. /

Задание принял к исполнению 01.04.2025

Студент / Сопрачев А. К. /

Реферат

На 31 с., 0 рис., 0 табл.

КЛЮЧЕВЫЕ СЛОВА:

Abstract

31 pages, 0 figures, 0 tables.

KEYWORDS:

Содержание

Введение

Глава 1. Обзор предметной области

1.1 Актуальность

Обусловлена популярностью пилотной версии PolyMap. В первую неделю учебного семестра, приложением пользовалось более 6000 студентов, что составляет около 80% от всей возможной аудитории.

1.2 Цели и задачи (В введении)

Сервис PolyMap является продолжением бакалаврской работы, в которой была реализована пилотная версия приложения, она обладала следующими ограничениями:

- Поддерживалась только iOS платформа
- Поддерживалась только одна карта Политеха, которая была жёстко закодирована в приложении
- Распространялось в виде приложения, которое требовалось устанавливать на устройство

1.2.1 Цели

Цели магистерской работы вытекают из ограничений пилотной версии приложения:

1. Разработать кроссплатформенное решение, которое будет доступно прямо в браузере, и будет адаптировано под управление как с помощью мыши на компьютере, так и с помощью сенсорного экрана на мобильных устройствах.
2. Реализовать возможность динамического просмотра разных карт, которые будут загружаться из удалённого сервера по запросу пользователя.
3. Серверная часть приложения должна справляться с вариативными нагрузками с высокими пиками. Должно быть быстрое время ответа в разных регионах мира.

1.2.2 Задачи

Для достижения поставленных целей необходимо решить следующие задачи:

1. Спроектировать и реализовать гибкую клиент-серверную архитектуру приложения
2. Разработать веб-приложение, которое будет отображать интерактивные карты в формате Extended-IMDF (формат карт, используемый в приложении PolyMap)
   - 2.а. Реализовать мобильную и компьютерную версии приложений.
   - 2.б. Интерфейс должен быть адаптирован под разные устройства.
   - 2.в. Управление картой на мобильных устройствах должно поддерживать жесты несколькими пальцами (для вращения и приближения карты).
3. Реализовать серверную часть, которая будет хранить карты и предоставлять их пользователю по запросу, а так же обрабатывать сопутствующие запросы веб-приложения (функция поделиться, генерация QR коров, сокращение ссылок).
4. Добавить в конструктор карт возможность загрузки карты на сервер.

Глава 2. Архитектура. Serverless подход

В этой главе будет рассмотрена верхнеуровневая архитектура сервиса PolyMap, будет подобран оптимальный подход к разработке серверной части, а так же будет рассмотрен выбор оптимального облачного провайдера для реализации Serverless архитектуры.

2.1 Анализ требований и проектирование

Архитектура сервиса в первую очередь определяется требованиями и задачами, которые перед ней ставятся. Одним из подходов к постановке технического задания является использования пользовательских сценариев. Они позволяют описать требования к системе с точки зрения пользователя, что позволяет лучше понять, как система будет использоваться в реальной жизни.

2.1.1 Ролевая модель. Сценарии использования

В сервисе PolyMap выделяются три основные роли:

- **Заказчик** – это человек, который заказывает разработку карты.
- **Пользователь** – это человек, который использует уже созданную карту.
- **Художник** – это человек, который по поручению заказчика оцифровывает карту в конструкторе сервиса.

При этом заказчик и художник могут быть одним и тем же человеком, так как в большинстве случаев, заказчик сам будет оцифровывать карту. Однако, в некоторых случаях, заказчик может поручить эту задачу художнику как своему представителю, так и заказать обрисовку карты у PolyMap.

2.1.1.1 Для заказчика

1. Как **Заказчик**, я хочу смотреть примеры других карт, чтобы определиться с выбором и качеством сервиса.
2. Как **Заказчик**, я хочу иметь возможность заказать размещение, чтобы получить карту.
3. Как **Заказчик**, я хочу делегировать задачу отрисовки карты чтобы самому не тратить на это время.
4. Как **Заказчик**, я хочу смотреть на карту во время отрисовки, чтобы контролировать процесс.
5. Как **Заказчик**, я хочу выбрать кастомный URL на котором будет карта, чтобы его было легко запомнить (например `polymap.ru/spbstu`).
6. Как **Заказчик**, я хочу иметь возможность создавать приглашения на мероприятия с закодированным маршрутом, чтобы рассылать их по почте и пользователю не надо было ничего дополнительно делать.
7. Как **Заказчик**, я хочу иметь возможность создавать векторные QR-коды приглашений, чтоб использовать их в печатной продукции.
8. Как **Заказчик**, я хочу чтобы сервера сервисы были в России, чтобы снизить инфраструктурные риски .

2.1.1.2 Для пользователя

1. Как **Пользователь**, я хочу , чтобы .
2. Как **Пользователь**, я хочу открывать карту прямо в браузере, чтобы не скачивать её как отдельно приложение.
3. Как **Пользователь**, я хочу чтобы карта быстро загружалась, чтобы не приходилось ждать.
4. Как **Пользователь**, я хочу чтобы при повторном открытии карта не скачивалась заново, чтобы не ждать загрузку при повторном открытии.
5. Как **Пользователь**, я хочу мобильный интерфейс, чтобы удобно управлять картой жестами, приближать и отдалять её.
6. Как **Пользователь**, я хочу компьютерный интерфейс, чтобы можно было изучить карту на компьютере.
7. Как **Пользователь**, я хочу автоматическое переключение между ПК и мобильной версией, чтобы не задумываться об этом.

8. Как **Пользователь**, я хочу просматривать не только планировку этажей, а и прилежащую территорию, чтобы изучать взаимное расположение корпусов.
9. Как **Пользователь**, я хочу создавать приглашения и QR коды как и заказчик, чтобы приглашать на студенческие события без излишней бюрократии (например собрание профсоюза).
10. Как **Пользователь**, я хочу строить бесшовные маршруты от любой одной аннотации до любой другой аннотации, даже если это кабинеты в разных корпусах, чтобы лучше планировать маршрут.
11. Как **Пользователь**, я хочу расчёт времени на маршрут с учётом лестниц и входов в здание, чтобы следить за временем и не опаздывать.
12. Как **Пользователь**, я хочу текстовый поиск по аннотациям, чтобы ввести номер кабинета, нажать на него и он отобразился на карте.
13. Как **Пользователь**, я хочу видеть информацию о кабинете, если такая есть, например расписание столовой или любую текстовую заметку, чтобы получать из карты ещё больше полезной информации.

2.1.1.3 Для художника

Большая часть задач художника уже была решена в рамках бакалаврской работы, ниже приведён список недостающих задач.

1. Как **Художник**, я хочу удобную привязку к сетке, чтобы не приходилось вручную выравнивать стены.
2. Как **Художник**, я хочу прямо из конструктора отправлять карту на сервер, чтобы не заниматься бесполезной работой по экспорту в файл.
3. Как **Художник**, я хочу создавать тестовые версии карт, чтобы во время процесса отрисовки смотреть как карта будет выглядеть в реальном приложении.
4. Как **Художник**, я хочу более удобным чем Git механизм совместной работы над картой, чтобы синхронизировать прогресс без навыков использовать Git.

Таким образом, верхнеуровневую архитектуру сервиса можно представить в виде трёх основных компонентов:

1. Клиентская часть – приложение отображающее интерактивную карту.
2. Конструктор карт – доступен для заказчиков карт и художников, позволяет создавать и редактировать карты, а так же загружать их на сервер.
3. Серверная часть – отвечает за хранение карт и сопутствующих данных, а так же за обработку запросов от клиентской части.

Необходимо предусмотреть возможность расширения клиентской части до нативных мобильных приложения на iOS и Android, а так же возможность в будущем, предоставлять доступ к конструктору карт неограниченному числу лиц.

Так же важным требованием является размещение серверной части в России, что позволит снизить инфраструктурные риски, связанных с политической ситуацией в мире.

