**Введение в OAuth 2**

**Введение**

OAuth 2 представляет собой фреймворк для авторизации, позволяющий Приложениям осуществлять ограниченный доступ к Пользовательским аккаунтам на HTTP Сервисах, например, на Facebook, GitHub и DigitalOcean.

Он работает по принципу делегирования аутентификации пользователя сервису, на котором находится аккаунт пользователя, позволяя стороннему приложению получать доступ к аккаунту пользователя. OAuth 2 работает в вебе, на десктопных и мобильных приложениях.

Эта статья предназначена для разработчиков приложений и предоставляет обзор ролей, типов авторизации и типичных сценариев использования OAuth 2.

Начнём с ролей OAuth!

**Роли OAuth**

OAuth определяет четыре роли:

* Владелец ресурса
* Клиент
* Сервер ресурсов
* Авторизационный сервер

Далее мы рассмотрим каждую из ролей.

**Владелец ресурса:  *Пользователь***

Владельцем ресурса является *пользователь*, который авторизует *приложение* для доступа к своему аккаунту. Доступ приложения к пользовательскому аккаунту ограничен “областью видимости” (scope) предоставленных прав авторизации (например, доступ на чтение или запись).

**Клиент:  *Приложение***

Клиентом является *приложение*, которое хочет осуществить доступ к аккаунту Владельца ресурса. Перед осуществлением доступа приложение должно быть авторизовано Владельцем ресурса, а авторизация должна быть одобрена со стороны API.

**сервер Авторизации**

Авторизационный сервер проверяет подлинность информации, предоставленной Владельцем ресурса, а затем создаёт авторизационные токены для Клиента, с помощью которых Клиент будет осуществлять доступ к пользовательским данным.

**сервер Ресурсов**

Сервер ресурсов непосредственно хранит защищённые данные аккаунтов Владельцев ресурсов.

С точки зрения разработчика Приложения, API сервиса одновременно выполняет и роль сервера Ресурсов и роль сервера Авторизации. Далее мы будем считать эти две роли одной, и называть её ***Сервис*** или ***API***.

**Абстрактное описание протокола**

Теперь, когда у нас есть представление о ролях, используемых в OAuth, рассмотрим диаграмму их взаимодействия друг с другом.



Рассмотрим описание последовательности шагов на этой диаграмме:

1. *Приложение* запрашивает у *пользователя* авторизацию на доступ к серверу ресурсов.
2. Если *пользователь* авторизует запрос, *приложение* получает разрешение на авторизацию (authorization grant).
3. *Приложение* запрашивает авторизационный токен у *сервера авторизации* (API) путём предоставления информации о самом себе и разрешении на авторизацию от пользователя.
4. Если подлинность приложения подтверждена и разрешение на авторизацию действительно, *сервер авторизации* (API) создаёт токен доступа для приложения. Процесс авторизации завершён.
5. *Приложение* запрашивает ресурс у *сервера ресурсов* (API), предоставляя при этом токен доступа для аутентификации.
6. Если токен действителен, *сервер ресурсов* (API) предоставляет запрашиваемый ресурс *приложению*.

Фактический порядок шагов описанного процесса может отличаться в зависимости от используемого типа разрешения на авторизацию, но в целом процесс будет выглядеть описанным образом. Далее мы рассмотрим различные типы разрешений на авторизацию.

**Регистрация приложения**

Перед тем, как начать использовать OAuth в вашем Приложении, вам необходимо зарегистрировать своё Приложения в Сервисе. Это делается путём регистрации в разделе “developer” или “API” сайта Сервиса, где вам необходимо предоставить следующую информацию (*возможно, включая некоторые детали о вашем приложении*):

* Название Приложения
* Сайт Приложения
* Redirect URL или callback URL

Redirect URL - это URL, на который Сервис будет перенаправлять пользователя после авторизации (или отказа в авторизации) вашего приложения.

**Идентификатор клиента и секрет клиента**

Сервис, после регистрации Приложения, создаст учётные данные Приложения - идентификатор Приложения (client ID) и секрет Приложения (client secret):

- Идентификатор Приложения (client ID) представляет собой публично доступную строку, которая используется API сервиса для идентификации приложения, а также используется для создания авторизационных URL для Пользователей.

- Секрет Приложения (client secret) используется для аутентификации подлинности Приложения для API сервиса, когда Приложение запрашивает доступ к аккаунту Пользователя. Секрет клиента должен быть известен только Приложению и Сервису.

**Разрешение на авторизацию**

В *абстрактном описании протокола* выше первые четыре шага касаются вопросов создания разрешения на авторизацию и токена доступа.

