# 36. SSH-форвардинг. Назначение, принципы работы.

OpenSSH клиент встроен в Linux и OS X. На Windows придется использовать PuTTY или другой SSH-клиент. Вернемся к Linux,откроем терминал и введем:

ssh root@host

Часть с SSH даст системе знать, что вы хотите открыть SSH соединение. **{user}** даст системе сведения об аккаунте к которому вы хотите подключиться. К примеру, вы захотите получить доступ в качестве **root** пользователя, который фактически является синонимом для администратора системы с полными правами на изменение любых данных в системе. **{host}** означает компьютер к которому вы хотите подключиться. Это может быть IP адрес или доменное имя. На машину, к которой мы подключаемся должен быть установлен SSH сервер, если это Ubuntu, то введем команду, которую использовали на семинарах: **sudo** **apt-get install** **ssh**

[**SSH**](https://ru.wikipedia.org/wiki/SSH)**туннель** — это туннель, создаваемый посредством SSH соединения и используемый для шифрования туннелированных данных. Используется для того, чтобы обезопасить передачу данных в интернете. Особенность состоит в том, что незашифрованный трафик какого-либо протокола шифруется на одном конце SSH соединения и расшифровывается на другом.

**Local TCP forwarding**

Начнем с простого — local TCP forwarding:

После успешного подключения к SSH-серверу на «host2», на «host1» SSH-клиент начинает слушать порт 9999. При подключении к порту 9999 на «host1», SSH-сервер на «host2» устанавливает соединение с localhost (коим и является для себя самого «host2») на порт 5432 и передает по этому соединению данные, принятые ssh-клиентом на «host1» на порт 9999.

**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*

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

Теперь нужно инициировать ssh-соединение со стороны «host2» вводом приведенной выше команды, а «host1» должен иметь белый IP-адрес и открытый порт SSH.

**37.** **Контейнеризация программных приложений. Основные понятия, использование.**

**Контейнеризация** (*виртуализация на уровне операционной системы*, *контейнерная виртуализация*, *зонная виртуализация*) — метод [виртуализации](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), при котором [ядро операционной системы](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) поддерживает несколько изолированных экземпляров [пространства пользователя](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) вместо одного. Эти экземпляры (обычно называемые *контейнерами* или *зонами*) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы. Для систем на базе [Unix](https://ru.wikipedia.org/wiki/Unix" \o "Unix) эта технология похожа на улучшенную реализацию механизма [chroot](https://ru.wikipedia.org/wiki/Chroot" \o "Chroot). Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

Программные контейнеры выступают в качестве стандартных модулей для развертывания программного обеспечения, которые могут содержать различный код и зависимости. Контейнеризация программного обеспечения позволяет разработчикам и ИТ-специалистам развертывать его в разных средах без каких-либо изменений или с минимальными изменениями.

Контейнеры также изолируют приложения друг от друга в общей операционной системе. Контейнерные приложения выполняются на основе узла контейнеров, который в свою очередь работает в операционной системе (Linux или Windows). Поэтому контейнеры требуют гораздо меньше ресурсов, чем образы виртуальных машин.

Еще одним преимуществом контейнеризации является масштабируемость. Вы можете быстро осуществлять горизонтальное масштабирование, создавая контейнеры для краткосрочных задач. С точки зрения приложения, создание экземпляра образа (контейнера) аналогично созданию экземпляра процесса, например для службы или веб-приложения. Но для обеспечения надежности при запуске нескольких экземпляров одного образа на нескольких серверах обычно желательно, чтобы контейнеры (экземпляры образа) выполнялись на разных серверах или виртуальных машинах в разных доменах сбоя.

Иными словами, контейнеры предоставляют такие преимущества, как изоляция, переносимость, гибкость, масштабируемость и контроль, на протяжении всего жизненного цикла приложения. Самым важным преимуществом является изоляция среды разработки от рабочей среды.