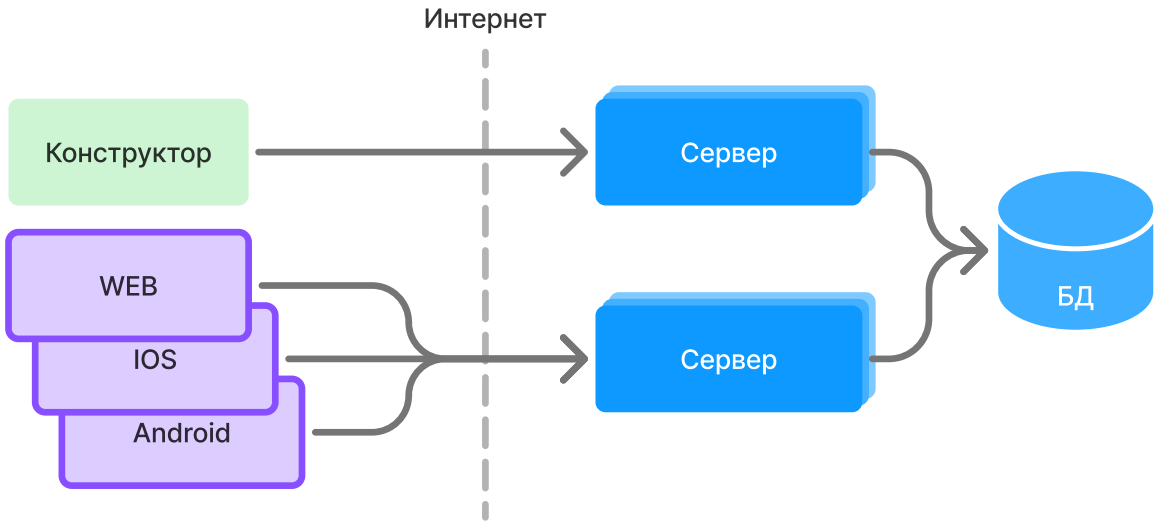


Рис. 1. Общая схема архитектуры приложения

2.2 Клиентская часть

Для соответствия требованиям кроссплатформенности, и скорости загрузки, было решено использовать веб-технологии для разработки клиентской части. Это позволит использовать единое веб-приложение как на мобильных устройствах, так и на компьютерах. Кроме того, это позволит избежать необходимости установки приложения на устройство, что значительно упростит процесс использования сервиса и даст пользователям более комфортный опыт (открыть сайт с картой быстрее и безопаснее для пользователя, чем скачивать и устанавливать приложения).

Клиентское веб-приложение всё ещё требует адаптации под разные устройства и способы пользовательского ввода. Для этого необходимо использовать адаптивный интерфейс, который будет подстраиваться под размер экрана устройства, а так же поддерживать способы управления картой мышкой и сенсорным экраном.

Клиентское приложение должно поддерживать следующие функции:

- Отображение карты в формате Extended-IMDF
  - Просмотр карты
  - Приближение и вращение карты
  - Просмотр планировок этажей
  - Переключения между этажами текущего здания
- Отображение аннотаций на карте и этажах
- Поиск по аннотациями на карте
- Просмотр информации об аннотации
  - Текстовое описание аннотации
  - Расположение аннотации на карте
- Построением маршрута по карте
  - Маршрут между двумя аннотациями
  - Маршрут от стандартной точки начала до аннотации (точка начала своя для каждой карты, их может быть несколько)
  - Маршрут должен бесшовно переходить от прилежащей территории к зданию и между этажами
- Должна быть возможность поделиться маршрутом с другими пользователями
  - С помощью постоянной URL ссылки
  - С помощью QR-кода

Подробная реализация будет рассмотрена в Раздел 3.2.

2.3 Конструктор карт

Конструктор карт является отдельным приложением, которое доступно для художников и заказчиков карт после авторизации. В нём можно создавать и редактировать карты, выгрузка их на сервер осуществляется из специального меню конструктора, которое будет доступно только для авторизованных пользователей. При планировании архитектуры сервиса, необходимо предусмотреть будущие развитие конструктора до публичной веб-версии, к которой будет иметь доступ любой желающий. Общее видение проекта подразумевает бизнес-модель аналогичную конструкторам сайтов, таким как Tilda или Wix. То есть, любой желающий сможет спроектировать карту, загрузить её на сервер и просмотреть, однако, для того, чтоб сделать её общедоступной, необходимо будет оплатить подписку.

Реализация веб-версии конструктора выходит за рамки текущий работы, однако планируется в дальнейших обновлениях сервиса, что накладывать ряд дополнительных требований на архитектуру:

- Весь процесс взаимодействия и обработки новых карт должен быть автоматизирован.
- Необходимо предусмотреть возможность добавления сервера синхронизации для совместной работы над одной картой.
- Необходимо заложить возможность масштабирования серверной части, отвечающей за обработку новых и изменения старых карт.

2.4 Серверная часть

Сервис будет сталкиваться с неравномерной нагрузкой с высокими пиками, которые будут возникать в связи со следующими причинами:

- Повышенный спрос на карту университетов в начале учебного семестра
- Повышенный спрос на карту университетов в начале каждого дня. По утрам, в 10:00 и в 12:00, перед началом пар, студенты будут открывать карту, чтобы найти нужную аудиторию. Что подтверждается статистикой использования пилотной версии.

TODO: Добавить график использования пилотной версии PolyMap

- При использовании карты на временных конференциях и выставках, большинство пользователей будут открывать карту одновременно, что создаст пиковую нагрузку на сервер.

Для решения этих проблем необходимо предусмотреть горизонтальное масштабирование серверной части, для этого **был выбран микросервисных подход** к разработке.

2.5 Serverless подход

Для оркестрации микросервисов наиболее распространено использовать Kubernetes, однако последние несколько лет, растёт популярность альтернативного подхода – Serverless. Он позволяет запускать и масштабировать микросервисы без необходимости взаимодействовать с виртуальными или выделенными серверами. Вместо этого, Serverless предлагает арендовать у облачных провайдеров только вычислительные ресурсы, которые фактически были использованы.

Основными недостатками Kubernetes выделают сложность настройки качественной инфраструктуры, высокий риск ошибок при ручной настройке, сложность управления версиями, а так же высокую стоимость кластера, особенно на начальных этапах разработки и тестовых средах.

2.5.1 Преимущества и недостатки

Как и любой другой подход, Serverless имеет свои плюсы и минусы. Плюсы Serverless вытекают из минусов Kubernetes:

1. Простота настройки и управления – при использовании Serverless, разработчик не взаимодействует с виртуальными серверами, а только с облачными ресурсами, которые предоставляют достаточно простой и однозначный интерфейс управления.
2. Низкая вероятность ошибок – так как разработчик не взаимодействует с виртуальными серверами, то вероятность ошибок при настройке инфраструктуры значительно снижается. Кроме того, облачные провайдеры предоставляют таки нестойки, которые не позволят допустить существенную ошибку.
3. Низкая стоимость – Serverless распространяется по модели pay-as-you-go, то есть необходимо оплачивать только те ресурсы, которые были фактически использованы, что хорошо видно на Рис. 2. Это позволяет значительно снизить затраты на инфраструктуру.

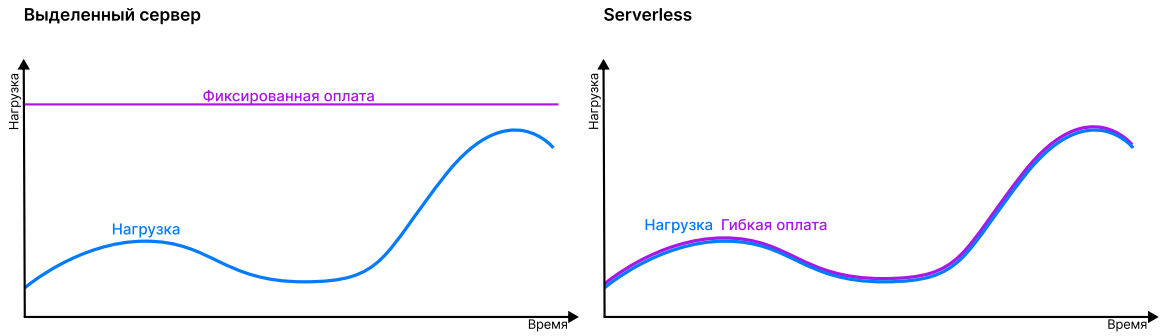


Рис. 2. Сравнение затрат на инфраструктуру

За счёт того, что микросервисы в Serverless подходе запускаются только во время запроса пользователя и отключаются после завершения обработки, они потребляют ресурсы облака только в момент фактической работы, в отличии от Kubernetes, в котором вы арендуете виртуальные сервера вне зависимости от их использования. При переменно нагрузки, Kubernetes будет простаивать значительное время, что приведёт к нерациональным затратам, которых позволяет избежать Serverless.

