Основы DevOps

Компьютерные сети

Введение в сети

Введение

- Знание компьютерных сетей крайне важно для DevOps инженера. Ведь именно на его плечи ложится настройка сетевой инфраструктуры, развертывание серверов (как компьютеров, на которых установлены веб-приложения, так и специальных программ - веб-серверов, которые обслуживают эти приложения) и настройка безопасного взаимодействия между различными компонентами системы (компоненты могут располагаться на различных серверах). Кроме того, обязанности DevOps включают в себя работу с облачными технологиями, что в принципе сложно представить без знания сетей.

- Чтобы относительно свободно чувствовать себя в сетевом мире необходимо, во-первых, понимать то, как один компьютер может взаимодействовать с другим (за это отвечают абстрактная модель **OSI** и модель **TCP/IP**), во-вторых, представлять, как происходит адресация в сети, т.е. как могут найти друг друга несколько компьютеров, не соединенных напрямую (за это отвечает ряд сетевых протоколов - **IP, DHCP, NAT, DNS**), и, наконец, владеть инструментами, которое все вышеперечисленное позволяют настраивать - мы будет изучать инструменты, которые предлагают операционные системы на ядре Linux (**ip**, **ping**, **dig**, **traceroute**, **ufw** и другие).

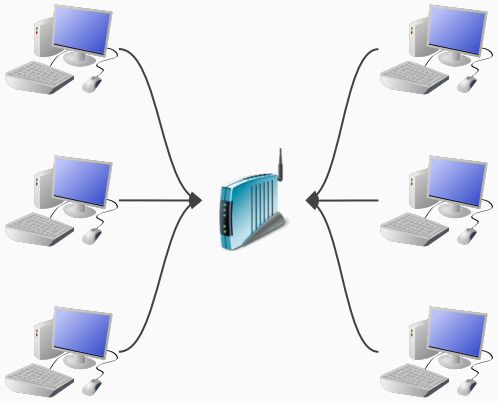
Типизация сетей

- **Сеть** - группа компьютеров или иных устройств, соединенных вместе каким-либо способом. Соединение может быть осуществлено при помощи провода или радиосигнала - это не имеет принципиального значения. Однако крайне важно понимать другое отличие - между локальной и глобальной сетью.

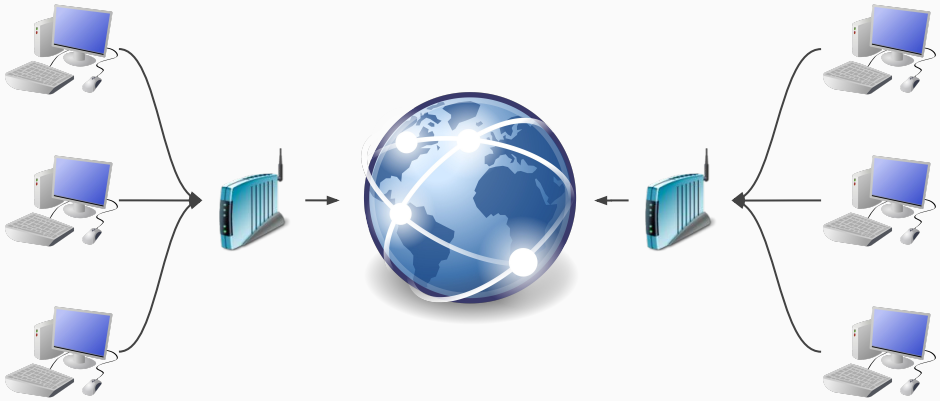
- **Локальная сеть** (**LAN**) объединяет устройства в закрытой локации - в квартире или даже в целом корпоративном здании. Для доступа в глобальную сеть может использоваться специальное устройство - маршрутизатор (роутер), который подключен к провайдеру (помогает связать локальную сеть с глобальной) с помощью проводного или беспроводного соединения. Кроме доступа в глобальную сеть, роутер идентифицирует устройства в локальной сети и обеспечивает маршрутизацию между ними.

- **Глобальная сеть** (**WAN**) объединяет несколько локальных сетей. Основным способом объединения является все те же маршрутизаторы, которые способны перенаправлять данные из одной локальной сети в другую и обратно. Это стало возможно благодаря IP (Internet Protocol) адресам - числовым идентификатором, которые должны быть уникальны в рамках одной сети. Маршрутизатор имеет один IP-адрес в первой сети и совершенно другой - во второй - тем самым он может обеспечить обмен данными между сетями.

Визуальное представление сетей



Локальная сеть (LAN)



Глобальная сеть (WAN)

Адресация внутри локальной сети (DHCP)

- Как уже было упомянуто, устройства в локальной сети могут взаимодействовать друг с другом, используя IP-адреса. При подключении нового устройства к сети, обычно роутер автоматически назначает ему IP-адрес. Этот IP-адрес позволяет другим устройствам в сети отправлять запросы этому новому устройству. Вся вышеописанная технология носит имя DHCP (Dynamic Host Configuration Protocol) Кроме того, сетевым администраторам можно вручную назначить IP-адрес устройства, учитывая настройки сети, при условии, что этот адрес еще не используется другим устройством. IP-адреса используются для маршрутизации данных в сети и определения, находятся ли устройства внутри этой сети или за ее пределами.

- Помимо IP-адресов, при передаче данных внутри локальной сети важную роль играют также MAC-адреса (Media Access Control) - уникальные идентификаторы сетевых адаптеров устройств. В отличие от IP-адресов, которые отвечают на вопрос "кому" (логическая адресация), MAC-адреса отвечают на вопрос "где" (физическая адресация), так как они помогают определить физическое расположение устройства в сети. Важно отметить, что MAC-адреса действуют только в пределах локальной сети и не могут использоваться для передачи данных между разными сетями.

