

Управление доступом Подключение и аутентификация



Авторские права

© Postgres Professional, 2015–2022

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Конфигурационные файлы

Простые методы аутентификации

Аутентификация по паролю

Внешняя аутентификация и сопоставление имен

Идентификация

определение имени пользователя БД
имя может отличаться от указанного (при внешней аутентификации)

Аутентификация

действительно ли пользователь тот, за кого себя выдает?
обычно требуется подтверждение (например, пароль)

Авторизация

имеет ли право данный пользователь подключаться к серверу?
частично пересекается с функционалом привилегий

При подключении клиента сервер должен выполнить несколько задач.

Во-первых, идентифицировать пользователя, то есть определить его имя. Для этого пользователь представляется, но указанное имя может отличаться от имени пользователя БД (например, если пользователь представляется своим именем в ОС).

Во-вторых, аутентифицировать пользователя, то есть проверить, что он действительно тот, за кого себя выдает. Простой пример — ввод пароля.

В-третьих, авторизовать пользователя, то есть определить, имеет ли он право подключения (эта задача частично пересекается с функционалом привилегий).

Иногда все три задачи называют общим словом «аутентификация». PostgreSQL позволяет гибко настроить этот процесс.

До сих пор мы подключались к серверу, никак не подтверждая свое имя. Дело в том, что в Ubuntu настройки по умолчанию позволяют подключаться к локальному серверу без авторизации, если имена пользователя базы данных и операционной системы совпадают. А в виртуальной машине курса сделаны дополнительные настройки, разрешающие вообще любые локальные подключения.

pg_hba.conf

конфигурационный файл, при изменении нужно перечитать строка — набор полей, разделитель — пробел или табуляция
пустые строки и текст после # игнорируются

Поля

тип подключения	}	параметры подключения
имя базы данных		
имя пользователя		
адрес узла		
метод аутентификации		
необязательные дополнительные параметры в виде <i>имя=значение</i>		

Настройки аутентификации хранятся в конфигурационном файле, который используется подобно postgresql.conf, но отличается от него по формату. Файл называется pg_hba.conf (от «host-based authentication»), его расположение определяется параметром *hba_file*. При изменении этого файла необходимо перечитать настройки (как обычно, с помощью pg_ctl reload или вызовом функции pg_reload_conf).

Файл pg_hba.conf состоит из строк, каждая из которых считается отдельной записью. Пустые строки и комментарии (все после символа #) игнорируются. Строка состоит из полей, разделенных пробелами или табуляциями.

Количество полей может различаться в зависимости от их содержимого, общий список приведен на слайде.

<https://postgrespro.ru/docs/postgresql/13/auth-pg-hba-conf>

Записи просматриваются сверху вниз

Применяется первая запись, которой соответствует подключение (тип, база, пользователь и адрес)

выполняется аутентификация и проверка привилегии CONNECT

если результат отрицательный, доступ запрещается

если ни одна запись не подошла, доступ запрещается

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		trust
# IPv4 local connections:					
host	all		all	127.0.0.1/32	trust
# IPv6 local connections:					
host	all		all	::1/128	trust

5

Конфигурационный файл обрабатывается сверху вниз. Для каждой строки определяется, подходит ли она к запрашиваемому клиентом подключению (по соответствию типа подключения, имени БД, имени пользователя и IP-адресу). Если подходит, то выполняется аутентификация указанным в строке методом. Если результат успешен, то подключение разрешается, иначе — запрещается (другие строки при этом уже не рассматриваются).

Если ни одна из строк не подошла, то доступ запрещается.

Таким образом, записи в файле должны идти сверху вниз от частного к общему.

Внизу слайда приведен фрагмент файла по умолчанию при сборке из исходных кодов (при установке из пакета возможны другие настройки). В этом примере видно три строки. Первая относится к локальным не-TCP подключениям (local) для любых баз (all) и пользователей (all). Вторая относится к удаленным подключениям (host) с адреса 127.0.0.1 (то есть localhost), третья — то же самое, но для IPv6.