Однако Serverless имеет и свои недостатки:

1. Требования читой архитектуры – Serverless подход требует от разработчиков строгого соблюдения принципов чистой архитектуры:
   - микросервисы быть независимыми друг от друга
   - связи должны быть сделаны через очереди сообщений
   - микросервисы должны быть State Less – то есть не должно храниться внутреннее состояние, оно будет потеряно при перезапуске микросервиса
2. Привязанность к облачному провайдеру – Serverless часть есть у большинства крупных облачных провайдеров, однако конечная реализация может отличаться, что сильно усложняет миграцию между облачными провайдерами.
3. Ограниченность возможностей – не любую микросервисную архитектуру удастся реализовать с помощью Serverless. Например, большинство баз данных не поддерживают Serverless. Однако облака предоставляют доступ к Managed версиям баз данных, к которым можно обращаться из Serverless микросервисов. Кроме того, не все технологии смогут корректно работать в условиях короткого жизненного цикла.
4. Должно быть обеспечено быстрое время холодного старта – так как микросервисы запускаются только во время запроса, то время их запуска должно быть минимальным. Это

накладывает ограничения на используемые технологии, подробнее о выборе подходящего технологического стека рассмотрено в Раздел 3.3.2.

2.5.2 Cloud Native

В современном мире, для разработки и развёртывания приложений, всё чаще используется Cloud Native подход. Он подразумевает разработку приложений, которые будут работать в облаке, используя облачные ресурсы для хранения данных и вычислений. Это позволяет в некоторых аспектах упростить разработку и развёртывание приложений, а так же значительно снизить затраты на поддержку и гарантии доступности.

Serverless является дочерним подходом к Cloud Native разработке. Именно этот подход и был выбран для разработки серверной части приложения PolyMap.

2.5.3 Выбор облачного провайдера с применением СППР

Наиболее крупным игроком в сфере Serverless является Amazon Web Services (AWS), однако официально в России он не доступен, что создаёт дополнительные инфраструктурные риски. По этому, в качестве облачного провайдера необходимо было выбрать отечественную альтернативу. На текущий момент в России существует несколько крупных облачных провайдеров предоставляющих Serverless решения:

- Яндекс Облако – крупнейший в России облачный провайдер, который активно развивает как Serverless, так и Cloud Native решения. Яндекс Облако предоставляет широкий спектр облачных услуг, включая Serverless Container, API Gateway, YDB и другие.
- Selectel – старый игрок на российском рынке, в первую очередь ориентируется на выделенные серверы, однако Serverless решения тоже присутствуют.
- Сбер Облако (переименовали в CloudRu) – относительно новый провайдер, активно развивается, имеет очень выгодные тарифы и бесплатные квоты.
- VK Cloud – является ещё одним новым провайдером, Serverless решения не являются приоритетом.

Стоит отметить, что все эти хостинги соответствуют требованиям российского законодательства о хранение данных (ФЗ-152), что является важным фактором при выборе облачного провайдера.

Для выбора облачного провайдера было принято решение воспользоваться системой поддержки принятия решений (СППР), которая позволяет сравнить облачные провайдеры по заданным критериям. В качестве критериев были выбраны следующие:

- Вычисления – функции (FaaS), наличие serverless контейнеров, технические лимиты, квоты, удобство использования
- Интеграции – удобство вызовов функций/контейнеров (API Gateway), триггеры, очереди, CRON задачи
- БД, Очереди, Уведомления – управления данными – наличие serverless решений, удобство работы, поддерживаемые протоколы
- DevOps & DX – наличие DevOps инструментов, документация, удобство автоматизации процессов. Оценивается CLI, SDK, наличие Terraform провайдера, наличие собственных GitHub/GitLab интеграций, активное сообщество (популярность)
- Мониторинг, логи – наличие инструментов мониторинга и логирования, удобство работы с ними, внутренний функционал (создание своих графиков, алертов, язык запросов и т.д.), а так же интеграция с внешними системами.

Кроме отечественных облачных провайдеров, в таблице представлены и зарубежные:

- AWS – на текущий момент является лидером в области Serverless, однако официально недоступен в России.
- Google Cloud Platform – второй по популярности облачный провайдер, который активно развивает Serverless решения.
- Azure – облачный провайдер от Microsoft, в нём тоже присутствует Serverless, однако на нём не делают акцент.

Для всех критериев были проставлены оценки по шкале от 0 до 10:
0–3 – функционал полностью отсутствует, либо пользоваться крайне неудобно.
4–5 – функционал присутствует, но выполняет базовый минимум задач, серьёзно уступает альтернативам.
6–7 – работает, но есть заметные пробелы, требует ручной настройки.
8–9 – работает хорошо, покрывает все задачи, работать удобно, но есть небольшие недочёты.
10 – самый полный и удобный на рынке функционал в данной категории.

С использованием 5 алгоритмов был рассчитан общий балл для каждого облачного провайдера. В Приложение А. находится подробный отчёт СППР. Использовалась собственная реализация СППР с открытым исходным кодом.

Вариант	Дом	Блок	Typ	Sjp	Sjm	ИТОГО	Место
Yandex	5	6	5	5	6	27	3
Selectel	5	6	2	2	6	21	6
VK	5	6	1	1	6	19	7
Sber	5	6	3	3	6	23	5
AWS	7	7	7	7	7	35	1
Google	6	6	6	6	6	30	2
Azure	5	6	4	4	6	25	4

Таблица 1. Результат работы СППР

Как видно из таблицы 1, наибольшим требуемым функционалом обладает AWS, однако он, как и Google и Azure недоступны в России, наивысший балл среди отечественных сервисов получило Яндекс Облако, поэтому оно было выбрано в качестве облачного провайдера для проекта. Sber Cloud активно развивает свои Serverless решения, однако на момент написания работы, всё ещё сильно отставал от Яндекс Облака. Selectel и VK имеют очень урезанный Serverless функционал, который не позволяет реализовать проект в полном объёме.

2.5.4 Обзор Serverless компонентов Яндекс Облака

Подход к Serverless в Яндекс Облаке реализован с помощью следующих компонентов:
- Cloud Functions – позволяет запускать код в ответ на события, такие как HTTP запросы или CRON задачи.
- Serverless Container – позволяет запускать Docker контейнеры в ответ различные события.
- API Gateway – позволяет описывать REST API для микросервисов в формате OpenAPI. Является публичной точкой входа в микросервисы.
- Message Queue – позволяет организовать асинхронную связь между микросервисами. Совместим с Amazon SQS API.
- Yandex Data Base (YDB) – Serverless база данных, которая может работать и в реляционном и в документо-ориентированном режимах. Совместимо DynamoDB API.

- Object Storage – объектное хранилище, которое позволяет хранить и раздавать бинарные объекты по протоколу S3.
- Cloud Postbox – сервис позволяет организовывать Email рассылки. Совместим с Amazon SES API.

В проекте не используется Cloud Functions, так как все микросервисы реализованы в виде Docker контейнеров, это позволяет использовать более привычный подход к разработке и тестированию, а так уменьшает привязанность к облачному провайдеру. Архитектура состоящая полностью из контейнеров может быть легко перенесена как на локальную машину, так и в другие облака. Кроме того, использование контейнеров позволяет использовать больший набор технологий, в том числе и те, которые не поддерживаются в Cloud Functions.

Кроме непосредственное Serverless компонентов, Яндекс Облако предоставляет и другие сервисы, которые будут полезны в проекте:
- Container Registry – позволяет хранить Docker образы, которые будут использоваться в Serverless Container.
- Monitoring – позволяет хранить и отслеживать состояния работы проекта: число запросов, время ответов и другие метрики. Кроме того, позволяет настраивать Alerting, который будет уведомлять о проблемах в работе проекта при срабатывании настроенных условий.
- Cloud Logging – позволяет собирать, хранить, фильтровать и просматривать логи из всех микросервисов в одном месте.

Yandex Cloud активно развивает Serverless раздел облака, и регулярно добавляет новые компоненты и улучшения. Например, на момент написания магистерской работы, в облако был добавлен Notification Service, который позволяет отправлять персональные уведомления пользователям через SMS, Push и Email. Это позволит в будущем расширить функционал сервиса PolyMap, сильно упростив реализацию личного кабинета заказчика.

