Сервисы на linux

В этом разделе мы будем рассматривать различные сервисы которые можно настроить на Linux.

- Установка и настройка apache2 на ubuntu
- Как использовать SSH-ключи для авторизации
- Руководство по установке и настройке SSH на сервере с Ubuntu
- Что такое DNS-сервер
- Как установить Nextcloud на Ubuntu 22.04
- Как настроить балансировку нагрузки с помощью Nginx.
- Виртуальная память, swap
- Установка и настройка Samba на Ubuntu и Debian
- Установка Mattermost на Ubuntu
- Как использовать Nessus для сканирования уязвимостей в Ubuntu 22.04
- Установка и настройка ArchLinux руками (без скрипта archinstall)

Установка и настройка apache2 на ubuntu

Под веб-серверами понимают как физические машины, так и специальное программное обеспечение. С точки зрения софта, веб-сервер — это программа, которая реализует логику сервера в клиент-серверной архитектуре: принимает HTTP-запросы от клиентов и возвращает соответствующие ответы.

На рынке представлено большое количество различных веб-серверов, которые предлагают своим пользователям дополнительный функционал. В зависимости от своих потребностей пользователь может выбрать наиболее подходящее его задачам решение.

Самые распространенные веб-серверы в 2023 году — это Nginx и Apache. В этом материале мы расскажем, как установить и настроить Apache на операционной системе Ubuntu 22.04.

Что такое Apache

Apache HTTP Server или просто Apache — это бесплатный кроссплатформенный веб-сервер с открытым исходным кодом. Он был разработан в 1995 году группой разработчиков для устранения недостатков популярного на тот момент веб-сервера NCSA HTTPd.

NCSA HTTPd был одним из первых веб-серверов. Его разработали в 1993 году в NCSA, университет Иллинойса. Он распространялся бесплатно и позволял пользователям размещать в интернете первые страницы, но NCSA HTTPd имел ограниченные возможности по сравнению с современными веб-серверами и ряд других недостатков, которые по итогу вылились в появление Apache.

Через год после выхода Apache получил популярность среди хостинг-компаний и разработчиков благодаря новой функциональности и кроссплатформенности. В 2005 году порядка 70% всех серверов в сети работало на Apache. Сейчас эта цифра держится в районе 20%, а основной конкурент Apache — это Nginx.

Apache состоит из двух основных компонентов: ядра и модулей. Ядро выполняет базовые функции веб-сервера: обрабатывает конфигурационные файлы, выполняет действия, связанные с HTTP, и загружает дополнительные модули. Модули позволяют расширить базовый функционал ядра: поддержка новых языков программирования, авторизация пользователей, повышение уровня безопасности. Над ядром работает исключительно

команда Apache.

В целом, к плюсам Apache можно отнести:

- Бесплатное ПО;
- Кастомизация: веб-сервер на Apache можно легко подстроить под конкретные цели и задачи благодаря большому количеству дополнительных модулей и открытому коду.
- Большое комьюнити;
- Кроссплатформенность;
- Хороший уровень производительности и безопасности.

А к минусам:

- Требовательность к ресурсам, особенно при обслуживании большого количества одновременных запросов;
- Ограниченная многопоточность: Apache использует технологию мультипроцессинга, помещая каждое соединение в отдельный поток. Количество таких потоков ограниченно, что негативно сказывается при большом количестве запросов;
- Сложность настройки из-за большого количества настроек.

Установка Apache

Установка Apache выполняется в несколько шагов:

Шаг 1. Обновите индексы пакетов apt

Перед установкой любого программного обеспечения на Ubuntu в первую очередь необходимо обновить индексы пакетов. Благодаря этому в репозитории будут находиться последние пакеты, доступные для установки.

Для обновления индекс пакетов APT до последней версии выполните следующую команду:

```
sudo apt update
```

Шаг 2. Установка веб-сервера Apache

Установка веб-сервера Apache на Ubuntu — это простой процесс, включающий в себя выполнение одной команды и перезагрузку системы. Чтобы установить Apache, выполните следующую команду:

```
sudo apt install apache2
```

После чего перезагрузите систему.

Шаг 3. Запуск и автозагрузка Apache

Чтобы запустить службу Apache, выполните эту команду:

```
sudo systemctl start apache2
```

Эту команду придется выполнять каждый раз при запуске сервера. Чтобы избежать этого, добавим Apache в автозагрузку:

```
sudo systemctl enable apache2
```

Шаг 4. Проверка установки сервера Apache

Проверим статус службы Apache, чтобы убедиться в успешности установки:

```
service apache2 status
```


Настройка брандмауэра

Теперь, когда вы установили Apache на Ubuntu, необходимо разрешить внешние подключения через брандмауэр UFW.

UFW (Uncomplicated Firewall) — это интерфейс командной строки для iptables, который представляет собой фаервол для ОС Linux. Он предназначен для упрощения управления правилами фаервола, делая его более доступным для новичков. UFW позволяет легко настраивать правила фаервола, такие как открытие или закрытие портов, блокировка или разрешение доступа к сети и т.д.

UFW является простым и эффективным способом обеспечения безопасности вашего сервера или компьютера. Он может использоваться как для домашних пользователей, так и для предприятий.

Вы можете пропустить этот раздел, если на вашем сервере не запущен UFW или не установлен брандмауэр. Но это не совсем безопасно, поэтому для обеспечения безопасности вашего устройства мы рекомендуем вам использовать брандмауэр.

При включенном брандмауэре вы можете обнаружить, что подключение к серверу Apache с удаленного устройства невозможно. Это связано с тем, что порты, которые использует

Apache, по умолчанию закрыты. Речь идет о 80 порту (для HTTP) и 443 порту (для HTTPS). Откроем подключение к обоим портам

Давайте начнем с открытия 80 порта. Через этот порт будут проходить подключения к Apache через HTTP.

Даже если вы планируете работать только с HTTPS-соединениями, не лишним будет открыть подключения на 80 порту, чтобы вы могли перенаправлять их на HTTPS.

В первую очередь проверим, включен ли файрвол UFW:

```
sudo ufw status
```

Мы должны увидеть статус `Active`. Если это не так, запустите сервис ufw следующей командой:

```
sudo ufw enable
```

Чтобы разрешить доступ к порту 80 с помощью UFW, мы можем использовать следующую команду в терминале:

```
sudo ufw allow 80
```

Также если вы планируете использовать HTTPS с сервером Apache на Ubuntu, вам также необходимо открыть порт 443.

Порт 443 — это порт, через который HTTPS работает по умолчанию. Поэтому если вы посетили сайт, использующий протокол "https://", ваш веб-браузер будет использовать именно этот порт.

Вы можете разрешить этот порт с помощью этой команды:

```
sudo ufw allow 443
```

Посещение вашего веб-сайта

Теперь, когда вы установили веб-сервер Apache на Ubuntu и открыли подключения в брандмауэре, давайте попробуем зайти на него.

Посетив страницу сервера, мы увидим только страницу по умолчанию. Это хороший показатель того, что все работает правильно.

Если планируется подключения с удаленного устройства, то в первую очередь необходимо узнать IP-адрес сервера Apache. Есть несколько способов узнать это.

Самый простой способ — использовать команду `hostname` с опцией `-I`. Команда в качестве результата выведет список IP-адресов, назначенных вашему устройству.

```
hostname -I
```

Например, нашему тестовому серверу присвоен только локальный IP-адрес, который мы видим ниже.

```
192.168.0.215
```

По этому адресу нужно перейти в любом браузере. Если вы получаете доступ непосредственно с вашего сервера Ubuntu, вы можете использовать `127.0.0.1` или `localhost` вместо этого.

После перехода по этому адресу вы должны увидеть страницу, аналогичную той, что показана на скриншоте ниже.

cffe6c21-820c-4d2b-91fc-578b2958b09d?width=728&height=400

Это говорит о том, что вы успешно запустили Apache на Ubuntu.

Заключение

В рамках этого материала мы рассмотрели установку Apache на Ubuntu 22.04, настройку брандмауэра и запуск сервера. При разработке сайта или веб-приложение эти шаги будут первыми на пути к готовому продукту.

Как использовать SSH-ключи для авторизации

Огромное количество облачных приложений построены на популярном протоколе SSH — он широко используется для управления сетевой инфраструктурой, передачи файлов и выполнению удаленных команд.

Название расшифровывается как Secure Socket Shell или *Безопасная оболочка сокета*. Это означает, что протокол предоставляет оболочку (интерфейс командной строки) вокруг соединения между несколькими удаленными хостами, которая является безопасной (зашифрованной и аутентифицированной).

SSH-соединение доступно во всех популярных операционных системах: Linux, Ubuntu, Windows, Debian и так далее. Благодаря закрытому и открытому ключу протокол устанавливает зашифрованный канал связи внутри незащищенной сети.

Ключи — основа SSH

Начнем с того, что протокол работает по клиент-серверной модели. Это значит, что на стороне пользователя есть SSH клиент (терминал в Linux или оконное приложение в Windows), а на стороне сервера активен так называемый *демон* (*daemon*), который принимает подключения от клиентов.

На практике общение по каналу SSH представляет собой управление терминалом удаленного сервера. Иными словами, после успешного подключения все, что вводится в консоль на локальном компьютере, тут же выполняется на удаленном сервере.

В наборе SSH есть два базовых ключа, необходимые для шифровки и дешифровки информации:

- открытый
- закрытый

Ключи связаны между собой некой математической информацией. Открытый ключ является публичным (можно передавать кому угодно), расположен на сервере и нужен для шифровки данных. Закрытый ключ непубличный (держится в тайне), размещен на клиенте и

предназначен, соответственно, для дешифровки данных.

Разумеется, ключи генерируются не вручную, а с помощью специальных программ— *keygen*-ов. Такие утилиты формируют новые ключи, задействуя основные алгоритмы шифрования, применяемые в технологии SSH.

Подробнее о работе SSH

Обмен публичными ключами

SSH основывается на так называемом симметричном шифровании. Два хоста, желающих вести защищенный обмен сообщениями друг с другом, формируют уникальный сеансовый ключ, криптографически основанный на публичных и непубличных данных каждого из хостов.

Например, хост *A* создает открытый и закрытый ключи. Открытый ключ отправляется хосту *B*. Хост *B* проделывает то же самое — свой открытый ключ он посылает хосту *A*.

Благодаря алгоритму Диффи-Хеллмана можно создать ключ из закрытого ключа хоста *A* и публичного ключа хоста *B*. Полученный новый ключ будет идентичен другому ключу, который генерируется на основе публичного ключа хоста *A* и закрытого ключа хоста *B*.

Таким образом оба хоста независимо друг от друга получают симметричный (одинаковый) ключ шифрования, после чего начинается защищенный обмен сообщениями. Отсюда и само название — *симметричное шифрование*.

Верификация сообщений

Для верификации сообщений хосты используют хеш-функцию, которая выдает строку фиксированного размера на основе следующих данных:

- Симметричного ключа.
- Номера пакета.
- Текста зашифрованного сообщения.

Результат хеширования этих данных называется HMAC — *код аутентификации сообщения на основе хэша*. Клиент создает свой HMAC и отправляет его серверу. Сервер получает HMAC клиента, создает свой HMAC (на основе имеющихся у него данных) и сравнивает их. Если они совпадают, значит верификация прошла успешно — никто никого не обманул.

Аутентификация хостов

Защищенное соединение между хостами — лишь полдела. Необходимо также выполнить аутентификацию пользователя, который подключается к удаленному хосту — вполне возможно, у него нет прав доступа для выполнения команд.

Вариантов много.

Например, пользователь может отправить на сервер некий пароль в зашифрованном виде. Если он верный, сервер будет «откликаться» на команды клиента.

Другой способ — сертификат. Пользователь первично отправляет серверу пароль и открытую часть сертификата, после чего взаимодействие между хостами функционирует без регулярного ввода паролей.

Алгоритмы шифрования

Ключевой элемент стойкости SSH — расшифровка симметричного ключа возможна только за счет закрытого ключа, а не открытого. Даже несмотря на то, что симметричный ключ состоит из обоих.

Для получения такого свойства необходим соответствующий алгоритм.

Существует три класса таких алгоритмов: RSA, DSA и алгоритмы на основе эллиптических кривых. У каждого есть свои особенности.

- RSA. Алгоритм, основанный в 1978 году на так называемой *целочисленной факторизации*. Учитывая, что разложение на множители больших полупростых чисел само по себе довольно сложно, существует зависимость между размером выбранных множителей и временем, необходимым для вычисления решения. Длина ключа варьируется от 1024 до 16384 бит.
- DSA. Алгоритм основан на дискретном логарифмировании и модульном возведении в степень. DSA похож на RSA, однако несколько иначе математически связывает открытый и закрытый ключи между собой. Длина ключа не превышает 1024 бит.
- ECDSA и EdDSA. Эти алгоритмы используют эллиптические кривые (в отличие от DSA, который основан на модульном возведении в степень), предполагая, что не существует эффективного решения дискретной логарифмической задачи. Ключи этих алгоритмов короче, но при этом уровень безопасности не хуже.

Так или иначе, подробное объяснение математических основ работы алгоритмов SSH останется за пределами этой статьи — такая информация в практике прикладной разработки используется довольно редко.

Генерация ключей

В каждой операционной системе есть свои утилиты, позволяющие оперативно сгенерировать SSH-ключ.

Например, в UNIX-подобных системах для получения пары ключей есть соответствующая команда — ssh-keygen:

```
ssh-keygen -t rsa
```

Обратите внимание, что тип (тот самый алгоритм шифрования) указывается после специального флага `-t`. Другие типы называются так: `dsa`, `ecdsa`, `ed25519`.

Аналогично можно конкретизировать длину ключа с помощью флага `-b`. Однако, с этим стоит быть осторожным — от длины ключа зависит степень безопасности будущего соединения.

```
ssh-keygen -b 2048 -t rsa
```

После ввода команды терминал выдаст предложение ввести полный путь и названия файлов, в которых сохранятся генерируемые ключи. Однако, вы можете принять указанный путь по умолчанию, нажав *Enter* — тогда будут установлены стандартные названия `id_rsa` (секретный) и `id_rsa.pub` (открытый).

Иными словами, открытый ключ окажется в файле `название.pub`, а закрытый в файле `название` без расширения.

Далее команда предложит ввести парольную фразу. И хотя она не является обязательной (она не имеет никакого отношения непосредственно к SSH-протоколу), рекомендуется ее указать для защиты от несанкционированного использования ключа внутри системы Linux неким сторонним пользователем. Во втором случае потребуется постоянно вводить пароль при установлении соединения.

Кстати, пароль можно в любой момент изменить:

```
ssh-keygen -p
```

Или можно даже указать все предварительные параметры целиком одной командой:

```
ssh-keygen -p password_old -N password_new -f path_to_files
```

Что касается Windows, есть два пути.