Можно сделать вывод о том, что по умолчанию PostgreSQL допускает только локальные соединения (как сетевые, так и нет).

Далее возможные значения полей рассматриваются более подробно.

Содержимое файла pg_hba.conf

Расположение конфигурационного файла:

```
=> SHOW hba_file;

          hba_file
-----
/etc/postgresql/13/main/pg_hba.conf
(1 row)
```

Мы можем посмотреть сам файл (опустим пустые строки и комментарии):

```
student$ sudo egrep '^[^#]' /etc/postgresql/13/main/pg_hba.conf

local   all             postgres                                trust
local   all             all                                trust
host     all             all             127.0.0.1/32          md5
host     all             all             ::1/128             md5
local    replication     all                                trust
host     replication     all             127.0.0.1/32          md5
host     replication     all             ::1/128             md5
```

При настройке виртуальной машины были внесены изменения в стандартный файл pg_hba.conf: метод peer был заменен на trust, чтобы роли alice, bob и charlie могли подключаться локально под пользователем student ОС.

Тип подключения

Имя базы данных

Адрес узла

Имя роли

local

локальное подключение через unix-domain socket

host

подключение по TCP/IP

(обычно требуется изменение параметра *listen_addresses*)

hostssl

шифрованное SSL-подключение по TCP/IP

(сервер должен быть собран с поддержкой SSL,
также требуется установить параметр *ssl*)

hostnssl

нешифрованное подключение по TCP/IP

В поле типа подключения можно задать одно из значений, перечисленных ниже.

Слово «local» — соответствует локальному подключению через доменный сокет (без использования сетевого подключения).

Слово «host» — соответствует любому подключению по TCP/IP. Поскольку по умолчанию PostgreSQL слушает соединения только с локального адреса (localhost), скорее всего потребуются задать другой адрес с помощью параметра сервера *listen_address*.

Слово «hostssl» — соответствует только шифрованному SSL-подключению по TCP/IP. Для таких соединений сервер должен быть собран с поддержкой SSL. Кроме того, требуется установить параметр *ssl* = on.

Слово «hostnssl» — соответствует только нешифрованному подключению по TCP/IP.

all

подключение к любой БД

sameuser

БД, совпадающая по имени с ролью

samerole

БД, совпадающая по имени с ролью или группой, в которую она входит

replication

специальное разрешение для протокола репликации

БД

БД с указанным именем (возможно, в кавычках)

имя[, имя ...]

нескольких имен из вышеперечисленного

В поле базы данных можно задать одно из значений, перечисленных ниже, или несколько таких значений через запятую.

Слово «all» — соответствует любой базе данных.

Слово sameuser — соответствует базе данных, совпадающей по имени с пользователем.

Слово samerole — соответствует базе данных, совпадающей по имени с какой-либо ролью, в которую входит пользователь (в том числе совпадающей по имени с самим пользователем, поскольку пользователь — та же роль).

Наконец, можно указать имя конкретной базы данных.

Вместо перечисления имен можно сослаться на файл с помощью @. В файле имена могут быть разделены запятыми, пробелами, табуляциями или переводами строк. Допускаются вложенные подключения файлов (@) и комментарии (#).

all

любой IP-адрес

IP-адрес/длина_маски

указанный диапазон IP-адресов (например, 172.20.143.0/24)

или альтернативная форма в два поля (172.20.143.0 255.255.255.0)

samehost

IP-адрес сервера

samenet

любой IP-адрес из любой подсети, к которой подключен сервер

доменное_имя

IP-адрес, соответствующий указанному имени (например, domain.com)

допускается указание части имени, начиная с точки (.com)

В поле адреса может быть указано одно из следующих значений.

Слово «all» — соответствует любому IP-адресу клиента.