2.5.5 Content Delivery Network (CDN)

Для решения задачи географического масштабирования, необходимо использовать Content Delivery Network (CDN) сети. Они пропускают запросы пользователей через географияески ближайший к ним узел, и в случае, если запрашиваемый ресурс разрешен для кеширования и уже есть в кеше, отдают его пользователю не перенаправляя запрос на основной сервер. Главным источником трафика в сервисе является раздача карт – это большие и тяжёлые файлы, которые будет скачивать каждый пользователь при каждом открытии карты. Сами карты являются статическими файлами, которые редко меняются, а так же, в большинстве случаев привязаны к конкретной локации. Поэтому их можно эффективно кешировать в CDN сети. Это позволит значительно снизить объём трафика и скорость открытия сайта, что положительно скажется на пользовательском опыте.

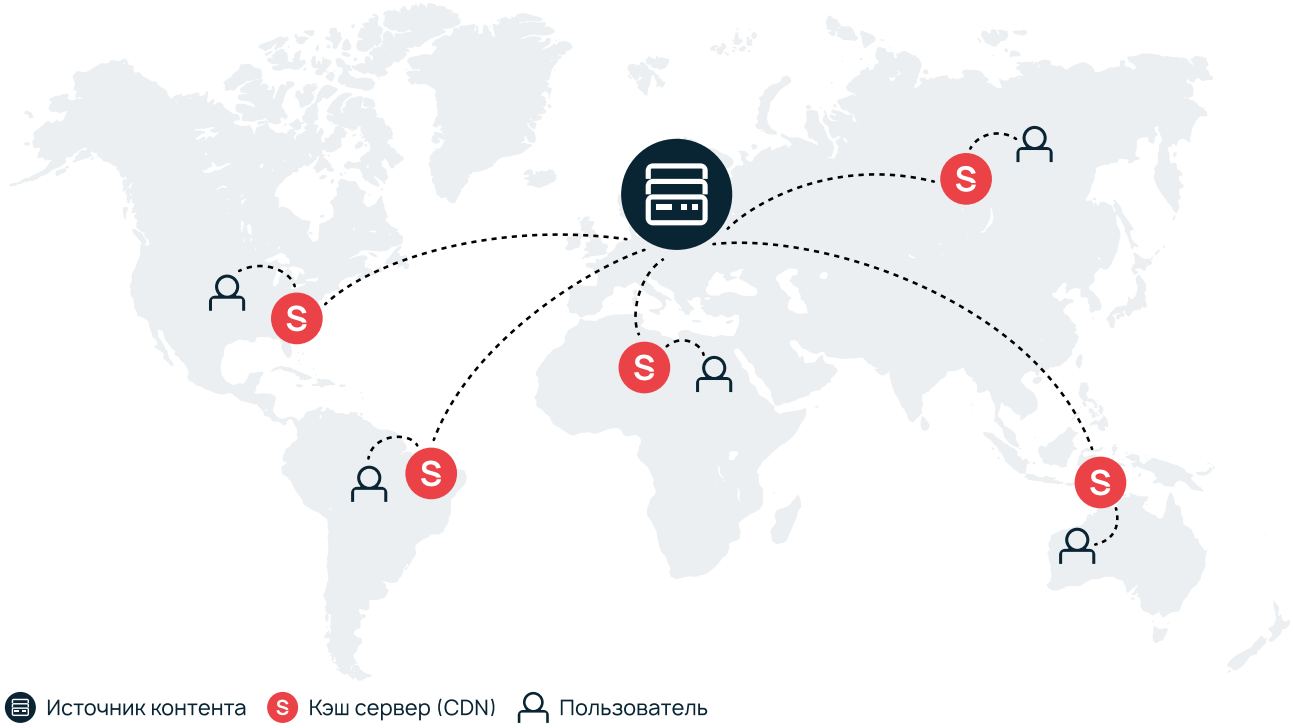


Рис. 3. Схема работы CDN

Общий принцип работы CDN состоит в передаче DNS управления к CDN провайдеру, после чего, провайдер будет разрешать запросы к ближайшему узлу.

2.5.6 Жизненный цикл запроса

При использовании Serverless в Yandex Cloud, жизненный цикл обработки запросы выглядит следующим образом:

1. Пользователь через DNS сеть CDN определяет ближайший к нему узел, и отправляет запрос на него.
2. Узел CDN проверяет, есть ли запрашиваемый ресурс в кеше. Если он есть, то отдает его пользователю, если нет, то проксирует запрос в Yandex Cloud.
3. Yandex Cloud получает запрос и по домену находит соответствующий `API Gateway`, который будет обрабатывать запрос.
4. `API Gateway` проводит часть проверок (авторизацию, валидацию параметров) и перенаправляет запрос на соответствующий `Serverless Container`, который будет обрабатывать запрос.
5. Если нет запущенного экземпляра `Serverless Container`, то Yandex Cloud создаёт новый экземпляр контейнера из локального `Docker Registry`, и перенаправляет запрос на него.
6. `Serverless Container` обрабатывает запрос, при этом имеет возможность обращаться к прочим ресурсам Облака, таким как базы данных, очереди, и т.д.
7. После обработки запроса, `Serverless Container` возвращает ответ обратно в `API Gateway`, который в свою очередь возвращается к пользователю через CDN Proxy.
8. Если указаны заголовки кеширования, то CDN Proxy кеширует ответ на региональном узле, и в дальнейшем будет отдавать его пользователям, не перенаправляя запрос на основной сервер.

В случае обновления статического контента, есть возможность вызвать инвалидацию CDN кеша с помощью специального API. Это позволит удалить из кеша старую карту, чтобы новая версия была доступна пользователям сразу после обновления.

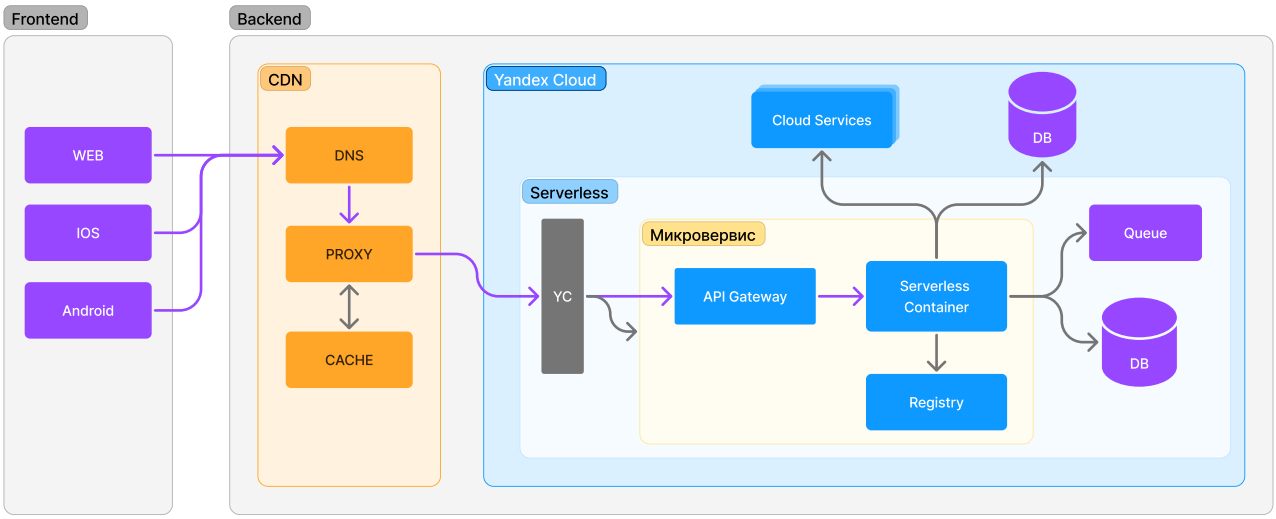


Рис. 4. Жизненный цикл запроса

2.5.7 Ценообразование

Основным преимуществом Serverless подхода является снижение инфраструктурных затрат относительно классических походов. Рассмотрим ценообразование в Yandex Cloud.

Важным фактором затрат на Serverless в Yandex Cloud является квоты на число бесплатно предоставляемых ресурсов. Эти квоты сбрасываются в начале каждого месяца, таким образом, низкая нагрузка в течение месяца, позволяет пользоваться инфраструктурой полностью бесплатно. После исчерпания квот, тарификация происходит по модели pay-as-you-go, то есть необходимо оплачивать только те ресурсы, которые были фактически использованы.

