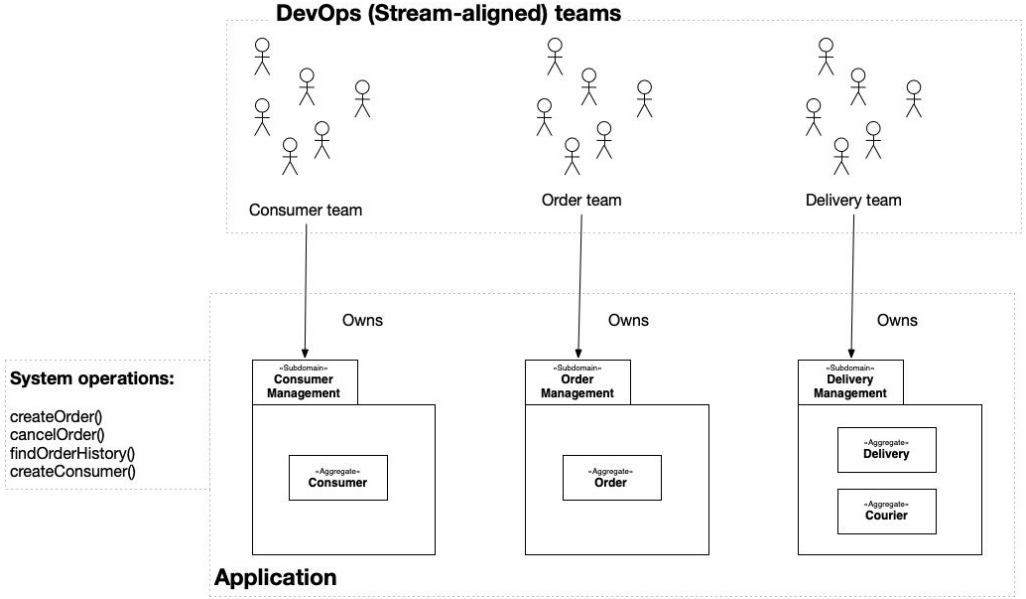
Microservices

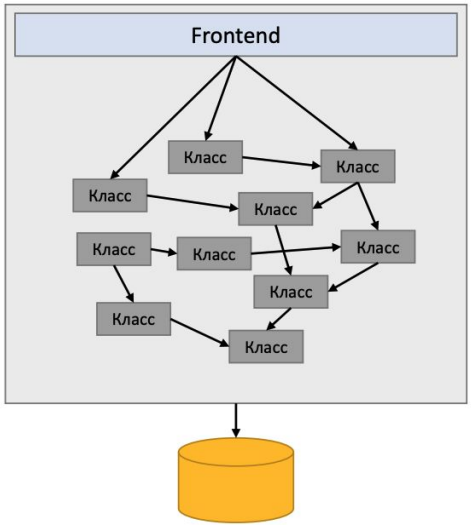
А в чем проблема?

- Вам нужно доставлять изменения быстро, часто и надежно (по Agile)
- Команда отвечает за один или несколько поддоменов.
- Как организовать поддомены в один или несколько развертываемых/исполняемых компонентов?

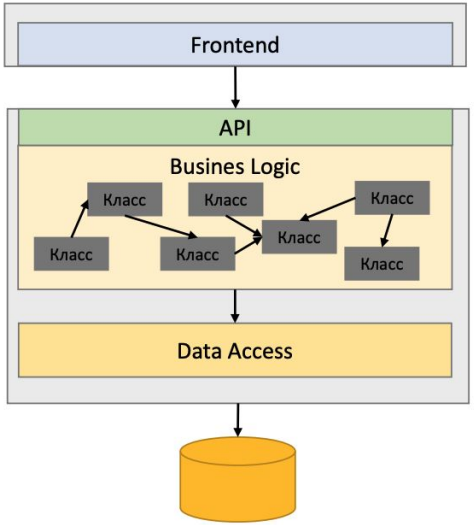


***Поддомен** — это реализуемая модель части бизнес-функций, также известной как бизнес-возможность.*
Поддомены реализуют поведение приложения, состоящее из набора (системных) операций.