Адресация в контексте глобальных сетей (NAT)

- Передача данных между разными сетями является гораздо более сложным концептом, чем взаимодействие устройств на локальном уровне. Сначала устройство определяет, что IP-адрес не принадлежит к его локальной сети и перенаправляет запрос к маршрутизатору. Маршрутизатор запоминает кто прислал ему запрос в этой сети, перенаправляет его от своего имени в другую сеть (где у маршрутизатора другой IP-адрес) и ждет ответа. Когда ответ приходит, маршрутизатор принимает его, условно “возвращается” в первоначальную сеть и отправляет ответ первоначальному устройству. Весь вышеописанный механизм имеет название NAT (Network Address Translation).

- Если маршрутизатор не подключен к той сети, куда направлен запрос, он может сделать следующие вещи. Во-первых, маршрутизаторы могут иметь т.н. адрес по умолчанию - этот маршрут обычно используется, чтобы отправить запрос на другой маршрутизатор, который имеет более широкий доступ к сети. Во-вторых, если положительного ответа не вернется, маршрутизатор просто проинформирует изначальное устройство о невозможности перенаправить запрос дальше.

Структура IP-адреса

Введение (IPv4)

- IPv4 (самая популярная версия IP адреса на данный момент) появилась в 1982 году. Данный адрес представляет собой 32-х битное число, которое вмещает в себя 4294967296 возможных значений. Обычно записывается в виде четырех десятизначных чисел от 1 до 255, которые разделены точками. Далее пример:

```
85.1.32.254
```

- Внутри самой большой глобальной сети в мире - сети Интернет - на 2023 год заняты абсолютно все оригинальные значения IPv4. Однако есть три блока адресов, которые запрещено использовать глобально. Блоки перечислены ниже:
  - **10.0.0.0** до **10.255.255.255**
  - **127.0.0.1** до **127.255.255.255**
  - **172.16.0.0** до **172.31.255.255**
  - **192.168.0.0** до **192.168.255.255**

  Эти адреса используются для сетей, подключенных к Интернету. Например, в Интернете находится роутер с IP-адресом 99.99.99 (такие адреса называются публичными). Одновременно роутер находится в своей сети с адресом 10.0.0.1 - такая сеть может содержать до 16777216 адресов и быть соединенной с другими сетями. Таким образом, проблемы исчерпания всех адресов IPv4 для устройств не существует - существует проблема исчерпания публичных адресов IPv4 .

Проблема определения сети

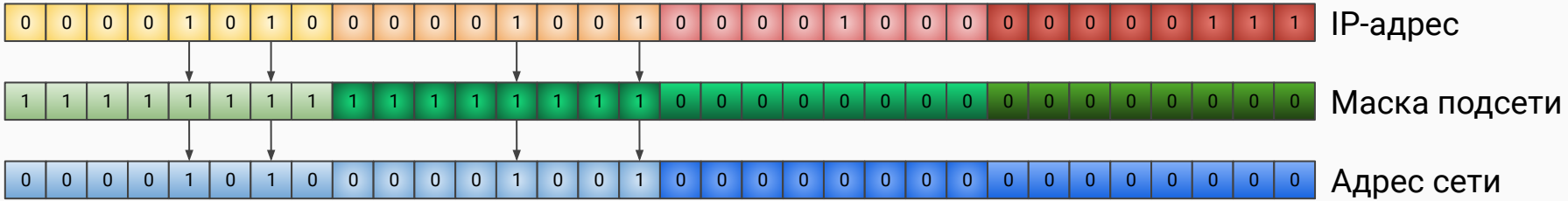
- Если присмотреться повнимательнее, то можно понять, что адрес IPv4 состоит из двух частей. Возьмем тот самый пример с адресом 10.0.0.1 - если у нас используется сеть 10.x.x.x, что в этой сети всегда будет заняты только первые восемь бит адреса, а непосредственно для самой адресации оставлено 24 бита (от 10.0.0.0 до 10.255.255.255). Таким образом, неизменная первая часть обозначает адрес сети, а оставшаяся часть - адрес устройства (также может называться адресом хоста).

- Предыдущий пункт дает понять способ, какой применяется устройством для определения того, в какой сети находится адресат запроса. Если IP адрес самого устройства имеет такой же адрес сети, какой у адресата запроса, они находятся в одной локальной сети. Во всех остальных случаях - это разные сети.

- Однако можно задать вопрос - как точно определить адрес сети? Ведь может быть сеть, которая начинается, например, не с 10.x.x.x (блок запрещенный в сети Интернет целиком), а с 10.10.x.x (только часть запрещенного блока). Что будет, если надо будет отправить запрос по адресу 10.5.0.1? Ответ на этот вопрос есть и заключается он в т.н. маске подсети, которая есть у каждого узла, и которую каждый узел прекрасно знает.

Маска подсети

- Как мы уже знаем, адрес IPv4 представляет собой 32-х битное число, в котором есть информация об адресе сети и адресе устройства в рамках этой сети. Чтобы точно узнать эти адреса, применяется т.н. маска подсети. Это также 32-х битное число, но те биты слева, которые относятся к адресу сети, у него установлены в единицу, остальные - в ноль. Для определение адреса сети берут два этих числа и применяют к ним операцию логического побитового “И”. В результате получим адрес сети. Предположим, что у нас есть IP-адрес 10.9.8.7 и маска подсети 255.255.0.0 - применим к ним логическое “И” и узнаем адрес сети (10.9.0.0):



- Обычно объединенную запись адреса и маски подсети пишут следующим образом: **10.9.8.7/16**. В этом случае до слеша указан сам IP адрес, а после слеша - те левые биты маски подсети, которые выставлены в единицу (если 16 - первые шестнадцать бит, которые в десятичном формате могут быть записаны вот так - 255.255.0.0, если 8, то в десятичном формате - 255.0.0.0 и т.д.).