Тип разрешения на авторизацию зависит от используемого Приложением метода запроса авторизации, а также от того, какие типы разрешения поддерживаются со стороны Сервиса.

OAuth 2 определяет четыре разных типа разрешения на авторизацию, каждый из которых полезен в определённых ситуациях:

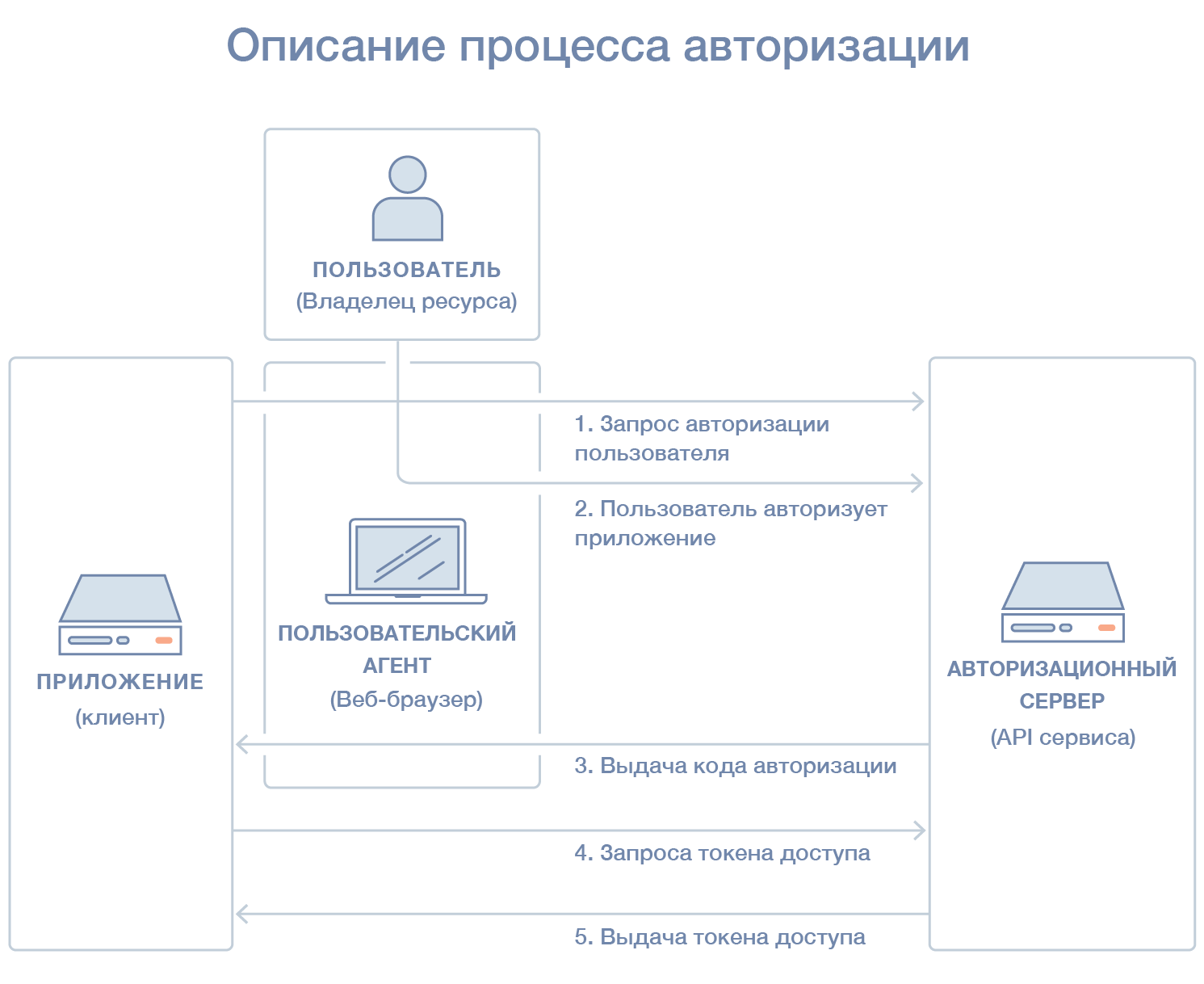
* **Код авторизации (Authorization Code)**: используется с серверными приложениями (server-side applications).
* **Неявный (Implicit)**: используется мобильными или веб-приложениями (приложения, работающие на устройстве пользователя).
* **Учётные данные владельца ресурса (Resource Owner Password Credentials)**: используются доверенными Приложениями, например Приложениями, которые являются частью самого Сервиса.
* **Учётные данные клиента (Client Credentials)**: используются при доступе Приложения к Сервису.