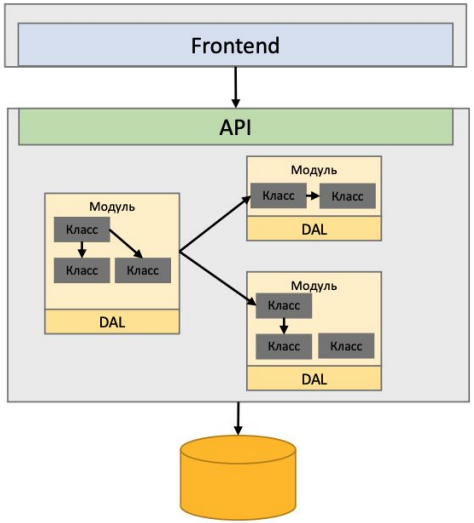
Монолитная архитектура



Big Ball of Mud



Layers arch



Modules arch

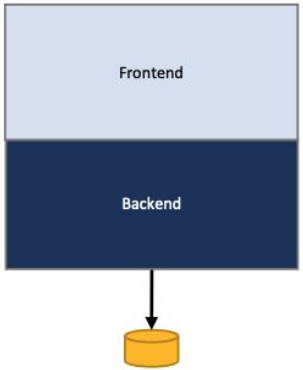
Преимущества Монолита

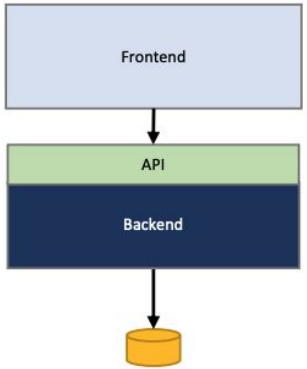
- **Меньше зависимости от других команд (*)**: вы можете сделать все, что вам нужно, самостоятельно
- **Простота развертывания (*)**: вам нужно развернуть только один монолит
- **Меньше требований к компетенциям**: мы следуем уже заранее заложенным арх стандартам
- **ACID транзакционность**: вся транзакция происходит в одном месте

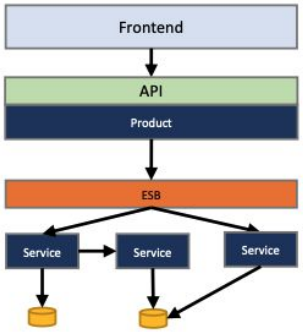
Недостатки Монолита

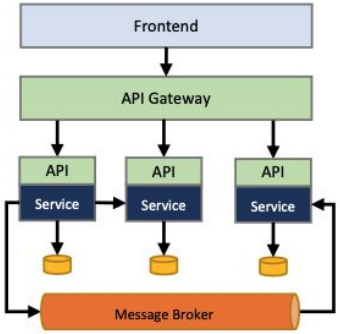
- **Сложность**: 1) в понимании (много кода), 2) внесения изменений (MR встают в очередь, они конфликтуют), 3) тестировании, 4) управлении ЖЦ
- **High Coupling**: сложно понять, как взаимодействуют различные части приложения, что приводит к увеличению времени разработки и повышению риска ошибок.
- **Масштабирование**: особенно когда некоторые компоненты должны обрабатывать большой объем трафика.
- **Привязанность к тех стеку** (который заложен исторически)
- **Развертывание**: сложный и трудоемкий процесс.
- **Отказоустойчивость:** если один модуль упадет, то завалится весь монолит