Ресурс		Стоимость	Бесплатно
API Gateway		120₽/1M запросов	Первые 100k в месяц
Serverless Containsers		16₽/1M вызовов	Первые 1M вызовов в месяц
		3.2₽/ГБ×час ОЗУ	Первые 1ГБ×час в месяц
		4.8₽/vCPU×час	Первые 1vCPU×час в месяц
Message Queue		48.76₽/1M запросов	Первые 100k в месяц
Yandex Data Base		21.38₽/1M Request Units	Первый 1M в месяц
		21.38₽/1ГБ×месяц	Первый 1ГБ в месяц
Object Storage (Standard)	GET	0.39₽/10000	Первые 100k в месяц
	POST	0.48₽/1000	Первые 10k в месяц
	DELETE	Бесплатно	
Исходящий трафик	< 1ТБ	1.53₽/1ГБ	Первые 100Гб в месяц
	< 50ТБ	1.28₽/1ГБ	
	< 100ТБ	1.20₽/1ГБ	
	> 100ТБ	1.15₽/1ГБ	

Таблица 2. Стоимость используемых Serverless ресурсов в Yandex Cloud на 01.05.2025

2.5.8 Сравнение затрат на Serverless и Kubernetes

С использованием цен актуальных на момент написания работы (Таблица 2) можно ориентировочно рассчитать общие затраты на инфраструктуру в месяц. Возьмём ориентировочную нагрузку на пилотную версию PolyMap iOS (Таблица 3).

Ресурс	Объём
Открытие карты	600 000
Пользователей	8 000
Просмотров информации об аннотации	70 000
Построено маршрутов	10 000
Открыто приглашений	250

Таблица 3. Потребляемые ресурсы пилотной версии PolyMap iOS за сентябрь 2024

Стоит отметить, что приложение не имеет официальный статус, и его не использовали для приглашений на мероприятия. В связи с этим, нагрузка на открытия приглашений сильно занижена, при реальном использовании на конференциях, число открытий приглашений будет пропорционально пришедшим пользователям.

Перенесём статистику на примерный механизм работы Serverless:

- API Gateway: 600 000 открытий карт + 70 000 просмотров аннотаций + 10 000 построений маршрутов + 250 открытий приглашений = 680 250 запросов
- Serverless Containers: 680 250 вызовов (среднее потребление: 0.5 ГБ памяти, 0.2 с процессора на вызов)
- Message Queue: 680 250 запросов
- Object Storage GET: 680 250 операций
- Исходящий трафик: 600 000 загрузок карт по 2 МБ = 1 200 ГБ

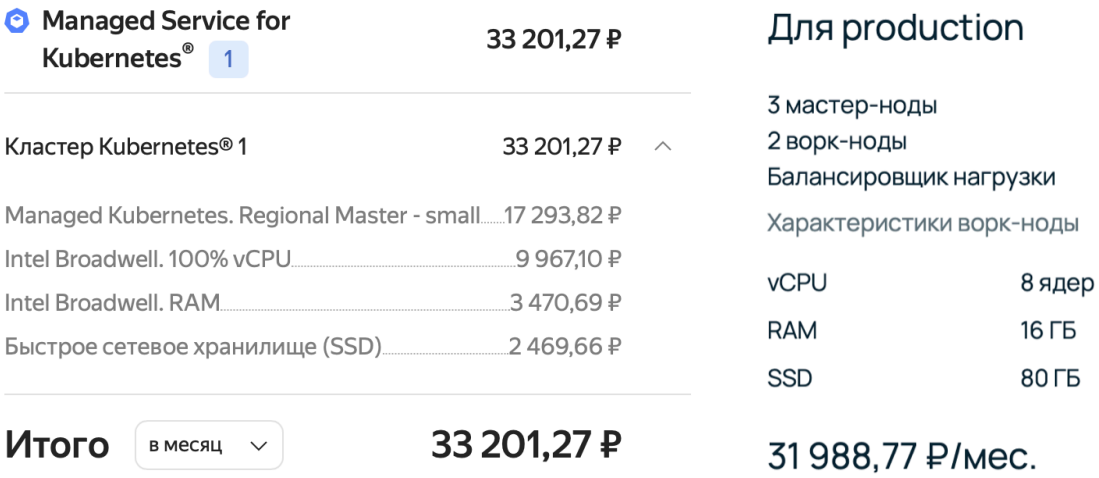
Ресурс	Объём	Бесплатно	Платно	Стоимость, ₽
API Gateway	680k запросов	100k	580k @120 ₽/1 млн	≈ 69.6
Вызовы контейнеров	680k вызовов	1M	—	0
Память (ГБ×ч)	18.9	1	17.9 @3.2 ₽/ГБ×ч	≈ 57.3
vCPU (vCPU×ч)	10.5	1	9.5 @4.8 ₽/vCPU×ч	≈ 45.6
Message Queue	680k запросов	100k	580k @48.76 ₽/1 млн	≈ 28.3
YDB RU	680k	1M RU	—	0
Object Storage GET	680k операций	100k	580k @0.39 ₽/10k	≈ 22.6
Исходящий трафик	1 200 ГБ	100ГБ	1 100 @1.28 ₽/ГБ	≈ 1 408
ИТОГО				**≈ 1 640**

Таблица 4. Оценка ежемесячных затрат на Serverless ресурсы

Как видно из расчёта (Таблица 4) на самый нагруженный месяц – сентябрь, приблизительные затраты на Serverless инфраструктуру составят **1600 рублей**, и большая часть затрат это исходящий трафик, который порождается раздачей статичных файлов карты, однако, большинство этих запросов не будут доходить для серверов, а будут обрабатываться на узлах кеша CDN, по этому реальные затраты будут значительно ниже.

Сравним это с затратами на Kubernetes кластер. Так как нагрузка не очень высокая, возьмём минимальную production конфигурацию кластера, которая будет состоять из 3 нод с 4 vCPU и 4 ГБ RAM. Три ноды позволят обеспечить доступность и отказоустойчивость сервера, а 4 vCPU и 4 ГБ RAM будет достаточно для обработки аналогичной нагрузки. Такой кластер в Yandex Cloud будет стоить **33000 рублей** в месяц, что значительно выше, чем затраты на Serverless. Для сравнения, на Рис. 5.б есть стоимость минимально возможного кластера в Selectel, он имеет меньшую отказоустойчивость, так как находится в одной зоне доступности,

однако обладает большими характеристиками: 8 vCPU и 16 ГБ RAM, такой кластер будет стоить **31000 рублей** в месяц.



(а) Yandex Cloud (б) Selectel

Рис. 5. Стоимость Kubernetes кластера в Yandex Cloud и Selectel

Так же, не стоит забывать, что кроме production кластера, необходимо будет поддерживать тестовый кластер, для разработки и тестирования новых обновлений, что значительно увеличит затраты на инфрастуктуру, в то время, как в Serverless подходе, тестовые среды даже не будут выходить из бесплатных квот. За три года активной разработки, я ни разу не использовал больше 10% от выделенных квот, что позволяет полностью бесплатно разрабатывать и тестировать новые обновления.

2.6 Детальная архитектура

TODO: переработать, дописать про клиенты, подробно раскрыть каждый блок архитектуры

2.6.1 Микросервисы

Использование микросервисов позволит разделить приложение на независимые части, и те из них, на которые будет повышенная нагрузка, можно будет масштабировать отдельно от остальных.

Для полного покрытия функционала сервиса, необходимо разработать следующие микросервисы:

1. Сервис раздачи карт (`map-storage`) – заниматься раздачей файлов карты пользователям в зависимости от запроса.
2. Сервис обслуживания конструктора (`constructor-back`) – занимается обработкой запросов от конструктора карт, сохранять карты на сервер.
3. Сервис хранения бинарных объектов (`blob-storage`) – хранит бинарные объекты, которые можно будет размещать на карте (например кастомные изображения аннотаций, планы этажей и т.д.).
4. Сервис синхронизации совместного редактирования (`constructor-WS`) – используется для синхронизации совместного редактирования карты в реальном времени. Позволяет нескольким пользователям одновременно редактировать карту, и видеть изменения друг друга в реальном времени.

5. Сервис генерации приглашений (`share-back`) – используется для функции создания приглашений на маршрут, позволяет приложить к приглашению ещё и текстовое сообщение, которое будет отображаться в приложении при открытии приглашения.
6. Сервис генерации QR-кодов (`qr-generator`) – используется для функции создания QR-кода для функции «поделиться маршрутом». Функционал вынесен из клиента в серверно приложение, для того, чтобы в будущем можно было использовать его в нативных мобильных версиях.
7. Сервис раздачи фронтеда (`web-map-back`) – используется для генерации OpenGraph метаданных для index.html страницы карт.

