SSH

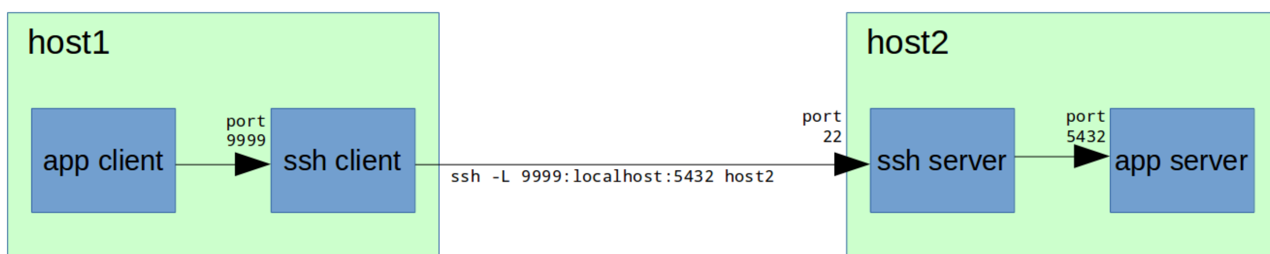
С SSH многие знакомы давно, но, как и я, не все подозревают о том, какие возможности таятся за этими магическими тремя буквами. Хотел бы поделиться своим небольшим опытом использования SSH для решения различных административных задач.

Оглавление:

1) Local TCP forwarding
2) Remote TCP forwarding
3) TCP forwarding chain через несколько узлов
4) TCP forwarding ssh-соединения
5) SSH VPN Tunnel
6) Коротко о беспарольном доступе
7) Спасибо (ссылки)

1) Local TCP forwarding

Начнем с простого — local TCP forwarding:



Имеем удаленный сервер «host2» с неким приложением, допустим, PostgreSQL server, которое принимает TCP-соединения на порту 5432. При этом вполне логично, что на этом сервере стоит файрвол, который прямых соединений извне на порт 5432 не разрешает, но при этом есть доступ по SSH (по-умолчанию порт 22, рекомендую его изменить). Требуется подключиться с нашего рабочего места «host1» клиентским приложением к серверу PostgreSQL на «host2».

Для этого на «host1» в консоли набираем:

```
host1# ssh -L 9999:localhost:5432 host2
```

Теперь на «host1» мы можем соединяться с PostgreSQL сервером через локальный порт 9999:

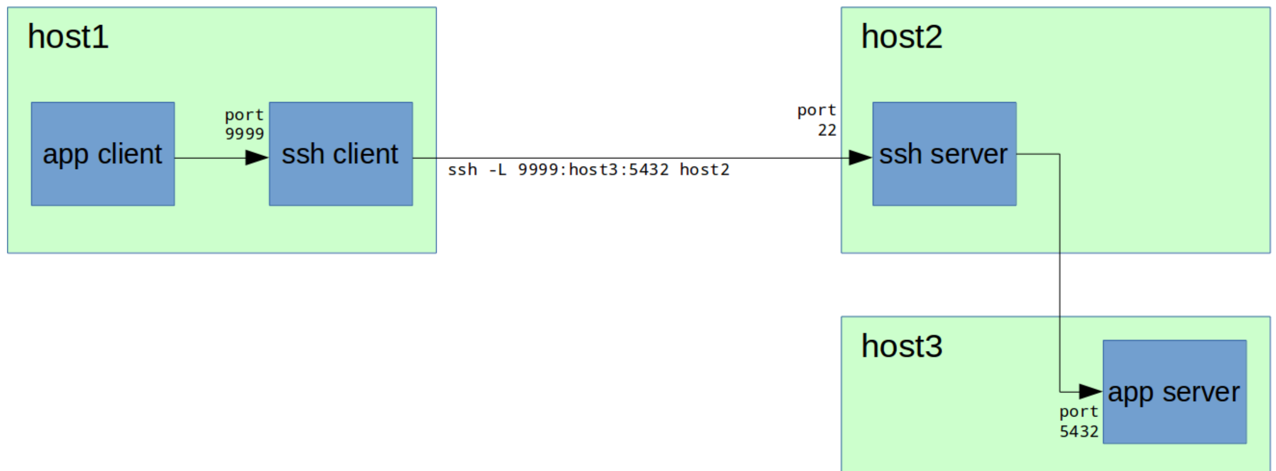
```
host1# psql -h localhost -p 9999 -U postgres
```