В первом используется аналогичная команда `ssh-keygen` из OpenSSH клиента, повторяющая ту же самую последовательность действий из Linux.

Во втором используется оконное приложение PuTTY, где с помощью нажатия одной кнопки получается открытый и закрытый ключ.

Установка клиентской и серверной части

Для SSH-соединения на платформе Linux (как на клиенте, так и на сервере) есть соответствующий инструмент под названием OpenSSH. И хотя, как правило, он уже установлен в операционной системе, в некоторых случаях его требуется установить самостоятельно (например, в Ubuntu).

Как правило, для установки SSH используется следующая универсальная команда с последующим введение пароля суперпользователя:

```
sudo apt-get install ssh
```

Тем не менее во многих операционных системах SSH иногда поделен на отдельные компоненты как для клиента, так и для сервера.

Для клиента

Чтобы узнать, есть ли клиент SSH на вашей локальной машине, просто введите `ssh` в консоль:

```
ssh
```

Если система поддерживает SSH, то терминал выведен описание этой команды. Если этого не произошло, тогда выполните ручную установку:

```
sudo apt-get install openssh-client
```

При установке также потребуется ввести пароль суперпользователя. После чего SSH-подключение станет доступно.

Для сервера

Аналогично клиенту, на сервере необходима серверная часть программного инструментария OpenSSH.

Чтобы проверить, доступен ли сервер SSH на используемом вами удаленном хосте, можно попытаться *локально* подключиться через `ssh`:

```
ssh localhost
```

Если SSH-демон действительно включен, то выйдет сообщение об успешном подключении. Если нет — демон нужно установить:

```
sudo apt-get install openssh-server
```

Как и на клиенте, терминал потребует ввести пароль суперпользователя. После установки можно снова проверить активность SSH. Кстати, помимо подключения к `localhost` есть и другой способ проверки:

```
sudo service ssh status
```

Кстати, после подключения можно произвольно менять настройки SSH. Для этого потребуется отредактировать конфигурационный файл `./ssh/sshd_config`.

Например, можно поменять стандартный порт на свой. Только не забудьте, что для применения новой конфигурации службу SSH нужно вручную перезапускать:

```
sudo service ssh restart
```

Копирование SSH-ключа на сервер

Специальная команда копирования

После создания публичного ключа его можно использовать в качестве авторизованного ключа на сервере. Это необходимо, чтобы быстро устанавливать соединение с сервером без регулярного ввода пароля.

Наиболее распространенный способ — использовать специализированную команду `ssh-copy-id`:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub name@server_address
```

В данном случае предполагается, что при генерации ключа использовались пути и имена по умолчанию. Если нет — имя `~/.ssh/id_rsa.pub` нужно просто заменить на свое.

Вместо `client` пишется имя пользователя, которое используется на удаленном сервере, а вместо `server_address` — адрес хоста. При этом команду можно сделать несколько короче, если пользовательские имена на клиенте и сервере совпадают:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub server_address
```

Если вы предварительно устанавливали секретный пароль на SSH-ключ, то в консоли выведется сообщение, предлагающее его ввести. Если перед этим вы не устанавливали пароль, то сразу выполнится копирование.

Иногда реализация удаленного сервера предполагает SSH-подключение по нестандартному (стандартный — 22) порту. В таком случае его потребуется явно указать с помощью дополнительного флага `-p`:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub -p 8129 name@server_address
```

Полуручное копирования

Есть операционные системы, в которых по тем или иным причинам команда `ssh-copy-id` может не поддерживаться вообще, хотя подключение по SSH к серверу возможно. В таком случае операция копирования разбивается на несколько связанных между собой команд.

```
ssh name@server_address 'mkdir -m 700 ~/.ssh; echo ' $(< ~/.ssh/id_rsa.pub) ' >> ~/.ssh/authoriz
```

Эта последовательность команд создаст специальную папку протокола на сервере (в случае, если отсутствует) с соответствующими разрешениями (700) на запись и чтение. Далее создастся файл `authorized_keys` — в нем хранятся публичные ключи всех разрешенных пользователей. Поэтому именно в него будет записана информация из локального файла (со стандартным названием) с открытым ключом клиента. Если `authorized_keys` уже есть на удаленном сервере, то он будет просто дополнен. По итогу эта команда дополнительно устанавливает соответствующее разрешение на файл `authorized_keys` внутри системы удаленного хоста.

В дальнейшем соединение с сервером происходит с помощью той же команды, но уже с проверкой пользователя по открытому ключу, который был добавлен в `authorized_keys`.

```
ssh name@server_address
```

Ручное копирование

Некоторые хост-платформы предлагают управление сервером через другие каналы связи. Например, через пользовательский веб-интерфейс в личном кабинете. В таком случае, в них должен присутствовать функционал для добавления публичного ключа на сервер. Иногда в веб-интерфейсе имитируется терминал серверной консоли.

Так или иначе, удаленный хост обязательно будет содержать соответствующий файл `~/.ssh/authorized_keys` — внутри него содержатся все разрешенные открытые ключи пользователей.

Логично, что именно в этот файл нужно скопировать открытый ключ клиента, размещенный в `~/.ssh/id_rsa.pub` (при стандартном названии).

В том случае, когда генерация ключей выполнялась в оконном приложении (обычно это PuTTY в Windows), открытый ключ копируется из него напрямую — то есть скопированный текст просто добавляется к тексту, который уже размещен в файле `authorized_keys`.

Подключение к серверу

Для подключения к удаленному серверу в операционной системе Linux нужно ввести в консоль:

```
ssh name@server_address
```

Либо, если наименование локального пользователя идентично наименованию удаленного:

```
ssh server_address
```

Далее система попросит вас ввести пароль. При этом ввод пароля не будет графически отображаться в терминале — его нужно просто набрать на клавиатуре и нажать *Enter*.

Как и в случае с командой `ssh-copy-id` при подключении к удаленному серверу можно явно указать порт:

```
ssh client@server_address -p 8129
```

Теперь вам доступен контроль над удаленной машиной через терминал — все вводимые команды будут выполняться на стороне сервера.

Заключение

Сегодня SSH — один из самых популярных протоколов в разработке и системном администрировании. Именно поэтому так важно иметь базовое представление о его функционировании.

Задача данной статьи — в общих чертах описать устройство SSH-соединения, коротко рассказать об алгоритмах шифрования (RSA, DSA, ECDSA и EdDSA) и продемонстрировать, как через применение публичного и секретного ключей можно устанавливать безопасное соединение с собственным сервером, обмениваясь сообщениями, которые будут оставаться недоступны третьим лицам.

Были показаны основные команды UNIX-подобных операционных систем, с помощью которых можно генерировать оба ключа и давать клиенту разрешение на SSH-доступ с помощью копирования открытого ключа на сервер с последующим подключением.

Руководство по установке и настройке SSH на сервере с Ubuntu

SSH (Secure Shell) представляет собой сетевой протокол, спроектированный для обеспечения безопасной связи между клиентом и сервером. Каждая отправляемая команда и полученная информация шифруются, что гарантирует безопасное управление удаленным хостом, передачу файлов и решение других задач.

Для успешной настройки SSH соединения вам следует выполнить следующие шаги:

1. Установить компоненты SSH-сервера на серверной машине. Для этого используйте пакет openssh-server, который обеспечивает функциональность сервера SSH.
2. Убедитесь, что на клиентской машине присутствует клиентский компонент SSH. В большинстве дистрибутивов Linux и BSD он уже предустановлен, однако для операционной системы Windows может потребоваться установка дополнительных утилит. Один из наиболее распространенных вариантов для Windows – это PuTTY, который предоставляет возможность установки SSH-соединения с удаленным хостом.

Активация SSH

Изначально, для безопасности, удаленный доступ к серверу через защищенный сетевой протокол отключен по умолчанию. Тем не менее, в Ubuntu настройка SSH выполняется легко и просто. Для начала, откройте консоль на сервере, на котором требуется активировать SSH в Ubuntu.

Далее, выполните обновление пакетного менеджера для обеспечения актуальности системы.

```
apt update
```

Для установки необходимого программного обеспечения, выполните следующее:

```
sudo apt install openssh-server
```

Обратите внимание: для выполнения обеих операций необходимы права суперпользователя, которые можно получить, используя sudo. По умолчанию, в Ubuntu, OpenSSH запускается автоматически после установки. Однако, если вам нужно проверить его текущий статус, вы можете воспользоваться командой:

```
sudo systemctl status ssh
```

При выполнении проверки статуса службы OpenSSH в выводе должно отображаться, что служба запущена и настроена на автозапуск при загрузке системы:

```
ssh.service - OpenBSD Secure Shell server
    Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
    Active: active (running) since Mon 2022-03-21 12:34:00 CEST; 1m ago
...
```

Это означает, что команда “Install SSH Ubuntu” выполнилась успешно. Чтобы вернуться в командную строку, нажмите клавишу “q”.

Если служба не активирована, вы можете запустить ее вручную, воспользовавшись следующей командой:

```
sudo systemctl enable --now ssh
```

В Ubuntu встроен инструмент управления брандмауэром, известный как UFW. Если брандмауэр активен в вашей системе, не забудьте разрешить доступ к порту SSH:

```
sudo ufw allow ssh
```

Теперь у вас есть возможность установить SSH-соединение с вашей системой Ubuntu с помощью любого удаленного компьютера.

Формирование пары ключей

Для усиления безопасности соединения и обеспечения удобной аутентификации, используется ключевая пара, состоящая из открытого и закрытого ключей. Открытый ключ хранится на хосте, в то время как закрытый ключ — на компьютере пользователя.

Давайте рассмотрим процесс создания ключей на разных операционных системах. Начнем с Ubuntu.

Для генерации новой пары ключей RSA с длиной 2048 бит вам потребуется открыть терминал и выполнить следующую команду:

```
ssh-keygen -t rsa
```

При создании пары ключей возникнет вопрос о месте их сохранения. Если вы просто нажмете Enter, ключи будут автоматически сохранены в подкаталоге .ssh в вашей домашней папке. Вы также можете выбрать другой путь для сохранения, но наилучшей практикой является использование каталога по умолчанию, так как это упрощает управление ключами в будущем.

Если на клиентском компьютере уже существует пара ключей, вам будет предложено перезаписать их. Решение о перезаписи остается на ваше усмотрение, но будьте осторожны. Если вы решите перезаписать ключи, то прежняя пара ключей будет удалена и больше не сможет быть использована для доступа к серверу. Чтобы избежать конфликта, лучше всего присваивать каждой новой паре уникальное имя, хотя каталог для хранения можно оставить неизменным.

Также вам будет предложено задать парольную фразу, что добавит дополнительный уровень безопасности, предотвращая доступ неавторизованных пользователей к вашему хосту. Если вы не желаете устанавливать парольную фразу, просто нажмите Enter.

Для убедительности в создании ключей можно выполнить следующую команду:

```
ls -l ~/.ssh/id_*.pub
```

В терминале будет отображен список сгенерированных ключей. Аналогично можно создать ключевую пару на macOS.

Если вы используете Windows, более простым вариантом будет загрузить PuTTY, который включает в себя утилиту PuTTYgen. С помощью этой утилиты можно создать пару ключей простым перемещением мыши. В PuTTYgen также можно указать папку для сохранения ключей и добавить парольную фразу для усиления безопасности.

Внесение ключа на сервер

Закрытый ключ остается на компьютере и никому не передается. В то время как открытую часть ключа необходимо перенести на сервер. Если у вас есть доступ к хосту с использованием пароля, то можно передать открытый ключ с помощью команды ssh-copy-id. Пример такой команды:

```
ssh-copy-id ubuntu@192.168.1.10
```

Вместо “ubuntu” введите ваше имя пользователя, а вместо “192.168.1.10” укажите IP-адрес сервера. После этого вам будет предложено ввести пароль, и после успешной аутентификации открытая часть ключа будет передана на хост.

Для установления подключения с использованием ключей выполните следующую команду:

```
ssh ubuntu@192.168.18.76
```

Замените “ubuntu” на ваше имя пользователя и ‘192.168.1.10’ на IP-адрес вашего сервера. Если вы не настроили парольную фразу, вы сможете войти в систему без дополнительной аутентификации. Система безопасности проведет проверку открытой и закрытой части ключа, и при совпадении установит соединение.

Настройка конфигурации

Для настройки Ubuntu Server используется файл /etc/ssh/sshd_config. Однако, перед внесением в него изменений, рекомендуется создать резервную копию. Это предотвратит потерю данных и сэкономит время, если случится ошибка.

Для создания резервной копии выполните следующую команду:

```
sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.factory-defaults
```

В файле /etc/ssh/sshd_config.factory-defaults будут сохранены стандартные параметры. Вы будете вносить изменения и редактировать файл /etc/ssh/sshd_config.

Выключение возможности аутентификации с использованием пароля

Использование аутентификации по паролю на SSH Server Ubuntu может быть удобным, однако создание длинных и сложных паролей может привести к их небезопасному хранению. Для повышения уровня безопасности аутентификации рекомендуется использовать ключи шифрования вместо паролей. В таком случае, пароль может стать излишним. В целях увеличения безопасности, рекомендуется его отключить.

При этом, имейте в виду следующее:

1. Отключение аутентификации по паролю может повлечь за собой риск блокировки вашего доступа к серверу.
2. Существует риск блокировки доступа, если вы потеряете закрытый ключ или испортите файл ~/.authorized_keys.
3. При блокировке доступа, вы можете потерять возможность доступа к данным и приложениям на сервере.
4. Рекомендуется отключать аутентификацию по паролю только если вы хорошо знакомы с аутентификацией с использованием ключей и полностью осознаете потенциальные последствия блокировки доступа.

Для отключения аутентификации по паролю выполните следующие шаги:

1. Подключитесь к серверу с правами root.
2. Отредактируйте файл sshd_config, изменяя параметр PasswordAuthentication с ‘Yes’ на ‘No’.

3. После внесения изменений, перезапустите службу SSH с помощью следующей команды, выполненной от имени пользователя root:

```
sudo systemctl restart sshd
```

После выполнения этого действия, вы больше не сможете использовать пароли для аутентификации. Для подключения останется только возможность использовать SSH-ключи в Linux.

Отмена привилегий root

Для повышения безопасности вашей удаленной системы Ubuntu, рассмотрите вариант отключения доступа пользователя root через SSH. В Ubuntu, установленной на удаленном хосте, отредактируйте файл конфигурации и настройте соответствующий параметр, чтобы исключить доступ root-пользователя через SSH.

Откройте файл конфигурации SSH в Ubuntu для внесения изменений.