Сервис хорошо подходит для реализации в микросервисной архитектуре, все микросервисы независимы друг от друга и могут работать обособленно.

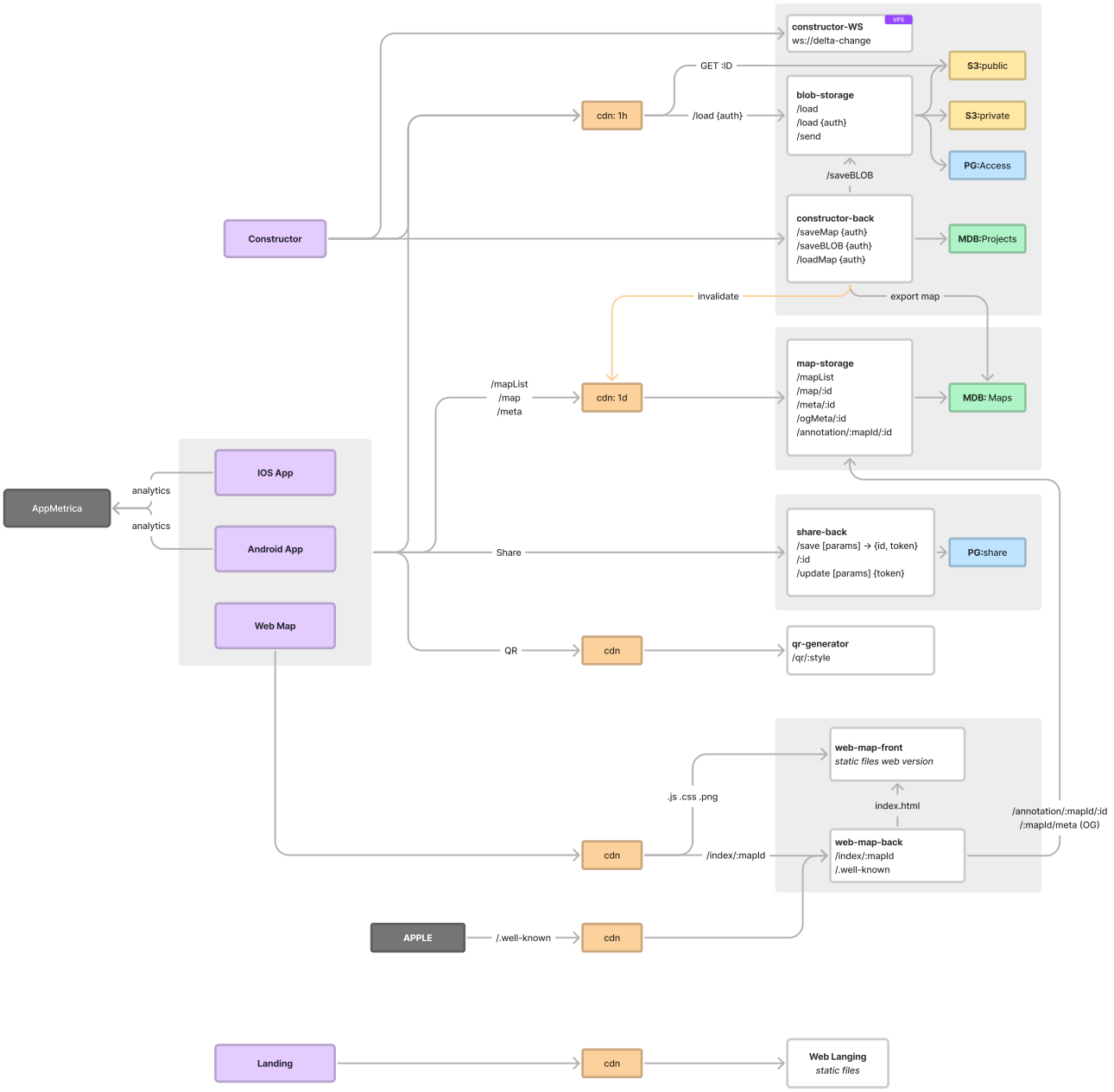


Рис. 6. Детальная схема архитектуры PolyMap

Глава 3. Реализация

3.1 Система контроля версий

При разработке проекта в микросервисном подходе выделают два способа организации системы контроля версий:

1. Монорепозиторий – все микросервисы хранятся в одном репозитории. Плюсом этого подходя является сквозное версионирование всех микросервисов, каждый коммит порождает новую общую версию приложения, которая гарантирована не нарушит совместимость. Минусом такого подхода является сильная связанность между микросервисами, что усложняет их независимую разработку, тестирование и развёртывание.
2. Полирепозиторий – каждый микросервис хранится в своём репозитории. Плюсом такого подхода является независимость разработки, тестирования и развёртывания каждого микросервиса. Минусом такого подхода является сложность в управлении версиями. Может случиться так, что при обновление одного микросервиса сломается совместимость с другим.

Я выбрал второй подход, так как он лучше позволяет разделить кодовые базы и вести независимую разработку. Сложность версионирования решается гарантиями обратной совместимости – ни одна новая версия не должна ломать совместимость с предыдущими версиями. Это требование и так необходимо соблюдать, что бы корректно работало горизонтальное масштабирование, при котором в одним момент времени могут работать несколько версий одного микросервиса.

3.1.1 Выбор системы контроля версий

В качестве хранилища системы контроля версий Git можно использовать несколько систем:

- GitHub
- GitLab
- Bitbucket

Каждая из них обладает своими плюсами и минусам. Для выбора была составлена сравнительная таблица функционала, который потребуется для разработки проекта.

TODO: Добавить сравнительную таблицу систем контроля версий

Как видно из таблицы, наибольшим числом плюсов обладает GitHub, поэтому он был выбран в качестве системы контроля версий для проекта.

В GitHub была создана отдельная огранизация, в которой хранятся все репозитории микросервисов проекта.

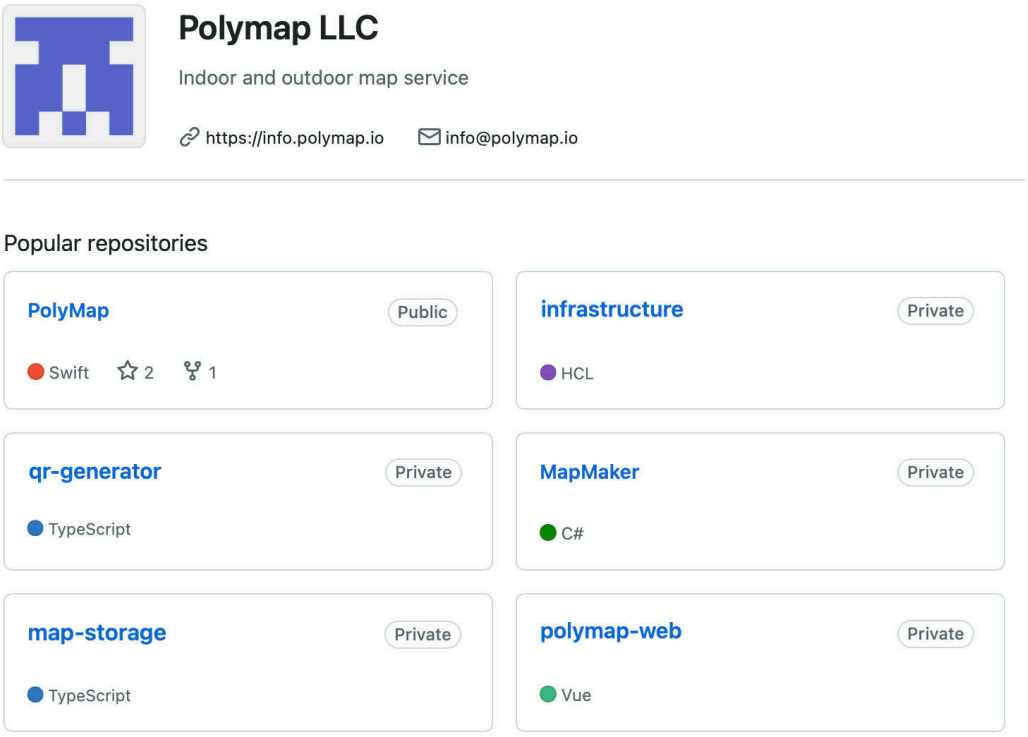


Рис. 7. Организация Polymap в GitHub

3.1.2 CI/CD

При разработке по микросервисной архитектуре крайне важно на самых ранних этапах автоматизировать процесс сборки и развёртывания приложения. Это обусловлено тем, что при таком подходе появляется множество проектов с частыми обновлениями, автоматизация рутинных процессов позволяет снизить количество ошибок, которые могут возникнуть при ручном развёртывании. А так же позволяет добавить автоматизированные тесты для контроля качества кода с самых ранних этапов.

