

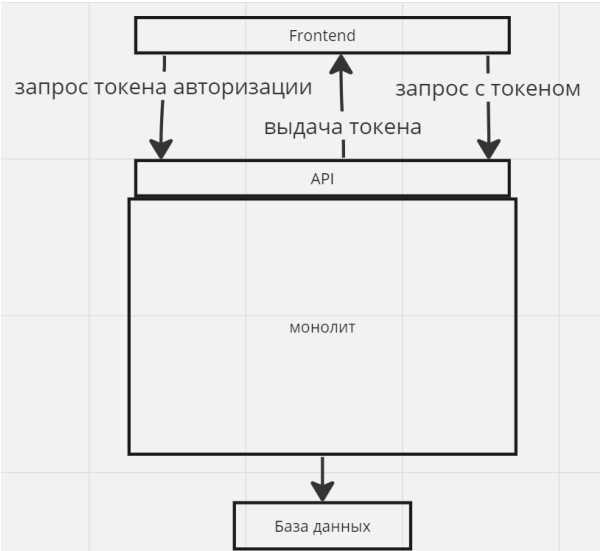
Обсудили с вами что такое микросервисы, цель внедрения, и паттерны создания микросервисов.

Рассмотрим паттерн аутентификации и авторизации (Access Token).

Access Token — это часть механизма аутентификации и авторизации, который используется для подтверждения того, что пользователь или система имеют право доступа к определенным ресурсам. В простейшем случае это строка символов, которую сервер выдает клиенту после успешной аутентификации. Далее клиент использует этот токен для доступа к защищенным ресурсам, предоставляя его в заголовке HTTP-запроса.

