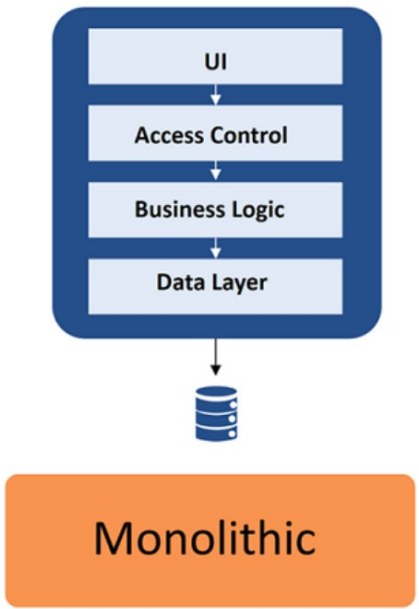
Микросервисы

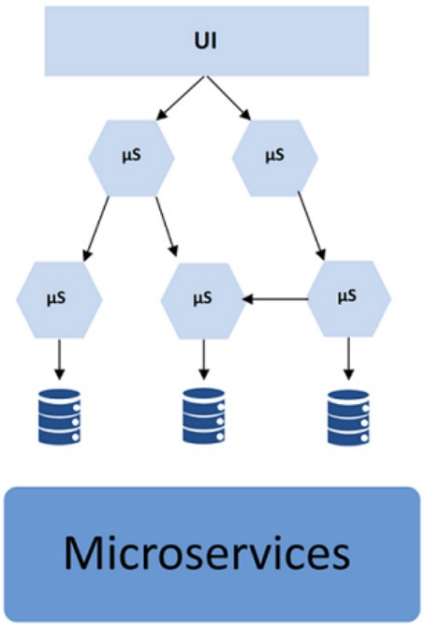
Содержание

- Микросервесная архитектура
- Преимущества и недостатки
- Обмен сообщениями
- Форматы сообщений
- Децентрализованное управление данными
- Вспомогательные сервисы
- Развёртывание программного обеспечения
- Безопасность

Монолитная архитектура VS Микросервисная



Monolithic



Microservices

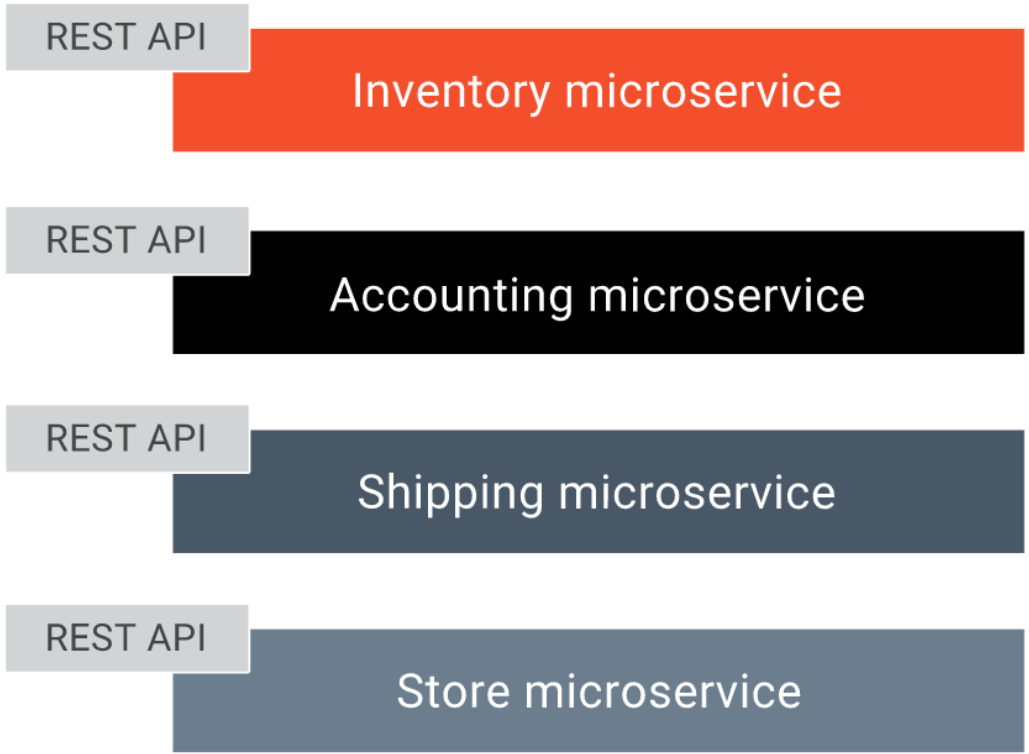
Преимущества микросервисной архитектуры

- Горизонтальная маштобируемость
- Модульность
- Изоляця компонентов
- Отсутсвие приложености к одному технологическому стеку
- Не страшно экспериментировать с новыми технологиями
- В каком-то смысле легкий вход для новых сотрудников
- Существует множество написанных сервисов и инструментов