Далее мы рассмотрим эти типы разрешения на авторизацию, примеры их использования.

**Тип разрешения на авторизацию: Код авторизации**

**Код авторизации** является одним из наиболее распространённых типов разрешения на авторизацию, поскольку он хорошо подходит для *серверных приложений*, где исходный код приложения и *секрет Приложения* (client secret)  не доступны посторонним. Процесс в данном случае строится на перенаправлении (redirection), что означает, что Приложение должно быть в состоянии взаимодействовать с *Пользовательским агентом* (user-agent), например, веб-браузером, и получать коды авторизации API, перенаправляемые через пользовательский агент.

Опишем процесс на диаграмме:



*\* Тут сделаны дополнительные 2 шага 3-й и 4-й, скорее всего для того, чтобы Пользователь не видел токена доступа, а видел только промежуточный код авторизации*

**Шаг 1: Ссылка с кодом авторизации**

Сначала Приложением Пользователю предоставляется ссылка следующего вида:

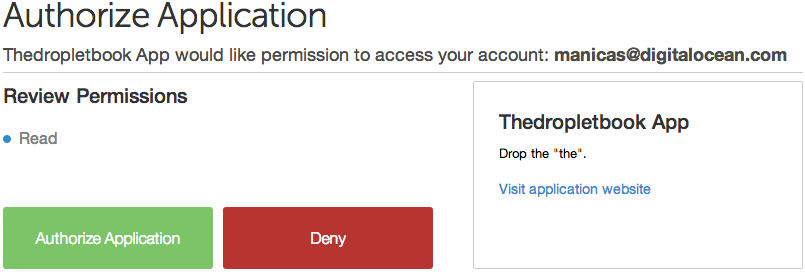
* https://cloud.digitalocean.com/v1/oauth/authorize?response\_type=code & client\_id=CLIENT\_ID & redirect\_uri=CALLBACK\_URL & scope=read

Рассмотрим компоненты ссылки:

* [**https://cloud.digitalocean.com/v1/oauth/authorize**](https://cloud.digitalocean.com/v1/oauth/authorize): входная точка API авторизации (API authorization endpoint).
* **response\_type=code**: указывает на то, что Приложение запрашивает доступ с помощью кода авторизации.
* **client\_id=CLIENT\_ID**: идентификатор Приложения (с помощью этого идентификатора API понимает, какое Приложение запрашивает доступ).
* **redirect\_uri=CALLBACK\_URL**: URL, на который Сервис перенаправит Пользовательского агента (браузер) после выдачи авторизационного кода.
* **scope=read**: задаёт уровень доступа приложения (в данном случае - доступ на чтение).

**Шаг 2: Пользователь авторизует приложение**

Когда Пользователь нажимает на ссылку, он должен сперва осуществить вход в систему для подтверждения своей личности (если он, конечно, ещё не залогинен). После этого Сервис предложит Пользователю авторизовать или отказать в авторизации Приложению для доступа к аккаунту Пользователя. Пример такого диалога представлен ниже:



На этом скриншоте экрана авторизации DigitalOcean мы можем видеть, что приложение “Thedropletbook App” запрашивает доступ на чтение к аккаунту “[manicas@digitalocean.com](mailto:manicas@digitalocean.com)”.

**Шаг 3: Приложение получает код авторизации**

Если Пользователь выбирает “Авторизовать приложение”, Сервис перенаправляет Пользовательский агент (браузер) вместе с *кодом авторизации* по URL перенаправления (redirect URL), который был задан на этапе регистрации Приложения. Ссылка будет выглядеть похожим образом (в данном примере приложение называется “dropletbook.com”):

* https://dropletbook.com/callback?code=AUTHORIZATION\_CODE

**Шаг 4: Приложение запрашивает токен доступа**

Приложение запрашивает токен доступа у API путём отправки авторизационного кода и аутентификационной информации (включая *секрет Приложения клиента*) Сервису. Ниже представлен пример POST-запроса для получения токена DigitalOcean:

* https://cloud.digitalocean.com/v1/oauth/token?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&grant\_type=authorization\_code&code=AUTHORIZATION\_CODE&redirect\_uri=CALLBACK\_URL

**Шаг 5: Приложение получает токен доступа**

Если авторизация прошла успешно, API возвращает токен доступа (а также, опционально, токен для обновления токена доступа - refresh token). Весь ответ сервера может выглядеть следующим образом:

* {"access\_token":"ACCESS\_TOKEN","token\_type":"bearer","expires\_in":2592000,"refresh\_token":"REFRESH\_TOKEN","scope":"read","uid":100101,"info":{"name":"Mark E. Mark","email":"mark@thefunkybunch.com"}}

Теперь Приложение авторизовано! Оно может использовать токен для доступа к Пользовательскому аккаунту через API Сервиса с заданными ограничениями доступа до тех пор, пока не истечёт срок действия токена или токен не будет отозван.