IP-адрес с указанием длины маски подсети (CIDR) — определяет диапазон допустимых IP-адресов. Альтернативно можно записать отдельно IP-адрес и в следующем поле маску подсети. Также поддерживаются IP-адреса в нотации IPv6.

Слово «samehost» — соответствует IP-адресу сервера (фактически, аналог 127.0.0.1 для систем, где такой адрес не разрешен).

Слово «samenet» — соответствует любому IP-адресу из любой подсети, к которой подключен сервер.

Наконец, адрес можно указать доменным именем (или частью доменного имени, начиная с точки). PostgreSQL определит принадлежность IP-адреса клиента указанному домену: для этого сначала по IP-адресу определяется доменное имя (reverse lookup), а затем проверяется, что такому домену действительно соответствует исходный IP-адрес (forward lookup). Таким образом проверяется соответствие владельца сети и владельца доменного имени для отсекаемых скомпрометированных адресов:

https://en.wikipedia.org/wiki/Forward-confirmed_reverse_DNS

`all`

любая роль

`роль`

роль с указанным именем (возможно, в кавычках)

`+роль`

роль, входящая в указанную роль

`имя[, имя...]`

несколько имен из вышеперечисленного

В поле имени пользователя можно указать одно из значений, перечисленных ниже, или несколько таких значений через запятую.

Слово «all» — соответствует любой роли.

Имя роли — соответствует роли с указанным именем. Если перед именем роли стоит знак +, то имя соответствует любому пользователю, входящему в указанную роль.

Вместо перечисления имен можно сослаться на файл с помощью @. В файле имена могут быть разделены запятыми, пробелами, табуляциями или переводами строк. Допускаются вложенные подключения файлов (@) и комментарии (#).

Ничего не проверяет

`trust`

допустить без аутентификации

`reject`

отказать без аутентификации

В поле метода аутентификации можно указать различные методы. Для начала познакомимся с двумя самыми простыми, а остальные рассмотрим ниже.

Метод «trust» безусловно доверяет пользователю и не выполняет проверку. В реальной жизни имеет смысл применять разве что для локальных соединений.

Метод «reject» безусловно отказывает в доступе. Можно использовать, чтобы отсеять любые соединения определенного типа или с определенных адресов (например, запретить нешифрованные соединения).

Что обозначает приведенная ниже настройка?

#	TYPE	DATABASE	USER	ADDRESS	METHOD
hostnossl	all		all	all	reject
host	sameuser		all	samenet	trust
host	pub		+reader	all	trust

Обратите внимание, что первую строку нельзя сместить вниз — смысл настройки изменится.

1. Запрещаются нешифрованные соединения.
2. Разрешается доступ пользователям из своей подсети к анонимным базам данных.
3. Разрешается доступ пользователям, входящим в роль reader, к базе данных pub.

Редактирование файла pg_hba.conf

Сохраним файл pg_hba.conf, чтобы иметь возможность восстановить его после экспериментов.

```
student$ sudo cp -n /etc/postgresql/13/main/pg_hba.conf ~/pg_hba.conf.orig
```

Другой способ получить содержимое pg_hba.conf — представление pg_hba_file_rules:

```
=> SELECT line_number, type, database, user_name, address, auth_method
FROM pg_hba_file_rules;
```

line_number	type	database	user_name	address	auth_method
89	local	{all}	{postgres}		trust
94	local	{all}	{all}		trust
96	host	{all}	{all}	127.0.0.1	md5
98	host	{all}	{all}	::1	md5
101	local	{replication}	{all}		trust
102	host	{replication}	{all}	127.0.0.1	md5
103	host	{replication}	{all}	::1	md5

(7 rows)

Представление читает сам файл, а не показывает уже прочитанные значения. Этим можно воспользоваться, чтобы проверить корректность сделанных изменений.