Если на «host1» Windows

Как это работает

Настройка SSH-сервера

Мы также можем соединяться с приложением не на самом «host2», а на любой доступной ему машине:

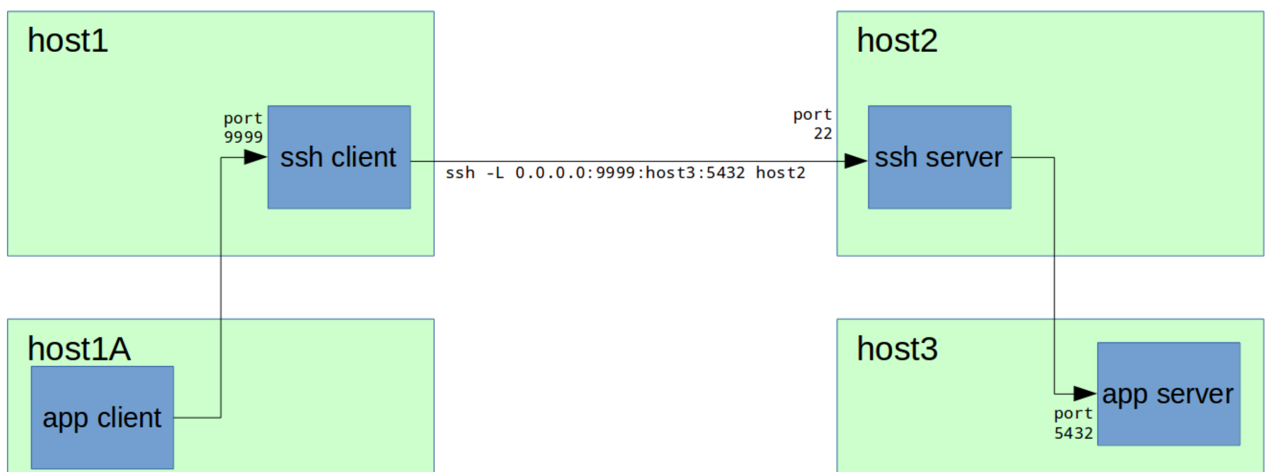


Для этого при пробросе портов вместо «localhost» указываем имя хоста, например «host3»:

```
host1# ssh -L 9999:host3:5432 host2
```

Тут важно заметить, что «host3» должен быть известен (если это имя, а не IP-адрес) и доступен для машины «host2».

Также можно через «host1» предоставить доступ любому другому узлу (назовем его «host1A») к сервису на «host3»:



Для этого нужно вставить в команду соединения ssh IP-адрес интерфейса, на котором будет поднят локальный порт 9999:

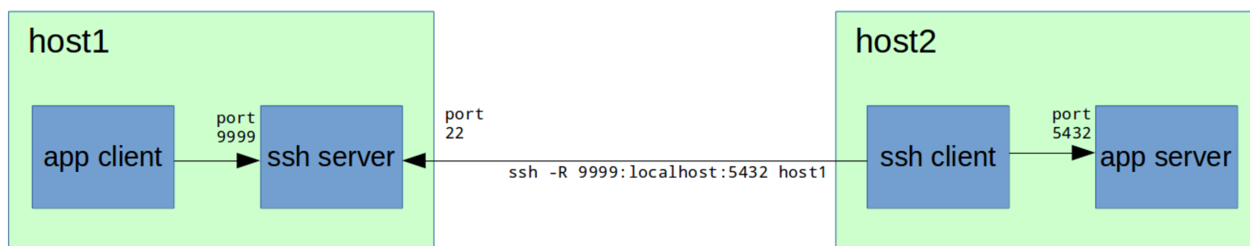
```
ssh -L 0.0.0.0:9999:host3:5432 host2
```

В данном примере порт 9999 будет открыт на всех доступных на «host1» IPv4 интерфейсах.