```
sudo vi /etc/ssh/sshd_config
```

Измените параметр PermitRootLogin на ‘No’.

Также имеется возможность разрешить аутентификацию пользователя root при использовании любого другого разрешенного механизма, за исключением пароля. Для этого установите параметру PermitRootLogin значение ‘prohibit-password’.

С такой настройкой вы сможете войти в систему как пользователь root с использованием закрытого ключа. Главное условие – перед перезапуском службы SSH-сервера необходимо скопировать открытый ключ в систему.

Для применения обновленной конфигурации permitrootlogin в SSHD перезапустите службу:

```
sudo systemctl restart sshd
```

Модификация порта по умолчанию

Исходно Open Server Ubuntu использует порт 22 по умолчанию. Для усиления безопасности, рекомендуется установить альтернативный порт. Часто рекомендуется выбирать значения из верхнего диапазона, например, от 50000 до 65000, и лучше всего использовать порты, в которых все цифры различны, например, 56789.

Необходимо открыть файл конфигурации:

```
sudo vi /etc/ssh/sshd_config
```

Раскомментируйте строку “Port 22”, заменив 22 на другой номер, например 56789. Сохраните внесенные изменения и закройте файл.

Чтобы применить эту конфигурацию, выполните перезапуск службы:

```
sudo systemctl restart sshd
```

После успешного перезапуска, проверьте, что теперь соединение осуществляется через новый порт:

```
ssh -p 56789 user@ip_server
```

Не забывайте, что после каждой настройки требуется выполнить перезапуск службы. В противном случае, SSH-подключение в Linux будет работать согласно предыдущей конфигурации.

Настройка создания туннелей

Туннелирование представляет собой метод передачи нешифрованного трафика или информации по зашифрованному каналу. Этот процесс может быть применен не только для передачи файлов, но также для обеспечения доступа к службам внутренней сети через брандмауэры и для создания виртуальных частных сетей (VPN).

Существуют три разновидности туннелирования:

- Локальное
- Удаленное
- Динамическое

Для настройки определенных видов туннелирования потребуется внести изменения в конфигурационный файл SSH.

Локальная переадресация

Этот процесс представляет собой перенаправление порта с клиентской машины на удаленную машину, а затем соединение переадресуется на другой порт целевой машины.

SSH-клиент проверяет наличие соединения на указанном порту. Когда он получает запрос на установление соединения, он пересылает его на указанный порт на удаленном хосте. После этого хост устанавливает соединение с другой целевой машиной через настроенный порт.

Обычно локальное перенаправление используется для удаленного доступа к службам из внутренней сети. Например, это может использоваться для доступа к базе данных или удаленного обмена файлами.

Для настройки локального перенаправления используется аргумент -L. Пример команды:

```
ssh networkadmin@server.example -L 8080:server1.example:3000
```

Теперь запустите веб-браузер на вашем локальном компьютере. Вместо того, чтобы обращаться к удаленному приложению по адресу server.example:3000, вы можете воспользоваться localhost:8080 для доступа к нему.

Удаленная переадресация

Возможность удаленного перенаправления позволяет устанавливать соединение с локальным компьютером из удаленного и работать с его ресурсами. По умолчанию SSH не включает функцию удаленного перенаправления портов, поэтому для использования этой функции вам необходимо внести дополнительные настройки в файл конфигурации SSH и выполнить дополнительную настройку сервера Ubuntu.

Для начала, откройте файл конфигурации:

```
sudo vi /etc/ssh/sshd_config
```

Установите значение ‘Yes’ для параметра GatewayPorts. Сохраните внесенные изменения и выполните перезапуск службы:

```
sudo systemctl restart sshd
```

Для настройки перенаправления, воспользуйтесь опцией -R. Пример команды:

```
ssh -R 8080:127.0.0.1:3000 -N -f user@remote.host
```

После выполнения данной команды, сервер будет прослушивать порт 8080 и перенаправлять всю сетевую активность на порт 3000, который открыт на вашем локальном компьютере. Удаленное перенаправление, как правило, используется для предоставления внешнему пользователю доступа к внутренним службам.

Динамическая переадресация

Локальное и удаленное перенаправление позволяют устанавливать туннели и обмениваться данными через один порт, в то время как динамическое перенаправление позволяет вам работать с несколькими портами одновременно.

Динамическое туннелирование создает на вашем локальном компьютере сокс-сервер, действующий как сервер SOCKS. Обычно он слушает порт 1080 по умолчанию. Когда удаленный хост подключается к этому порту, трафик пересылается сначала на удаленную

машину, а затем на целевой динамический порт.

Для настройки динамического туннелирования используется параметр SSH -D. Вот пример соответствующей команды:

```
ssh -D 9090 -N -f user@remote.host
```

После настройки туннелирования вы можете сконфигурировать ваше приложение для его использования, например, настроить прокси-сервер в вашем браузере. Важно помнить, что перенаправление трафика должно быть настроено отдельно для каждого приложения, которое вы хотите использовать через туннель.

Отключение SSH

Для деактивации сервера OpenSSH, приостановите работу службы SSH, выполнив следующую команду:

```
sudo systemctl disable --now ssh
```

Для повторного запуска службы, выполните следующую команду:

```
sudo systemctl enable --now ssh
```

Команда “Enable SSH” в Ubuntu не переустанавливает программное обеспечение, поэтому вам не придется выполнять повторную настройку. Она просто активирует ранее установленную и настроенную службу.

Заключение

В данной статье мы освоили основы применения SSH на Ubuntu. Теперь у вас есть знания о том, как установить необходимое программное обеспечение для настройки безопасного соединения, настроить его, настроить туннелирование и даже отключить службу, если она не требуется.

Умение подключаться к Ubuntu через SSH – это весьма распространенная навык, который может оказаться полезным вам, будь то в области разработки, администрирования или для личных нужд. Например, это может пригодиться для создания безопасного соединения между разными устройствами в локальной сети.

Что такое DNS-сервер

Из этой статьи вы узнаете ряд нюансов работы глобальной сети. Осветим, что такое Domain Name System, как работает технология, какие DNS-серверы бывают и другие важные вопросы.

Что такое DNS

Прежде чем начать говорить о DNS-серверах, расскажем о самой технологии DNS (Domain Name System). DNS — это технология, которая позволяет браузеру вроде Firefox, Chrome или Edge найти запрошенный пользователем сайт по его имени.

Как работает DNS

Принцип работы DNS похож на поиск и вызов контактов из телефонной книги смартфона. Ищем имя, нажимаем «позвонить», и телефон соединяет нас с нужным абонентом. Понятно, что смартфон в ходе звонка не использует само имя человека, вызов возможен только по номеру телефона. Если вы внесете имя без номера телефона, позвонить человеку не сможете.

Так и с сайтом. Каждому имени сайта соответствует набор цифр формата XXX.XXX.XXX.XXX. Каждый октет адреса занимает ровно один байт, поэтому цифры актуальны только на отрезке от 0 до 255. Этот набор называется IP-адресом, примером реального IP-адреса является 192.168.0.154 или 203.113.89.134. Когда пользователь вводит в адресной строке браузера имя сайта, например google.com, компьютер запрашивает IP-адрес этого сайта на специальном DNS-сервере и после получения корректного ответа открывает сам сайт.

Что такое DNS-сервер

Это как раз и есть «книга контактов» интернета. DNS-сервер — это специализированный компьютер (или группа), который хранит IP-адреса сайтов. Последние, в свою очередь, привязаны к именам сайтов и обрабатывает запросы пользователя. В интернете много DNS-серверов, они есть у каждого провайдера и обслуживают их пользователей.

Зачем нужны DNS-серверы и какие они бывают

Основное предназначение DNS-серверов — хранение информации о доменах и ее предоставление по запросу пользователей, а также кэширование DNS-записей других серверов. Это как раз «книга контактов», о которой мы писали выше.

В случае кэширования все несколько сложнее. Дело в том, что отдельно взятый DNS-сервер не может хранить вообще всю информацию об адресах сайтов и связанных с ними IP-адресами. Есть исключения — корневые DNS-серверы, но о них позже. При обращении к сайту компьютера пользователя браузер первым делом проверяет локальный файл настроек DNS, файл hosts. Если там нет нужного адреса, запрос направляется дальше — на локальный DNS-сервер интернет-провайдера пользователя.

Локальный DNS-сервер в большинстве случаев взаимодействует с другими DNS-серверами из региона, в котором находится запрошенный сайт. После нескольких обращений к таким серверам локальный DNS-сервер получает искомое и отправляет эти данные в браузер — запрошенный сайт открывается. Полученные данные сохраняются на локальном сервере, что значительно ускоряет его работу. Поскольку единожды «узнав» IP-адрес сайта, запрошенного пользователем, локальный DNS сохраняет эту информацию. Процесс сохранения полученных ранее данных и называется кэшированием.

Если пользователь обратится к ранее запрошенному сайту еще раз, то сайт откроется быстрее, поскольку используется сохраненная информация. Правда, хранится кэш не вечно, время хранения зависит от настроек самого сервера.

IP-адрес сайта может измениться — например, при переезде на другой хостинг или сервер в рамках прежнего хостинга. Что происходит в этом случае? В этом случае обращения пользователей к сайту, чей IP-адрес поменялся, некоторое время обрабатываются по-старому, то есть перенаправление идет на прежний «айпишник». И лишь через

определенное время (например, сутки) кэш локальных серверов обновляется, после чего обращение к сайту идет уже по новому IP-адресу.

Где находятся главные DNS-серверы

DNS-серверы верхнего уровня, которые содержат информацию о корневой DNS-зоне, называются корневыми. Этими серверами управляют разные операторы. Изначально корневые серверы находились в Северной Америке, но затем они появились и в других странах. Основных серверов — 13. Но, чтобы повысить устойчивость интернета в случае сбоев, были созданы запасные копии, реплики корневых серверов. Так, количество корневых серверов увеличилось с 13 до 123.

В Северной Америке находятся 40 серверов (32,5%), в Европе – 35 (28,5%), еще 6 серверов располагаются в Южной Америке (4,9%) и 3 – в Африке (2,4%). Если взглянуть на карту, то DNS-серверы расположены согласно интенсивности использования интернет-инфраструктуры. Есть серверы в Австралии, Китае, Бразилии, ОАЭ и других странах, включая Исландию.

В России тоже есть несколько реплик корневых серверов DNS, среди которых:

- F.root (Москва),
- I.root (Санкт-Петербург),
- J.root (Москва, Санкт-Петербург),
- K.root (Москва, Санкт-Петербург, Новосибирск),
- L.root (Москва, Ростов-на-Дону, Екатеринбург).

Что такое DNS-зоны

В этой статье мы рассматриваем лишь вариант «один домен — один IP-адрес». На самом деле, ситуация может быть и сложнее. Так, с определенным доменным именем может быть связано несколько ресурсов — сайт и почтовый сервер. У этих ресурсов вполне могут быть разные IP-адреса, что дает возможность повысить надежность и эффективность работы сайта или почтовой системы. Есть у сайтов и поддомены, IP-адреса которых тоже могут быть разными.

Вся эта информация о связи сайта, поддоменов, почтовой системы хранится в специальном файле на DNS-сервере. Его содержимое называется DNS-зона. Файл содержит следующие типы записей:

- A — адрес веб-ресурса, который привязан к конкретному имени домена.

- MX — адрес почтового сервера.
- CNAME — чаще всего этот тип записи используется для подключения поддомена.
- NS — адрес DNS-сервера, который отвечает за содержимое других ресурсных записей.
- TXT — любая текстовая информация о доменном имени.
- SPF — данные с указанием списка серверов, которые входят в список доверенных для отправки писем от имени указанного домена.
- SOA — исходная запись зоны, в которой указаны сведения о сервере и которая содержит шаблонную информацию о доменном имени.

А что с новыми доменами?

После регистрации доменного имени нужно «рассказать» о нем DNS-серверам. Для этого нужно прописать ресурсные записи, что обычно делается в админке хостинг-провайдера или доменного провайдера. Примерно через сутки DNS-записи пропишутся в локальном сервере, также они попадут в реестры всех прочих DNS-серверов. Как только это произойдет, новый домен станет нормально открываться браузером. «DNS сайта», как иногда ошибочно называют доменное имя, активируется.

Еще немного о DNS

Инфраструктура DNS-серверов, вернее, ее основа, была заложена в начале 1980-х годов. С тех пор менялась она лишь незначительно — например, добавлялись новые доменные зоны. Так, в РФ в 2010 году появился кириллический домен .рф. До этого доменные имена могли быть лишь латинскими.

От DNS-инфраструктуры зависит нормальная работа всей глобальной сети, поэтому за работоспособностью серверов постоянно следят. В частности, предпринимаются меры по усилению безопасности системы. Кроме того, вводятся и меры на случай стихийных бедствий, проблем с электричеством и других экстренных ситуаций.

Как установить Nextcloud на Ubuntu 22.04

В инструкции мы расскажем, что такое Nextcloud, как скачать и установить его на свой сервер с дистрибутивом Ubuntu 22.04. Перед установкой подготовим все необходимые зависимости и конфигурации. А также выпустим бесплатный SSL-сертификат от сервиса Let’s Encrypt.

Что такое Nextcloud

Nextcloud — бесплатная и открытая облачная система для хранения данных и совместной работы. Это альтернатива популярным облачным сервисам вроде Dropbox или Google Drive. Но в отличие от этих сервисов, Nextcloud можно установить на свой сервер. Это означает, что вы будете полностью контролировать свои данные, не передавая их третьим лицам. Пользоваться сервисом можно как через браузер, так и установив клиентское приложение на компьютер (Windows, MacOS, Linux) или телефон (Android, iOS).

Кроме того, Nextcloud поддерживает множество плагинов (приложений) для расширения функционала. Вот несколько примеров:

1. **Calendar.** Позволяет управлять своим расписанием, поддерживает синхронизацию с другими календарями через CalDAV.
2. **Files**. Синхронизация файлов между устройствами, есть поддержка протокола WebDAV.
3. **Contacts.** Управление контактами и адресной книгой с возможностью синхронизации через CardDAV.
4. **Mail.** Интеграция электронной почты с облачным хранилищем.
5. **Notes.** Заметки и списки с возможностью синхронизации на разных устройствах.
6. **Talk.** Чат, аудио и видео звонки между пользователями Nextcloud.
7. **Tasks**. Планировщик задач, интегрированный с календарем и поддерживающий синхронизацию с другими устройствами.
8. **Document Viewer**. Онлайн-редактор документов, позволяющий создавать и редактировать текстовые файлы, таблицы и презентации.
9. **External storage support.** Интеграция и перенос данных из других облачных хранилищ, таких как Google Drive или Dropbox.

10. **Two-Factor Authentication (2FA).** Усиление безопасности аккаунта за счет подключения двухфакторной аутентификации.
11. **News**. RSS-ридер для чтения новостей и блогов.
12. **Group folders**. Создание общих папок и управление доступом к ним для разных групп пользователей.
13. **Social**. Интеграция социальных сетей и обмен подписками на федеративные платформы, такие как Mastodon.

Для начала создадим виртуальную машину, на которой будем устанавливать и настраивать Nextcloud

Подготовка к установке

Для начала создайте виртуальную машину, на которой будем устанавливать и настраивать Nextcloud. Затем, установим необходимые пакеты: веб-сервер Apache, базу данных MariaDB и различные модули для PHP:

```
sudo apt update
sudo apt install apache2 mariadb-server libapache2-mod-php php-gd
php-mysql php-curl php-mbstring php-intl php-gmp php-bcmath
php-xml php-imagick php-zip unzip
```

Сейчас мы установили пакеты для базовой инсталляции Nextcloud. Если вы будете устанавливать дополнительные приложения (аддоны), то для них могут потребоваться дополнительные пакеты. Перед установкой приложений читайте их требования и устанавливайте пакеты при необходимости.

Nextcloud хранит большинство своих настроек в базе данных. Поэтому создадим БД и пользователя, а Nextcloud после первого запуска уже сам создаст нужные ему таблицы, индексы и представления. Чтобы подключиться к нашему локальному серверу, воспользуемся простой консольной утилитой mysql.