IPv6

- В самом конце главы уделим внимание протоколу IPv6, который, хотя и уступает IPv4 по популярности на данный момент, в ближайшем будущем станет доминирующей технологией в сетях. Адрес IPv6 был утвержден еще в 1998 году в ответ на угрозу исчерпания публичных адресов версии IPv4. Очевидное отличие - IPv6 предлагает огромный диапазон адресов - 2^{128}. Это достигается тем, что новый адрес имеет увеличенный размер - 128 бит (16 байт). В связи с этим был изменен формат записи адреса - отображается в виде 8 шестнадцатеричных чисел, разделенных двоеточием (каждое число не превышает 2 байта - 16 бит):

```
2001:0DB8:3C4D:7777:0260:3EFF:FE15:9501
```

- Также IPv6 привносит качественные изменения. Во-первых, здесь не применяют широковещательные запросы ко всем устройствам к сети - вместо этого используются адреса мультикаста. Они начинаются с FF, отправляются только тем устройствам, которые готовы их получать (технология MLD - Multicast Listener Discovery). Во-вторых, уменьшается значение DHCP сервера - устройства сами умеют устанавливать себе адреса в локальной сети (технология SLAAC - Stateless Address Autoconfiguration). В-третьих, IPv6 по умолчанию включает в себя шифрование, аутентификацию и прочие важные аспекты безопасности с помощью технологии IPsec (Internet Protocol Security).

Стеки протоколов OSI и TCP/IP

Ведение

- Мы уже знаем, как устройства объединяются в сети и как они взаимодействуют внутри сети и за ее пределами. Кроме того, автор этих строк надеется, что стало понятно, как происходит идентификация и маршрутизация устройств. Однако все это было рассмотрено в контексте именно сетей. Теперь пришло время рассмотреть, как это происходит на уровне устройств - как возникают данные, как они упаковываются, и какие методологии для этого используются.

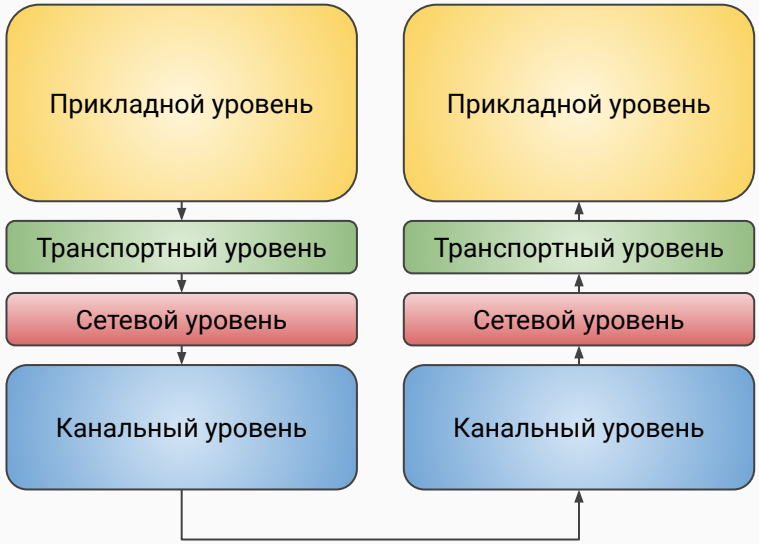
- Первая попытка формализовать передачу данных между устройствами была предпринята в конце 1970-х годов с созданием абстрактной модели OSI (Open Systems Interconnection). Модель описывает 7 иерархических уровней, через которые данные проходят перед тем, как попасть в сеть. Были подробно расписаны обязанности каждого уровня и принципы их взаимодействия (только на один уровень вверх или вниз).

- Модель OSI охватывает все подробности сетевого взаимодействия, но она не определяет конкретные протоколы. Для практического применения чаще используется модель TCP/IP, которая описывает 4 уровня и базируется на конкретных протоколах. Модель TCP/IP была представлена в 1974 году и основана на протоколах IP (Internet Protocol) и TCP (Transmission Control Protocol).

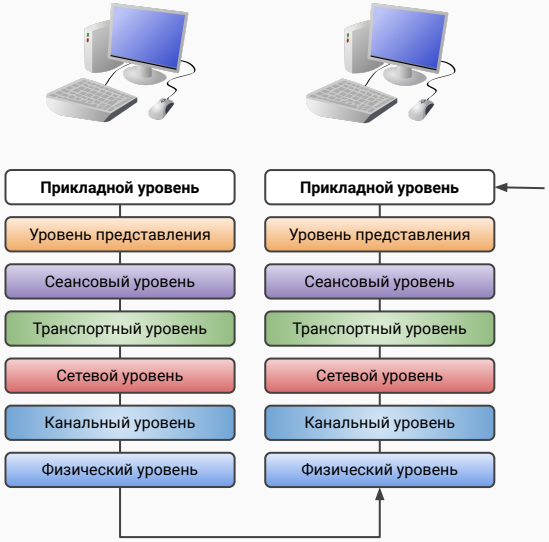
Визуальное представление моделей OSI и TCP/IP

Модель OSI

Модель TCP/IP



Прикладной уровень




```
POST /login HTTP/2.0
Host: bank.com
Content-Type: multipart/form-data
Content-Length: 30

username=Vasya&password=secret
```

- **Прикладной уровень** - это начальная точка для данных, которые зарождаются здесь. Здесь находятся прикладные программы и службы, созданные для удовлетворения потребностей пользователей - т.е. именно с этим уровнем пользователь взаимодействует напрямую. Примерами устройств уровня являются веб-браузеры, почтовые клиенты и серверы, где данные и запросы пользователя начинают свой путь. Протоколы, работающие на этом уровне, включают HTTP для веб-сервисов, FTP для передачи файлов и SMTP для электронной почты.

