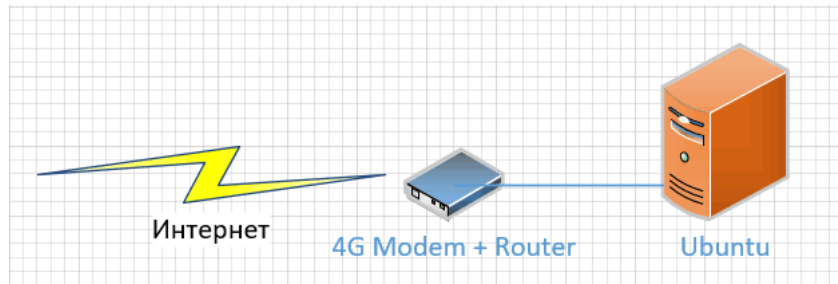


## Подключаемся к серверу за NAT при помощи туннеля SSH. Простая и понятная инструкция.

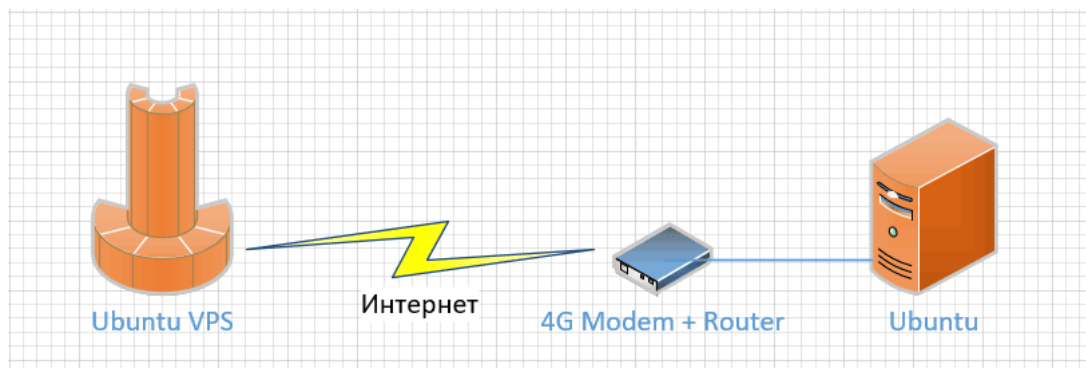
Однажды, в студеную зимнюю пору, возникла у меня потребность в открытии некоторых ресурсов внутреннего сервера частной сети внешним пользователям. Сервер сам удобно расположился в скромной серверной на краю промышленной зоны, в предместьях небольшой подмосковной деревушки. Где из всего многообразия способов подключения к сети Интернет, доступен только 4G модем. Да и тот, изредка пытается перейти на сеть 3G, несмотря на могучую внешнюю направленную антенну.

В довершение к нестабильной связи, российские сотовые операторы, не желают раскошелиться на современное аппаратное и программное обеспечение и, как следствие, пользователям доступны только IPv4 сети. Никаких IPv6 нет и в помине, да и не планируется в ближайшем столетии. Плюс все это находится за NAT провайдера. Другими словами, пробраться извне к серверу, расположенному в локальной частной сети, которая подключена к глобальной сети через частную сеть сотового оператора — задача, невыполнимая без использования посредника обладающего реальным IP-адресом. Если свой NAT на своем роутере еще можно как-то настроить для подключения извне, то с NAT провайдера поделить решительно ничего нельзя.



Типичная топология подключения в промзоне

Итак, для того, чтобы иметь хоть какую-то возможность получить доступ к ресурсам сервера в локальной сети промышленной зоны необходим посредник. В качестве такого посредника я решил использовать самый простой и дешевый вариант VPS-сервера у хостинг-провайдера. Ресурсы сервера мизерны и смехотворны. ОЗУ всего около 300 мегабайт, дисковое пространство порядка 10 гигабайт, а процессор вообще один. У меня телефон мощнее раз так в десять, чем этот VPS. Но у него имеется то, чего нет у моего телефона, а именно белый статический IP-адрес сети IPv4, да еще и привязанное доменное имя. А это значит, что он доступен с любой точки глобальной сети Интернет. И прокинув какой-то канал или, правильнее, туннель от сервера за NAT к VPS-серверу, можно будет подключаться к серверу в локальной сети из любой точки глобальной.



Вариант решения задачи с использованием внешнего сервера

Кстати, здесь и далее я буду использовать адресацию и примеры, построенные на основе двух виртуальных машин, работающих на одном компьютере (и настоятельно рекомендую, прежде чем устанавливать полноценный туннель между серверами, потренируйтесь на «виртуалках», потом будет меньше восстанавливать через удаленную консоль). Машина первая, Ubuntu-VPS работает через сетевой мост в локальной сети. Это значит, что я могу к ней подключиться с любого компьютера в моей локальной сети. Вторая виртуальная машина, назовем ее просто — Ubuntu, подключается к сети через NAT среды виртуализации. Она может выходить в локальную сеть, может достигаться до любого сервера в сети Интернет, а вот к ней попасть снаружи нельзя никак. NAT надежно защищает доступ ко второй виртуальной машине. На обеих виртуальных машинах установлена серверная версия Ubuntu 18.04 LTS с актуальной версией OpenSSH.