Например, допишем следующую строку к файлу pg_hba.conf:

```
student$ echo 'local all all trust' | sudo tee -a /etc/postgresql/13/main/pg_hba.conf
```

```
local all all trust
```

```
=> SELECT line_number, error
FROM pg_hba_file_rules
WHERE error IS NOT NULL;
```

line_number	error
104	invalid authentication method "trust"

(1 row)

Без представления мы бы узнали об ошибке только из журнала после перечитывания конфигурации.

Сервер запрашивает у клиента пароль

password

передается в незашифрованном виде

md5

передается MD5-хеш

scram-sha-256

используется протокол SCRAM

При парольной аутентификации сервер PostgreSQL запрашивает у пользователя пароль и проверяют его на соответствие паролю, который хранится либо в самой СУБД, либо во внешней службе.

<https://www.postgrespro.ru/docs/postgresql/13/protocol-flow#id-1.10.5.7.3>

Для паролей, хранящихся в СУБД, поддерживаются три метода.

Метод «**md5**» сравнивает MD5-дайджест пароля с MD5-дайджестом, хранящимся в базе. При запросе сервер отправляет клиенту так называемую «соль», клиент вычисляет MD5-хеш пароля, добавляет «соль», еще раз вычисляет MD5-хеш и отправляет его на сервер для сравнения с сохраненным хешем. Благодаря «соли» хеши одинаковых паролей получаются разными. Однако в настоящее время алгоритм MD5 считается недостаточно криптостойким.

Наиболее безопасный метод «**scram-sha-256**» использует при аутентификации протокол SCRAM и задействует криптостойкий алгоритм SHA-256. Метод реализует фреймворк SASL, отделяющий механизм аутентификации от прикладного протокола.

<https://postgrespro.ru/docs/postgresql/13/sasl-authentication>

Метод «**password**» предполагает передачу пароля в незашифрованном виде. Его не следует применять, если соединение клиент-сервер не зашифровано.

Установить пароль пользователя

```
[ CREATE | ALTER ] ROLE ...  
  PASSWORD 'пароль'  
[ VALID UNTIL дата_время ];
```

пользователю с пустым паролем будет отказано в доступе
при аутентификации по паролю

Пароли хранятся в системном каталоге

pg_authid

метод шифрования определяется параметром *password_encryption*

метод аутентификации должен совпадать с методом шифрования
(md5 автоматически переключается на scram-sha-256)

До сих пор мы создавали роли без указания пароля. Если установить метод аутентификации по паролю, таким пользователям будет отказано в доступе.

Пароль хранится в базе данных в таблице pg_authid.

Чтобы установить пароль, надо указать его — либо сразу при создании роли в команде CREATE ROLE, либо впоследствии в ALTER ROLE.

Пароль хранится в зашифрованном виде, алгоритм шифрования (MD5 или SCRAM-SHA-256) определяется параметром *password_encryption*.

Также можно указать время действия пароля.

Если сохраненный пароль был зашифрован алгоритмом SCRAM-SHA-256 и установлен метод аутентификации MD5, при обмене сообщениями будет использован более надежный метод SCRAM-SHA-256.

Вручную

Установить переменную PGPASSWORD

неудобно при подключении к разным базам
не рекомендуется из соображений безопасности

Файл с паролями

~/.pgpass на узле клиента
строки в формате *узел:порт:база_данных:имя_пользователя:пароль*
в качестве значения можно указать * (любое значение)
строки просматриваются сверху вниз, выбирается первая подходящая
файл должен иметь права доступа 600 (rw- --- ---)

Пользователь может вводить пароль вручную, а может автоматизировать ввод. Для этого есть две возможности.

Во-первых, можно задать пароль в переменной окружения PGPASSWORD (на клиенте). Однако это неудобно, если приходится подключаться к нескольким базам, и не рекомендуется из соображений безопасности.

Во-вторых, можно задать пароли в файле ~/.pgpass (его расположение можно изменить, задав переменную окружения PGPASSFILE). К файлу должен иметь доступ только владелец, иначе PostgreSQL проигнорирует его.