- Приведем пример работы с HTTP протоколом. Человек заполняет форму (она ссылается на ***https://bank.com/login***) на сайте и нажимает кнопку “Подтвердить”. Браузер формирует текст, представленный слева и отправляет его нижележащим уровням. После прохождения запроса по всем уровням на устройстве человека, по сети, а потом по уровням устройства сайта (но уже в обратном порядке), сайт наконец получит те данные, которые отправил человек (и в том же формате).

Уровень представления

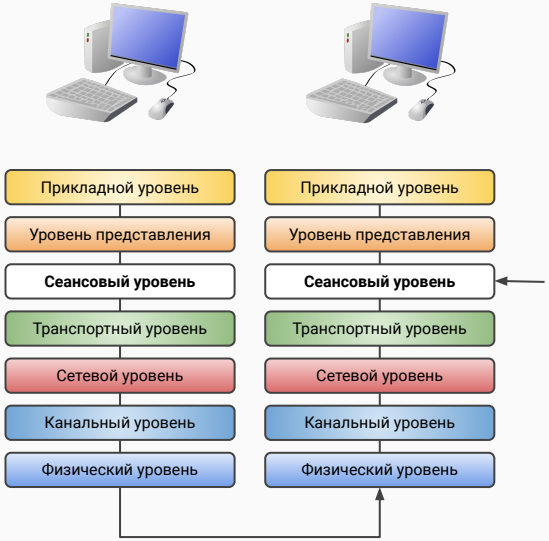

```
POST /login HTTP/2.0
Host: bank.com
Content-Encoding: gzip
Content-Type: multipart/form-data
Content-Length: 16

some-binary-data
```

- **Уровень представления** - теоретически этот уровень задумывался для обработки и преобразования данных так, чтобы они стали совместимыми для передачи через сеть. Здесь должно было происходить кодирование, сжатие данных и шифрование. Целью протоколов данного уровня было гарантировать, что данные могли быть правильно интерпретированы на другой стороне.

- На практике функции этого уровня переданы другим уровням. Например, сжатием данных занимается прикладной протокол HTTP. В процессе запроса протокол может может сжать данные и установить заголовок Content-Encoding, где будет указан алгоритм сжатия. Также допускается заголовок Accept-Encoding - он показывает разрешенные алгоритмы сжатия. В примере слева показан один из вариантов такого запроса. Функции шифрования обычно передаются протоколу TLS, который часто рассматривается как “протокол без точного уровня в иерархии протоколов” или как протокол транспортного уровня - его мы рассмотрим отдельно позднее.

Сеансовый уровень



- **Сеансовый уровень** - обеспечивает согласование и синхронизацию обмена данными между устройствами, а также обеспечивает восстановление сеансов связи между устройствами в случае сбоев в сети. Кроме того, этот уровень определяет порядок обмена данными между устройствами и контролирует, кто в данный момент имеет право передачи.

- В контексте модели TCP/IP функции сеансового уровня как правило принадлежат транспортному уровню, а именно протоколу TCP. Если нам необходим пример протокола конкретно сеансового уровня модели OSI, то одним из самых ярких представителей несомненно является X.255. Он занимается тем, что пытается восстановить соединение, если оно было потеряно. Также он наблюдает за тем, как давно было использовано соединение - если оно не использовалось длительное время, протокол может принять решение переподключиться.

Транспортный уровень

- **Транспортный уровень** - выполняет несколько функций. Во-первых, это сегментация данных (протоколы разделяют данные на основе пропускной способности сети). Во-вторых, это идентификация приложений с помощью портов (приложение на компьютере может быть идентифицировано номером от 1 до 65535 - этот номер и называется портом), ведь часто запрос приходит не для всего компьютера, а для программы на нем. В-третьих, транспортный уровень может устанавливать специальное соединение, чтобы убедиться, что получатель существует и готов принимать данные.

- Основные протоколы транспортной сети это TCP и UDP. Первых из них, TCP, отвечает за надежную передачу данных - он делит данные на сегменты (присваивая каждому сегменту свой код) и следит за тем, чтобы данные пришли в целости. UDP не столь надежен - он разбивает данные на датаграммы и отправляет их без подтверждения доставки. Каждый транспортный протокол добавляет данным свой заголовок, содержащий, например, порты отправителя и получателя.

Сетевой уровень

- **Сетевой уровень** - предназначен для определения оптимального пути передачи данных от источника к адресату. Это включает в себя принятие решений о том, через какие узлы и маршруты данные должны проходить. Роутеры (маршрутизаторы) работают на сетевом уровне и принимают эти решения на основе информации о текущей нагрузке и других факторов. Также этот уровень еще больше фрагментирует сегменты или датаграммы, присланные транспортным уровнем (если на данном участке сети малая пропускная способность) - эти части оборачиваются в т.н. пакеты, заголовок которых содержит IP-адрес отправителя и IP-адрес получателя.

- Примерами протоколов сетевого уровня являются уже многократно упомянутые IPv4 и IPv6, которые отвечают за адресацию и маршрутизацию в сетях. Каждый подвид протокола имеет свою четкую структуру (32 и 128 бит соответственно), а также маску подсети, которая позволяет узнать отдельно адрес сети и отдельно - адрес устройства.

Канальный уровень

- **Канальный уровень** - основная обязанность - обеспечение соединения на уровне сетевых карт. Оборачивает блоки данных верхнего уровня (пакеты с IP адресами) в так называемые кадры, которым внутри заголовков указывает MAC-адрес отправителя и MAC-адрес получателя, а также добавляет контрольную сумму, которая позволяет проверить, не повредился ли кадр при передаче. Канальный уровень практически всегда работает только в контексте локальной сети - в другие сети доступа не имеет.