Прототип VPS-сервера у провайдера в моей виртуальной машине имеет адрес 192.168.1.16. Этот адрес доступен из моей локальной сети 192.168.1.0/24 равно как и с основного компьютера. Я могу подключиться к серверу Ubuntu-VPS через SSH, могу его «пропинговать». Вторая виртуальная машина получила адрес 10.0.2.15 и я не могу подключиться к Ubuntu извне: с основного компьютера, из локальной сети, даже с Ubuntu-VPS. А вот с Ubuntu я могу выйти в Интернет, могу подключиться к Ubuntu-VPS по SSH, могу подключиться к любому компьютеру в моей локальной сети. Так работает NAT.

```
viad@UbuntuServerX64-2:~$ ping 192.168.1.16
PING 192.168.1.16 (192.168.1.16) 56(84) bytes of data:
64 bytes from 192.168.1.16: icmp_seq=1 ttl=63 time=1.72 ms
64 bytes from 192.168.1.16: icmp_seq=2 ttl=63 time=1.07 ms
64 bytes from 192.168.1.16: icmp_seq=3 ttl=63 time=1.06 ms
64 bytes from 192.168.1.16: icmp_seq=4 ttl=63 time=0.709 ms
64 bytes from 192.168.1.16: icmp_seq=5 ttl=63 time=1.07 ms
64 bytes from 192.168.1.16: icmp_seq=6 ttl=63 time=0.769 ms
```

Сервер за NAT может достигаться до внешнего сервера

В качестве порта, который нужно будет вывести наружу из-за NAT, я буду использовать обычный http-порт с номером 80. Чтобы что-то на нем отвечало, я установлю на сервер Ubuntu веб-сервер Apache (устанавливаем через `'sudo apt install apache2'`). И собственно задача по прокидыванию будет решена, когда я с любого компьютера в моей локальной сети смогу получить веб-страничку Apache с сервера Ubuntu.

Осталось только решить при помощи чего организовать туннель между Ubuntu-VPS и Ubuntu.

## Варианты строительства туннеля между серверами

Вообще проблема соединений, безопасных соединений, двух и более компьютеров через публичные сети — стара как мир компьютеров. Поэтому еще в старые, добрые, «ламповые» времена на свет появилась технология [VPN](#). Она-то как раз и предназначена для создания безопасного соединения между компьютерами. На страницах своего блога я уже рассматривал проблематику строительства VPN-сетей как минимум дважды: «[Подключаемся к удаленному роутеру ZyXEL по IPsec VPN через StrongSwan на Headless Ubuntu 14LTS](#)» и «[SoftEther VPN — проходящий сквозь огненную стену](#)». И вроде бы у технологий VPN есть все, что нужно, чтоб создать туннель между серверами и позволить внешним пользователям подключаться к серверу за NAT. Но тут, как всегда, есть определенные нюансы:

- Далеко не всегда провайдер, особенно мобильный, с радостью пропустит через свое подключение VPN-соединение. Да и не каждый клиентский роутер позволит такую вольность. Поэтому вид VPN-соединения нужно очень тщательно выбирать и подбирать под конкретные условия.
- Вариантов VPN-соединений существует масса: SSTP, L2TP, IPsec, OpenVPN, PPTP и тому подобные. А еще ведь есть и разные реализации всего перечисленного.
- Неподготовленному, начинающему пользователю будет тяжело не только разобраться, но и установить, да настроить VPN-соединение. VPN штука непростая.

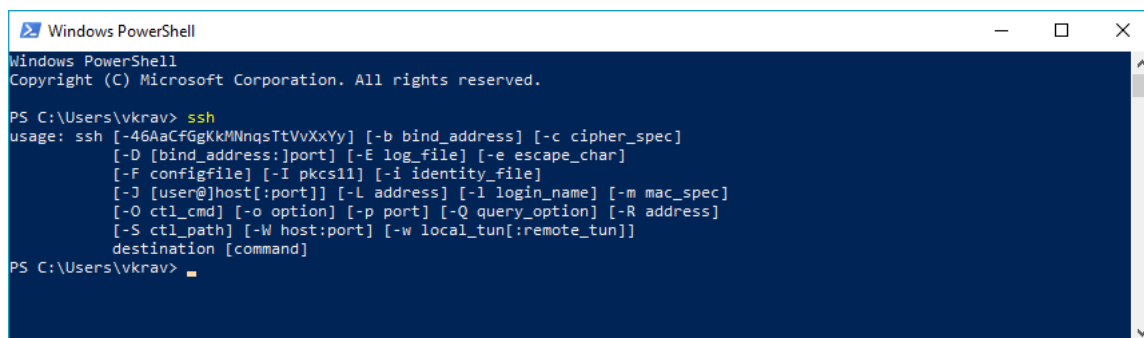
И как вывод: весь штат функций VPN несколько избыточен для решения такой простой задачи, как прокинуть порт-другой из-за NAT на внешний сервер, да открыть их там пользователям для доступа. Поэтому проводим изыскания далее.