Недостатки микросервисной архитектуры

- **Разработка распределенных систем**
- **Множество баз данных и управление транзакций**
- **Тестирование микросервисных приложений**
- **Монтирование приложений**

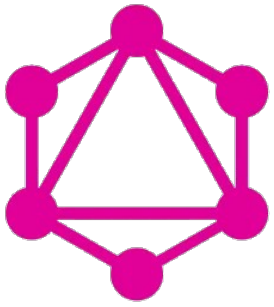
Обмен сообщениями в микросервисах



Синхронный обмен сообщениями

{ REST-API }

gRPC



GraphQL

Синхронный обмен сообщениями. REST

(REpresentational State Transfer) — это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP

Автор идеи и термина Рой Филдинг 2000г.

REST вытеснил остальные подходы, в том числе дизайн основанный на SOAP и WSDL

Синхронный обмен сообщениями. gRPC

gRPC технология межпроцессного взаимодействия и альтернативы сервисам RESTful.

Предназначена для подключения, вызова, управления и отлаживание распределенные гетерогенных приложении так же легко, как и выполнять локальный вызов функции.

В отличие от REST, можено определить контракт на обслуживание, используя язык определения интерфейса (IDL) на основе буфера протокола gRPC, а затем сгенерировать код сервиса и клиента для предпочитаемого языка программирования (Go, Java, Node ...).

Синхронный обмен сообщениями. GraphQL

GraphQL - это язык запросов для API и среда выполнения для выполнения этих запросов существующими данными.

GraphQL популярен для некоторых случаев использования, когда нет фиксированного контракта на обслуживание.

Отличается от взаимодействия клиент-сервер, посколькуклиенты определяют, данные и формат данных

Синхронный обмен сообщениями. Thrift

Thrift — язык описания интерфейсов, который используется для определения и создания служб под разные языки программирования. Является фреймворком к удалённому вызову процедур.

Используется Thrift в качестве альтернативы синхронному обмену сообщениями REST/HTTP.

Ассинхронный объмен сообщениями

Ассинхронный объмен сообщениями. Kafka

Apache Kafka — брокер сообщений, реализующий паттерн Producer-Consumer с хорошими способностями к горизонтальному масштабированию.

Обеспечивающая наращивание пропускной способности как при росте числа и нагрузки со стороны источников, так и количества систем-подписчиков.

Подписчики могут быть объединены в группы.

Это Open Source разработка, созданная компанией LinkedIn на JVM стеке (Scala).

Ассинхронный объмен сообщениями. NATS

NATS - это система обмена сообщениями с открытым исходным кодом.

Сервер NATS написан на языке программирования Go.

Клиентские библиотеки для взаимодействия с сервером доступны для десятков основных языков программирования.

Ассинхронный объмен сообщениями. AMQP

AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы.

Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию.

AMQP - это двоичный протокол. Информация организуется в фреймы различных типов. Фреймы содержат методы протокола и другую информацию. Все кадры имеют один и тот же общий формат: заголовок кадра, полезная нагрузка и конец кадра. Формат полезной нагрузки кадра зависит от типа кадра.

Ассинхронный объмен сообщениями. MQTT

MQTT (Message Queuing Telemetry Transport) — упрощённый сетевой протокол, работающий поверх TCP/IP, ориентированный для обмена сообщениями между устройствами по принципу издатель-подписчик.

Основное предназначение — работа с телеметрией от различных датчиков, устройств, использование шаблона подписчика обеспечивает возможность устройствам выходить на связь и публиковать сообщения, которые не были заранее известны или предопределены, в частности, протокол не вводит ограничений на формат передаваемых данных.

Ассинхронный объмен сообщениями. STOMP

STOMP (Simple Text Oriented Message Protocol) - ранее известный как TTMP, представляет собой простой текстовый протокол, разработанный для работы с промежуточным программным обеспечением, ориентированным на сообщения.