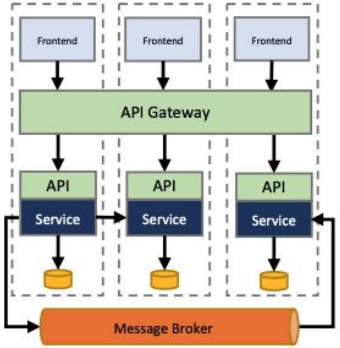
Возможные архитектуры









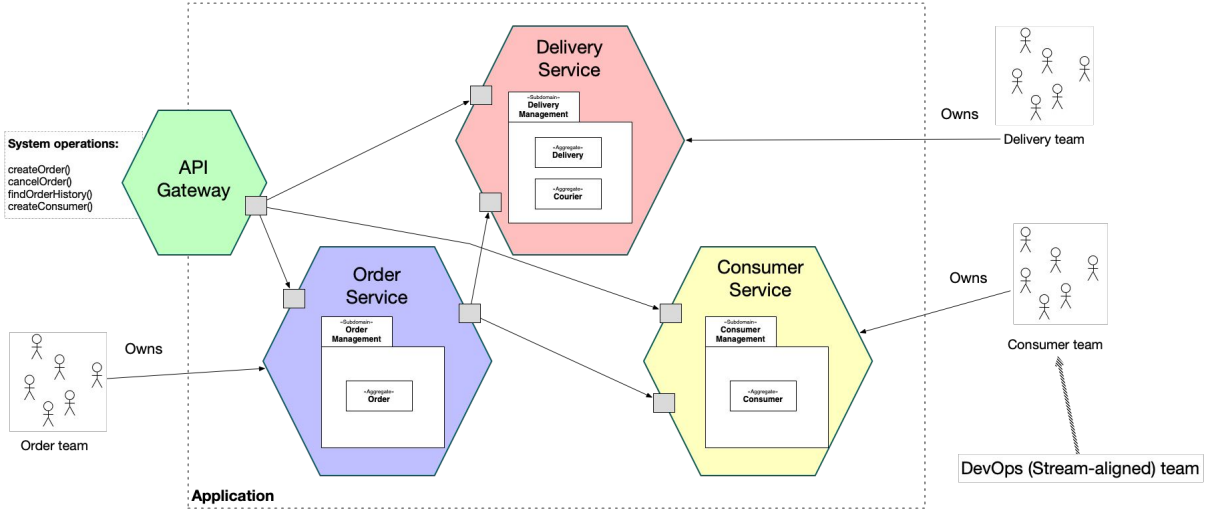


Монолит	Frontend + Монолит	SOA	Микросервисы	Микрофронтенды
Frontend и Backend единое целое	Frontend и Backend разворачивается отдельно, Backend предоставляет некоторое API, но все еще единый и тяжелый	Отдельные сервисы, корпоративная шина (ESB), и сервисы с API по продуктам. Пример: **Сервисы**: 1) услуг 2) оплата 3) чеки **Продукт**: оплата услуги с выдачей чека	Наследие SOA, но нет копр шины, все каналы коммуникации примитивные и без бизнес логики (API Gateway / Прокси, Message Broker), вся логика в **микросервисах**	Доп. декомпозиция Frontend, где каждому **микрофронту** соответствует свой Backend микросервис

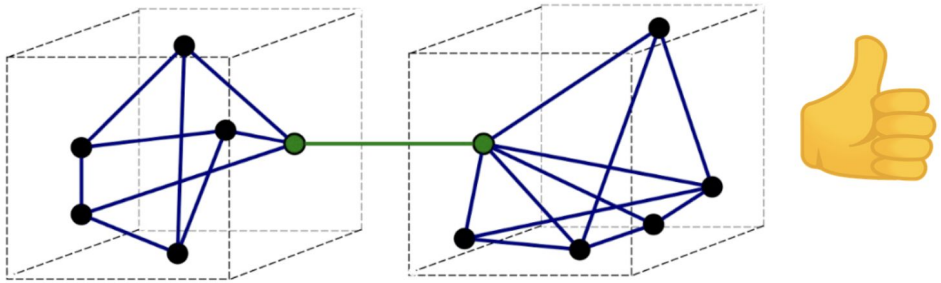
MSA

Архитектура, которая структурирует приложение как набор независимо развертываемых, слабо связанных компонентов, также называемых **сервисами**. Каждый сервис состоит из одного или нескольких поддоменов. Сервисы должны иметь low coupling и high functional cohesion.