Многие пользователи слышали, а некоторые активно используют проху-сервера. ~~Во времена, когда сетевые пакеты в Интернет можно было передавать дискетами.~~ Всего десять-пятнадцать лет тому назад пользователи подключались к глобальной сети посредством модемов по проводной телефонной сети. На стол ставилась коробочка, в которую подключались провода из телефонной розетки, коробочка дозванивалась до провайдера и начинала шипеть. Скорости тогда были мизерные, а каналы узенькие, и чтобы не состариться в ожидании, когда же прокачается очередная страничка, провайдеры устанавливали у себя могучие проху-сервера. Они кешировали популярные веб-странички у себя в памяти или на жестком диске и быстро отдавали сохраненное пользователю. Таким образом, существенно снижался трафик в сети, ведь пользователи постоянно ходят на одни и те же веб-сайты. Сейчас же проху-сервера используются больше для обхода различных блокировок. Пропал доступ к любимому сайту из-за того, что на нем разместили что-то нарушающее закон? Не беда! Подключаешься через проху и все заработало как и прежде! Но все, почему-то, забывают, что кроме привычных HTTP проху-серверов, человечеством была разработана технология [SOCKS](#). По сути, это те же самые проху, но разработанные специально для прохождения фаерволов (и как следствие NAT тоже).

Для операционной системы Ubuntu доступно сразу несколько реализаций SOCKS-серверов. Например, норвежский [Dante](#) или же российский [3proxy](#). И та и другая реализация куда легче, чем любой из традиционных VPN, настраиваются они тоже легче. Но, как оказалось, есть вариант еще проще и изящнее.

Протокол [SSH](#) появился на свет в своей первой инкарнации еще в 1995 году, когда Интернет еще только делала свои первые робкие шаги по планете в роли глобальной сети. С тех пор протокол, применяемый в основном для безопасного удаленного управления серверами, значительно окреп, возмужал и просочился даже в Windows. А в

Ubuntu OpenSSH вообще штатное средство, которое устанавливается в серверных редакциях системы даже без запроса пользователя.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\vkra> ssh
usage: ssh [-46AaCfGgKkMnNqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
          [-D [bind_address:]port] [-E log_file] [-e escape_char]
          [-F configfile] [-I pkcs11] [-i identity_file]
          [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]
          [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]
          [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
          destination [command]
```

SSH есть даже в PowerShell нод Windows 10

Все так привыкли, что SSH применяется только и исключительно для удаленного управления, что знания о том, что при помощи SSH можно создавать туннели, остались только у [аксакалов](#) из IT, да [саксаулов](#). А ведь банальный, штатный SSH, который уже установлен на обоих серверах, способен не только прокинуть порты из-за NAT, но и проверить куда более сложные задачи, вплоть до создания того же самого полноценного VPN-соединения, либо же работать просто как SOCKS-проху. Но не будем спешить и пройдемся по всем шагам, которые необходимо выполнить, чтобы тривиальная задача о проброске портов заработала. И если честно, то пришлось потратить изрядно времени, чтобы разобраться во всех нюансах настройки SSH-туннеля.

## Пробрасываем 80-й порт через NAT при помощи SSH-туннеля

Дальнейшая инструкция может быть не исчерпывающей. Но в моем случае она заработала без сбоев не только в боевых условиях, но и на двух независимых тестовых сборках в виртуальных машинах. Во избежание гибели человеческих жертв, настоятельно рекомендую проверить, что у вас, при попытке повторения эксперимента, есть консольный (не SSH) доступ к вашим серверам. Иначе, при простейшей ошибке в конфигурации вы можете оказаться без удаленного доступа к вашим машинам. И сразу же еще одно важное замечание для пользователей [SSHGuard](#) и подобных систем по защите от подбора паролей — будьте готовы, что он вас заблокирует, если вы начнете ломиться на сервер с ошибочными аутентификационными данными. Тут тоже поможет доступ через консоль (у VPS-хостеров, как правило, она доступна через веб). Сбросить временную блокировку вашего адреса можно перезагрузив SSHGuard следующей командой '[sudo service sshguard restart](#)'.

### Шаг 1. Создаем SSH-туннель между серверами.

Ну, что же. Приступим. У нас есть два сервера: Ubuntu-VPS и Ubuntu. На обоих уже установлена операционная система Ubuntu Server, а заодно и OpenSSH. На сервере Ubuntu-VPS желательно создать отдельного пользователя, который по своей настройке прав будет использоваться только и исключительно как пользователь для создания туннеля SSH. Каналы SSH хоть и надежны, но в случае компрометации сервера за NAT, злоумышленник сможет получить доступ и к VPS-серверу, если устанавливать туннель под каким-либо привилегированным пользователем.

При создании туннеля необходимо помнить, что иницируется туннелирование всегда с того сервера, что располагается за NAT. Соответственно в нашем случае мы должны создавать туннель с сервера Ubuntu. Поэтому логинимся на сервер Ubuntu и проверяем способность его подключиться к серверу Ubuntu-VPS при помощи команды **'ssh 192.168.1.16'**. Где 192.168.1.16 есть адрес сервера Ubuntu-VPS.