Парольная аутентификация

Настроим парольную аутентификацию для локальных подключений пользователя student. Сначала проверим, как шифруются пароли при сохранении.

```
=> SHOW password_encryption;
```

```
password_encryption
-----
md5
(1 row)
```

Нам нужен надежный алгоритм.

```
=> SET password_encryption='scram-sha-256';
```

```
SET
```

Теперь зададим пароль для пользователя student. Пароль может состоять из любых символов Unicode.

```
=> ALTER ROLE student PASSWORD 'пароль';
```

```
ALTER ROLE
```

Осталось настроить правила аутентификации:

```
student$ sudo tee /etc/postgresql/13/main/pg_hba.conf << EOF
local all postgres trust
local all student  scram-sha-256
EOF
```

```
local all postgres trust
local all student  scram-sha-256
```

Если бы мы указали метод MD5, при аутентификации пользователя student все равно использовался бы SCRAM-SHA-256. Но при этом другие пользователи могли бы сохранять пароли, зашифрованные менее криптостойким алгоритмом MD5.

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf
-----
t
(1 row)
```

Пытаемся угадать пароль:

```
student$ psql 'user=student password=1234'
```

```
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL:  password authentication failed for user "student"
```

Теперь с правильным паролем:

```
student$ psql 'user=student password=пароль' -c '\conninfo'
```

```
You are connected to database "student" as user "student" via socket in "/var/run/postgresql" at port "5432".
```

Восстановим сохраненный файл pg_hba.conf.