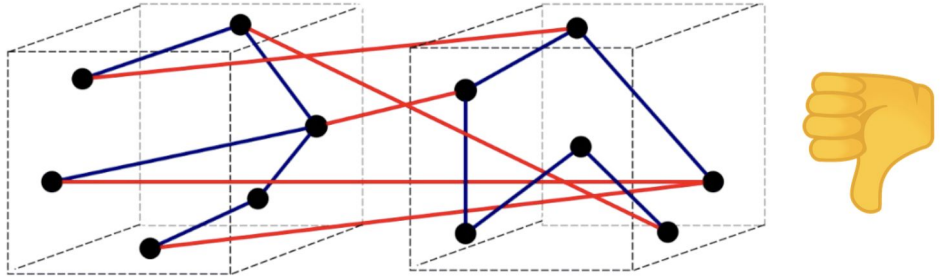
Для независимости развертывания сервис имеет
- свой собственный репозиторий исходного кода
- свой собственный конвейер развертывания (для сборки, тестирования и деплоя)



Правило декомпозиции



a) Low Coupling (низкая связанность) и High Cohesion (высокое зацепление)



b) High Coupling (высокая связанность) и Low Cohesion (низкое зацепление)

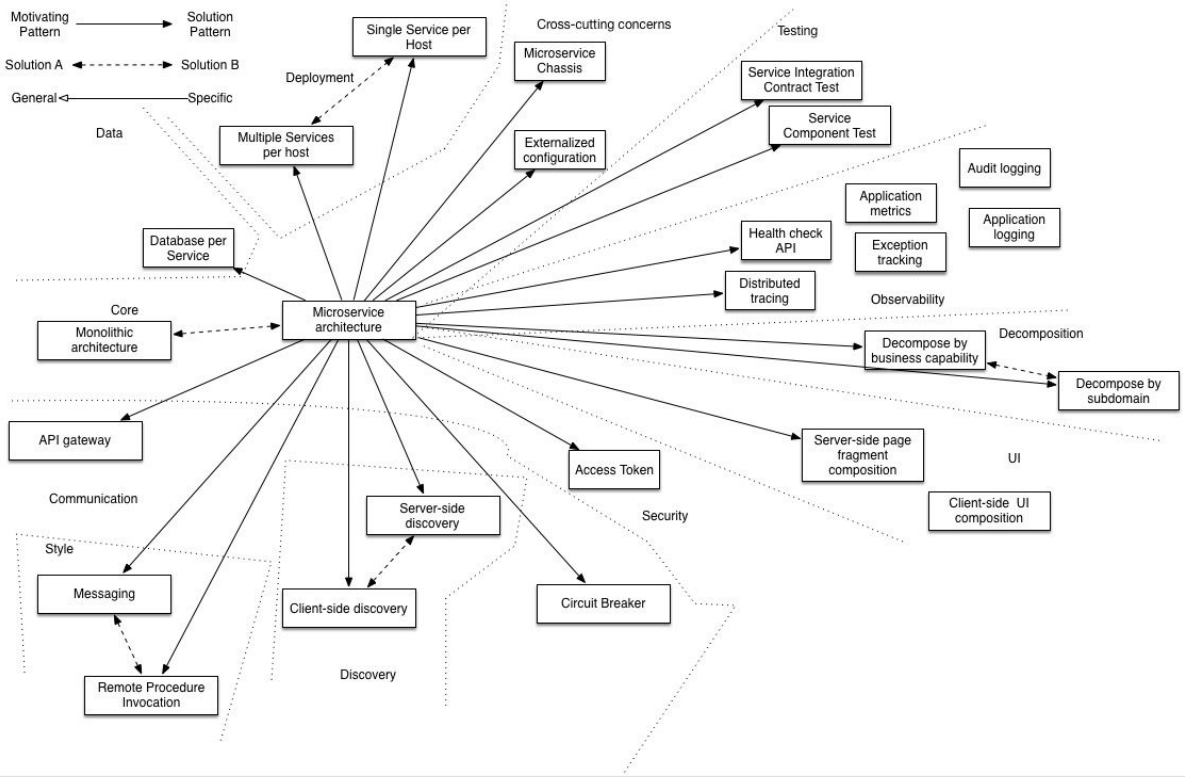
Преимущества

- **Simple services** - состоят из небольшого количества поддоменов (возможно, даже одного), поэтому их легче понимать и поддерживать.
- **Team autonomy** - команда может разрабатывать, тестировать и развертывать свою службу независимо от других команд.
- **Fast deployment pipeline** - быстро тестируются, поскольку они относительно небольшие и могут быть развернуты независимо.
- **Support multiple technology stacks** - могут использовать разные технологические стеки и могут обновляться независимо.
- **Segregate subdomains by their characteristics** - поддомены могут быть разделены по своим характеристикам на отдельные сервисы с целью улучшения масштабируемости, доступности, безопасности и т. д.

Недостатки

- Сложность **правильной декомпозиции**
- Присущи все сопутствующие сложности **распределенных систем**.