Если был создан токен для обновления токена доступа, он может быть использован для получения новых токенов доступа, когда истечёт срок действия старого токена.

**Тип разрешения на авторизацию: Неявный**

Неявный тип разрешения на авторизацию используется мобильными и веб-приложениями (приложениями, которые работают в веб-браузере), где конфиденциальность *секрета клиента* не может быть гарантирована. Неявный тип разрешения также основан на перенаправлении Пользовательского агента, при этом токен доступа передаётся Пользовательскому агенту для дальнейшей передачи Приложению. Это, в свою очередь, делает токен доступным Пользователю и другим Приложениям на устройстве Пользователя. Также при этом типе разрешения на авторизацию не осуществляется аутентификация подлинности приложения, а сам процесс полагается на URL перенаправления (зарегистрированный ранее в сервисе).

Неявный тип разрешения на авторизацию не поддерживает токены обновления токена доступа (refresh tokens).

Процесс выглядит следующим образом: приложение просит пользователя авторизовать себя, затем сервер авторизации передаёт токен доступа к пользовательскому агенту, который передаёт токен приложению. Далее мы опишем процесс в деталях.



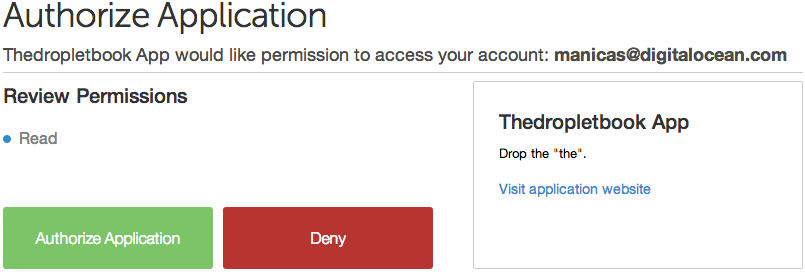
**Шаг 1: Ссылка для неявной авторизации**

При неявном типе разрешения на авторизацию пользователю предоставляется ссылка, запрашивающая токен у API. Эта ссылка выглядит почти так же, как ссылка для предыдущего способа (с кодом авторизации), за исключением того, что запрашивается *токен* вместо кода (обратите внимание на *response type* “token”):

* https://cloud.digitalocean.com/v1/oauth/authorize?response\_type=token&client\_id=CLIENT\_ID&redirect\_uri=CALLBACK\_URL&scope=read

**Шаг 2: Пользователь авторизует приложение**

Когда пользователь нажимает на ссылку, он должен сперва осуществить вход в систему для подтверждения своей личности (если он, конечно, ещё не залогинен). После этого сервис предложит пользователю авторизовать или отказать в авторизации приложению для доступа к аккаунту пользователя. Пример такого диалога представлен ниже:



На этом скриншоте экрана авторизации DigitalOcean мы можем видеть, что приложение “Thedropletbook App” запрашивает доступ на чтение к аккаунту “[manicas@digitalocean.com](mailto:manicas@digitalocean.com)”.

**Шаг 3: Пользовательский агент получает токен доступа с URI перенаправления**

Если пользователь выбирает “Авторизовать приложение”, сервис перенаправляет Пользовательский агент по URI перенправления приложения и включает в URI фрагмент, содержащий токен доступа. Это выглядит примерно вот так:

* https://dropletbook.com/callback#token=ACCESS\_TOKEN

**Шаг 4: Пользовательский агент следует по URI перенаправления**

Пользовательский агент следует по URI перенаправления, сохраняя при этом токен доступа (*куда, если шаги 5 и 6 делаются для передачи токена приложению???*).

**Шаг 5: Приложение выполняет скрипт извлечения токена доступа**

Приложение возвращает веб-страницу, которая содержит скрипт для извлечения токен доступа из полного URI перенаправления, сохранённого пользовательским агентом.

**Шаг 6: Токен доступа передаётся приложению**

Пользовательский агент запускает скрипт извлечения токена доступа, а затем передаёт извлечённый токен приложению.

Теперь Приложение авторизовано! Оно может использовать токен для доступа к Пользовательскому аккаунту, находящемуся на Сервисе, с заданными ограничениями доступа до тех пор, пока не истечёт срок действия токена или токен не будет отозван.