- Примером протокола канального уровня можно назвать Ethernet, который отвечает за передачу данных между устройствами с помощью MAC-адреса. Другой пример - протокол ARP, который позволяет узнать MAC-адрес другого устройства с помощью его IP-адреса.

Физический уровень

- **Физический уровень** - это самый нижний уровень в иерархии уровней OSI (и он также входит в стек протоколов TCP/IP как часть канального уровня). На этом уровне осуществляется преобразование данных (битов) в физические сигналы, которые затем передаются по физической среде, такой как провода, оптоволокно или радиоволны. Физический уровень не инициирует распространение сигналов, но отвечает за их передачу. Главной задачей физического уровня является обеспечение физической передачи данных через выбранную среду, а также обеспечение надежности передачи данных

- Данный уровень не имеет протоколов в том понимании, какое использовалось предыдущими уровнями, т.к. он работает не в логической среде, а в физическом мире.

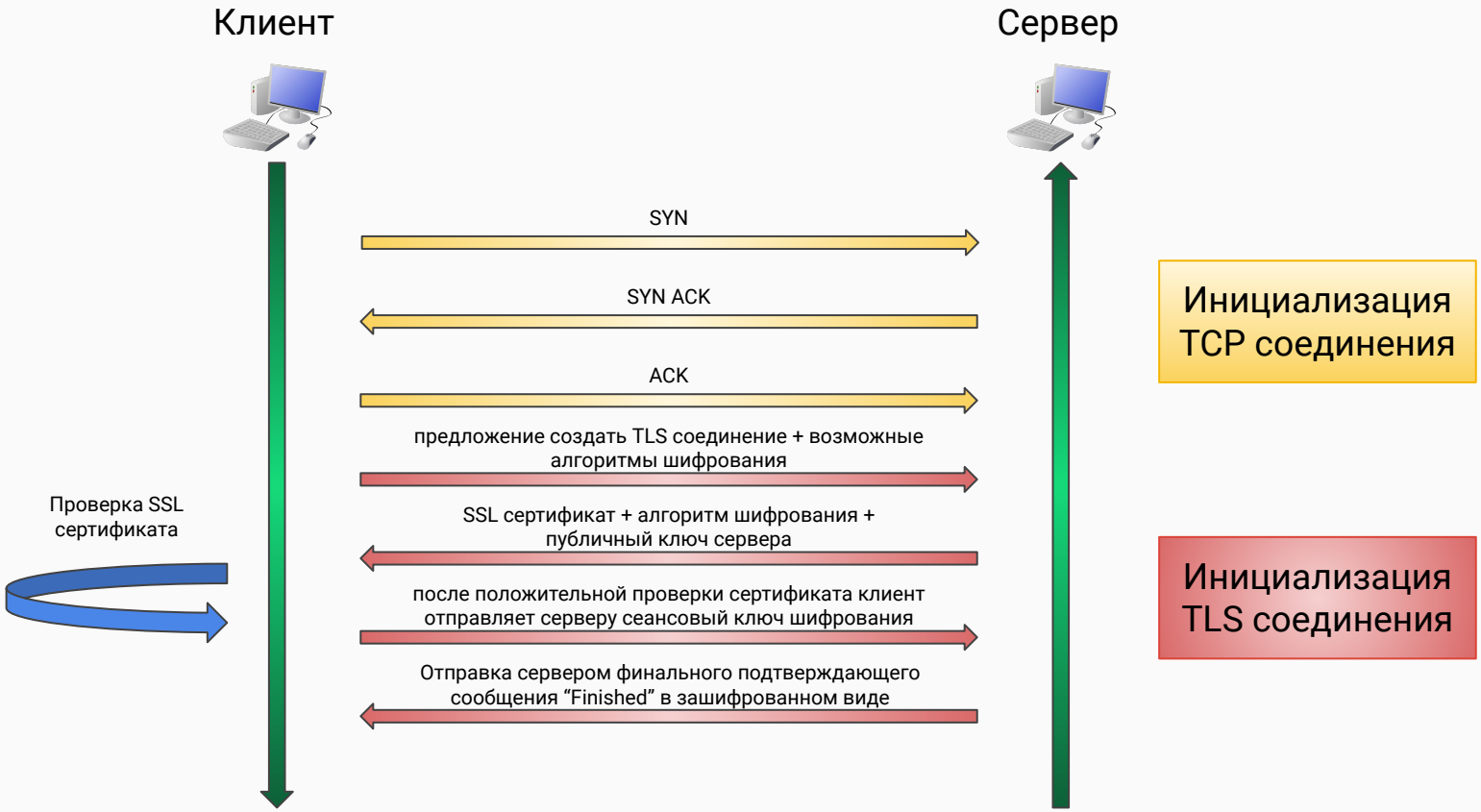
TLS протокол (введение)

- Особое место среди всех протоколов занимает TLS (ранее его место занимал SSL, но он устарел, поэтому рассматривать его мы не будем) - Transport Layer Security. Точное место TLS в уровнях иерархии не может быть точно установлено, но обычно его приписывают к транспортному уровню. Его основная задача - защищенная передача данных с гарантированной идентификацией адресата.

- В основе работы протокола TLS лежат два основных принципа. Во-первых, это специальный сертификат сервера, который клиентская программа (например, браузер) может автоматически проверить в некой авторитетной организации (браузер содержит предустановленный список корневых центров проверки). Во-вторых, это шифрование соединения между сервером и клиентом для предотвращения внедрения в запросы и ответы, подмены значения и т.д.