```
sudo mysql
```

Теперь создадим пользователя и БД. Введите следующие команды, но обязательно поменяйте пароль на более сложный. Также можно указать другое имя пользователя и БД при необходимости.

```
CREATE USER 'nextcloud_user'@'localhost' IDENTIFIED BY 'password';
CREATE DATABASE IF NOT EXISTS nextcloud CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
GRANT ALL PRIVILEGES ON nextcloud.* TO 'nextcloud_user'@'localhost';
FLUSH PRIVILEGES;
```

MariaDB настроена, выходим из командной строки SQL:

```
quit;
```

Установка и настройка Nextcloud

Теперь мы готовы к установке Nextcloud. Скачиваем самую последнюю версию и распаковываем ее:

```
wget https://download.nextcloud.com/server/releases/latest.zip
unzip latest.zip
```

Затем переместим распакованное приложение директорию /var/www:

```
sudo cp -r nextcloud /var/www
```

И сменим владельца каталога, чтобы Apache мог корректно обрабатывать все вложенные файлы:

```
sudo chown -R www-data:www-data /var/www/nextcloud
```

Настройка веб-сервера Apache

Теперь необходимо настроить Apache. Для начала создадим файл **/etc/apache2/sites-available/nextcloud.conf** с настройками виртуального хоста.

В качестве доменного имени мы будем использовать **cloud.my-domain.com**, а вам необходимо указать свой домен:

```
<VirtualHost *:80>
  DocumentRoot /var/www/nextcloud/
  ServerName  cloud.my-domain.com

  <Directory /var/www/nextcloud/>
    Require all granted
    AllowOverride All
    Options FollowSymLinks MultiViews
```

```
    <IfModule mod_dav.c>
      Dav off
    </IfModule>
  </Directory>
</VirtualHost>
```

Чтобы Apache начал использовать этот файл конфигурации, выполните в shell-консоли команду:

```
a2ensite nextcloud.conf
```

Далее активируем несколько модулей Apache, которые необходимы Nextcloud для корректной работы:

```
a2enmod rewrite
a2enmod headers
a2enmod env
a2enmod dir
a2enmod mime
```

Перезапустим Apache, чтобы применить все выполненные настройки:

```
service apache2 restart
```

Получение TLS-сертификата

Nextcloud уже готов к работе, но сейчас сейчас он работает по протоколу HTTP и не шифрует трафик. Чтобы повысить защиту сервера, выпустим и установим бесплатный TLS-сертификат от сервиса Let’s Encrypt. Для начала установим утилиту certbot, которая автоматически выпускает и обновляет TLS-сертификаты.

```
sudo apt-get install -y certbot
```

Запросим сертификат с помощью команды (не забудьте в параметре указать свое доменное имя):

```
sudo certbot --nginx -d chat.my-domain.com
```

Далее Certbot попросит ввести адрес электронной почты и согласиться с условиями использования. На этот адрес будут приходить письма с информацией об окончании срока сертификата. Сертификаты Let’s Encrypt действуют 90 дней. Чтобы обновить сертификат автоматически, добавьте cron-задание или systemd-таймер, который будет запускать следующую команду:

```
sudo certbot renew --quiet
```

Теперь необходимо включить соответствующие модули в Apache, чтобы он начал использовать SSL-сертификаты:

```
a2enmod ssl
a2ensite default-ssl
```

И еще раз перезапустим Apache, чтобы он применил новые настройки:

```
service apache2 reload
```

Инициализация облака Nextcloud и начальный интерфейс

Откроем в браузере страницу https://cloud.my-domain.com и увидим окно первоначальной настройки Nextcloud. Создадим учетную запись администратора, а также укажите название БД, пользователя и пароль, которые мы заводили на этапе настройки MariaDB. При необходимости также можете поменять каталог, где будут размещаться пользовательские данные.

4.png

Далее появится окно входа, в котором нужно ввести логин и пароль только что созданного пользователя. После первоначальной настройки мы попадем в основной интерфейс Nextcloud. Тут для примера уже загружено несколько ознакомительных файлов и активированы базовые приложения, вроде синхронизации файлов, просмотра фотографий, ведение заметок и другие.

5.png

Nextcloud установлен и готов к работе. Теперь вы можете создавать пользователей, устанавливать дополнительные приложения и кастомизировать Nextcloud различными методами под свои нужны.

Заключение

Мы установили и настроили облако Nextcloud на собственном сервере. Установили все необходимые зависимости, подготовили MariaDB и Apache к установке, защитили сервер бесплатным SSL-сертификат от Let’s Encrypt для шифрования трафика.

Как настроить балансировку нагрузки с помощью Nginx.

Современные приложения могут обрабатывать множество запросов одновременно, и при этом, даже при высокой нагрузке, они должны возвращать пользователям корректную информацию. Масштабировать приложения можно разными способами:

- Вертикальное масштабирование -— просто «накинуть» оперативной памяти, мощностей процессора — арендовать или купить более мощный сервер. На ранних этапах развития приложения это просто, но у такого подхода есть недостатки — цена и ограничения современного железа.

  80d59a9d-c558-4e7b-acd6-34af128d41cb?width=1292&height=482

- Горизонтальное масштабирование — добавить больше инстансов приложения. Поднять второй сервер, развернуть на нём точно такое же приложение и каким-то образом распределять трафик между этими экземплярами приложений.

  588116fc-513c-4e07-8e74-a2ecbf72a2d9?width=1600&height=696

Горизонтальное масштабирование, с одной стороны, может быть дешевле и менее ограничивать нас в железе — можно просто добавить ещё инстансов приложения. Однако теперь нам необходимо распределять пользовательские запросы между разными экземплярами приложения.

Балансировка нагрузки (load balancing) — это способ распределения запросов к приложению (сетевого трафика) между некоторым количеством устройств.

Балансировщик нагрузки — это программа-посредник, которая располагается между пользователем и группой приложений. Общая логика следующая:

1. Пользователь заходит на сайт по определенному домену, за которым скрывается IP-адрес балансировщика нагрузки.
2. Балансировщик на основе настроек определяет, на какой из экземпляров приложения перенаправлять трафик от пользователя.
3. Пользователь получает ответ от нужного экземпляра приложения.

   ecbe7056-30e0-41d7-af5c-e1d333bf5772?width=705&height=688

Какие проблемы решает балансировка нагрузки?

- **Повышение доступности приложения:** Балансировщики нагрузки обладают функционалом для обнаружения аварийных ситуаций на серверах. В случае отказа одного из серверов балансировщик может автоматически перенаправить трафик на другой адрес, обеспечивая бесперебойную работу приложения для пользователей.
- **Масштабируемость:** Одной из основных задач балансировщика является распределение нагрузки между экземплярами приложения. Это позволяет применять горизонтальное масштабирование, добавляя новые экземпляры приложения и увеличивая общую производительность системы.
- **Улучшение безопасности:** Балансировщики нагрузки могут включать в себя функции, связанные с безопасностью. Они могут отслеживать трафик, фильтровать запросы, обеспечивать маршрутизацию через фаерволы и другие механизмы, что способствует повышению безопасности приложения.

Какие есть инструменты балансировки сетевого трафика?

Существует довольно много приложений, которые могут выступать в качестве балансировщика нагрузки, однако одним из самых популярных является Nginx. Его и рассмотрим в данном гайде.

Nginx

Nginx — это универсальный веб-сервер. Он отличается хорошей производительностью, низким потреблением ресурсов и своими широкими возможностями. Nginx можно использовать как:

- Веб-сервер
- Реверс-прокси и балансировка нагрузки
- Почтовый прокси-сервер
- И многое другое.

На сайте можно узнать подробнее про возможности Nginx. Ну а мы перейдём к практике.

Установка Nginx на Ubuntu

Nginx можно установить на все популярные дистрибутивы Linux: Ubuntu, CentOS и другие. В статье мы будем использовать Ubuntu. Чтобы установить Nginx, используем следующие команды:

```
sudo apt update
sudo apt install nginx
```

Чтобы убедиться, что всё прошло успешно, можно использовать команду:

```
systemctl status nginx
```

Файлы конфигураций для Nginx находятся в каталоге `/etc/nginx/sites-available/`. По умолчанию в этом каталоге создаётся файл `default`. В нём мы и будем писать нашу конфигурацию.

Пример настройки

начала откроем файл конфигурации по умолчанию:

```
cd /etc/nginx/sites-available/

sudo nano default
```

Поместим сюда следующую конфигурацию:

```
   upstream application{
      server 10.2.2.11; # ip-адреса серверов для распределения запросов между ними
      server 10.2.2.12;
      server 10.2.2.13;
   }

   server {
      listen 80; # по этому порту будет открываться nginx


      location / {
         # описываем, куда перенаправлять трафик от nginx
          proxy_pass http://application;
      }
   }
```

Для настройки балансировки нагрузки в Nginx в конфигурации нужно определить два блока:

- `upstream` — определяет адреса серверов, между которыми будет распределяться сетевой трафик. Тут мы указываем IP-адреса, порты и, при необходимости, методы балансировки нагрузки. Их мы обсудим далее.
- `server` — определяет способ, с помощью которого Nginx будет получать запросы. Обычно тут указывается порт, доменное имя и другие параметры.
- Путь `proxy_pass` — описывает, куда эти запросы надо перенаправлять. Это название указанного выше `upstream`.

Таким образом, Nginx используется не только как балансировщик нагрузки, но ещё и как обратный прокси (reverse proxy). Реверс-прокси — это сервер, который располагается между клиентом и экземплярами приложений. Он перенаправляет запросы от клиентов в бэкенд, и при этом может обеспечивать нас дополнительными функциями, например, такими как SSL-сертификаты, логирование и др.

Методы балансировки нагрузки

Round Robin

Существует довольно много методов балансировки. Nginx по умолчанию использует алгоритм Round Robin. Он довольно прост. Допустим, у нас есть приложения 1, 2 и 3. Балансировщик нагрузки отправит первый запрос на первое приложение:

> 6fe3b70b-dc2b-414b-ad69-ae8c6dc21640?width=444&height=304

Затем на 2:

> 8ddde93c-8f0f-49de-a01d-4f65b6e79cec?width=439&height=271

Затем на 3:

> 9534b017-c8ae-45b7-849a-7598b4ab81c8?width=418&height=276

И далее снова на 1:

> cc60e815-29e5-4c7e-a710-35907e27b610?width=438&height=293

Рассмотрим на примере. Я развернул два приложения и настроил балансировку нагрузки с помощью Nginx для них.

```
upstream application {
   server 172.25.208.1:5002#first
   server 172.25.208.1:5001; #second
}
```

Давайте посмотрим, как это работает на практике:

Первый запрос отправляет нас на первый сервер, второй запрос — на второй сервер, а затем снова на первый. Однако этот алгоритм имеет ограничение — экземпляры бэкенда будут простаивать просто потому, что ждут своей очереди.

Round Robin с указанием весов

Чтобы избежать простоя серверов, можно использовать некоторый числовой приоритет. У каждого сервера появляется свой *вес*, который определяет, как много трафика распределяется на конкретный экземпляр приложения. Таким образом мы гарантируем, что более мощные серверы получат больше трафика.

В Nginx приоритет указывается с помощью *server weights* следующим образом:

```
upstream application{
      server 10.2.2.11 weight=5;
      server 10.2.2.12 weight=3;
      server 10.2.2.13 weight=1;
   }
```

При такой настройке сервер с адресом 10.2.2.11 получит наибольшее количество трафика, так как у него указан наибольший вес.

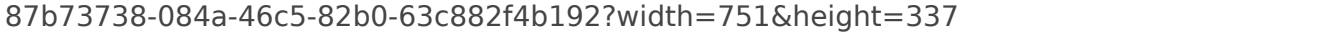
Такой подход более надёжен, чем обычный Round Robin, однако он всё ещё имеет недостаток — вручную мы можем указать вес, основываясь на мощности сервера, но при этом сами запросы также могут различаться по скорости выполнения: есть более долгие и тяжёлые, а есть быстрые и незатратные.

Рассмотрим этот метод на примере.

Настройка:

```
upstream application {
   server 172.25.208.1:5002 weight=3; #first
   server 172.25.208.1:5001 weight=1; #second
}
```

Результат:



Как мы видим, теперь каждый четвёртый запрос отправляется на второй сервер.

Least Connection

Что, если отойти от Round Robin? Распределять запросы между серверами можно не просто по порядку, а, например, основываясь на каких-либо параметрах. Отлично подойдёт количество активных соединений с сервером.

Алгоритм Least Connection обеспечивает равномерное распределение нагрузки между экземплярами приложения, как раз основываясь на количестве соединений с сервером. Чтобы его настроить, в блоке upstream надо указать `least_conn;`:

```
upstream application{
      least_conn;
      server 10.2.2.11;
      …
  }
```

Вернёмся к нашему примеру.

Чтобы проверить работу этого алгоритма, я написал скрипт, который отсылает 500 запросов параллельно и смотрит, на какое из приложений попал каждый запрос.

Вот вывод этого скрипта:

ae66ccbf-d410-42cc-8882-1194728d4e72?width=316&height=45

Кроме того, этот алгоритм может использоваться вместе с весами для адресов, по аналогии с Round Robin. В этом случае веса будут обозначать количество соединений с этим адресом по отношению к другим адресам — например, в случае с весами 1 и 5, на адрес с весом 5 попадёт в пять раз больше соединений, чем на адрес с весом 1.

Пример такой настройки:

```
upstream application{
      least_conn;
      server 10.2.2.11 weight=5;
      …
   }
```

А вот настройка для примера:

```
upstream loadbalancer {
   least_conn;
   server 172.25.208.1:5002 weight=3; #first
   server 172.25.208.1:5001 weight=1; #second
}
```

И вывод скрипта:

4a2496fb-8cc3-4ca3-b3bd-65bdb90cd7d7?width=322&height=93

Как мы видим, запросов на первый сервер ровно в три раза больше, чем на второй.

IP Hash

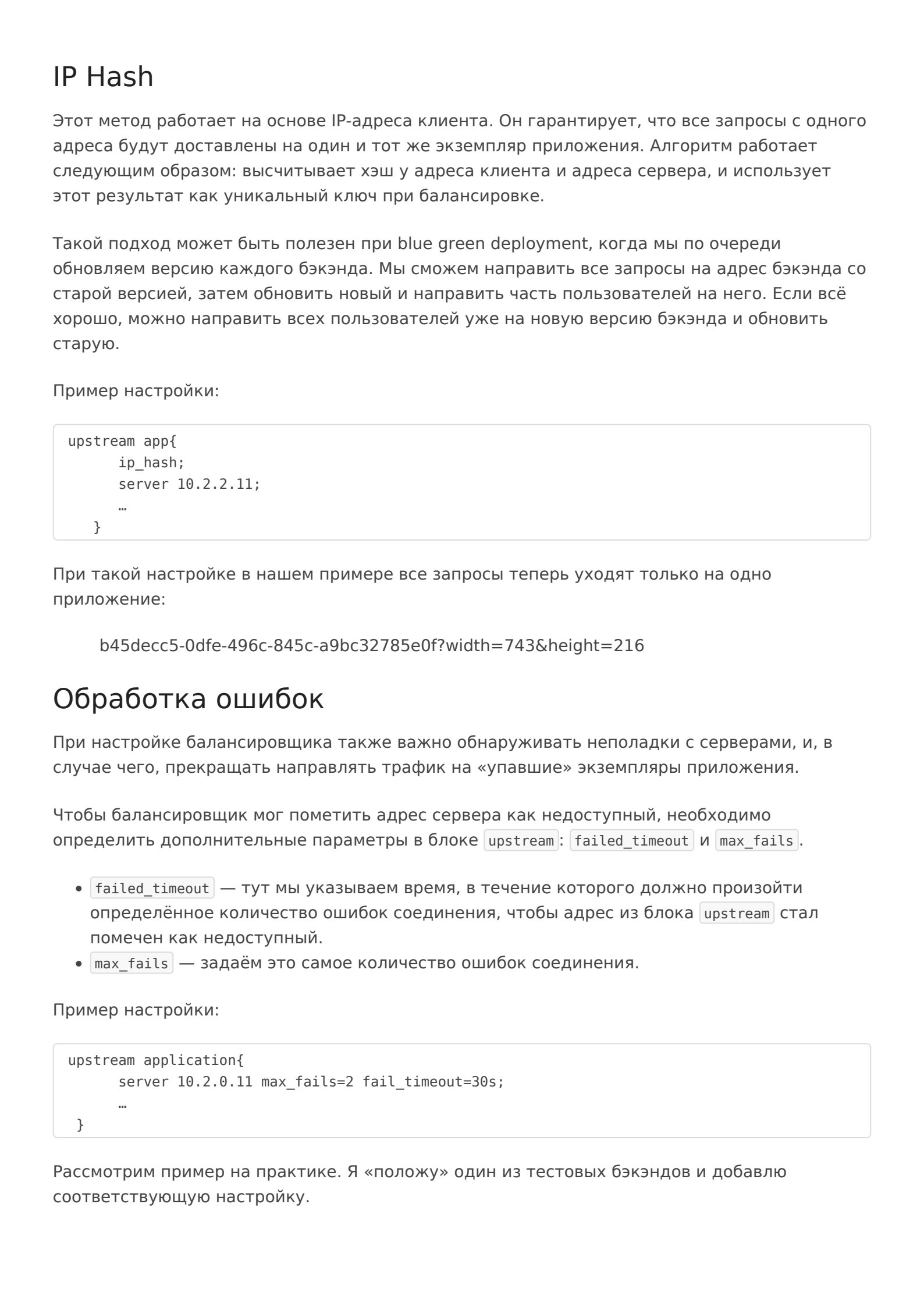
Этот метод работает на основе IP-адреса клиента. Он гарантирует, что все запросы с одного адреса будут доставлены на один и тот же экземпляр приложения. Алгоритм работает следующим образом: высчитывает хэш у адреса клиента и адреса сервера, и использует этот результат как уникальный ключ при балансировке.

Такой подход может быть полезен при blue green deployment, когда мы по очереди обновляем версию каждого бэкэнда. Мы сможем направить все запросы на адрес бэкэнда со старой версией, затем обновить новый и направить часть пользователей на него. Если всё хорошо, можно направить всех пользователей уже на новую версию бэкэнда и обновить старую.

Пример настройки:

```
upstream app{
      ip_hash;
      server 10.2.2.11;
      …
   }
```

При такой настройке в нашем примере все запросы теперь уходят только на одно приложение:



Обработка ошибок

При настройке балансировщика также важно обнаруживать неполадки с серверами, и, в случае чего, прекращать направлять трафик на «упавшие» экземпляры приложения.

Чтобы балансировщик мог пометить адрес сервера как недоступный, необходимо определить дополнительные параметры в блоке `upstream`: `failed_timeout` и `max_fails`.

- `failed_timeout` — тут мы указываем время, в течение которого должно произойти определённое количество ошибок соединения, чтобы адрес из блока `upstream` стал помечен как недоступный.
- `max_fails` — задаём это самое количество ошибок соединения.

Пример настройки:

```
upstream application{
      server 10.2.0.11 max_fails=2 fail_timeout=30s;
      …
 }
```

Рассмотрим пример на практике. Я «положу» один из тестовых бэкэндов и добавлю соответствующую настройку.

e7d009e0-7307-4891-9607-e68431e1a5ba?width=1020&height=217

Первый экземпляр бэкэнда из примера теперь отключен.

88aa0bfd-46f4-4c20-9d31-7de36925e08e?width=769&height=290

Nginx перенаправляет трафик только на второй сервер.

Сравнительная таблица алгоритмов распределения трафика

ac141a50c2fd35cba03e1.png

5f876dfe29b588865610f.png

Заключение

В этой статье мы погрузились в тему load balancing. Узнали, какие существуют методы балансировки нагрузки в Nginx и разобрали их на примере.

Виртуальная память, swap

Представьте себе, что вы находитесь в библиотеке. Полки - это ваш жёсткий диск, книги - это файлы, а вы - ядро операционной системы. Перед вами есть стол - это оперативная память. Ваша работа - читать книги. Только, как и в реальной жизни, вы не можете читать книгу целиком - вы читаете страницами. Когда вы берёте из полки книжку, вы копируете нужные страницы и кладёте их на стол.

Т.е. на столе у вас разложены страницы. И вы, как Цезарь, многозадачны, читаете множество книг одновременно. Когда стол большой, т.е. когда много оперативки - всё хорошо, вы можете разместить кучу страниц. Но большой стол стоит больших денег, поэтому обычно вы берёте средненький стол, рассчитанный под ваши задачи, ну и чтобы оставалось немного свободного места.

Есть определённые книжки, которые вам нужны не постоянно, но вы с ними работаете относительно часто. Скажем, вы читаете книжку, там много неизвестных терминов и вам периодически нужен словарь. Не то чтобы он нужен всегда на столе, но если у вас есть немного места, то почему бы временно не положить словарь на стол? Чтобы постоянно не тратить время на копирование словаря. А если понадобится место под другие книжки - ничего, уберёте словарь.

Если посмотреть вывод команды:

```
free -m
```

можно увидеть значение - столько-то места сейчас используется для буфера/кэша. Когда вы хотите прочесть файл с файловой системы, вы получаете блоки. Но в оперативке вы работаете не с блоками, а со страницами, т.е. вам нужно предварительно превратить содержимое блоков в страницы.

И вот этот переходный пункт, где вы блоки превращаете в страницы, а также записываете за какой страницей какой блок - называется буфером. Т.е. буфер содержит метаданные файловой системы и блоки, которые пишутся или считываются с диска. А в кэше хранятся уже страницы. И эти страницы нужны как для более быстрого чтения, так и для более производительной работы. Представьте, что вам нужно несколько раз прочесть одну и ту же страницу. Вместо того, чтобы каждый раз загружать её в память, вы её держите в оперативке и считываете её оттуда.

А касательно записи, представьте, что вам нужно что-то дописать в словаре. Если один раз - написали на листе, отнесли и добавили в книгу. А если вы часто что-то пишете? Вместо того, чтобы после каждой строчки относить и добавлять в книгу, вы можете подержать страницу пока на столе - а может ещё что-то понадобится дописать? Когда вы изменяете страницу в оперативке, у неё появляется метка, что она грязная - dirty page. И уже потом, скажем, раз в 5 секунд, вы относите грязные листы и добавляете в книги, т.е. сохраняете изменённые данные с оперативки на диск.

Т.е. данные не сразу пишутся на диск, а некоторое время хранятся в оперативке, в кэше. И чтобы вручную синхронизировать изменённые в оперативке данные с диском, надо выполнить команду sync. Размер грязных страниц можно посмотреть в /proc/meminfo:

```
grep Dirty /proc/meminfo
```

И вот небольшой тест с созданием файла. Сначала делаем sync, чтобы убедиться, что все данные записаны на диск. Затем с помощью dd генерируем файл размером в 100 мегабайт. Сразу после этого смотрим размер грязных страниц - 100 мб. Это означает, что файл пока что в оперативке, а не на диске. Делаем sync и снова проверяем - теперь всё записалось на диск.

Такой режим кэша, когда данные сначала сохраняются в оперативке, а потом пишутся на диск, называется writeback. Но если вдруг компьютер потеряет питание и информация не успеет синхронизироваться, грязные страницы пропадут. Соответственно, есть риск потерять данные. Т.е. такой режим очень быстрый, но не самый надёжный. Также есть режим кэша writethrough - когда данные пишутся одновременно и на диск. Т.е. условно при каждом изменении страницы вы относите страницу в книгу на полку. Так всё работает чуть

медленнее, зато при внезапном выключении вы ничего не теряете. Есть и другие режимы, но это два основных.

Память можно поделить на 4 типа по двум категориям. Она может быть личной и общей - т.е. доступна только для одного процесса или нескольких. А также может быть анонимной и основанной на файле, т.е. файловой. File-backed - это когда вы загружаете файл в оперативку - взяли книжку и положили листы на стол. А анонимная - это когда процесс просит вас выделить ему место, не привязанное к какому-либо файлу. Скажем, не книжку взять, а просто чистый А4 положить. Процесс там будет хранить какие-то данные во время работы. Таким образом получается 4 типа - anonymous private, anonymous shared, file-backed private и file-backed shared.

И теперь представим, что у вас на столе осталось мало места, а вам надо ещё добавить листов, т.е. мало оперативки осталось. Первым делом вы поищете листы, которые вам больше не нужны - т.е. освободите память, которую использовали завершённые процессы. Но этого может не хватить. Тогда в расход пойдут старые листы, которые как бы и нужны, но используются очень редко и основаны на книжках - т.е. файловые. Вы избавитесь от этих страниц, а если вдруг они потом понадобятся - то опять возьмёте их из книг, так как вы знаете, что это за файл. А вот от анонимных листов избавиться не получится. То что там написано - нет ни в одной книге. Поэтому выкидывать эти листы не вариант.

И тут вам на помощь приходят ящички стола. Туда вы можете сложить старые листы, которые сейчас на столе не нужны, а при необходимости оттуда достать. Это ваш swap - место на диске, выделенное для отгрузки редкоиспользуемых страниц с оперативки. Этот механизм, когда вы что-то кладёте в swap или достаёте оттуда называется подкачка страниц. В качестве места может быть как отдельный файл на файловой системе, так и целый раздел, в зависимости от этого место может называться файл подкачки или раздел подкачки.

Т.е. swap не выступает продолжением оперативки, он выступает временным хранилищем для редкоиспользуемых страниц. И не обязательно, чтобы он использовался только когда оперативки не хватает. Иногда полезней использовать оперативку для кэша, чем для редкоиспользуемых страниц.

Чтобы вся эта подкачка работала для программ прозрачно, чтобы разработчики сами не заботились о swap-е, ядро ОС выделяет процессам вместо реальной памяти виртуальную, которая может быть больше оперативки. Виртуальная память состоит из оперативки и swap-а. Т.е. условно, книжка может предварительно попросить места под 100 листов на столе. И пусть у вас стол будет поменьше, но, если что, редкоиспользуемые страницы вы добавите в ящик.

Также swap используется для гибернации. Это как спящий режим, только, если в спящем режиме у вас всё оборудование начинает меньше потреблять энергии, но всё ещё потребляет по чуть-чуть, то при гибернации все данные из оперативки сохраняются в swap-е, а компьютер полностью выключается, т.е. ничего не потребляет. А при запуске всё из swap-а возвращается в оперативку. Это позволяет больше сэкономить энергии, но запуск чуть дольше, чем при спящем режиме.

Пара популярных вопросов:

- нужен ли swap, если много оперативки?

Если большой излишек оперативки, т.е. много много оперативки не тратится - то можно обойтись и без swap-а. С другой стороны swap даже на таких системах может пригодится - будет чуть больше места для кэша.

- стоит ли делать swap на ssd?

С одной стороны, из-за скорости подкачка страниц будет быстрее, что увеличит производительность. С другой стороны, у ssd есть ограниченное число циклов перезаписи, и использование swap-а активно их расходует. Но, если у вас оперативки достаточно - подкачка будет происходить довольно редко. Ну и даже с относительно средним использованием, современные SSD проживут довольно долго, а их цена не такая кусачая. Точные сроки никто назвать не сможет, так как это индивидуально для конкретной системы и SSD.

- раздел подкачки или файл подкачки - что лучше?

Раньше из-за прослойки с файловой системой swap-файл был чуть помедленнее, но ядро после версии 2.6 работает с блоками swap-файла минуя файловую систему. С одной стороны пропала разница в производительности с разделом подкачки, с другой - нельзя просто взять и переместить файл подкачки на другую файловую систему, swap надо предварительно отключить. Но файл обычно легче увеличить, чем раздел, хотя если swap-раздел на LVM и есть свободное место - тоже несложно.

- сколько места выделять под swap?

Тут немного индивидуально - зависит от софта, который будет использоваться. Какие-то программы требуют больше swap-а, какие-то меньше. Но есть общие рекомендации - если у вас оперативки меньше 2 гигабайт - то под свап стоит выделить места в два раза больше, чем в оперативке. Если ещё и предполагаете использование гибернации - то в 3 раза больше. Ну и дальше по табличке.

Хорошо, что такое swap и для чего он нужен мы разобрались. Насколько используется swap мы можем увидеть с помощью утилиты free, либо утилиты swapon. Как видно, у нас для swap-а используется отдельный раздел, а точнее lvm:

```
sudo blkid | grep swap
```

Его размер - 2 гига, а используется небольшой процент. Кроме того видно, что у swap-а есть приоритет. Мы можем на одной системе сделать несколько swap разделов и файлов. Т.е. сначала будем складывать страницы в ящичек с высоким приоритетом, а если он забьётся - в ящик с приоритетом пониже. Скажем, мы можем выделить под swap место как на ssd, так и на жёстком диске. Также можно приоритет поставить одинаковым - тогда одновременно будут использоваться оба swap-а.

Попробуем добавить файл подкачки и перевести наш swap с раздела на файл. Для начала создадим файл размером в 2 гигабайта, для этого используем команду dd:

```
dd if=/dev/zero of=/swapfile bs=1K count=2M
```

Здесь bs - размер блока - 1Килобайт; 2M - 2 миллиона. Т.е. 2 миллиона блоков по 1 килобайту - получается 2 гигабайта. Помните, как после создания раздела мы записывали на него файловую систему - mkfs? Для swap-а мы делаем что-то похожее:

```
sudo mkswap /swapfile
```

Был бы раздел - указали бы название раздела. Команда вывела нам предупреждение, что у файла не безопасные права - 644, и рекомендуется выставить 600. Что ж, сделаем:

```
sudo chmod 600 /swapfile
```

И дальше, по аналогии того, как мы монтировали файловую систему, мы должны активировать swap, заодно зададим ему приоритет повыше:

```
sudo swapon -p 1 /swapfile
```

После этого посмотрим активные swap-ы - swapon. В списке появился наш файл.

Но, как и с файловыми системами, чтобы swap работал после перезагрузки, надо его добавить в fstab:

```
sudo nano /etc/fstab
```

Пропишем наш swap-file, а заодно закомментируем старый раздел.

Чтобы без перезагрузки избавиться от swap раздела, используем команду swapoff:

```
sudo swapoff /dev/dm-1
swapon
```

Таким образом мы убираем все страницы из раздела обратно в память или на другой swap.

Также стоит упомянуть такой параметр, как swappiness:

```
sysctl vm.swappiness
```

Этот параметр определяет, насколько сильно будет задействован swap. Тут значения от 0 до 100 - это баланс между кэшем и выгрузкой анонимных страниц. При низких значениях swap будет стараться держать анонимные страницы как можно дольше в памяти, при этом чаще выгружая кэш. Это может быть полезно, если программа сама управляет своим кэшем, а не отдаёт это ядру операционной системы. Например, так делают системы управления

базами данных. Если же значение ближе к 100, то система агрессивнее будет заниматься подкачкой - перемещать анонимные страницы в swap и обратно. Это обычно системы, где программы сами не кэшируют, но кэш важнее. Например, частая работа с данными, при этом производительностью программ можно немного пожертвовать.

Если мы хотим поменять это значение и чтобы оно оставалось после перезагрузки, нужно создать файл или подредактировать существующий в директории `/etc/sysctl.d/`, прописав в нём `vm.swappiness` и нужное значение, после чего применить:

```
sudo sysctl -p
```

Также стоит упомянуть, что существует модуль ядра zswap. Он создаёт в оперативной памяти сжатую область, где и хранится swap. Это позволяет подкачке меньше использовать диск и работать быстрее. Но если места в оперативке нет - данные всё же выгружаются на swap, который на диске.

Подведём итоги. Сегодня мы с вами разобрали механизмы работы виртуальной памяти - что такое кэш, что такое грязные страницы, анонимные страницы, разобрались, зачем нужна подкачка, как она работает, как создавать раздел или файл подкачки и как это всё настраивать. Со swap-ом работают относительно редко - один раз настраивается во время установки системы, а больше делать ничего и не надо. Поэтому очень важно сразу всё настроить правильно.

Источник

Установка и настройка Samba на Ubuntu и Debian

Рассмотрим процесс установки ПО Samba на ОС Ubuntu 20.04. Для работы будем использовать облачные ресурсы. Также эта инструкция подойдет в том числе для установки Samba на Debian и Astra Linux. А начнем с краткого описания этого программного обеспечения.

Что такое Samba

Samba представляет собой программный пакет, разработанный для обеспечения совместимости и взаимодействия UNIX-подобных систем с Windows. ПО распространяется по свободной лицензии уже на протяжении 30 лет. Samba обеспечивает беспроблемную интеграцию серверов и ПК, работающих под управлением UNIX, в систему AD (Active Directory). Это ПО может использоваться как контроллер и в качестве стандартной составляющей домена. Таким образом, у пользователей появляются возможности по гибкой настройке облачных файловых хранилищ. Samba обеспечивает широкий функционал по управлению правами доступа к файлам и базам данных с назначением определенных групп пользователей.

Добавление пользователя

Здесь всё просто, введите команду:

```
sudo useradd -p new_server_pass new_server_user
```

Вместо `new_server_pass` и `new_server_user` можно использовать любой пароль и любое имя пользователя. Указанные здесь имя и пароль приведены просто для примера. Обратите внимание, что мы сразу ввели пароль, что стало возможным благодаря команде `-p`. Теперь используем команду `update` для обновления хранилища, и можно переходить к дополнительным настройкам.

Дополнительные настройки

Для подготовки к установке Linux Samba потребуется выполнить еще пару действий:

- Синхронизировать время. Это необходимо сделать для того, чтобы впоследствии корректно отображались даты. Это поможет в дальнейшем облегчить поиск данных, с поиском которых при некорректно отображаемом времени могли бы возникнуть проблемы.
- Настроить работу брандмауэра. Хотя в большинстве систем на основе Linux все порты открыты, некоторые нужные для работы Samba (чаще всего это 445 и порты 137-139) могут быть закрыты. Поэтому проверьте их доступность и при необходимости откройте их.

Для синхронизации времени используйте специальную утилиту `chrony` — добавленная в автозагрузку, она будет поддерживать правильные временные значения. Это добавление осуществляется с помощью простой команды:

```
sudo systemctl enable chrony
```

Теперь запускаем программу, используя команду `start`, и выставляем нужный часовой пояс, а корректность установки смотрим при помощи команды `date`. Вывод должен быть примерно таким:

```
Fri Oct 28 11:25:34 UTC 2022
```

Вот и всё, сервер готов к установке и настройке Samba на Ubuntu.

Установка Samba на Ubuntu 20.04

Для удобства разбили процесс установки на отдельные этапы.

Шаг 1. Подготовка

Для старта процесса установки используйте следующую команду:

```
sudo apt install samba -y
```

Теперь нужно запомнить системное имя сервиса. В большинстве случаев это `smbd`. Поэтому, если хотите вызвать сервис, то вводите именно это значение.

- И для начала настроим автозапуск, что делается командой `enable smbd`.
- Теперь запускаем, используя уже знакомую команду `start`.
- Затем проверяем статус системы при помощи `status`.

- Для остановки Samba используется `stop`.
- Чтобы перезапустить сервис, введите команду `restart`.
- Если хотите, чтобы Samba больше не запускалась автоматически, используйте инструкцию `disable`.
- Команда `reload` нужна для обновления конфигурации.

Следующая команда позволит принудительно открыть порт 445, а также 137-139 (помните про настройки брандмауэра?). Чтобы разрешить их в брандмауэре *ufw*, используйте:

```
sudo ufw allow Samba
```

Шаг 2. Настройка анонимного доступа

Допустим, у нас есть какой-то удаленный сервер, который находится за пределами нашего облака. Правила сетевой безопасности требуют ни в коем случае не открывать прямой доступ к нему через IP. Сделать это можно только через VPN-туннель, который к тому же уже настроен. Обычно серверы, к которым предоставлен VPN-доступ, имеют адрес 10.8.0.1, и именно с этим адресом в дальнейшем нам и предстоит работать.

Чтобы расшарить данные и предоставить к ним анонимный доступ, сначала откройте конфигурационный файл. Он находится здесь: `/etc/samba/smb.conf`. Рекомендуем сделать бэкап чистого файла — это поможет затем быстро восстановить исходное состояние программы без необходимости ее переустановки. Теперь удалите все комментарии, оставив только код, и введите команду `testparm`, чтобы убедиться, что программа работает нормально. В настройках общей папки введите следующие параметры:

```
[share]
    comment = шара
    path = /data/public_share
    public = yes
    writable = yes
    read only = no
    guest ok = yes
```

Также следите за тем, чтобы в следующих четырех полях (`mask` и `mode`) совпадали числовые значения (например, 0777). Что касается конкретных строк, то здесь:

- `[share]` — имя расшаренной папки, которое будет видно всем подключившимся к вашему серверу;
- `comment` — комментарий, который может быть каким угодно;
- `path` — путь к папке для хранения данных;
- `public` — дает разрешение на общий доступ: если не хотите, чтобы пользователи могли просматривать содержимое папки, установите значение `no`;
- `writable` — определяет, можно ли записывать данные в папке;
- `read only` — указывает, что папка только для чтения: чтобы пользователи могли создавать новые, поставьте значение `no`;
- `guest ok` — определяет, могут ли гости получать доступ к папке.

Таким образом, название папки и путь могут отличаться в зависимости от того, какие значения вы указали для общей папки. Комментарий тоже может быть любым, а для нижних четырех параметров значения устанавливаются в виде `yes` или `no`. Теперь перезапускаем программу и на всякий случай проверяем, можно ли подключиться к серверу из-под Windows.

Шаг 3. Настройка доступа по учетным данным пользователя

Для создания доступа по логину и паролю нужно в первую очередь создать новую директорию и настроить права. В файле конфигурации установите все параметры на `no` (см. выше), кроме `writable`: в этой строке должно стоять значение `yes`, то есть запись в папке должна быть включена.

Командой `mkdir` создаем новую директорию, далее с помощью `useradd someone` (вместо *someone* может стоять любое имя пользователя) создаем пользователя и задаем ему пароль командой `passwd`. Например, так:

```
passwd something
```

Теперь командой `sudo smbpasswd -a someone` добавляем нового пользователя и пытаемся выполнить вход: если всё настроено корректно, то нам будет открыт доступ к папке.

Шаг 4. Настройка группового доступа

Это понадобится для создания ограниченного доступа отдельных групп пользователей. В `smb.conf` после строки `guest ok` дополнительно пропишите следующие строки (все имена пользователей были сгенерированы просто для примера):

```
valid users = admin, vlasova_ulyana, usova_elena, sokolova_eva, spiridonova_kseniya
write list = admin, spiridonova_kseniya
```

В строке `valid users` указываются пользователи, которым предоставлен доступ к директории. А в строке `write list` указаны те, кто может изменять данные в папке. Кроме того, после строки `force directory mode` добавьте еще одну строку со следующим значением:

```
inherit owner = yes
```

Она необходима для того, чтобы можно было включать наследование создаваемых объектов. Теперь записываем настройки и перезапускаем сервис, после чего новые настройки должны вступить в силу.

Шаг 5. Подключение к ресурсу из-под Windows и Linux

- Для быстрого подключения к Самбе из-под Windows нажмите *Ctrl+E* и введите путь. Только учтите, что нужно использовать \\ для указания пути к ресурсу в сети. А чтобы не соединяться с сервером постоянно, можно выбрать опцию подключения ресурса как диска, если это позволяет ваша политика безопасности. В новом окне задайте букву для наименования диска и заполните необходимые данные.
- Для подключения к Самбе из-под Linux используются утилиты `cifs`, которые устанавливаются с помощью команды `sudo apt install cifs-utils -y`. Далее ресурс монтируется и выполняется подключение. Это делается с помощью `mount.cifs //10.8.0.1/our_share /share` (путь и название ресурса могут быть любыми). Можно выполнять монтирование и автоматическим способом, для чего используется файл конфигурации `fstab` со своими настройками.

Шаг 6. Настройка сетевой корзины

Эта операция понадобится, чтобы избежать случайного безвозвратного удаления файлов. Для этого создаем такую директорию:

```
[Recycle]
    comment = Корзина для временного хранения файлов
    path = /directory/recycle
    public = yes
    browseable = yes
    writable = yes
    vfs objects = recycle
    recycle:repository = .recycle/%U
    recycle:keeptree = Yes
    recycle:touch = Yes
    recycle:versions = Yes
    recycle:maxsize = 0
    recycle:exclude = *.tmp, ~$*
    recycle:exclude_dir = /tmp
```

Теперь рассмотрим построчно, что означают эти параметры (кроме первых шести строк, с ними понятно):

- `vfs objects = recycle` — указание на использование соответствующей подсистемы;
- `repository` — путь для хранения удаленных данных;
- `keeptree` — сохранять ли дерево каталогов после удаления;
- `touch` — нужно ли изменять временные метки файлов при их помещении в корзину;
- `versions` — нужно ли указывать номер версии, если удаляются файлы с одинаковыми именами;
- `maxsize` — максимальный размер помещаемого в корзину файла. Значение 0 отключает лимиты;
- `exclude` — какие типы файлов нужно исключить;

- `exclude_dir` — какие каталоги нужно исключить.

Заключение

На этом всё, теперь вы знаете, что нужно делать перед установкой Самбы, как ее ставить на облачный сервер и настраивать под собственные нужды.

Установка Mattermost на Ubuntu

Mattermost — это платформа для обмена сообщениями и совместной работы, которую можно установить на собственных серверах или в облаке. Mattermost является альтернативой таким мессенджерам как Slack и RocketChat.

В этом гайде мы рассмотрим тарифный план Free, который включает в себя безлимитную историю сообщений и групповые звонки (подробнее о тарифах). Клиенты Mattemost доступны на мобильных платформах (IOS, Android), в десктоп-версиях (Windows, Linux, Mac), также поддерживается браузерная версия.

> Для тарифа Free доступна только Self Hosted версия; далее будет разбираться установка на Ubuntu. Ознакомиться с другими способами установки (в том числе и Docker-образ) можно на официальном сайте.

Технические требования

На 1000 пользователей минимально потребуется 1 CPU, 2GB RAM и PostgreSQL v11+ (необязательно использовать именно Postgres).

Установка Mattermost

Теперь можно переходить к самой установке. Для установки подключим репозиторий `deb.packages.mattermost.com/repo-setup.sh`.