```
vlad@UbuntuServerX64-2:~$ ssh 192.168.1.16
vlad@192.168.1.16's password:
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.15.0-20-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Aug 11 02:01:27 MSK 2018

System load:  0.05          Processes:      85
Usage of /:   13.9% of 19.56GB Users logged in: 1
Memory usage: 12%          IP address for enp0s3: 192.168.1.16
Swap usage:   0%

 * Introducing Minimal Ubuntu for docker and clouds. 30 MB base image and
   optimised kernels on public clouds. Made for machines and containers.

   - https://bit.ly/minimal-ubuntu

142 packages can be updated.
64 updates are security updates.

Last login: Sat Aug 11 02:00:57 2018 from 192.168.1.12
```

*Подключение по SSH от сервера за NAT к удаленному серверу*

Если все хорошо и подключение установлено, то выходим из него через **'exit'** и приступаем к созданию туннеля при помощи команды **'ssh -N -R 8080:localhost:80 vlad@192.168.1.16'**. Где, параметр **'-N'** означает, что после создания туннеля никакая команда на той стороне не будет выполняться. Параметр **'-R 8080:localhost:80'** означает, что создается так называемый обратный (reverse) туннель. **8080** означает, что входной точкой туннеля на удаленном сервере (в нашем случае это 192.168.1.16 или же Ubuntu-VPS) будет выступать порт 8080. Параметр **'localhost'** означает адрес того сервера, чей порт будет пробрасываться. В нашем случае, когда используется ресурс с самого сервера можно использовать не только **'localhost'**, но и 127.0.0.1, либо же вообще прописать доменное имя или сетевой адрес нашего сервера Ubuntu. И последний параметр **'vlad@192.168.1.16'** передает имя пользователя, под которым на удаленном (Ubuntu-VPS) сервере будет создан туннель, а так же и сам адрес (или доменное имя) удаленного сервера. Напомню, что пока мы находимся на сервере Ubuntu.

```
vlad@UbuntuServerX64-2:~$ ssh -N -R 8080:localhost:80 vlad@192.168.1.16
vlad@192.168.1.16's password:
```

*Устанавливаем SSH-туннель между сервером за NAT и внешним сервером*

При подключении удаленный SSH-сервер запросит ввести пароль пользователя указанного для создания туннеля (в нашем случае это vlad). А после... После ничего происходить не будет, кроме как отображение мигающего стимула командной строки. Собственно туннель уже создан и он работоспособен. Осталось только его проверить. Для этого подключаемся к серверу Ubuntu-VPS (на сервере Ubuntu установлен Apache2 со штатным конфигом) и запускаем команду **'wget localhost:8080'**

```
vlad@UbuntuServerX64-2:~$ wget localhost:8080
--2018-08-11 02:21:31-- http://localhost:8080/
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10918 (11K) [text/html]
Saving to: 'index.html.1'