```
student$ sudo cp ~/pg_hba.conf.orig /etc/postgresql/13/main/pg_hba.conf
```

```
student$ sudo pg_ctlcluster 13 main reload
```

ldap [*параметры*]

пароль хранится на сервере LDAP

radius [*параметры*]

пароль хранится на сервере RADIUS

ram [*параметры*]

пароль хранится в подключаемом модуле PAM

Пароль может храниться не в СУБД, а во внешней службе.

Методы «ldap», «radius» и «ram» используют сервер LDAP, сервер RADIUS или подключаемый модуль аутентификации (Pluggable Authentication Module) соответственно. Эти методы требуют указания дополнительных специфичных параметров. Подробно они не рассматриваются.

Проводится вне СУБД

`peer [map=...]`

запрос имени пользователя у ядра ОС (для локальных подключений)

`cert [map=...]`

аутентификация с использованием клиентского SSL-сертификата

`gss [map=... и другие параметры]`

аутентификация Kerberos по протоколу GSSAPI

`sspi [map=... и другие параметры]`

аутентификация Kerberos/NTLM для Windows

В этой группе методов и идентификация, и аутентификация происходит вне СУБД. В результате успешной аутентификации PostgreSQL получает:

1. имя, указанное при подключении (внутреннее имя СУБД);
2. имя, идентифицированное внешней системой (внешнее имя).

Поэтому все перечисленные методы позволяют указать как минимум один дополнительный параметр `map`. Он определяет правила сопоставления внутренних и внешних имен (подробнее об этом на следующем слайде).

Метод `peer` запрашивает имя пользователя у ядра ОС. Поскольку ОС уже аутентифицировала этого пользователя (скорее всего, запросив пароль), мы просто верим ей.

Метод `cert` использует аутентификацию на основе клиентского сертификата и предназначен только для SSL-соединений.

Метод `gss` использует аутентификацию Kerberos по протоколу GSSAPI (RFC1964 <https://tools.ietf.org/html/rfc1964>). Поддерживается автоматическая аутентификация (single sign-on).

Метод `sspi` использует аутентификацию Kerberos или NTLM для систем на Windows. Поддерживается автоматическая аутентификация.

pg_ident.conf

еще один конфигурационный файл
строка — набор полей, разделитель — пробел или табуляция
пустые строки и текст после # игнорируются

Поля

название соответствия
(указывается в параметре `map` в `pg_hba.conf`)
внешнее имя
(считается регулярным выражением, если начинается с косой черты)
внутреннее имя пользователя БД

Правила сопоставления имен определяются в отдельном файле `pg_ident.conf`; его расположение определяется параметром `ident_file`. Файл имеет такую же структуру, как и `pg_hba.conf`. Записи состоят из трех полей: название соответствия, внешнее имя, внутреннее имя.

Название соответствия необходимо, чтобы разграничивать разные правила сопоставления внутри одного файла `pg_ident.conf` (которые могут потребоваться для разных записей в `pg_hba.conf`).

Внешнее имя должно совпадать с именем, возвращаемым внешней системой аутентификации или указанным в сертификате. Если это поле начинается с косой черты, то его значение считается регулярным выражением. Это позволяет обработать ситуации, когда внешнее и внутреннее имена отличаются только префиксами или суффиксами.

Внутреннее имя должно совпадать с именем пользователя базы данных.

Запись, сопоставляющая внутреннее и внешнее имена, означает, что данному внешнему пользователю разрешено подключаться к СУБД под данным внутренним пользователем (конечно, при условии успешной аутентификации).

<https://postgrespro.ru/docs/postgresql/13/auth-username-maps>

Что обозначает приведенная ниже настройка?

pg_hba.conf

#	TYPE	DATABASE	USER	ADDRESS	METHOD	
hostssl	sameuser	all	all	cert	map=m1	
local	all	all	all	peer	map=m2	
host	all	all	samehost	md5		

pg_ident.conf

#	MAPNAME	SYSTEM-USERNAME	PG-USERNAME
m1		/^(.*)@domain\.com\$	\1
m2		student	alice
m2		student	bob

SSL-соединения аутентифицируются по клиентскому сертификату. Предполагается, что имя (common name) в сертификате имеет вид «пользователь@domain.com», и *пользователь* считается именем роли. Для локальных соединений PostgreSQL запрашивает имя пользователя у операционной системы. Соответствие m2 определяет, что пользователь ОС student может подключаться под ролями alice и bob. Сетевое соединение с локальным сервером аутентифицируются по паролю (зашифрованному MD5).

Настройки аутентификации определяются
в конфигурационных файлах

Можно использовать аутентификацию по паролю
(с хранением пароля в СУБД или вне ее) и внешнюю
аутентификацию

1. Измените конфигурационные файлы (предварительно сохранив оригиналы) таким образом, чтобы:
безусловно разрешить локальное соединение суперпользователям student и postgres;
разрешить сетевые подключения всем пользователям к любым базам данных с аутентификацией по паролю с использованием MD5.
2. Создайте роль alice с паролем, зашифрованным MD5, и роль bob с паролем, зашифрованным SCRAM-SHA-256.
3. Проверьте возможность подключения под созданными ролями.
4. Под суперпользовательской ролью посмотрите пароли пользователей alice и bob в системном каталоге.
5. Восстановите исходные конфигурационные файлы.

1. Настройка аутентификации

Сохраним исходный файл настроек:

```
student$ sudo cp -n /etc/postgresql/13/main/pg_hba.conf ~/pg_hba.conf.orig
```

Теперь перезапишем pg_hba.conf с нуля:

```
student$ sudo tee /etc/postgresql/13/main/pg_hba.conf << EOF
local all postgres trust
local all student trust
host all all all md5
EOF
```

```
local all postgres trust
local all student trust
host all all all md5
```

```
student$ sudo pg_ctlcluster 13 main reload
```

2. Создание ролей

```
=> SHOW password_encryption;
```

```
password_encryption
-----
md5
(1 row)
```

```
=> CREATE ROLE alice LOGIN PASSWORD 'alice';
```

```
CREATE ROLE
```

```
=> SET password_encryption='scram-sha-256';
```

```
SET
```

```
=> CREATE ROLE bob LOGIN PASSWORD 'bob';
```

```
CREATE ROLE
```

3. Проверка подключения

Поскольку настройки требуют ввода пароля, мы укажем его явно в строке подключения.

При выполнении этого задания лучше ввести пароль вручную, чтобы убедиться в том, что он запрашивается.