```
curl -o- https://deb.packages.mattermost.com/repo-setup.sh | sudo bash -s mattermost
```

Здесь в `sudo bash -s mattermost` передается аргумент `mattermost` для добавления репозитория только Mattermost. Аргументом по умолчанию в данном скрипте является `all`, тогда будут также добавлены репозитории Ngnix, PostgreSQL и Certbot.

Установим сервис. Он будет установлен по пути `/opt/mattermost`. Пользователь `mattermost` и группа `mattermost` будут созданы автоматически.

```
sudo apt update
sudo apt install mattermost -y
```

После установки понадобится создать `config.json` с необходимыми правами (он создается на основе `config.defaults.json`). Чтение и запись в файл будут доступны только для пользователя владельца (в нашем случае это `mattermost`).

```
sudo install -C -m 600 -o mattermost -g mattermost /opt/mattermost/config/config.defaults.json /
```

В конфиге необходимо заполнить параметры:

```
sudo nano /opt/mattermost/config/config.json
```

- SiteUrl — сюда необходимо прописать созданный домен с протоколом https (в блоке `ServiceSettings`). Позже мы подключим SSL-сертификат.

  7b483263-249e-4b7b-9621-61cddbc1fcd6?width=730&height=500

- DriverName — по умолчанию стоит postgres (в блоке `SqlSettings`).
- DataSource — указать пользователя, пароль, хост и имя БД в ссылке подключения (в блоке `SqlSettings`).

  367a08a5-0b32-424e-9399-9f63923513da?width=799&height=402

Остальные конфиги не важны для первого запуска, и их можно будет изменить позднее в административной консоли Mattermost.

Запускаем сервис Mattermost:

```
sudo systemctl start mattermost
```

Проверим, что ничего не упало и Mattermost поднялся:

```
sudo systemctl status mattermost.service
```

8627ec6b-d788-4bf9-a99b-b17d36b1a65f?width=854&height=133

А также, что он доступен на порту 8065.

e328427a-7bef-4a62-90d7-bd2c3abccc9d?width=1574&height=980

Если сайт не открывается, проверьте настройки фаервола. Также можно обратиться к порту локально с сервера:

```
curl -v localhost:8065
```

Дополнительно добавим автозапуск сервиса:

```
sudo systemctl enable mattermost.service
```

Nginx

Мы не будем напрямую использовать порт 8065 — он будет позже закрыт фаерволом. Для проксирования запросов установим Nginx.

```
sudo apt install nginx
```

Теперь добавим конфигурацию:

```
sudo nano /etc/nginx/sites-available/mattermost
```

В строке `server_name` необходимо указать свой домен, в нашем случае *mattermost.twc1.net*.

Обратите внимание, что проксируется как HTTP, так и websocket-протоколы.

```
upstream backend {
  server 127.0.0.1:8065;
  keepalive 32;
}

proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=mattermost_cache:10m max_size=3g inactive

server {
  listen 80;
  server_name mattermost.twc1.net;
  location ~ /api/v[0-9]+/(users/)?websocket$ {
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    client_max_body_size 50M;
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Frame-Options SAMEORIGIN;
    proxy_buffers 256 16k;
    proxy_buffer_size 16k;
    client_body_timeout 60;
    send_timeout 300;
    lingering_timeout 5;
    proxy_connect_timeout 90;
    proxy_send_timeout 300;
    proxy_read_timeout 90s;
    proxy_pass http://backend;
  }
  location / {
    client_max_body_size 50M;
```

```
    proxy_set_header Connection "";
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Frame-Options SAMEORIGIN;
    proxy_buffers 256 16k;
    proxy_buffer_size 16k;
    proxy_read_timeout 600s;
    proxy_cache mattermost_cache;
    proxy_cache_revalidate on;
    proxy_cache_min_uses 2;
    proxy_cache_use_stale timeout;
    proxy_cache_lock on;
    proxy_http_version 1.1;
    proxy_pass http://backend;
 }
}
```

Добавим линк конфигурационного файла в `sites-enabled`:

```
sudo ln -s /etc/nginx/sites-available/mattermost /etc/nginx/sites-enabled/mattermost
```

Уберем дефолтный конфиг:

```
sudo rm -f /etc/nginx/sites-enabled/default
```

Делаем рестарт сервиса Nginx:

```
sudo service nginx restart
```

Теперь выпустим сертификат от Let’s Encrypt для домена через `certbot`:

```
sudo apt install python3-certbot-nginx && certbot
```

Certbot попросит указать почту, домен и после добавит сертификат для домена.

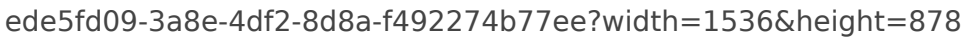
После установки сертификата `certbot` допишет конфигурацию nginx mattermost. Будет добавлена директива `listen` для обработки запросов с 443 порта, директивы с ключами и конфигами для SSL, а также редирект с HTTP на HTTPS.

Firewall

Я ограничил входящие TCP запросы портами 22 для работы с SSH, 80 и 443 для TCP. Для сбора метрик на дашборд сервера необходимо открыть также порт 10050; список IP-адресов, для которых необходимо открывать порт, будет в `/etc/zabbix/zabbix_agentd.conf`.

Первый запуск

Теперь Mattermost будет доступен по адресу https://mattermost.twc1.net/.



Можно создать аккаунт и рабочее пространство сразу из браузерной версии.

После установки и при первом входе можно наблюдать проблему с коннектом Web-сокетов.

Рекомендую проверить конфиги, теперь сделать это можно из System Console.

Например, моя ошибка заключалась в URL сервера.

Из коробки будут доступны звонки, playbooks, магазин с плагинами, аутентификация Gitlab.
Также сам продукт имеет отличную документацию.

Источник

Как использовать Nessus для сканирования уязвимостей в Ubuntu 22.04

Nessus — это один из самых известных и широко используемых сканеров уязвимостей в мире. Разработанный компанией Tenable, Inc., Nessus предоставляет комплексное решение для обнаружения уязвимостей, позволяя организациям и индивидуальным пользователям идентифицировать и устранять потенциальные угрозы безопасности в их сетевой инфраструктуре. С его помощью можно проводить глубокий анализ безопасности, охватывая различные аспекты, от простого обнаружения уязвимостей до сложных проверок на соответствие стандартам безопасности.

Версии Nessus: Essentials, Professional, Expert

- Nessus Essentials. Бесплатная версия, предназначенная для домашних пользователей и тех, кто только начинает знакомство с областью безопасности. Эта версия предоставляет базовые функции сканирования и обнаружения уязвимостей.
- Nessus Professional. Платная версия, предназначенная для профессионалов в области безопасности и крупных организаций. Она предлагает расширенные функции, такие как возможность сканирования больших сетей, интеграция с другими системами безопасности и дополнительные инструменты для анализа и отчетности.
- Nessus Expert.Это премиальная версия, которая включает в себя все функции Professional, а также дополнительные инструменты и возможности, такие как поддержка облачного сканирования, интеграция с системами управления инцидентами безопасности и дополнительные опции настройки.

Nessus предлагает следующий набор функций для сканирования уязвимостей:

- Обнаружение уязвимостей: Nessus обнаруживает уязвимости в различных системах и приложениях на основе своей базы данных известных уязвимостей.
- Проверка на соответствие: Nessus проводит проверки на соответствие различным стандартам безопасности и регулированиям.
- Интеграция с другими системами: предоставляется возможность интеграции с системами управления инцидентами, системами управления журналами и другими инструментами безопасности.
- Облачное сканирование: Версия Nessus Expert позволяет проводить сканирование в облачных средах, таких как AWS, Azure и Google Cloud.
- Визуализация данных: Nessus включает в себя дашборды и отчеты для представления результатов сканирования.
- Регулярные обновления: Nessus обновляет свою базу данных уязвимостей для отслеживания новых угроз.
- Гибкая настройка: предоставляет опции настройки для адаптации процесса сканирования к конкретной среде.

Установка Nessus

Nessus можно установить на Ubuntu двумя способами: как docker-контейнер и как deb-пакет. Рассмотрим оба способа.

Установка Nessus на Ubuntu через Docker

1. Подготовка к установке. Прежде всего, убедитесь, что на вашей системе установлен Docker.

2. Скачивание образа. Загрузите последнюю версию образа Nessus из Docker Hub, выполнив следующую команду:

```
docker pull tenable/nessus:latest-ubuntu
```

Загрузка может занять около 10 минут.

3. Создание и запуск контейнера. После того как образ загружен, создайте и запустите контейнер с помощью следующей команды:

```
docker run --name "nessus_tw" -d -p 8834:8834 tenable/nessus:latest-ubuntu
```

Где:

- `--name "nessus_tw"` задает имя контейнера.

- `-d` указывает Docker запускать контейнер в фоновом режиме.
- `-p 8834:8834` проксирует порт 8834 из контейнера на порт 8834 хоста, делая приложение доступным на localhost:8834.

4. Запуск контейнера после остановки. Если вам потребуется запустить контейнер после его остановки, используйте команду:

```
docker start nessus_tw
```

Установка Nessus в Ubuntu как deb-пакет

> Напомним, что для загрузки установщика с сайта tenable.com на сервере должен быть запущен VPN.

1. Скачивание установочного пакета: Сначала загрузим установочный пакет для Ubuntu с помощью команды `curl`:

```
curl --request GET \
  --url 'https://www.tenable.com/downloads/api/v2/pages/nessus/files/Nessus-10.6.1-ubuntu1404_am
  --output 'Nessus-10.6.1-ubuntu1404_amd64.deb'
```

2. Установка Nessus. Установочный файл Nessus был загружен в текущую директорию. Теперь используем `dpkg` для установки Nessus на нашу машину с Ubuntu. Введите следующую команду для установки:

```
sudo dpkg -i ./Nessus-10.6.1-ubuntu1404_amd64.deb
```

3. Запуск службы Nessus. После установки необходимо запустить службу Nessusd. Введите следующую команду:

```
sudo systemctl start nessusd.service
```

4. Проверим корректность работы nessusd. Выполним `systemctl status nessusd` и убедимся, что служба запущена и работает без ошибок:

```
Active: active (running)
```

5. Доступ к Nessus через браузер. Теперь вы можете получить доступ к Nessus в вашем локальном браузере по адресу `https://localhost:8834/`

> Порт 8834 является портом по умолчанию для Nessus. В большинстве браузеров при попытке доступа к Nessus может появиться предупреждение о безопасности с предложением вернуться назад. Однако

> это полностью безопасно, и вы можете нажать на «Дополнительно» и затем продолжить работу с веб-сайтом.

Первоначальная настройка Nessus

1. Переход на страницу настройки. После запуска контейнера откройте браузер и перейдите по адресу `https://localhost:8834`. Вы увидите, что выполняется загрузка необходимых компонентов.
2. Регистрация на сайте Tenable. Пока идет загрузка необходимых компонентов, рекомендуется зарегистрироваться на официальном сайте Tenable для получения кода активации. После регистрации код будет отправлен на указанный вами электронный адрес.

> “ При регистрации необходимо использовать ящик на домене, отличном от .ru.

3. Использование мастера установки.

- Как только компоненты загрузятся, начнется процесс настройки с помощью мастера установки. Нажмите «Continue».

  a6052b12-52ba-4ed8-b4df-09a36de9685b?width=1280&height=894

- Выберите версию «Nessus Essentials».

  37a4b461-d860-4bdf-b9db-e2e497d28d99?width=1279&height=991

- Введите код активации, который был отправлен на вашу электронную почту.

  54a20a24-8ac0-4c2e-ba69-6beebc496487?width=1280&height=990

- Создайте учетную запись пользователя, указав имя и пароль.

  b8609651-970b-449d-bf55-f45784d390a0?width=1278&height=987

4. Завершение установки. Дождитесь завершения установки и загрузки всех плагинов. Как только все процессы на странице https://localhost:8834/#/settings/about/events будут завершены, установка Nessus будет полностью завершена.

Настройка сервера beeBox

После успешной установки и настройки Nessus настало время проверить его в действии. Для этого нам потребуется целевая система, которую мы будем сканировать на наличие уязвимостей. В рамках этой статьи мы решили использовать виртуальную машину bee-box.

bee-box — это специализированная виртуальная машина на базе bWAPP (buggy web application). bWAPP создано специально с уязвимостями, что позволяет специалистам по безопасности, разработчикам и студентам практиковаться в обнаружении и предотвращении угроз. В beeBox представлены следующие уязвимости:

- Injection (HTML/SQL/LDAP/SMTP/…)
- Broken Authentication & Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Components with Known Vulnerabilities
- Unvalidated Redirects & Forwards
- XML External Entity Attacks (XXE)
- Server-Side Request Forgery (SSRF)

Это делает beeBox идеальным инструментом для демонстрации возможностей Nessus.

Установка bee-box в VirtualBox

В этом разделе мы рассмотрим процесс установки bee-box в VirtualBox версии 7.0. Отметим, что в других версиях VirtualBox процедура может незначительно отличаться.

1. Скачивание образа. Сначала скачайте образ виртуальной машины beeBox (файл bee-box_v1.6.7z) и распакуйте его.
2. Создание новой виртуальной машины. Запустите VirtualBox и нажмите кнопку «New». В разделе «Name and Operating System» укажите имя машины, выберите тип ОС Linux и версию «Oracle Linux (64-bit)».

   9f379244-c1ee-401c-9008-a4005266f4b1?width=712&height=605

3. Настройка оборудования. Выделите 1024 МБ RAM и 1 CPU.

   e796e1e5-0167-4f27-8d33-1495957eda4f?width=708&height=603

4. Выбор жесткого диска. В разделе «Hard Disk» выберите «Use an Existing Virtual Hard Disk File», затем «Add» и укажите путь к файлу `bee-box.vmdk`, который вы распаковали ранее.

ea5f0ee0-41c9-4f25-986e-391ac1d036f6?width=956&height=807

5. Настройка сети. Перед запуском машины перейдите в «Settings» -> «Network» и измените «Attached to» с «NAT» на «Bridged Adapter».

334e7edc-4ab5-4dee-901d-29761064f1af?width=946&height=437

6. Запуск виртуальной машины. Нажмите «Start» для запуска.

7. Настройка раскладки клавиатуры. После загрузки рабочего стола кликните на «USA» в верхнем меню, выберите «Keyboard Preferences», затем «Layouts» и в «Keyboard model» укажите «IBM Rapid Access II».

79c158a5-bf6a-4d33-8125-04ab67bae973?width=512&height=555

8. Получение IP-адреса. Откройте терминал и введите команду ip a для определения IP-адреса виртуальной машины. Затем обратитесь к этому IP с вашей основной машины, чтобы убедиться в доступности приложения.

5caaa196-7dab-420b-abc9-fff7323c70c9?width=660&height=461

Сканирование с использованием Nessus

Общие параметры Nessus

Перед тем как приступить к использованию Nessus, важно ознакомиться с его интерфейсом и настройками. Главный экран программы разделен на две основные вкладки: «Scans» и «Settings». Для начала давайте подробно рассмотрим раздел «Settings».

About:

- Overview. Этот подраздел предоставляет общую информацию о вашей установке Nessus, включая версию, детали лицензии и другую ключевую информацию.
- License Utilization. Здесь отображаются все IP-адреса, которые были просканированы. В бесплатной версии доступно сканирование до 16 хостов. Хосты, которые не сканировались в течение 90 дней, автоматически освобождаются из лицензии.
- Software Update. Позволяет настроить автоматическое обновление или запустить процесс обновления вручную.
- Encryption Password. Здесь можно установить пароль для шифрования данных Nessus. Если вы решите установить пароль, то вам необходимо запомнить его, так как без него восстановление данных будет невозможно.

- Events. Этот подраздел позволяет просматривать историю обновлений и другие ключевые события.

Advanced Settings:

- Этот раздел содержит дополнительные настройки Nessus. Хотя мы не будем подробно рассматривать каждую из них в этой статье, вы можете найти подробную информацию о каждой настройке на официальном сайте.

Proxy Server:

- Если ваша сеть требует использование прокси-сервера для доступа к интернету или целевым серверам, вы можете настроить параметры прокси в этом разделе.

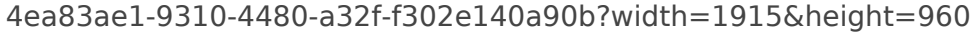
SMTP Server:

- Здесь можно настроить параметры SMTP-сервера, чтобы Nessus мог отправлять уведомления по электронной почте о результатах сканирования и других важных событиях.

Запуск базового сканирования

Теперь перейдем во вкладку Scans. Перед тем как приступить к использованию Nessus для сканирования уязвимостей, необходимо правильно настроить параметры сканирования. Это обеспечит максимальную эффективность и точность в обнаружении уязвимостей.

На главном экране нажмем на кнопку New Scan и попадем в мастер создания сканирования.



Выбор типа сканирования. Для нашего примера выберем Basic Network Scan.

Общие настройки

- General. Задайте имя, описание, выберите папку для результатов и в Targets укажите IP-адрес виртуальной машины bee-box.

- Schedule. Здесь можно настроить периодичность сканирования (опционально).
- Notifications. Укажите почтовые адреса для уведомлений. Для работы этой функции необходимо настроить подключение к SMTP-серверу в настройках.

Детальные настройки

- Discovery. Тут мы можем выбрать тип сканирования — common ports (4700 часто используемых портов, исходя из документации), all ports (все порты) или выбрать Custom с тонкой настройкой сканирования портов. В примере выберем common

ports.

- Assessment. Можем выбрать способ обнаружения уязвимостей. В примере выберем «Scan for all web vulnerabilities» для более быстрого выполнения сканирования. Вы также можете выбрать Custom. Все параметры доступные в настройке описаны в официальной документации.

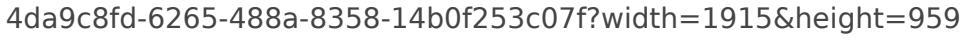
- Report. Настройте параметры формирования отчета (опционально). В примере ничего изменять не будем.

- Advanced. можно настроить скорость выполнения сканирования. При выборе ручной настройки, можно включить/отключить дебаг для плагинов. Подробнее про каждый пункт вы можете прочитать в документации. В примере установим значение в Default.

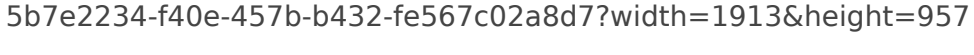
Дополнительные настройки

Помимо основных настроек сверху вы можете увидеть две вкладки — Credentials и Plugins.

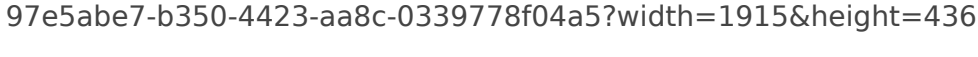
- Credentials. Вы можете установить данные для подключения к сервисам, запущенным на сканируемом хосте (например, для поиска уязвимостей, для которых необходим не привилегированный доступ).



- Plugins. Можем увидеть список плагинов, которые будут использоваться при сканировании. При выборе других типов сканирования, например, advanced scans, у вас будет возможность включать и отключать нужные плагины.



Завершение настройки и запуск сканирования. Нажмите **save**, затем вернитесь на главную страницу и нажмите **Launch** для запуска сканирования (находится рядом с крестиком).



Теперь сканирование запущено. За ходом выполнения вы можете наблюдать, нажав на созданный скан.

Просмотр результатов сканирования

После завершения сканирования для анализа результатов перейдите к выбранному сканированию.

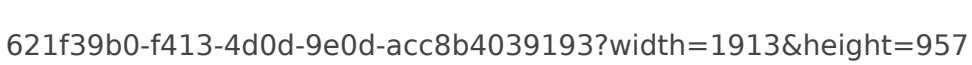
Центральная часть экрана представляет собой таблицу с детальной информацией о найденных уязвимостях:

- Severity: Отражает степень серьезности угрозы, основываясь на метрике CVSS.
- CVSS: Значение метрики CVSSv2, которое указывает на риск уязвимости.
- VPR: Альтернативная метрика от Tenable, предоставляющая дополнительную оценку риска.
- Name: Название обнаруженной уязвимости.
- Family: Категория или группа, к которой относится уязвимость.
- Count: Количество экземпляров данной уязвимости.

Важно отметить, что некоторые уязвимости могут быть объединены в группу «Mixed». Чтобы изменить это поведение, перейдите в Settings -> Advanced и установите параметр Use Mixed Vulnerability Groups в значение «No».

Слева от таблицы представлена информация о целевом хосте, а также диаграмма, демонстрирующая распределение обнаруженных уязвимостей по степени их серьезности.

Для детального изучения конкретной уязвимости, просто кликните по ее названию. Возьмем для примера уязвимость «Drupal Database Abstraction API SQLi».



В основной части экрана будет представлено:

- Описание уязвимости: Краткое описание проблемы и версии ПО, в которой она была устранена.
- Детали обнаружения: Отчеты о найденной уязвимости и методах ее устранения.
- Технические детали: В данном случае, это SQL-запрос, который использовался для выявления уязвимости.

В левой части экрана представлена дополнительная информация:

- Информация о плагине: Описание плагина, который обнаружил уязвимость.
- Рейтинги VPR и CVSS: Оценки серьезности уязвимости по различным метрикам.
- Данные об эксплуатации: Информация о возможности эксплуатации уязвимости.
- Ссылки: Полезные ссылки на ресурсы, такие как exploit-db, nist.gov и другие, где можно узнать больше о данной уязвимости.

Заключение

В этом гайде мы подробно рассмотрели процесс установки, настройки и использования Nessus для сканирования уязвимостей. Nessus — это мощное автоматизированное средство, но его эффективность напрямую зависит от корректной настройки. Однако стоит помнить, что для обеспечения полноценной безопасности сети и системы нельзя полагаться исключительно на автоматизированные инструменты. Комплексный подход и постоянное обучение в области безопасности — ключ к надежной защите.

Источник

Установка и настройка ArchLinux руками (без скрипта archinstall)

> Это краткая и надеюсь понятная для новичков wiki документация, по установке Arch Linux. Рассмотрим мы все, от конфигурации дисков и до установки графической оболочки!

Не будем медлить и приступим к установке.

Первым делом скачиваем образ дистрибутива с официального сайта!

Скачиваете с удобным для вас торрент-клиентом по этому пути:

https://archlinux.org/download/

Создаем загрузочный образ на Flash-USB и настраиваем BIOS под автозагрузку нашей флешки.
Думаю эти пункты можно и не раскрывать т.к. если вы уже решили установить Arch, то вы обязаны знать базовые вещи))

1. Проверка интернет-соединения.

> В этой инструкции не будет информации про настройку подключения по Wi-Fi. Если вы собираетесь подключиться к беспроводной сети, то рекомендую прочитать эту статью.

1.1. Запускаем команду ping для проверки подключения к интернету.

```
ping wiki.hellnetwork.net
```

При успешном соединении должны "побежать" байтики.

2. Настройка дисков.

Одна из самых важных вещей - правильная настройка дисков, т.к. без этого невозможно даже запустить нашу машинку.

2.1. Сначала проверим доступные нам диски командой:

```
lsblk
```

В моем случае нужный мне диск это ***/dev/sdb.***

> Не забывайте, что в вашем случае диск скорее всего будет другой!

2.2. Подготовка диска.

В этом гайде мы будем пользоваться утилитами ***gdisk*** и ***cgdisk.***

Запускаем команду ниже, что бы **удалить все данные** на нашем диске, и подготовить его к разметке:

```
gdisk /dev/sdb/
```

На первую опцию пишем букву **x** - для того, что бы зайти в режим эксперта, затем **z** что бы удалить данные на диске.

2.3. Создание разделов.

Пора создавать разделы на нашем пустом диске. Для этого мы воспользуемся утилитой ***cgdisk*** (т.к банально, это легче для меня), вы можете воспользоватся удобной для вас

утилитой!

Открываем наш диск командой:

```
cgdisk /dev/sdb/
```

1. **/boot-раздел. *First Sector*** оставляем дефолтным. На втором пункте прописываем сколько места мы выделяем для нашего раздела. Я обычно выделяю 1024MiB. **Код раздела - ef00. Даем имя нашему разделу - boot.**
2. **Раздел /swap**. ***First Sector*** также оставляем пустым. Затем пишем сколько места мы выделяем для раздела подкачки, обычно я выбираю не меньше 8GiB. В зависимости от размера вашего диска вы можете выделять сколько вам угодно (но желательно больше 4GiB). **Код раздела - 8200. Даем имя нашему разделу - swap.**
3. **/root.** Создаем корневой раздел. ***First Sector*** также оставляем пустым. Затем прописываем сколько места мы выделяем для корневой директории (и да, лучше всего разделять корневую директорию и домашнюю, в целях безопасности.). Обычно я выбираю не больше 40GiB, т.к корневой директорию я особо не буду нагружать. **Код раздела - 8300. Даем имя нашему разделу - root.**
4. И последнее **/home** - домашний раздел. Здесь мы все пункты оставляем пустыми, ибо все остальное место выделяется для домашнего раздела. **Код раздела - 8300. Даем имя нашему разделу - home.**

2.4. Форматирование разделов.

Теперь нам нужно отморфатировать все наши разделы, для дальнейшего монтирования.

Запускаем команду **lsblk** и видим, что в нашем диске (в моем случае - **/dev/sdb/**) появились 4 раздела.
Форматирование происходит по порядку разделов, то есть сначала **/boot , /swap, /root и /home.**

1. **Начнем с /boot.** Вводим команду**:**

   ```
   mkfs.fat -F32 /dev/sdb1
   ```

2. Для **/swap** вводим команды:

   ```
   mkswap /dev/sdb2
   swapon /dev/sdb2
   ```

3. Затем последние разделы форматируем под ext4.

```
mkfs.ext4 /dev/sdb3
mkfs.ext4 /dev/sdb4
```

2.5 Проверяем результат форматирования.

Для этого вводим команду ***lsblk*** и в результате, в пункте **Type** наши разделы должны показываться как **part.**

2.6 Монтирование разделов.

Для того, что бы установить систему, разделы сначала нужно будет примонтировать на определенный путь.

Для корневой директории:

```
mount /dev/sdb3 /mnt
```

Затем создаем директории внутри **/mnt** , в которые мы будем монтировать наши разделы.

```
mkdir /mnt/boot
mkdir /mnt/home
mount /dev/sdb1 /mnt/boot
mount /dev/sdb4 /mnt/home
```

Затем снова пишем команду ***lsblk*** и видим, что разделы примонтированы в указанные директории.

3. Обновление зеркал.

Для начала забекапим файл зеркал:

```
cp /etc/pacman.d/mirrorlist /etc/pacman.d/mirrorlist.backup
```

Затем устанавливаем нужные нам пакеты:

```
sudo pacman -Sy pacman-contrib
```

и обновляем зеркала:

```
rankmirrors -n 6 /etc/pacman.d/mirrorlist.backup > /etc/pacman.d/mirrorlist
```

4. Установка основных пакетов и первоначальная настройка.

Теперь пришло время для установки важных пакетов нашей системы.

4.1. Установка базы линукс

```
pacstrap -K /mnt base linux linux-firmware base-devel
```

4.2. Генерируем **fstab** файл:

```
genstab -U -p /mnt >> /mnt/etc/fstab
```

4.3. Переходим к корневому каталогу устанавливаемой системы

```
arch-chroot /mnt
```

Устанавливаем нужные нам пакеты, в первую очередь ***nano*** для работы с файлами.

```
pacman -S nano
```

4.4. Настраиваем наши региональные данные.

Для этого открываем файл **/etc/locale.gen** и раскоментируем нужную нам строку. Для меня это **en_US.UTF-8.** Затем запускаем команду:

```
locale-gen
echo LANG=en_US.UTF-8 > /etc/locale.conf
export LANG=en_US.UTF-8
```

Дальше настройка timezone.

```
ln -s /usr/share/zoneinfo/Asia/Tashkent > /etc/localtime
hwclock --systohc --utc
```

Прописываем hostname системы:

```
echo yourhostname > /etc/hostname
```

Для поддержки 32-битной структуры в файле **/etc/pacman.d/mirrorlist** раскоментируем пункт **[multilib]** и все что с ним связано.
Затем обновляем пакеты:

```
sudo pacman -Sy
```

4.5. Создание пароля рута и создание нового пользователя

Для начала установим пароль от рута командой **passwd**
Затем создаем нового пользователя командой:

```
useradd -m -u users -G wheel,storage,power -s /bin/bash username
passwd username
```

4.6. Настройка бутлоудера

Для начала установим наш бутлоудер:

```
bootctl install
```

но для его работы нам нужно прописать entries. В файле **/boot/loader/entries/arch-entry.conf** прописываем все что внизу:

```
title <yourtitle>
linux /vmlinuz-linux
initrd /initframs-linux.img
```

Затем прописываем команду:

```
echo "options root=PARTUUID=$(blkid -s PARTUUID -o value /dev/sdb3) rw" >>
/boot/loader/entries/arch-entry.conf
```

4.7. Настройка сети.

Установка и настройка dhcp:

```
sudo pacman -S dhcpd
sudo systemctl enable dhcpcd@<your-net-card>.service
```

Установка менеджера сети:

```
sudo pacman -S networkmanager
sudo systemctl enable NetworkManager.service
```

Теперь мы можем перезагрузить систему и вытащить нашу флешку.