**Тип разрешения на авторизацию: учётные данные владельца ресурса**

При этом типе разрешения на авторизацию Пользователь предоставляет Приложению напрямую свои авторизационные данные в сервисе (имя пользователя и пароль). Приложение, в свою очередь, использует полученные учётные данные Пользователя для получения токена доступа от сервиса. Этот тип разрешения на авторизацию должен использоваться только в том случае, когда другие варианты не доступны. Кроме того, этот тип разрешения стоит использовать только в случае, когда Приложение пользуется доверием Пользователя (например, является частью самого сервиса, или операционной системы Пользователя).

**Процесс с учётными данными владельца ресурса**

После того, как Пользователь передаст свои учётные данные Приложению, Приложение запросит токен доступа у Авторизационного сервера. Пример POST-запроса может выглядеть следующим образом:

* https://oauth.example.com/token?grant\_type=password&username=USERNAME&password=PASSWORD&client\_id=CLIENT\_ID

Если учётные данные корректны, сервер Авторизации вернёт токен доступа Приложению. Теперь Приложение авторизовано!

**Внимание:** DigitalOcean в настоящее время не поддерживает тип разрешения на авторизацию с использованием учётных данных владельца ресурса, поэтому ссылка выше ведёт на воображаемый авторизационный сервер “oauth.example.com”.

**Тип разрешения на авторизацию: Учётные данные клиента**

Тип разрешения на авторизацию с использованием собственных учётных данных позволяет Приложению осуществлять доступ к своему собственному аккаунту сервиса. Это может быть полезно, например, когда Приложение хочет обновить собственную регистрационную информацию на Сервисе или URI перенаправления, или же осуществлить доступ к другой информации, хранимой в аккаунте Приложения на Сервисе, через API сервиса.

**Процесс с учётными данными клиента**

Приложение запрашивает токен доступа путём отправки своих учётных данных, своего идентификатора клиента и секрета клиента Авторизационному серверу. Пример POST-запроса может выглядеть следующим образом:

* https://oauth.example.com/token?grant\_type=client\_credentials&client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET

Если учётные данные корректны, Авторизационный сервер вернёт токен доступа для Приложения. Теперь Приложение авторизовано для использования собственного аккаунта!

**Внимание:** DigitalOcean в настоящее время не поддерживает тип разрешения на авторизацию с использованием учётных данных клиента, поэтому ссылка выше ведёт на воображаемый авторизационный сервер “oauth.example.com”.

**Пример использования токена доступа**

После того, как Приложение получит токен доступа, оно может использовать этот токен для доступа к Пользовательскому аккаунту через API Сервиса с заданными ограничениями доступа до тех пор, пока не истечёт срок действия токена или токен не будет отозван.

Ниже представлен пример запроса к API с использованием curl. Обратите внимание, что он содержит токен доступа:

* curl -X POST -H "Authorization: Bearer ACCESS\_TOKEN""https://api.digitalocean.com/v2/$OBJECT"

Если токен доступа валиден, API обработает полученный запрос. Если срок действия токена доступа истёк или токен некорректен, API вернёт ошибку “invalid\_request”.

**Обновление токена доступа**

После истечения срока действия токена доступа все запросы к API с его использованием будут возвращать ошибку “Invalid Token Error”. Если при создании токена доступа был создан и токен для обновления токена доступа (refresh token), последний может быть использован для получения нового токена доступа от авторизационного сервера.

Ниже представлен пример POST-запроса, использующего токен для обновления токена доступа для получения нового токена доступа:

* https://cloud.digitalocean.com/v1/oauth/token?grant\_type=refresh\_token&client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&refresh\_token=REFRESH\_TOKEN

**Заключение**

На этом мы завершаем наш обзор OAuth 2. Теперь у вас есть общее представление о том, как работает OAuth 2, а также о том, когда и как использовать существующие типы разрешения на авторизацию.

Если вы хотите узнать больше об OAuth 2, рекомендуем ознакомиться со следующими статьями:

* [Как использовать аутентификацию OAuth в DigitalOcean, если вы пользователь или разработчик](https://www.digitalocean.com/community/tutorials/how-to-use-oauth-authentication-with-digitalocean-as-a-user-or-developer). - https://www.digitalocean.com/community/tutorials/how-to-use-oauth-authentication-with-digitalocean-as-a-user-or-developer
* [Как использовать API v2 DigitalOcean](https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-api-v2). - https://www.digitalocean.com/docs/apis-clis/api/
* [Фреймворк авторизации OAuth 2.0](http://tools.ietf.org/html/rfc6749). - http://tools.ietf.org/html/rfc6749