```
=> \c "dbname=student user=alice host=localhost password=alice"
```

You are now connected to database "student" as user "alice" on host "localhost" (address "127.0.0.1") at port "5432".

```
=> \c "dbname=student user=bob host=localhost password=bob"
```

You are now connected to database "student" as user "bob".

4. Просмотр паролей

```
=> \q
```

```
student$ psql
```

```
=> SELECT rolname, rolpassword FROM pg_authid WHERE rolname IN ('alice','bob') \gx
```

```
-[ RECORD 1 ]-----
rolname      | alice
rolpassword  | md5579e43b423b454623383471aeb85cd87
-[ RECORD 2 ]-----
rolname      | bob
rolpassword  | SCRAM-SHA-256$4096:0PSRZLkLL0IZmPA+NdKpiA==$2NuyxFEsiQ2fYb/E97J2/NuHgZK3wY4PB28k6v4wqek=:EzWnMS00Z3kx0ompfCeeVUUs++liF/JmST5Nwypo+2U=
```

Пароли хранятся как значение хеш-функции, не допускающее расшифровки. Сервер всегда сравнивает между собой зашифрованные значения — введенный пароль и значение из pg_authid.

5. Восстановление исходных настроек

```
student$ sudo cp ~/pg_hba.conf.orig /etc/postgresql/13/main/pg_hba.conf
```

```
student$ sudo pg_ctlcluster 13 main reload
```

Определенным пользователям, список которых время от времени меняется, требуется разрешить локальный доступ без авторизации. Сложность в том, что любое изменение списка требует изменения файла `pg_hba.conf`.

1. Настройте аутентификацию, лишенную этого недостатка.
2. Убедитесь в правильности изменений.
3. Восстановите первоначальные настройки.

1. Используйте групповую роль.

1. Настройка аутентификации

Сохраним исходный файл настроек:

```
student$ sudo cp -n /etc/postgresql/13/main/pg_hba.conf ~/pg_hba.conf.orig
```

Будем управлять аутентификацией пользователей, добавляя их в группу locals.

Запишем файл pg_hba.conf с нуля:

```
student$ sudo tee /etc/postgresql/13/main/pg_hba.conf << EOF
local all student trust
local all +locals trust
EOF
```

```
local all student trust
local all +locals trust
```

```
student$ sudo pg_ctlcluster 13 main reload
```

Создадим групповую роль:

```
=> CREATE ROLE locals;
```

```
CREATE ROLE
```

2. Проверка

Алиса в группе locals:

```
=> CREATE ROLE alice LOGIN;
```

```
CREATE ROLE
```

```
=> GRANT locals TO alice;
```

```
GRANT ROLE
```

Боб пока не в группе:

```
=> CREATE ROLE bob LOGIN;
```

```
CREATE ROLE
```

```
student$ psql "dbname=student user=alice" -c "\conninfo"
```

You are connected to database "student" as user "alice" via socket in "/var/run/postgresql" at port "5432".

```
student$ psql "dbname=student user=bob" -c "\conninfo"
```

psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL: no pg_hba.conf entry for host "[local]", user "bob", database "student", SSL off

Включаем Боба в группу:

```
=> GRANT locals TO bob;
```

```
GRANT ROLE
```

```
student$ psql "dbname=student user=bob" -c "\conninfo"
```

You are connected to database "student" as user "bob" via socket in "/var/run/postgresql" at port "5432".

2. Восстановление исходных настроек

```
student$ sudo cp ~/pg_hba.conf.orig /etc/postgresql/13/main/pg_hba.conf
```

```
student$ sudo pg_ctlcluster 13 main reload
```