Форматы сообщений

JSON

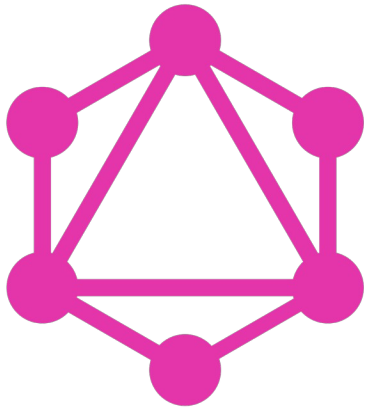
protobuf
Protocol Buffers

XML

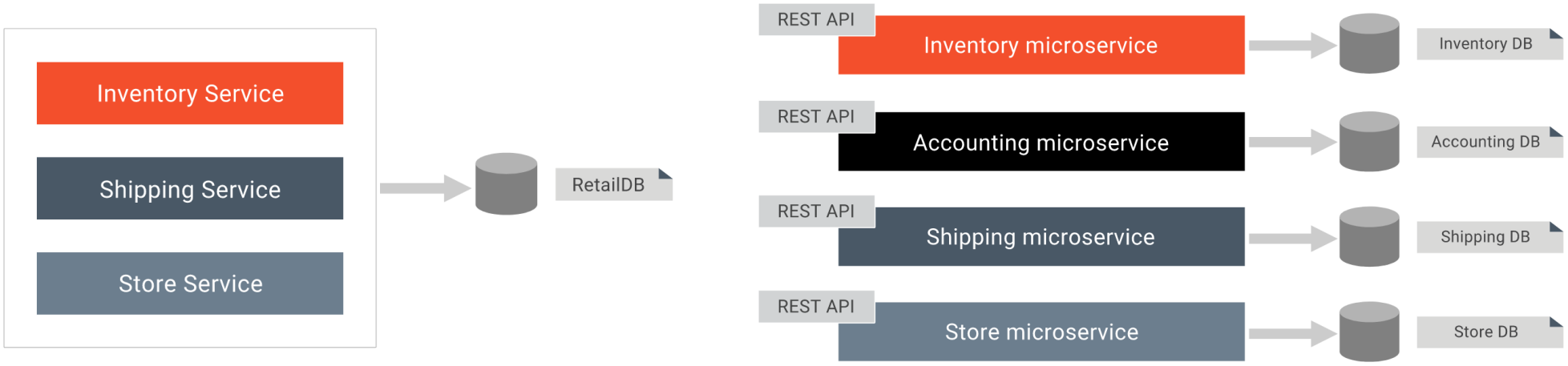
AVRO™

FlatBuffers
An open source project by FPL

Service Contracts



Децентрализованное управление данными



Service Registry и Service Discovery

Service Registry

Содержит метаданные экземпляров микросервиса (которые включают фактическое расположение микросервиса, такие как порт хоста, ip-адресс). Микросервисные экземпляры регистрируются в реестре служб при запуске и отменяются при завершении работы.

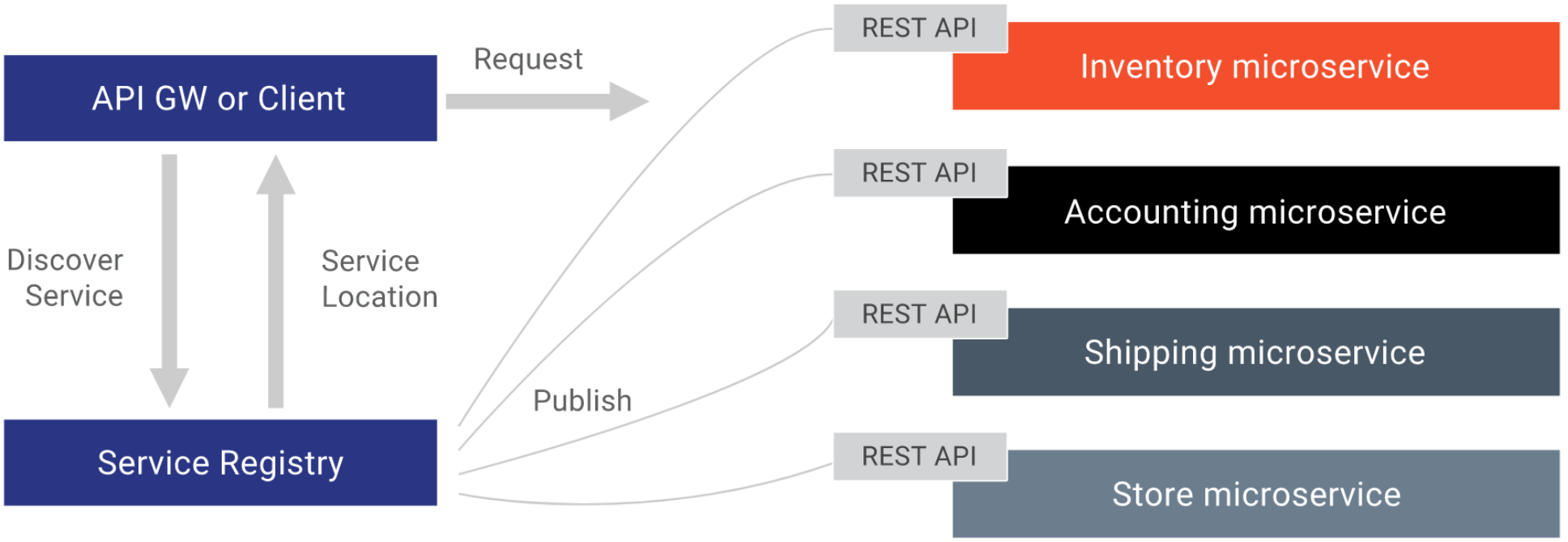
Потребители сервисов находят доступные микросервисы и их местоположение через реестр услуг.