- Приведем упрощенную схему защищенного соединения. При начальном запросе клиент уведомляет сервер, что хочет установить TLS соединение и присылает список возможных алгоритмов шифрования. Сервер выбирает один из алгоритмов и одновременно присылает свой сертификат и ключ, которым можно шифровать сообщения. Клиент проверяет сертификат и, если все хорошо, начинается обмен данными в зашифрованном виде.

TLS протокол (визуальное представление работы)

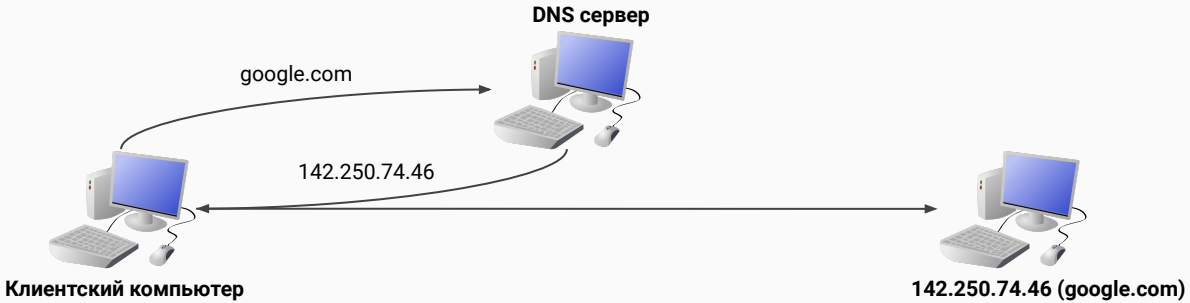


Особенности работы DNS

Доменное имя и IP-адрес (использование DNS)

- Теперь немного отойдем от сложных вещей и поговорим о том, как компьютеры общаются друг с другом в разных сетях казалось бы без помощи IP-адресов. Мы много раз видели, что при обращении к некому сайту мы вводим что-то вроде https://google.com или https://wikipedia.org - в данном случае мы не указываем никакого IP-адреса, но все равно получаем доступ к нужному нам содержимому.

- Дело в том, что на самом деле каждый сайт в сети Интернет обязан иметь публичный IP-адрес, к которому мы и обращаемся. Когда мы делаем запрос к https://google.com, наше устройство сначала делает запрос к специальным узлам в сети Интернет, которые называются DNS серверами. Какой-либо из этих узлов знает, что нашему запросу соответствует, например, адрес **142.250.74.46** и возвращает этот адрес нам. Потом наше устройство запоминает этот IP-адрес и при передаче данных по сети подменяет **google.com** именно на него.



DNS записи (введение)

- Как мы выяснили, протокол DNS и DNS сервера помогают преобразовывать красивые доменные имена сайтов в относительно сложные IP-адреса для маршрутизации в сети. При всем при этом у нас должен возникнуть вопрос - а каким образом DNS сервера знают об этих парах (имя - IP-адрес)?

- Дело в том, что эти пары может установить владелец имени сайта (также называется доменным именем) после покупки этого имени. Обычно это можно сделать на сайте той организации, где произошла покупка (в Латвии - сайт https://nic.lv). Однако мы можем не только устанавливать IP - также есть возможность установить параметры для электронной почты, перенаправлений и многого другого. Все эти параметры называются DNS записями - далее мы рассмотрим самые популярные из них.

Основные типы DNS записей

- **A (Address Record)** - используется для связи доменного имени с IPv4-адресом сервера сайта. Например, google.com - 142.250.74.46
- **AAAA (IPv6 Address Record)** - применяется для соединения имени с IPv6-адресом сервера сайта. Например, google.com - 0:0:0:0:0:ffff:8efa:4a2e
- **CNAME (Canonical Name Record)** - создает псевдонимы для доступного существующего доменного имени сайта. Например, google.com - www.google.com
- **MX (Mail Exchanger Record)** - указывает адрес серверов, которые принимают почту для домена. Например, google.com - mail.google.com
- **TXT (Text Record)** - может хранить произвольные описания сайта. Например, google.com - MS=E4A68B9AB2BB9670BCE15412F62916164C0B20BB
- **SRV (Service Record)** - в числе прочего позволяет указать порт, приоритет и уже существующее доменное имя сайта для обрабатываемого доменного имени.
- **NS (Name Server Record)** - устанавливает название авторитетных DNS серверов, которые будут отдавать первичные данные о DNS записях доменного имени. По сути, те узлы в сети Интернет, где точно можно узнать A, AAAA, MX и другие записи сайта.

Базовые инструменты для работы с сетью: ping, wget и curl

Команда **ping** - проверка существования узла в сети

- Чтобы начать работать с неким узлом в глобальной сети, нам сперва может потребоваться проверить его существование. Наилучшая командой для этого является ping, которая посылает пакеты определенного размера на указанное доменное имя или IP-адрес. Если мы запустим команду без дополнительных параметров, то она будет выполняться бесконечно - для остановки надо нажать **Ctrl C**. В результате мы получим информацию о том, сколько пакетов дошло успешно, а также минимальное, максимальное и среднее время ответа:

```
ping google.com
```

- Иногда мы не хотим бесконечно выполнять команду ping - чаще все это потребуется сделать конкретное число раз. Для этого можно применить параметр **-c**, после которого следует написать число повторений:

```
ping -c 4 8.8.4.4
```

- Еще один параметр **-f** позволяет произвести массовую отправку пакетов для нагрузочного тестирования (доступен только суперпользователю):

```
sudo ping -f localhost
```

Команда **wget** - загрузка файлов из сети

- Загрузка файлов из сети - одна из самых насущных потребностей в современном мире, которая не требует обоснования. Для загрузки файлы в операционных системах на ядре Linux традиционно используется команда **wget**, после которой следует указать ссылку на файл - загрузка начнется немедленно. Например, если мы с помощью **wget** обратимся по ссылке https://www.google.com/favicon.ico то мы получим в рабочей директории файл ***favicon.ico***:

```
wget https://google.com/favicon.ico
```