2) Remote TCP forwarding

Но что делать, если, например, «host2» не имеет белого IP-адреса, находится за NAT или вообще все входящие соединения к нему закрыты? Или, например, на «host2» стоит Windows и нет возможности поставить SSH-сервер?

Для этого случая есть Remote TCP forwarding:



Теперь нужно устанавливать ssh-соединение в обратном направлении — от «host2» к «host1». Т.е. наша административная рабочая станция будет SSH-сервером и будет доступна по SSH с «host2», а на «host2» нужно будет выполнить подключение SSH-клиентом:

```
ssh -R 9999:localhost:5432 host1
```

Если на «host2» Windows

Как это работает

Также у вас возникнут дополнительные сложности с обеспечением безопасности на «host1», если вы не доверяете узлу «host2». Однако это выходит за рамки данной статьи.

И, конечно, вы каким-то образом (сами или с посторонней помощью) должны инициировать ssh-соединение со стороны «host2» вводом приведенной выше команды, а «host1» должен иметь белый IP-адрес и открытый порт SSH.

После установки ssh-соединения все работает аналогично предыдущей главе.

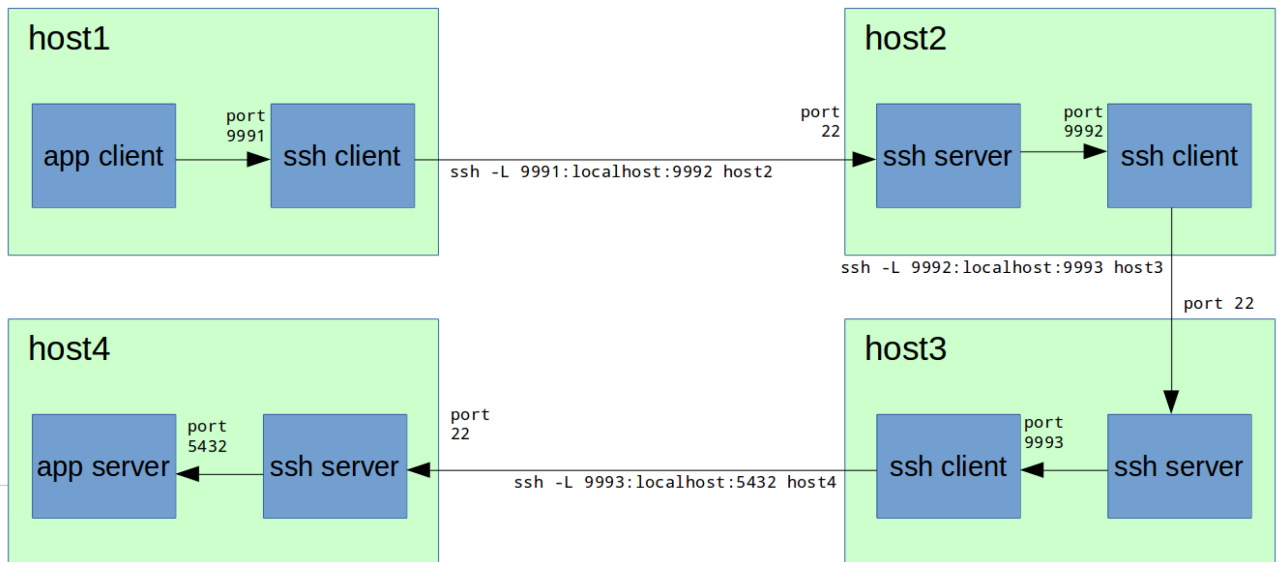
3) TCP forwarding chain через несколько узлов

В закрытых сетях часто бывает, что нужный нам узел напрямую недоступен. Т.е. мы можем зайти на нужный хост только по цепочке, например host1 → host2 → host3 → host4:

```
host1# ssh host2
host2# ssh host3
host3# ssh host4
host4# echo hello host4
```

Это может происходить например если эти узлы являются шлюзами, либо если на них доступны шлюзы только в соседние подсети.