Service Discovery

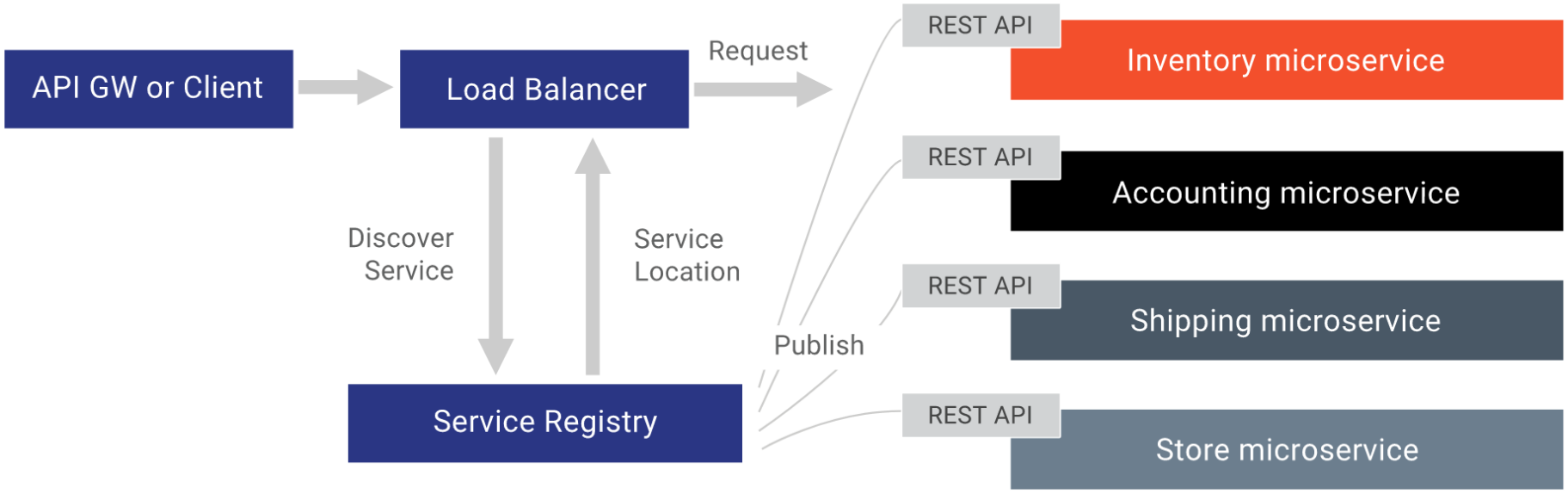
Чтобы найти доступные микросервисы и их местоположение, нужен механизм обнаружения сервисов.

Есть два типа механизмов обнаружения служб - обнаружение на стороне клиента и обнаружение на стороне сервера.

Обнаружение на стороне клиента (Client-side discovery)



Обнаружение на стороне сервера (Server-side discovery)



Развёртывание программного обеспечения (Deployment)

Развертывание микросервисов играет важную роль и имеет следующие ключевые требования:

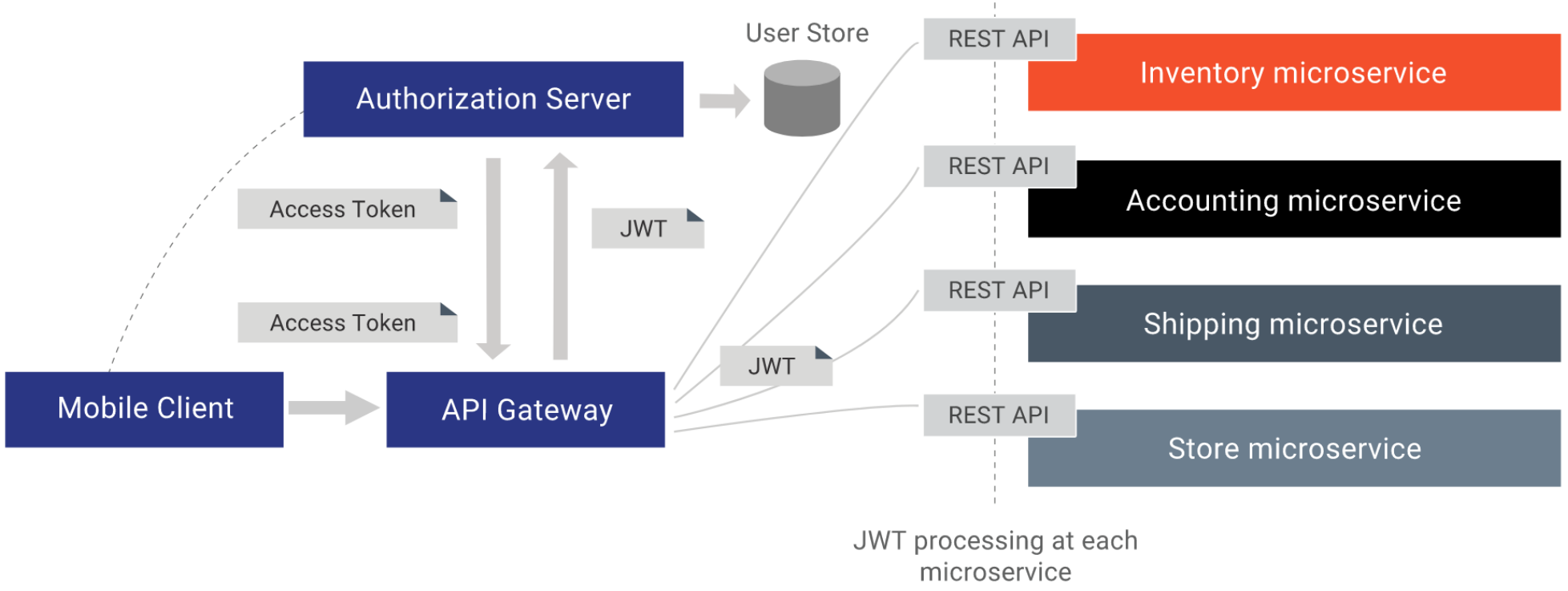
- **Возможность развертывания / отмены развертывания независимо от других микросервисов**
- **Должен иметь возможность масштабирования на каждом уровне микросервисов (данный сервис может получать больше трафика, чем другие сервисы)**
- **Быстрое развертывание микросервисов**
- **Сбой в одном микросервисе не должен влиять на другие сервисы**

Безопасность Монолитного приложения

В типичном монолитном приложении безопасность заключается в поиске «кто является вызывающим абонентом», «что может делать вызывающий абонент» и «как мы можем распространять эту информацию».

Обычно это реализуется в общем компоненте безопасности, который находится в начале цепочки обработки запросов. Этот компонент заполняет необходимую информацию с использованием основного пользовательского репозитория (или пользовательского хранилища).

Безопасность в Микросервисной архитектуре



Безопасность в Микросервисной архитектуре

- **Использовать OAuth 2.0 и OIDC-сервер, как сервер авторизации. В таком случае доступ к микросервисам будет успешно предоставлен, если кто-то имеет право использовать данные.**
- **Использовать стиль API-шлюза, в котором существует единая точка входа для всех клиентских запросов.**
- **Клиент подключается к серверу авторизации и получает токен доступа (токен по ссылке). Затем отправляет токен доступа к API-шлюзу вместе с запросом.**
- **Преобразование токена на шлюзе - API-шлюз извлекает токен доступа и отправляет его на сервер авторизации для получения JWT (токену по значению).**
- **Шлюз передает этот JWT вместе с запросом на уровень микросервисов.**
- **JWT содержат необходимую информацию для хранения пользовательских сеансов и других данных. Если каждая служба может понимать JWT, то вы распространили механизм идентификации, который позволяет вам передавать идентификационные данные по всей вашей системе.**
- **На каждом уровне микросервиса может быть компонент, который обрабатывает JWT, что является довольно тривиальной реализацией.**

Service Mesh