- При взаимодействии между различными сервисами возрастает **вероятность сбоев**.
- **Сложно управлять** большим количеством сервисов**.**
- Разработчику необходимо решить такие проблемы, как **задержка сети** и **балансировка нагрузки**.
- Некоторые распределенные операции могут быть связаны тесной связью между сервисами во время выполнения, что снижает их **доступность**.
- Каждый микросервис отвечает за сохранение своих данных. В результате обеспечение **согласованности данных** может стать проблемой.
- **Сложное тестирование** в распределенной среде.

Связанные паттерны

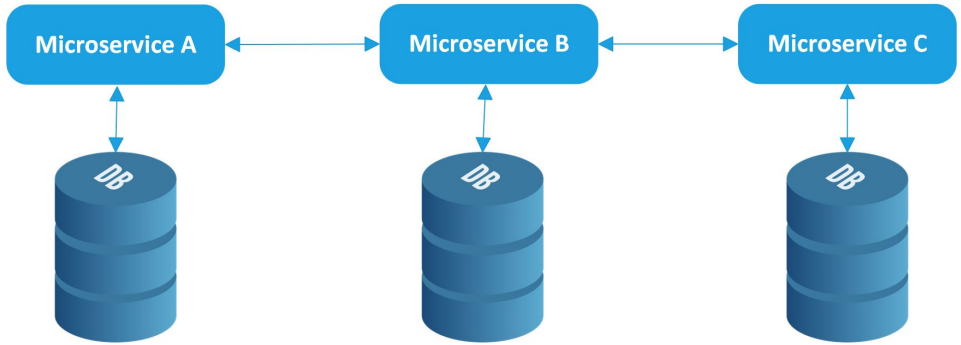


Паттерн: Database Per Service

Предоставить каждому сервису собственное хранилище данных, чтобы не было сильных зависимостей на уровне данных. При этом имеется в виду именно логическое разделение данных.

Решает проблемы

- Устранения влияния на данные
- Распределение нагрузки на БД
- Сохраняются bounded context-ы, тк сущности не пересекаются
- Технология может быть выбрана любая



Private-tables-per-service
Schema-per-service
Database-server-per-service

Другие типичные компоненты

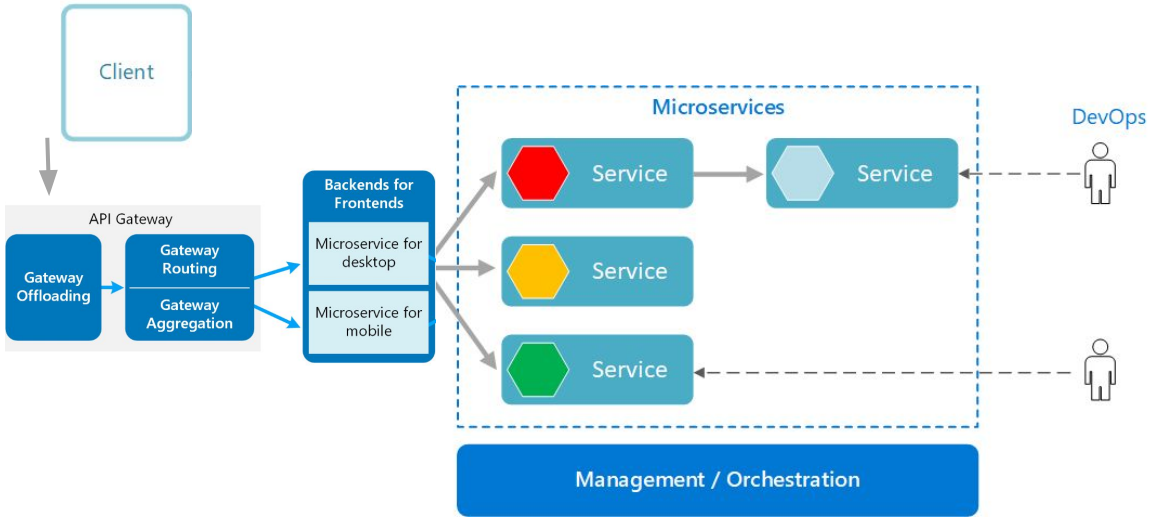
Помимо самих сервисов, в типичной архитектуре микросервисов присутствуют и некоторые другие компоненты:

Management/orchestration.
- размещение сервисов на узлах
- выявление сбоев
- балансировка нагрузки и т. д.

(ex, Kubernetes)

API Gateway - единая точка входа для клиента (routing, aggregation, offloading)

BFF - API Gateways для различных видов клиентов

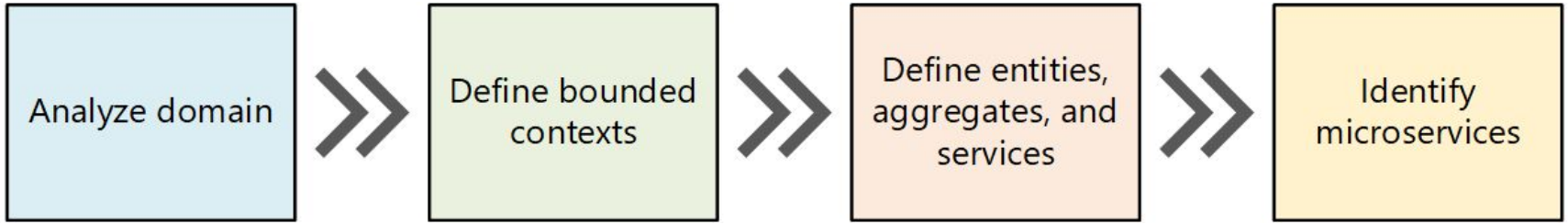


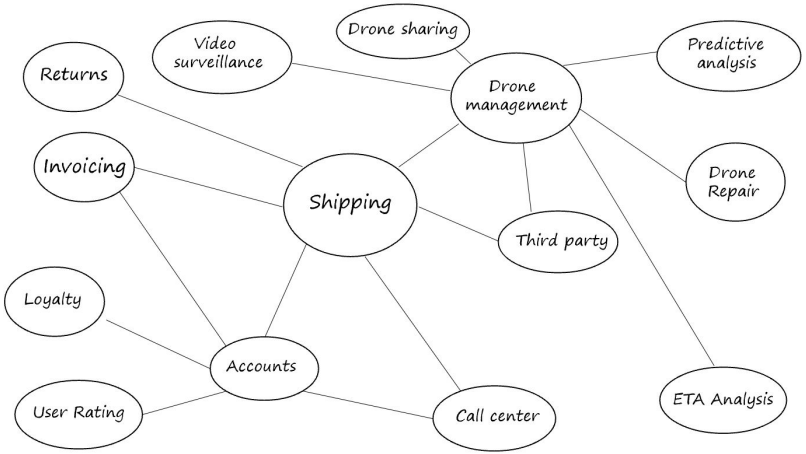
*Для успешного использования микросервисов необходима культура **DevOps**.*