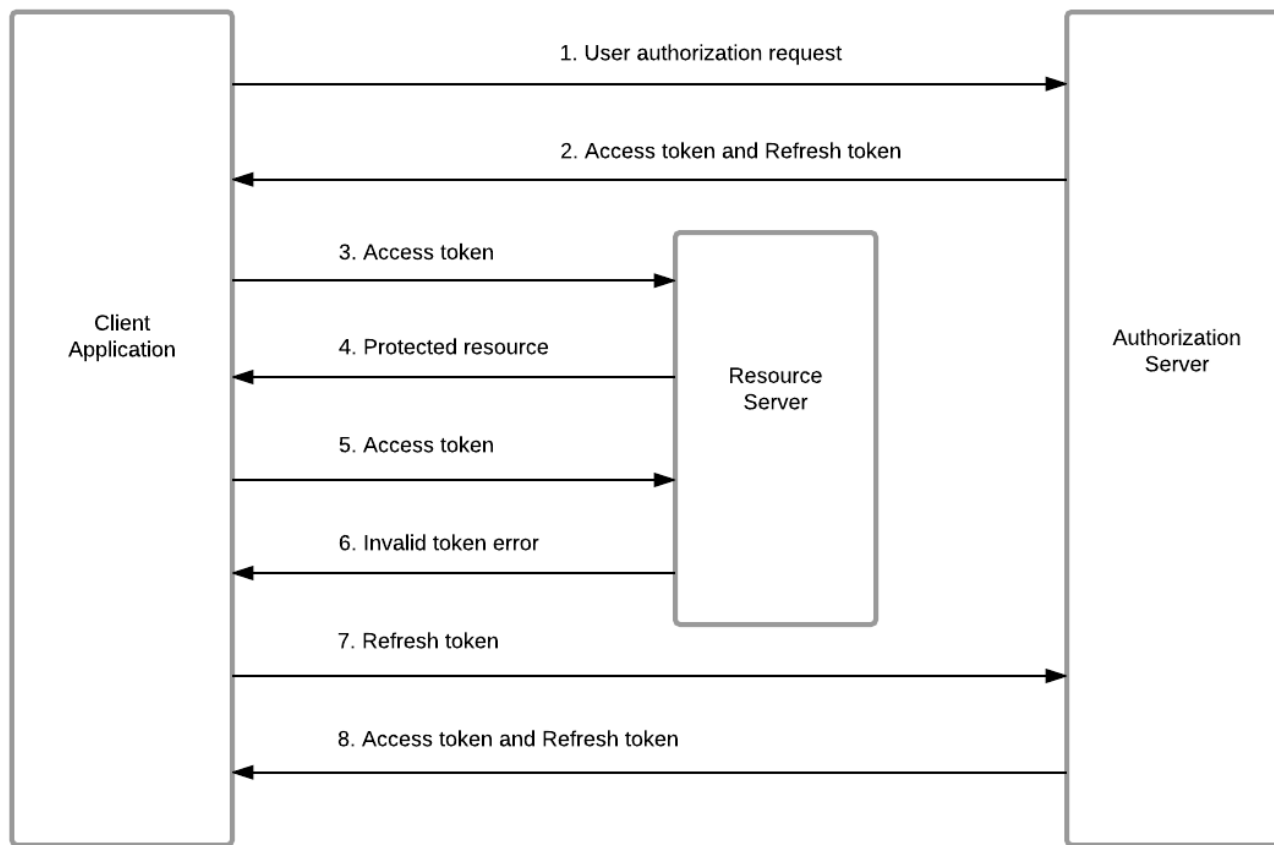
Как это работает в монолите

В монолитных системах токены часто используются для управления сессиями пользователя. После входа в систему пользователь получает токен, который сохраняется на клиенте (например, в cookie). При каждом следующем запросе токен отправляется обратно на сервер, где он проверяется, и в случае валидности предоставляется доступ к ресурсам.



Зачем этот паттерн в микросервисах

У вас теперь не один сервис, а очень много - и что, теперь каждый бэкенд микросервис должен уметь создавать токен? Это очень большая нагрузка. Поэтому выделяют отдельный один сервис авторизации, а все остальные микросервисы учат только проверять токены и всё.



На картинке видно, что выделен отдельный сервис аутентификации, а Resource server - это как раз наш обычный микросервис, который умеет только проверять токен. И таких Resource server(микросервисов) у нас много, но сервис аутентификации - один.

В микросервисной архитектуре Access Token особенно полезен:

1. **Децентрализация:** Микросервисы могут быть развернуты независимо и работать в разных средах. Токен позволяет каждому микросервису самостоятельно проверить права доступа без необходимости обращения к центральной точке аутентификации.
2. **Скорость и масштабируемость:** Так как каждый микросервис может самостоятельно проверить токен, это уменьшает латентность и увеличивает производительность системы.
3. **Безопасность:** Токены могут быть временными и содержать различную информацию, которая помогает валидировать запрос.

Как он реализуется

1. **Сервер Аутентификации:** Один из микросервисов является сервером аутентификации. Он отвечает за проверку учетных данных и выдачу токенов.
2. **Аутентификация:** Пользователь отправляет свои учетные данные на сервер аутентификации. После проверки сервер выдает токен.
3. **Использование токена:** При каждом запросе к любому из микросервисов, пользователь (или другой микросервис) прикрепляет токен в заголовке HTTP-запроса.
4. **Проверка токена:** Получив запрос, микросервис проверяет токен (иногда с помощью обращения к серверу аутентификации, иногда локально), и если токен валидный, предоставляет доступ к ресурсам.

Нюансы при использовании паттерна

1. **Срок жизни токена:** Слишком долгий срок жизни токена может представлять риски безопасности, слишком короткий — создать проблемы с удобством использования.
2. **Обновление токенов:** Как управлять процессом обновления токенов, если они истекают.
3. **Нагрузка на сервер аутентификации:** В случае, если для проверки токена необходимо каждый раз обращаться к серверу аутентификации, это может создать дополнительную нагрузку.
4. **Типы токенов:** Существуют различные типы токенов (JWT, OAuth, SAML etc.), каждый с своими преимуществами и недостатками.
5. **Шифрование:** В зависимости от уровня безопасности, токены могут быть зашифрованы или подписаны цифровой подписью для дополнительной проверки.
6. **Обработка ошибок:** Что делать, если токен невалидный, истек или был украден.