- Также **wget** позволяет скачать несколько файлов одновременно - для этого после самой команды надо перечислить все ссылки на файлы через пробел:

```
wget https://google.com/favicon.ico https://en.wikipedia.org/favicon.ico
```

- Иногда мы заходим получить файл не с оригинальным названием, а с каким-то другим - для этого используется параметр -O после которого следует написать новое название, а затем - ссылку на файл:

```
wget -O google.ico https://google.com/favicon.ico
```

Команда **wget** - настройка загрузки файлов

- Если в процессе загрузки файла произошел сбой, то у нас есть способ не начинать скачивать файл заново, а возобновить его загрузку с того момента, как у нас возникла ошибка. Для этого используется параметр **-c**:

```
wget -c https://releases.ubuntu.com/22.04.3/ubuntu-22.04.3-live-server-amd64.iso
```

- Находясь на сервере, который работает под большой нагрузкой, не следует пользоваться максимально возможной скоростью загрузки - ее надо ограничивать. Применим для этого параметр **--limit-rate**, после которого через равно надо написать максимальную скорость закачки, например, 100K:

```
wget --limit-rate=100K https://releases.ubuntu.com/22.04.3/ubuntu-22.04.3-live-server-amd64.iso
```

- Часто может потребоваться загрузить файл не в рабочую директорию, а по некоторому другому пути. Для этого следует подставить параметр **-P**, после которого указывается директория (уже существующая) для файла:

```
wget -P /home/user https://linux.org/favicon.ico
```

Команда **wget** - загрузка сайтов из сети

- Команда **wget** обладает дополнительными возможностями - с ее помощью можно загружать не только отдельные файлы, но и целые сайты. Если мы применим параметр **--mirror** и укажем ссылку к начальной странице некого сайта, то все его страницы будут загружены автоматически (с соблюдением файлов структуры и структуры каталогов). Для пример загрузим сайт, где хранятся ядра Linux - **https://www.kernel.org**:

```
wget --mirror https://www.kernel.org
```

- Предыдущая команда успешно загрузит сайт, но есть один нюанс - ссылки на пункты меню, стили, файлы JavaScript и картинки по прежнему будут указывать на изначальный сайт. Чтобы решить эту проблему, следует применить параметр **-p** (дополнительно загружает все стили, файлы JavaScript и картинки) и параметр **--convert-links** (автоматически будет преобразовывать ссылки сайта на ссылки внутри файловой структуры операционной системы):

```
wget --mirror -p --convert-links https://www.kernel.org
```

Команда **curl** - осуществление запросов по сети

- Команда **curl** очень похожа на **wget**, но включает в себя большее количество возможностей, т.к. она способна на практически любые типы сетевых запросов прикладного уровня с использованием протоколов HTTP, HTTPS, FTP, SMTP и многих других. По умолчанию **curl** не устанавливается на Ubuntu, поэтому нам следует проинсталлировать эту утилиту вручную (если она уже не установлена):

```
sudo apt install -y curl
```

- Базовый синтаксис команды **curl** элементарен - сначала идет сама команда, а затем послее нее указывается та ссылка, на которую следует отправить запрос. В результате в командной строке (**stdout**) отобразится тело самого запроса:

```
curl https://example.com
```

- Если нам необходимо получить не тело запроса, а его заголовки, то следует применить параметр **-I**. Если же нам надо получить как заголовки, так и само тело, то применяется параметр **-i**. Узнаем заголовки сайта **example.com**:

```
curl -I https://example.com
```

Команда **curl** - типы запросов

- По умолчанию **curl** отправляет запрос при помощи протокола HTTP и метода ***GET***, который предназначен просто для получения данных. Однако иногда нам надо отправить данные на сервер, притом данные эти не должны отображаться в адресной строке. Для этого нужен параметр **-d**, после которого в кавычках расписывается тело запроса - сам запрос в этом случае автоматически осуществляется методом ***POST***. Произведем запрос с данными на сайт ***reqres.in*** с целью создания тестового пользователя ***john***, с профессией ***god***:

```
curl -d '{"name": "john","job": "god"}' https://reqres.in/api/users
```

- Очень часто при обращении к различным API будет необходимо использовать другие типы запросов - ***PATCH***, ***PUT***, ***DELETE*** и другие. Для этого применяется параметр -X, после которого указывается непосредственно тип данных. Обновим на сайте ***reqres.in*** пользователя с id = 3 именем ***mary*** и профессией ***programmer*** с помощью метода ***PATCH***:

```
curl -X PATCH -d '{"name": "mary","job": "programmer"}' https://reqres.in/api/users/3
```

Команда **curl** - заголовки запросов

- Кроме самой ссылки и тела запроса, мы также можем передать заголовки запроса. Это осуществляется с помощью параметра **-H**, после которого в кавычках следует указывать название заголовка и его значение:

```
curl -H 'X-Api-Code: 12345678' https://httpbin.org/headers
```

- Также поддерживается отправка нескольких заголовков одновременно - просто надо повторить конструкцию с **-H** и заголовком несколько раз:

```
curl -H 'X-Api-Code: 12345678' -H 'Content-Type: application/json' https://httpbin.org/headers
```

Команда **curl** - загрузка файлов в сеть

- Одна из мощнейших функций **curl** связана с возможностью загрузить файлы в сеть. Сама по себе загрузка практически ничем не отличается от других запросов, единственные два отличия - перед названием параметра файла в теле запроса необходимо поставить знак @, кроме того - следует использовать параметр **-F**, который указывает, что тип содержимого равен **multipart/form-data** (т.е. файл). Приведем пример загрузки файла:

```
curl -X POST -F 'filename=@/path/to/file.txt' https://example.com
```