index.html.1      100%[=====>] 10.66K --.-KB/s   in 0s

2018-08-11 02:21:31 (217 MB/s) - 'index.html.1' saved [10918/10918]
```

*Проверка при помощи wget доступность ресурса сервера за NAT*

Если все работает, то утилита wget скачает html-страничку доступную на 8080-порту сервера Ubuntu-VPS. Это значит, что SSH-туннель уже есть и он работает! И им уже почти можно пользоваться. Если попробовать подключиться к серверу Ubuntu-VPS на порт 8080 с основной машины или же из локальной сети, то скорее всего браузер вернет ошибку подключения. Как от нее избавиться будет пояснено в шаге 2. Но если в вашей установке браузер сразу же отобразит стандартную html-страничку Apache, то можно смело переходить к шагу 3.

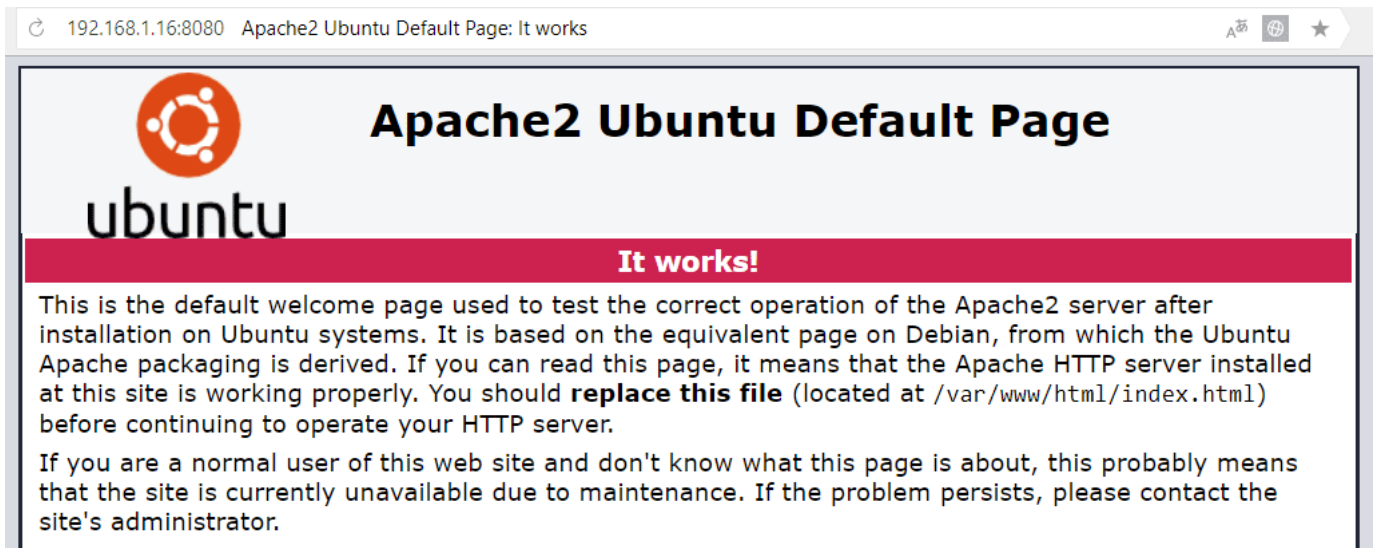
Небольшое замечание по поводу номера порта. Почему я выбрал на Ubuntu-VPS порт 8080, а не 80? Ответ тут прост. Конечно же, можно было бы сформировать параметр и как **'80:localhost:80'**, тогда страничка открывалась бы на стандартном порту, но использование портов с номерами ниже 1024 требует администраторских прав. Но ведь мы же не хотим создавать туннель с административным пользователем? Не хотим же? В крайнем случае, можно воспользоваться переназначением портов уже на Ubuntu-VPS, например, через IPTables.

Кстати, туннель можно запускать, не занимая терминальную сессию. Для этого можно применить ключ **'-f'**. В таком случае, после подключения и ввода пароля, ssh на машине-инициаторе туннель уйдет в фон и в этой же терминальной сессии можно будет продолжать работать как и прежде. Но, данный ключик доступен не во всех версиях SSH, поэтому попробуйте обновиться, если вдруг SSH будет ругаться на неверный ключ.

## Шаг 2. Делаем доступным порт 8080 для доступа извне.

Если при установленном и работающем туннеле получить страничку Apache можно только с сервера Ubuntu-VPS, то значит в настройках сервера SSH необходимо включить доступ к пробрасываемым через туннель портам.

Редактируем любым редактором конфигурационный файл SSH, расположенный по пути `'/etc/ssh/sshd_config'` на Ubuntu-VPS и записываем там строчку (или убираем символ комментария #) `'GatewayPorts yes'`. Сохраняем и перезапускаем SSH-сервер командой `'sudo service sshd restart'`.



*Дефолтная страничка от Apache получена с сервера посредника*

После перезагрузки сервера SSH, устанавливаем туннель и пробуем зайти на Ubuntu-VPS через браузер на основном компьютере. Все работает. И на этом можно было и закончить, если бы не потребность в большей автономности и автоматизации туннеля. Поэтому двигаемся далее.

### Шаг 3. Подключаемся без ввода пароля.

Профессионалы рекомендуют подключаться по SSH не посредством ввода пароля, а при помощи аутентификации по сертификату. Дескать, так работает быстрее, да и надежнее в плане безопасности. Не будем с ними спорить, а применим эту практику, тем более что без аутентификации через сертификат о нормальном автоматическом старте туннеля речь не идет. Кому-то так или иначе придется вводить пароль от пользователя, под которым осуществляется создание туннеля.

Программисты, в целом, народец ленивый. Поэтому для устранения лишних действий по генерации сертификатов и их передаче на удаленный сервер были реализованы соответствующие подпрограммы. Ничего странного в этом нет, легче один день потратить на написание кода по удаленной установке сертификатов, зато потом все сделать за пять минут.

Операцию необходимо проводить на том компьютере, который устанавливает туннель, в нашем случае это сервер Ubuntu. Наличие подключенного туннеля необязательно. Для начала генерируем ключи аутентификации при помощи команды `'ssh-keygen'`.