3.1.3 GitHub Actions

В выбранной системе контроля версий GitHub, для автоматизации процессов сборки и развёртывания, используется встроенный инструмент GitHub Actions. С его помощью можно настроить различные автоматические процессы, которые будут запускаться при определённых действиях в репозитории. Процессы могут состоять из нескольких последовательных и параллельных шагов.

3.1.4 Infrastructure as Code (IAC)

Для автоматизации развёртывания приложений в CloudNative среде можно использовать IAC (Infrastructure as Code) подход. Он позволяет декларативно описать инфраструктуру в виде кода, который можно хранить рядом с кодом приложения в системе контроля версий. Это позволяет версионировать инфраструктуру вместе с кодом приложение, что в свою очередь позволяет откатить инфраструктуру к предыдущей версии, если в ней были внесены ошибки. Так же такой подход сильно упрощает развёртывание для Serverless приложений, который состоят из множества ресурсов, которые необходимо связать между собой.

3.1.5 Terraform

Для реализации IAC подхода наиболее популярным инструментом является Terraform. Все облачные провайдеры в первую очередь добавляют поддержку именно этого инструмента, в том числе и Yandex Cloud который используется в моём случае.

В Terraform инфраструктура описывается кодом на специальном языке HCL (HashiCorp Configuration Language). Базовой сущьностью в Terraform является ресурс, который описывает отдельный компонент инфраструктуры (Api GateWay, Serverless Container, Docker Registry). Внутри блока ресурса описываются его параметры. Внутри одного Terraform проекта можно ссылаться на другие ресурсы. Например, ресурс Serverless Container должен в своих параметрах ссылаться на ресурс Docker Registry, в котором хранится образ контейнера.

3.2 Клиентская часть

3.2.1 Vue

3.2.2 Three.js и WebGL

3.2.3 AppleMapKit

3.2.4 OpenGraphMeta

3.3 Серверная часть

3.3.1 Стек технологий

TODO: Указать что для Serverless круто использовать Bun

3.3.2 Выбор оптимальной технологии для Serverless

3.3.3 Микросервисы

Рассмотрим некоторые микросервисы подробнее

3.3.4 Раздача карта и CDN

Глава 4. Тестирование и оценка качества

TODO: Папйплайн CI/CD

4.1 Оценивание качества кода

4.2 Модульное тестирование

4.3 UI тестирование

4.4 End to End тестирование

4.5 Нагрузочное тестирование

Глава 5. Заключение

TODO: Анализ результатов

Заключение

Список литературы

ПРИЛОЖЕНИЕ А.
Отчёт СППР о выборе облачного провайдера

А.1. Исходные данные

	Вес	Yandex	Selectel	VK	Sber	AWS	Google	Azure
↑ **Вычисления**	2	9	6	5	7	10	9	9
↑ **Интеграции**	1	9	6	5	8	10	9	8
↑ **БД, Очереди, Уведомления**	0.8	8	7	6	8	10	9	8
↑ **DevOps & DX**	0.8	8	6	6	7	10	9	9
↑ **Мониторинг, логи**	0.9	8	6	6	8	9	9	8

Таблица 1. Исходные данные

А.1.1. Нормализованные веса

Для корректной работы алгоритма веса критериев были нормализованы.

- **Вычисления** - 0.36
- **Интеграции** - 0.18
- **БД, Очереди, Уведомления** - 0.15
- **DevOps & DX** - 0.15
- **Мониторинг, логи** - 0.16

А.2. Ход решения

Для определения оптимального варианта были построены матрицы бинарных отношений (БО), после чего к ним были применены следующие механизмы:

- Механизмы доминирования
- Механизмы блокирования
- Турнирный механизм
- Механизм K-max

По каждому механизму был выбран лучший вариант, после чего была составлена сводная таблица с результатами

А.2.1. Механизм доминирования

Сколько раз варианты доминируют по всем БО. В матричном виде выбираются варианты, у которых в строках все значения равны 1.

- **Yandex** не блокирует ни в одной категории **=> 0**
- **Selectel** не блокирует ни в одной категории **=> 0**
- **VK** не блокирует ни в одной категории **=> 0**
- **Sber** не блокирует ни в одной категории **=> 0**
- **AWS** доминируют в категориях: **Вычисления, Интеграции, БД, Очереди, Уведомления, DevOps & DX, Мониторинг, логи** => `0.36 + 0.18 + 0.15 + 0.15 + 0.16 = 1`
- **Google** доминируют в категориях: **Мониторинг, логи** => **0.16**
- **Azure** не блокирует ни в одной категории **=> 0**

Вариант	Баллы	Место
Yandex	0	3
Selectel	0	3
VK	0	3
Sber	0	3
AWS	1	1
Google	0.16	2
Azure	0	3

Таблица 2. Сводная таблица результатов механизма доминирования

A.2.2. Механизм блокирования

Сколько раз варианты блокируют по всем БО. В матричном виде выбираются варианты, у которых в столбцах все значения равны 1.

- **Yandex** не блокирует ни в одной категории **=> 0**
- **Selectel** не блокирует ни в одной категории **=> 0**
- **VK** не блокирует ни в одной категории **=> 0**
- **Sber** не блокирует ни в одной категории **=> 0**
- **AWS** блокируют в категориях: **Вычисления, Интеграции, БД, Очереди, Уведомления, DevOps & DX** => 0.36 + 0.18 + 0.15 + 0.15 = **0.84**
- **Google** не блокирует ни в одной категории **=> 0**
- **Azure** не блокирует ни в одной категории **=> 0**

Вариант	Баллы	Место
Yandex	0	2
Selectel	0	2
VK	0	2
Sber	0	2
AWS	0.84	1
Google	0	2
Azure	0	2

Таблица 3. Сводная таблица результатов механизма блокирования

A.2.3. Турнирный механизм

Сколько раз варианты предпочтительнее

- **Yandex** - в категории:
  - **Вычисления** – опережает **Selectel, VK, Sber** => 0.36 + 0.36 + 0.36 = **1.09**
    Симметрично с **Google, Azure** => 0.36 / 3 = **0.12**
  - **Интеграции** – опережает **Selectel, VK, Sber, Azure** => 0.18 + 0.18 + 0.18 + 0.18 = **0.73**
    Симметрично с **Google** => 0.18 / 2 = **0.09**
  - **БД, Очереди, Уведомления** – опережает **Selectel, VK** => 0.15 + 0.15 = **0.29**
    Симметрично с **Sber, Azure** => 0.15 / 3 = **0.05**
  - **DevOps & DX** – опережает **Selectel, VK, Sber** => 0.15 + 0.15 + 0.15 = **0.44**
  - **Мониторинг, логи** – опережает **Selectel, VK** => 0.16 + 0.16 = **0.33**
    Симметрично с **Sber, Azure** => 0.16 / 3 = **0.05**
- **Selectel** - в категории:

- **Вычисления** – опережает **VK** => **0.36**
- **Интеграции** – опережает **VK** => **0.18**
- **БД, Очереди, Уведомления** – опережает **VK** => **0.15**
- **DevOps & DX** – симметрично с **VK** => 0.15 / 2 = **0.07**
- **Мониторинг, логи** – симметрично с **VK** => 0.16 / 2 = **0.08**