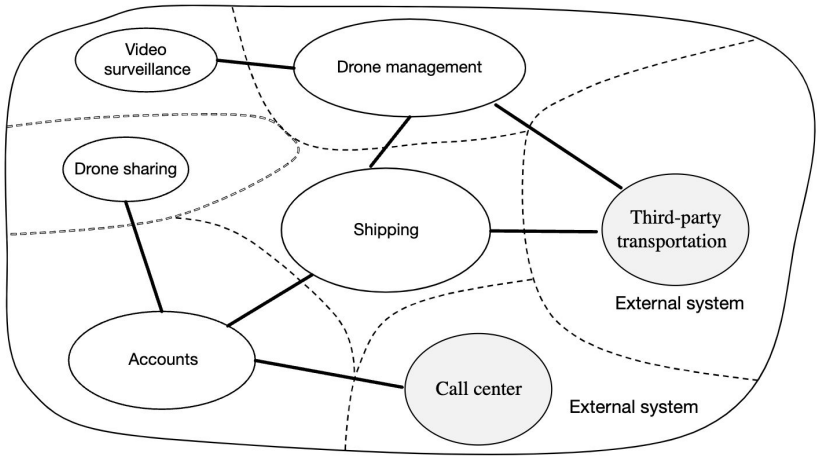
System design

1. Scope Refinement - общее прикидки по системе
2. Functional requirements - функциональные требования
3. NFR - не функциональные требования (например, наблюдаемость)
4. Load estimation* - подсчет трафика и данных
5. ***Define bounded contexts****
6. High-lvl design
7. Component design
8. Database selection
9. Scalability

Define bounded contexts (DDD)







Демонстрация: eCommerce Shop

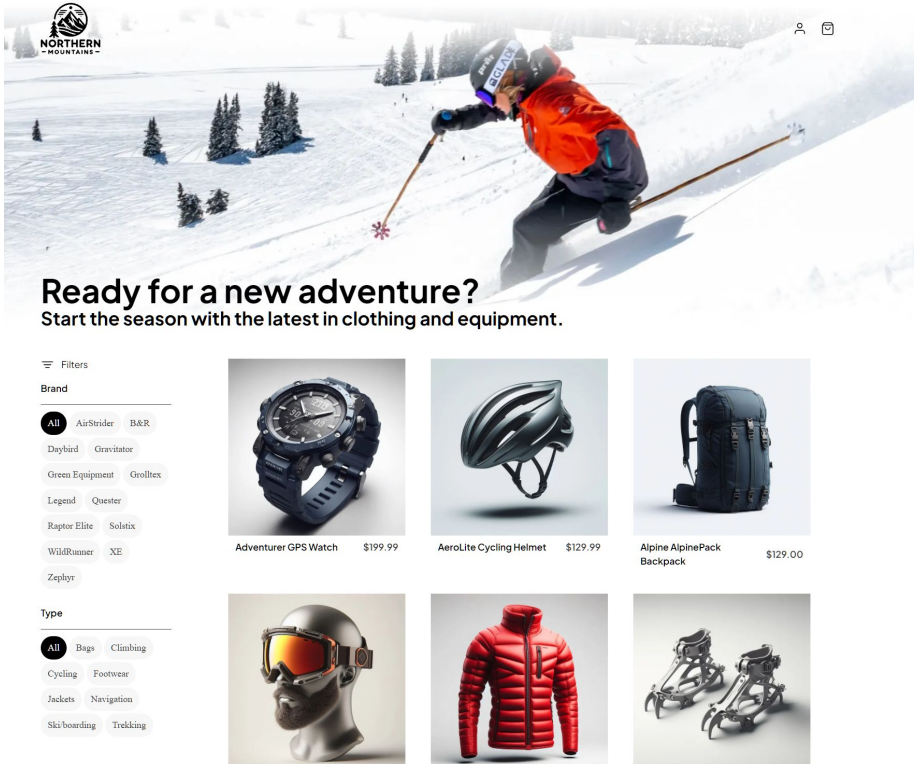
Functional Req:

- Login/out to site
- Search in the catalog
- Add items to basket
- Pay of order

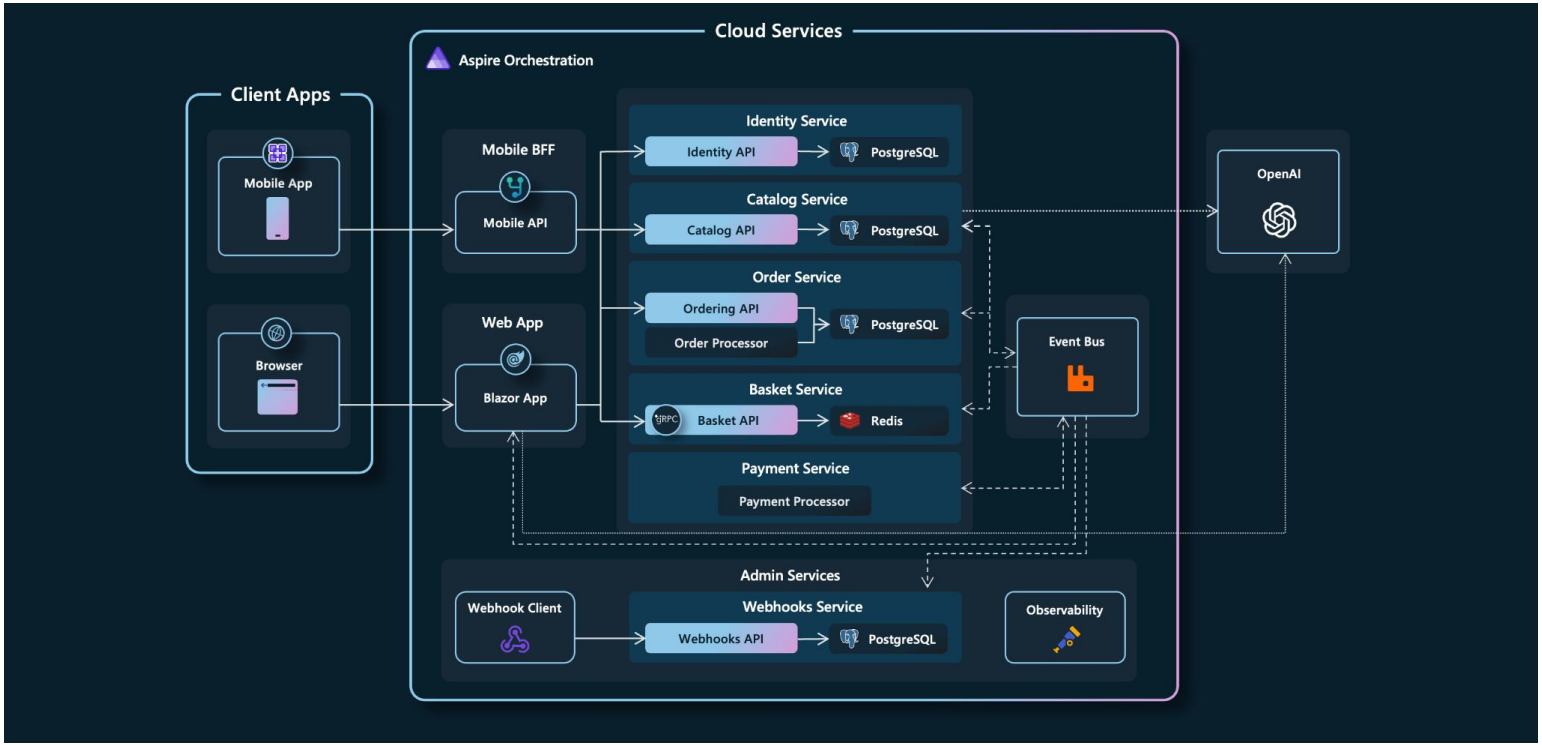
Non-Functional Req

- Observability
- Few clients: web, mobile

GitHub: https://github.com/dotnet/eShop



Демонстрация: eCommerce Shop High-lvl design



Задание: MSA High-lvl design

Оценка: 2

Описание: Выберите архитектурный кейс (например, из ArchKatas или классические - WhatsApp, Netflix, Uber etc.). Декомпозируйте функциональные требования на микросервисы и сделайте небольшие оценки трафика и данных. Спроектируйте MSA, используя паттерны.

DoD: UML или другие виды диаграмм для описания системы + видеоотчет.