```
vlad@UbuntuServerX64-2:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vlad/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vlad/.ssh/id_rsa.
Your public key has been saved in /home/vlad/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:x4/k/eBmCo4W3zHaBNBz91zSiUQ8M vlad@UbuntuServerX64-2
The key's randomart image is:
+----[RSA 2048]-----+
|
| . o = 0
| . B = +
| . +=
| . o + 0
| S . 0
| . 0 + B
| oo = + o E .
| oo . 0 0 0 + 0
| . + .
+----[SHA256]-----+
```

*Генерация сертификатов*

При генерации сертификатов нет необходимости вводить кодовую фразу (passphrase), ее желательно оставить пустой. Это уменьшает безопасность в целом, но, в противном случае, кодовую фразу придется вводить каждый раз при подключении к удаленному серверу, либо выдумывать очередные грабли по обходу. Следующим шагом необходимо передать наш сертификат на удаленный сервер, т.е. на Ubuntu-VPS. Операция осуществляется всего одной командой `'ssh-copy-id -i ~/.ssh/id_rsa.pub vlad@192.168.1.16'`. Где, ключ `'-i ~/.ssh/id_rsa.pub'` указывает на публичный ключ сгенерированный ранее. Обратите внимание, что публичный и секретный ключи были сгенерированы в домашней директории пользователя, под которым и производилась генерация, в папке `'ssh'`. Второй параметр `'vlad@192.168.1.16'` определяет пользователя, под которым произойдет подключение к удаленному SSH-серверу для установки сертификата публичного ключа. При подключении придется ввести пароль этого пользователя. Соответственно сам пользователь у нас `'vlad'` и адрес `192.168.1.16` принадлежат нашему серверу Ubuntu-VPS.

После установки сертификата можно провести проверку, работает ли сертификат. Для этого подключаемся по SSH к Ubuntu-VPS командой `'ssh vlad@192.168.1.16'`. Если все настроено верно, то SSH-сессия откроется без запроса пароля, и мы увидим консоль удаленной машины. Все работает, выходим из сессии SSH через `'exit'`.

Далее, необходимо добавить сгенерированный секретный ключ в агента сертификации ssh-agent. На этом этапе могут возникать различные по своей природе трудности, истинная причина которых может быть ясна далеко не всегда и с ходу. На всех моих серверах при попытке добавления сертификата секретного ключа в агента выпадает ошибка `"Could not open a connection to your authentication agent"`.

Как правило, проблема заключается в том, что на сервере не запущен SSH-Agent и не установлена переменная среды окружения '**SSH\_AUTH\_SOCK**'. Без этой переменной SSH, при подключении, не знает, куда именно подключаться для получения сертификата секретного ключа. Опять же речь тут идет про сервер-инициатор создания туннеля.

```
vlad@UbuntuServerX64-2:~$ eval "$(ssh-agent)"
Agent pid 13632
vlad@UbuntuServerX64-2:~$ ssh-add
Identity added: /home/vlad/.ssh/id_rsa (/home/vlad/.ssh/id_rsa)
```

*Добавление сертификата в ssh-agent*

Но запустить просто так агента не выйдет. В сети есть несколько вариантов того, как можно побороться с ошибкой. Мне гарантировано помогает следующий способ: запускаем сертификационного агента через команду '**eval "\$(ssh-agent)"**'. Агент запускается и устанавливает требуемую переменную окружения. После чего можно запускать и '**ssh-add**', который добавит сертификат куда следует.

Обратите внимание, что если вы не добавите сертификат в сертификационного агента, то возможно возникновение проблем при запуске туннеля в автоматическом режиме.

PS. Вообще, после поверхностного изучения функций SSH-Agent создается впечатление, что все работает и без него, если сертификаты созданы без использования passphrase. А агент требуется как раз для того, чтобы запускать соединение SSH с сертификатом который был зашифрован при помощи кодовой фразы, но без ее ввода. При этом сам агент должен быть запущен на сервере, чтоб данный механизм сработал.

## Шаг 4. Автоматически восстанавливаем SSH-туннель.

Если в условиях виртуальной машины соединение между двумя ее экземплярами сеть настолько стабильная, насколько стабилен сам компьютер, на котором крутится виртуалка, то в реальной жизни каналы связи оставляют желать много лучшего. Особые радости жизни привнесут соединения через радиоканал, например, как в поставленной задаче. В этом случае разумно было бы как-то отслеживать наличие соединения и его автоматически восстанавливать.

Поступить тут можно двумя способами. В первом случае мы мониторим работоспособность туннеля и в случае его отсутствия перезапускаем туннель. В SSH присутствует стандартная схема для мониторинга состояния подключения, причем как на стороне сервера, так и на стороне клиента. Суть ее заключается в том, что SSH будет проверять наличие рабочего подключения и в случае отсутствия такового будет просто завершать SSH-сеанс, избавляя нас от зависших сессий.

На серверной части, а именно на Ubuntu-VPS, редактируем конфигурационный файл SSH-сервера '**/etc/ssh/sshd\_config**'. В него добавляем (или раскомментируем) следующие параметры:

```
TCPKeepAlive yes

ClientAliveInterval 30

ClientAliveCountMax 3
```

И перезагружаем SSH-сервер командой '**sudo service sshd restart**'. Параметр '**TCPKeepAlive yes**' включает мониторинг работоспособности ssh-подключения. Параметр '**ClientAliveInterval 30**' проверяет живость клиентского подключения каждые 30 секунд, впрочем, параметр можно установить на усмотрение администратора. Параметр '**ClientAliveCountMax 3**' означает количество попыток, которые будут произведены с интервалом указанным в ClientAliveInterval до того, как соединение будет закрыто со стороны сервера. В нашем примере, если что-то произойдет с клиентом, например, компьютер просто отключится от сети, то через 90 секунд Ubuntu-VPS закроет туннельное соединение.

При создании туннельного подключения со стороны клиента есть возможность указать аналогичные параметры при подключении. Устанавливаются они через параметр '**-o**'. В нашем случае строка подключения примет следующую форму: '**ssh -N -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -R 8080:localhost:80 vlad@192.168.1.16**'. Обратите внимание, что параметры на клиенте слегка отличаются от тех, что на сервере. Важно их не перепутать.

Далее, любым автоматизированным средством, например, через периодические задачи cron, мы отслеживаем наличие процесса с ssh-туннелем в списке задач и в случае его отсутствия запускаем процесс еще раз. Но есть вариант удобнее и практичнее. Саксаулы рекомендуют использовать [AutoSSH](#), как вершину ленивости программистской мысли.

AutoSSH уже входит в репозитории Ubuntu, поэтому устанавливаем ее через '**sudo apt install autossh**' на сервер-клиент Ubuntu.

После установки AutoSSH наша строка для подключения на Ubuntu примет следующую форму '**autossh -M 0 -N -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -R 8080:localhost:80 vlad@192.168.1.16**'. В команде мало что поменялось, ssh заменился на autossh, добавился параметр '**-M 0**'. Изначально предполагалось, что AutoSSH будет мониторить порт (и даже два, один для отправки пакета KeepAlive, второй для его приема), указанный в параметре '**-M**' на наличие отклика. Этот порт должен быть свободным, да и на той стороне что-то должно отвечать, что, мол, сервис жив, все работает. Но, при наличии KeepAlive в самом SSH, необходимость тратить порты и городить ответную часть — отпала. Поэтому просто выключаем эту функцию через приведенный параметр.

При некоторых пертурбациях на сервере, проброска порта может не состояться, даже если был поднят туннель. То есть, со стороны SSH-сервера и SSH-клиента, все будет выглядеть нормально, туннель будет существовать и работать, но вот порт не будет проброшен от одного сервера к другому. Для автоматической перезагрузки туннеля в таком случае стоит применить дополнительный параметр '**-o "ExitOnForwardFailure=yes"**'. В таком случае, если возникает проблема с проброской портов, то SSH-туннель будет отключаться. И повторно подключаться через AutoSSH. Результирующая строка подключения трансформируется в следующее: '**autossh -M 0 -N -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -o "ExitOnForwardFailure=yes" -R 8080:localhost:80 vlad@192.168.1.16**'.

## Шаг 6. Поднимаем SSH-туннель при перезапуске сервера.

На текущем шаге все работает, но вот только после перезагрузке сервера необходимо каждый раз запускать руками ssh-туннель. Небольшое неудобство, которое стойко перенесет бухгалтер или инженер, но никак не программист. Поэтому необходимо каким-то образом запустить туннель в автоматическом режиме при старте сервера. Аксакалы сходу предложат вариант через CRON, но есть вариант более жесткий с применением systemd. Рассмотрим оба способа.



## Создаем сервис в systemd

Создаем в директории `/etc/systemd/system/` сервера Ubuntu файл с наименованием нашего сервиса, например, `'autossh-tunnel.service'`. Не забыв про права суперпользователя, вызываем редактор `'sudo nano /etc/systemd/system/autossh-tunnel.service'` и забиваем туда следующий текст:

```
[Unit]

Description=AutoSSH tunnel to Ubuntu-VPS

After=network-online.target

[Service]

Environment="AUTOSSH_GATETIME=0"

ExecStart=/usr/bin/autossh -M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -o "ExitOnForwardFailure=yes" -i /home/vlad/.ssh/id_rsa -N -R 8080:localhost:80 vlad@192.168.1.16

[Install]

WantedBy=multi-user.target
```

Здесь стоит дать некоторые пояснения. Параметр `'After=network-online.target'` означает, что запуск произойдет только тогда, когда на сервере будет осуществлено подключение к сети, когда сервер получит свой IP-адрес и вообще сможет выходить в сеть. Параметр `'Environment="AUTOSSH_GATETIME=0"'` действует аналогично ключу `'-f'`, т.е. запускает исполнение программы в виде фонового процесса. Попытка использования `'-f'` в systemd приведет к ошибке. Параметр `'WantedBy=multi-user.target'` означает, что сервис будет запущен, когда произойдет нормальная загрузка системы и будет доступна неграфическая оболочка для пользователей.

Поскольку по умолчанию туннель на клиентской машине запускается от имени суперпользователя, то в строку инициализации AutoSSH добавляем новый параметр `'-i /home/vlad/.ssh/id_rsa'`. Он указывает путь к сертификату секретного ключа, который был создан на шаге 3.

Если при подключении к удаленному серверу возникает ошибка `"Host key verification failed"`, то следует применить следующее:

- Осуществить запуск AutoSSH от имени суперпользователя `'sudo /usr/bin/autossh -M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -o "ExitOnForwardFailure=yes" -i /home/vlad/.ssh/id_rsa -N -R 8080:localhost:80 vlad@192.168.1.16'`. При этом SSH попросит подтвердить открытый сертификат удаленного сервиса, после этого ошибка `"Host key verification failed"` пропадет.
- Альтернативой можно добавить новый параметр в команду запуска SSH, даже два, которые позволяют пропускать проверку сертификата удаленного сервера. Для этого добавляем `'-o "UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no"'`.

Свистопляска связана с тем, что изначально мы запускали наш туннель исключительно под пользователем vlad, и все ключи были сгенерированы именно для него. Поэтому нужно было либо принять ключ удаленного сервера для пользователя root, т.е. запустив туннель через sudo и приняв приглашение системы о приеме ключа. Либо установить опцию об игнорировании проверки удаленного ключа. Игнорирование позволит продолжать работать с удаленным сервером, даже если на нем была произведена замена операционной системы, изменены сертификаты ключей, подменен сервер на другой. Конечно, это повышает стабильность работы туннеля, но с другой стороны совершенно не гарантирует безопасность такой работы. Поэтому стоит поискать компромисс...

Двигаемся далее. Перезагружаем сервисы в systemd при помощи команды `'sudo systemctl daemon-reload'`. Запускаем сервис командой `'sudo systemctl start autossh-tunnel.service'`. И проверяем, что у нас поднялся туннель. Для этого можно просто зайти через браузер на основной машине по адресу `'http://192.168.1.16:8080'`. Должна открыться страничка от Apache. Если же страничка не открывается, то необходимо смотреть и проверять лог-файл syslog на предмет сообщений об ошибках.

И если все работает так как следует, то включаем автозагрузку сервиса при старте системы командой `'sudo systemctl enable autossh-tunnel.service'`.

## Создаем задачу в планировщике

Запускать туннель от имени суперпользователя, наверное, не очень здорово, но мне так и не удалось запустить его от пользователя vlad при автоматической загрузке через systemd. Несмотря на все мои ухищрения, запуск упорно происходит от пользователя root при любых раскладах. Поэтому я поступил, как самый банальный из нас, и прописал запуск туннеля SSH в обычный планировщик на сервере Ubuntu.