- **VK** - в категории:
  - **DevOps & DX** – симметрично с **Selectel** => 0.15 / 2 = **0.07**
  - **Мониторинг, логи** – симметрично с **Selectel** => 0.16 / 2 = **0.08**
- **Sber** - в категории:
  - **Вычисления** – опережает **Selectel, VK** => 0.36 + 0.36 = **0.73**
  - **Интеграции** – опережает **Selectel, VK** => 0.18 + 0.18 = **0.36**
    Симметрично с **Azure** => 0.18 / 2 = **0.09**
  - **БД, Очереди, Уведомления** – опережает **Selectel, VK** => 0.15 + 0.15 = **0.29**
    Симметрично с **Yandex, Azure** => 0.15 / 3 = **0.05**
  - **DevOps & DX** – опережает **Selectel, VK** => 0.15 + 0.15 = **0.29**
  - **Мониторинг, логи** – опережает **Selectel, VK** => 0.16 + 0.16 = **0.33**
    Симметрично с **Yandex, Azure** => 0.16 / 3 = **0.05**
- **AWS** - в категории:
  - **Вычисления** – опережает **Yandex, Selectel, VK, Sber, Google, Azure** => 0.36 + 0.36 + 0.36 + 0.36 + 0.36 + 0.36 = **2.18**
  - **Интеграции** – опережает **Yandex, Selectel, VK, Sber, Google, Azure** => 0.18 + 0.18 + 0.18 + 0.18 + 0.18 + 0.18 = **1.09**
  - **БД, Очереди, Уведомления** – опережает **Yandex, Selectel, VK, Sber, Google, Azure** => 0.15 + 0.15 + 0.15 + 0.15 + 0.15 + 0.15 = **0.87**
  - **DevOps & DX** – опережает **Yandex, Selectel, VK, Sber, Google, Azure** => 0.15 + 0.15 + 0.15 + 0.15 + 0.15 + 0.15 = **0.87**
  - **Мониторинг, логи** – опережает **Yandex, Selectel, VK, Sber, Azure** => 0.16 + 0.16 + 0.16 + 0.16 + 0.16 + 0.16 = **0.82**
    Симметрично с **Google** => 0.16 / 2 = **0.08**
- **Google** - в категории:
  - **Вычисления** – опережает **Selectel, VK, Sber** => 0.36 + 0.36 + 0.36 = **1.09**
    Симметрично с **Yandex, Azure** => 0.36 / 3 = **0.12**
  - **Интеграции** – опережает **Selectel, VK, Sber, Azure** => 0.18 + 0.18 + 0.18 + 0.18 = **0.73**
    Симметрично с **Yandex** => 0.18 / 2 = **0.09**
  - **БД, Очереди, Уведомления** – опережает **Yandex, Selectel, VK, Sber, Azure** => 0.15 + 0.15 + 0.15 + 0.15 + 0.15 = **0.73**
  - **DevOps & DX** – опережает **Yandex, Selectel, VK, Sber** => 0.15 + 0.15 + 0.15 + 0.15 = **0.58**
    Симметрично с **Azure** => 0.15 / 2 = **0.07**
  - **Мониторинг, логи** – опережает **Yandex, Selectel, VK, Sber, Azure** => 0.16 + 0.16 + 0.16 + 0.16 + 0.16 = **0.82**
    Симметрично с **AWS** => 0.16 / 2 = **0.08**
- **Azure** - в категории:
  - **Вычисления** – опережает **Selectel, VK, Sber** => 0.36 + 0.36 + 0.36 = **1.09**
    Симметрично с **Yandex, Google** => 0.36 / 3 = **0.12**
  - **Интеграции** – опережает **Selectel, VK** => 0.18 + 0.18 = **0.36**
    Симметрично с **Sber** => 0.18 / 2 = **0.09**
  - **БД, Очереди, Уведомления** – опережает **Selectel, VK** => 0.15 + 0.15 = **0.29**
    Симметрично с **Yandex, Sber** => 0.15 / 3 = **0.05**

- **DevOps & DX** – опережает **Yandex, Selectel, VK, Sber** => `0.15 + 0.15 + 0.15 + 0.15 = 0.58`
  Симметрично с **Google** => `0.15 / 2 = 0.07`
- **Мониторинг, логи** – опережает **Selectel, VK** => `0.16 + 0.16 = 0.33`
  Симметрично с **Yandex, Sber** => `0.16 / 3 = 0.05`

Вариант	Баллы	Место
Yandex	3.19	3
Selectel	0.85	6
VK	0.15	7
Sber	2.19	5
AWS	5.92	1
Google	4.31	2
Azure	3.04	4

Таблица 4. Сводная таблица результатов турнирного механизма

A.2.4. Механизм K-max

	HRo+ ER+ NR	HRo+ NR	HRo+ ER	HRo	Sjp	Sjm
Yandex	5	3	5	3	16 * 0.36 = 5.82	-
Selectel	1	1	1	1	4 * 0.36 = 1.45	-
VK	0	0	0	0	0	-
Sber	2	2	2	2	8 * 0.36 = 2.91	-
AWS	6	6	6	6	24 * 0.36 = 8.73	8.73 (Строго наиб.)
Google	5	3	5	3	16 * 0.36 = 5.82	-
Azure	5	3	5	3	16 * 0.36 = 5.82	-

Таблица 5. Таблица результатов механизма K-max для категории Вычисления

	HRo+ ER+ NR	HRo+ NR	HRo+ ER	HRo	Sjp	Sjm
Yandex	5	4	5	4	18 * 0.18 = 3.27	-
Selectel	1	1	1	1	4 * 0.18 = 0.73	-
VK	0	0	0	0	0	-
Sber	3	2	3	2	10 * 0.18 = 1.82	-
AWS	6	6	6	6	24 * 0.18 = 4.36	4.36 (Строго наиб.)
Google	5	4	5	4	18 * 0.18 = 3.27	-
Azure	3	2	3	2	10 * 0.18 = 1.82	-

Таблица 6. Таблица результатов механизма K-max для категории Интеграции

	HRo+ ER+ NR	HRo+ NR	HRo+ ER	HRo	Sjp	Sjm
Yandex	4	2	4	2	12 * 0.15 = 1.75	-
Selectel	1	1	1	1	4 * 0.15 = 0.58	-
VK	0	0	0	0	0	-
Sber	4	2	4	2	12 * 0.15 = 1.75	-
AWS	6	6	6	6	24 * 0.15 = 3.49	3.49 (Строго наиб.)
Google	5	5	5	5	20 * 0.15 = 2.91	-
Azure	4	2	4	2	12 * 0.15 = 1.75	-

Таблица 7. Таблица результатов механизма K-max для категории БД, Очереди, Уведомления

	HRo+ ER+ NR	HRo+ NR	HRo+ ER	HRo	Sjp	Sjm
Yandex	3	3	3	3	12 * 0.15 = 1.75	-
Selectel	1	0	1	0	2 * 0.15 = 0.29	-
VK	1	0	1	0	2 * 0.15 = 0.29	-
Sber	2	2	2	2	8 * 0.15 = 1.16	-
AWS	6	6	6	6	24 * 0.15 = 3.49	3.49 (Строго наиб.)
Google	5	4	5	4	18 * 0.15 = 2.62	-
Azure	5	4	5	4	18 * 0.15 = 2.62	-

Таблица 8. Таблица результатов механизма K-max для категории DevOps & DX

	HRo+ ER+ NR	HRo+ NR	HRo+ ER	HRo	Sjp	Sjm
Yandex	4	2	4	2	12 * 0.16 = 1.96	-
Selectel	1	0	1	0	2 * 0.16 = 0.33	-
VK	1	0	1	0	2 * 0.16 = 0.33	-
Sber	4	2	4	2	12 * 0.16 = 1.96	-
AWS	6	5	6	5	22 * 0.16 = 3.6	-
Google	6	5	6	5	22 * 0.16 = 3.6	-
Azure	4	2	4	2	12 * 0.16 = 1.96	-

Таблица 9. Таблица результатов механизма K-max для категории Мониторинг, логи

Вариант	Сумма sJp	Место sJp	Сумма sJm	Место sJm
Yandex	14.55	3	0	2
Selectel	3.38	6	0	2
VK	0.62	7	0	2
Sber	9.6	5	0	2
AWS	23.67	1	20.07	1
Google	18.22	2	0	2
Azure	13.96	4	0	2

Таблица 10. Сводная таблица результатов механизма K-max

A.3. Результат

По результатам всех механизмов, в зависимости от полученного места были начислены баллы каждому варианту

Вариант	Дом	Блок	Typ	Sjp	Sjm	ИТОГО	Место
Yandex	5	6	5	5	6	27	3
Selectel	5	6	2	2	6	21	6
VK	5	6	1	1	6	19	7
Sber	5	6	3	3	6	23	5
AWS	7	7	7	7	7	35	1
Google	6	6	6	6	6	30	2
Azure	5	6	4	4	6	25	4

Таблица 11. Итоговая таблица результатов

A.3.1. Итоговый вариант

Максимальную сумму баллов набрал вариант **AWS** с суммой **35** баллов

ПРИЛОЖЕНИЕ Б.
ТЕСТ

тест