В таком случае мы также можем делать TCP forwarding по цепочке:



Здесь порты 9991, 9992, 9993 выбраны для наглядности, на практике можно использовать один и тот же порт (например, 9999), если он свободен на всех узлах.

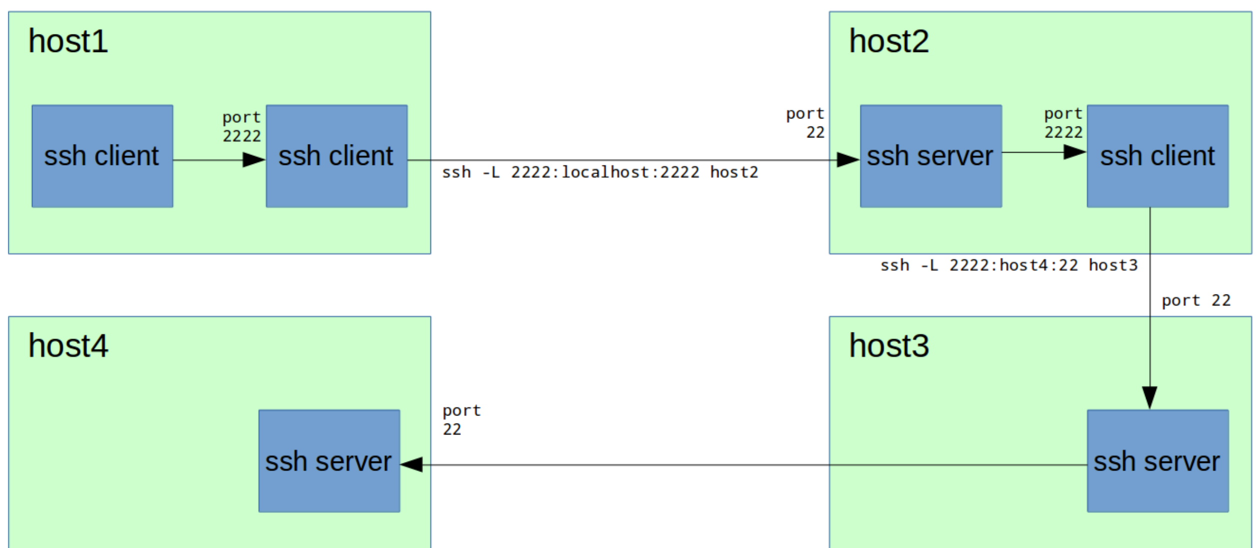
Итого нужно выполнить следующую цепочку команд:

```
host1# ssh -L 9991:localhost:9992 host2
host2# ssh -L 9992:localhost:9993 host3
host3# ssh -L 9993:localhost:5432 host4
```

Как это работает

4) TCP forwarding ssh-соединения

Иногда бывает нужно соединиться по ssh с сервером, который напрямую недоступен, а доступ возможен только по цепочке ssh-серверов (см. предыдущую главу). Теперь мы обладаем нужными знаниями чтобы сделать следующее:



```
host1# ssh -L 2222:localhost:2222 host2
host2# ssh -L 2222:host4:22 host3
```

Таким образом, на порту 2222 на «host1» у нас теперь есть форвардинг на порт SSH (22) на «host4». Можем соединиться:

```
host1# ssh -p 2222 localhost
host4# echo hello host4
```

Казалось бы, зачем это нужно? Например, вот зачем:

```
# копируем файл на host4
host1# scp -P 2222 /local/path/to/some/file localhost:/path/on/host4
# копируем файл с host4
host1# scp -P 2222 localhost:/path/on/host4 /local/path/to/some/file
# делаем еще один замечательный TCP forwarding на host4
host1# ssh -p 2222 -L 9999:localhost:5432 localhost
host1# psql -h localhost -p 9999 -U postgres
# обратите внимание, что порт для команды ssh задается ключем -p в нижнем
регистре,
# а для команды scp -P в верхнем регистре
```

Ну и вообще, здорово что теперь «host4» так близко :)

Вывод: можно делать TCP forwarding большого уровня вложенности.