```
CPU [ 0.7%] Tasks: 32, 70 thr; 1 running
Mem [ 85.0M/985M] Load average: 0.93 0.34 0.12
Swp [ 0K/947M] Uptime: 00:00:57

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
733 vlad 20 0 4628 816 748 S 0.0 0.1 0:00.00 /bin/sh -c /usr/bin/autossh -M 0 -o
745 vlad 20 0 4528 772 712 S 0.0 0.1 0:00.00 /usr/lib/autossh/autossh -M 0 -o Ser
793 vlad 20 0 48128 5812 5140 S 0.0 0.6 0:00.04 /usr/bin/ssh -o ServerAliveInterval
827 root 20 0 72296 5688 4964 S 0.0 0.6 0:00.00 /usr/sbin/sshd -D
```

*Проверка под каким пользователем загружен процесс*

Для этого я, не находясь с правами суперпользователя, т.е. без использования sudo, ввел команду `'crontab -e'`. После чего ввел следующую инструкцию планировщика: `'@reboot /usr/bin/autossh -M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" -o "ExitOnForwardFailure=yes" -i /home/vlad/.ssh/id_rsa -N -R 8080:localhost:80 vlad@192.168.1.16'`. Не забыв в конце текста вставить переход на новую строку и возврат каретки (просто нажал Enter). В результате, при запуске системы, еще до входа пользователя, автоматически поднимается туннель от не root пользователя. Собственно, что и требовалось реализовать.

## Шаг 7. Прокидываем несколько портов через SSH-туннель.

При желании можно создать сразу несколько туннелей. Для этого можно просто клонировать сервисы для systemd, размножать задачи в CRON или же запускать туннели в ручном режиме с установкой флага `-f`.

## Вместо заключения

При работе туннелей я не заметил какой-то уж заметной нагрузки, как на сервер, так и на клиента. Похоже, что даже дохлые современные сервера способны справляться с нагрузкой туннелирования играючи.

Знаток Linux автоматически посоветуют начать копать как в сторону ключа `'-w'`, так и в сторону директивы `sshconfig`. Но, оставлю данное упражнение для любителей залезть поглубже в тему.

**Upd1.** Если запуск из-под cron не выходит, ssh-сессия завершается с ошибкой, например как-то так `"ssh exited prematurely with status 255; autossh exiting"`, то стоит попробовать сделать три вещи:

1. Прописать в строку запуска autossh в cron `'-i /home/vlad/.ssh/id_rsa'`, тем самым указав программе точно, где лежит ключ (сертификат).
2. Дополнительно стоит попробовать указать ключ `'-f'` сразу после autossh. При помощи этого ключа мы указываем, что сессия ssh должна запуститься в фоновом режиме.
3. Прописать пути ко всем зависимым директориям в crontab включая и директорию где располагаются ключи (сертификаты). В моем случае путь выглядит примерно так `PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/home/vlad/.ssh`

Во всех случаях выше необходимо заменить `vlad` на пути в ваших системах.

## Полезные ссылки

- Боремся с ошибкой `ssh-add Could not open a connection to your authentication agent`. [Часть 1](#).
- Боремся с ошибкой `ssh-add Could not open a connection to your authentication agent`. [Часть 2](#).
- Боремся с ошибкой `ssh-add Could not open a connection to your authentication agent`. [Часть 3](#).
- [Автоматическое](#) подключение туннеля SSH при помощи AutoSSH и использование сервиса для запуска туннеля при автозагрузке.
- [Различия](#) в `network*.target` в systemd.
- [Mosh](#) – еще одна альтернатива SSH для нестабильных соединений.
- [ssh-agent](#) и `ssh-add`.