Замечания про RSA fingerprint

5) SSH VPN Tunnel

TCP port forwarding — это отличная возможность. Но что если нам нужно больше? Доступ по UDP, доступ к множеству портов и хостов, доступ к динамическим портам? Ответ очевиден — VPN. И всемогущий SSH начиная с версии 4.3 и здесь придет нам на помощь.

Забегая вперед скажу: этот функционал SSH хорошо работает если вам нужно временное решение для каких-то административных задач. Для построения постоянных VPN этот вариант далеко не самый подходящий, т. к. он предполагает TCP-over-TCP, что плохо скажется на скорости соединения.

Еще про TCP forwarding

Настройка SSH-сервера:
PermitTunnel в настройках sshd по-умолчанию выключен, его нужно включить в /etc/ssh/sshd_config:

```
PermitTunnel yes
```

или

```
PermitTunnel point-to-point
```

ВАЖНО: для поднятия нового сетевого интерфейса туннеля и на ssh-клиенте, и на ssh-сервере необходимы права суперпользователя. Можно долго спорить о том, насколько это небезопасно, но в большинстве случаев на ssh-сервере достаточно настройки:

```
PermitRootLogin without-password
```

Таким образом вы запрещаете вход root по паролю, а разрешаете только другими средствами, например, по ключу RSA, что гораздо безопаснее.

Перезапускаем sshd:
```
sudo service sshd restart # centos
```
или
```
/etc/init.d/ssh restart # (debian/ubuntu)
```

Туннель поднимается при использовании магического ключа -w:

```
host1# sudo ssh -w 5:5 root@host2
```

Где 5:5 — номер интерфейса на локальной машине и на удаленной соответственно. Здесь вас может смутить, что ifconfig не выдаст в списке интерфейса «tun5». Это потому что он в состоянии «down», а вот если вызвать «ifconfig -a» или «ifconfig tun5», то интерфейс будет виден:

```
host1# ifconfig tun5
tun5 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00
POINTOPOINT NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Назначаем интерфейсам IP-адреса и поднимаем их:

```
host1# sudo ifconfig tun5 192.168.150.101/24 pointopoint 192.168.150.102
host2# sudo ifconfig tun5 192.168.150.102/24 pointopoint 192.168.150.101
```

Если есть файрвол, не забываем разрешить соединения с интерфейса tun5:

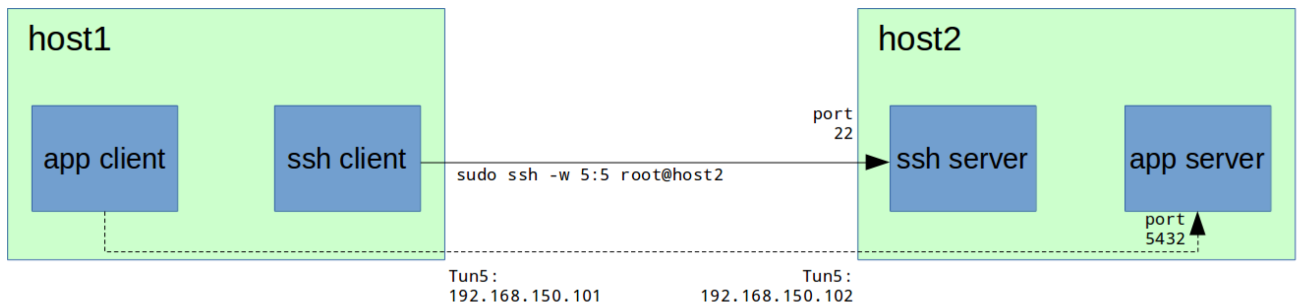
```
host1# # сохраняем исходные правила файрвола
host1# sudo iptables-save > /tmp/iptables.rules.orig
host1# sudo iptables -I INPUT 1 -i tun5 -j ACCEPT
host2# # сохраняем исходные правила файрвола
host2# sudo iptables-save > /tmp/iptables.rules.orig
host2# sudo iptables -I INPUT 1 -i tun5 -j ACCEPT
```

На host1 это делать необязательно, здесь это сделано лишь для того чтобы ping работал в обе стороны.

Наслаждаемся пингом:

```
host1# ping 192.168.150.102
host2# ping 192.168.150.101
```

Если рассмотреть более ранний пример с PostgreSQL, то теперь схема будет такая:



А команда для подключения к серверу PostgreSQL будет выглядеть так:

```
host1# psql -h 192.168.150.102 -U postgres
```

Ну а далее можно делать какой-либо из этих узлов шлюзом, если нужно обеспечить доступ не к одному узлу, а к сети. Например:

```
host2# # разрешаем IP forwarding
host2# sudo sysctl -w net.ipv4.ip_forward=1
host2# # разрешаем IP forwarding с host1
host2# sudo iptables -I FORWARD 1 -s 192.168.150.101 -j ACCEPT
host2# # разрешаем IP forwarding на host1
host2# sudo iptables -I FORWARD 1 -d 192.168.150.101 -j ACCEPT
host2# # маскируем IP адрес host1
host2# sudo iptables -t nat -A POSTROUTING -s 192.168.150.101 -j
MASQUERADE

host1# # Предположим, у host2 есть доступ к сети 192.168.2.x, куда нам
нужно попасть с host1
host1# # Прописываем host2 как шлюз в сеть 192.168.2.x
host1# sudo ip route add 192.168.2.0/24 via 192.168.150.2
host1# # Наслаждаемся доступом в сеть с host1
host1# ping 192.168.2.1
```

После окончания работы не забываем вернуть net.ipv4.ip_forward и файрвол в исходное состояние.

```
host1# sudo iptables-restore < /tmp/iptables.rules.orig
host2# sudo iptables-restore < /tmp/iptables.rules.orig
```

6) Коротко о беспарольном доступе

Думаю, все уже знают что авторизация по паролю это не про нас. Но на всякий случай впихну сюда краткую инструкцию по настройке аутентификации по ключу RSA:

1. На клиентских машинах генерируем пользователю свой ключ RSA:

```
client1# ssh-keygen -t rsa
```

По-умолчанию приватный ключ сохраняется в ~/.ssh/id_rsa, а открытый — в ~/.ssh/id_rsa.pub. Приватный ключ храните как зеницу ока и никому не давайте, никуда не копируйте.

При создании ключа можно задать пароль (passphrase), которым ключ будет зашифрован.

2. Клиентские открытые ключи нужно сохранить на ssh-сервере в файле ~/.ssh/authorized_keys (~ это домашняя директория того пользователя, которым будете логиниться), каждый на отдельной строке. Для того чтобы это не делать вручную, на каждом клиенте можно воспользоваться командой:

```
ssh-copy-id user@sshserver
```

Где user — имя пользователя на сервере, sshserver — имя или IP-адрес ssh-сервера.
Права на файл ~/.ssh/authorized_keys

3. Проверьте, что можете зайти на сервер по ключу, без ввода пароля (не путать с passphrase):
```
ssh user@sshserver
```
Рекомендую не закрывать хотя бы одну активную ssh-сессию с сервером до тех пор, пока окончательно не закончите настройку и не убедитесь что все работает.

4. Отключите на SSH-сервере возможность входа по паролю в файле /etc/ssh/sshd_config:

```
PasswordAuthentication no
```

Возможность входа по открытому ключу обычно уже включена по-умолчанию:

```
PubkeyAuthentication yes
```

Я обычно также отключаю две следующие опции:

```
GSSAPIAuthentication no
UseDNS no
```

В некоторых случаях это позволяет ускорить процесс соединения (например, когда на сервере нет доступа в Интернет).

5. Перезапустите sshd:
```
service sshd restart
```
или
```
/etc/init.d/ssh restart
```

В случае ошибок полезно бывает смотреть лог /var/log/secure либо использовать опции -v, -vv или -vvv для вывода детального лога соединения:
```
ssh -vvv user@sshserver
```