- Также есть возможность загрузить несколько файлов одновременно - в этом случае следует повторить конструкцию с параметром -F и телом запроса соответствующее число раз:

```
curl -X POST -F 'filename=@/path/to/file.txt' -F 'filename2=@/path/to/file2.txt' https://example.com
```

Команда **curl** - загрузка файлов из сети

- После загрузки файлов в сеть было бы логично посмотреть как **curl** загружает файлы из сети. Это производится при помощи параметра **-O**. В результате мы получим файл с оригинальным названием:

```
curl -O https://www.google.com/favicon.ico
```

- Если мы хотим получить файл с названием, отличным от оригинального, нам следует применить параметр **-o**, после которого следует указать новое название:

```
curl -o fav-renamed.ico https://www.google.com/favicon.ico
```

- В числе прочего **curl** позволяет сказать несколько файлов (каждому надо подставить параметр **-O**):

```
curl -O https://www.google.com/favicon.ico -O https://www.africau.edu/images/default/sample.pdf
```

Команда **curl** - аутентификация

- Часто тот узел, которые необходим для доступа команде **curl**, защищен паролем - наиболее распространенным вариантом является т.н. базовая аутентификация. Для прохождения этой аутентификации нам всего навсего следует использовать параметр **-u**, после которого через двоеточие следует указать логин и пароль:

```
curl -u 'username:password' https://example.com
```

- Как правило, после аутентификации часто происходит перенаправление на другую страницу. Чтобы разрешить это перенаправление, необходимо применить параметр **-L**, а для повторного применения параметров аутентификации на всех последующих страницах следует использовать параметр **--location-trusted**:

```
curl -L --location-trusted -u 'username:password' https://example.com
```

- Чтобы в файл сохранить ***cookie***, которые содержат необходимую для аутентификации информацию, следует использовать параметр **-c** и название файла после него:

```
curl -c cookie.txt https://example.com
```

- Для совершения обратного действия - отправки ***cookie*** сайту - следует применить параметр **-b** и имя файла, где эти ***cookie*** хранятся:

```
curl -b cookie.txt https://example.com
```

Анализ сетевого трафика

Команда **traceroute** - трассировка пути запроса

- В сети часто происходят перебои - даже если у нас все замечательно, где то на пути к целевому узлу может произойти нечто, что может помешать нашему запросу достичь финальную цель. Как мы уже знаем, запрос идет не напрямую, а от одного роутера к другому. Чтобы узнать адреса этих роутеров, а также выяснить, какой узел потенциально вышел из строя, используется команда **traceroute**, которую сначало надо установить следующим способом:

```
sudo apt install -y traceroute
```

- Для получения списка узлов по пути к целевому узлу надо ввести команду **traceroute**, а после нее указать доменное имя или IP-адрес нужного узла. Команда начнет отправлять пакеты (обернутые в UDP диаграммы) на адрес узла, но с одним условием - каждому пакету будет присвоено время жизни (TTL) - сначала 1, потом 2 и т.д. Каждый промежуточный узел, получающий пакет со временем жизни равным 0, должен вернуть ответ источнику запроса. Таким образом мы получаем информацию о каждом узле по пути к нашей цели:

```
traceroute google.com
```

Команда **ss** - проверка портов на локальном компьютере

- Когда мы рассматривали транспортный уровень модели OSI, мы упоминали, что программам, которые работают по сети, должны быть присвоены уникальные номера, которые называются портами. Очень часто мы хотим посмотреть список занятых портов, чтобы узнать, какое приложение на каком из них работает. Для этого можно использовать много утилит, но самой удобной является **ss**.

- Сама по себе команда **ss** предлагает довольно сложную информацию, поэтому упомянем важные параметры, которые традиционно применяются в этом контексте. Параметр **-t** - показывает приложения, которые слушают TCP соединения, **-u** - слушатели UDP соединений, **-l** - отобразит только те программы, которые в данный момент не обслуживают соединения, а слушают их, **-p** - отобразит процессы, которые используют соединение, **-a** - показывает одновременно слушающие и активные соединения, **-n** - программа будет пытаться показывать именно порты слушающих служб, а не названия служб:

```
sudo ss -tulpan
```

Команда **nmap** - проверка портов на удаленном компьютере

- Иногда нам может потребоваться проанализировать безопасность сети и выполнить ряд других задач, связанных с обнаружением сетевых ресурсов на удаленных компьютерах. Для этого идеально подходит утилита **nmap**, которая сканирует порты (по умолчанию - 1000 самых популярных) и определяет устройства, которые на этих портах работают. Очевидно, что такие действия часто совершают злоумышленники, поэтому **ns** следует применять на своих компьютерах или сайте ***scanme.nmap.org***. Установим **nmap** следующей командой:

```
sudo apt install -y nmap
```

- В самом базовом случае после команды **nmap** следует указать имя или IP-адрес нужного нам узла. В результате мы получим список портов (***opened*** - готовы принимать соединения, ***filtered*** - не готовы) и работающих на них устройств:

```
nmap scanme.nmap.org
```

- Для определения подробных версий программ используется параметр **-A**, чтобы попытаться узнать операционную систему применяется параметр **-O**, а для того, чтобы просканировать определенные порты, следует применить параметр **-p** и после надо указать (через запятую или через тире) список нужных нам портов:

```
sudo nmap -O -A -p 1-1000 scanme.nmap.org
```

Команда **nslookup** - получение DNS записей

- Для получения DNS данных о каком либо узле в сети можно использовать команду **nslookup** - она закономерно умеет показывать IP-адрес по доменному имени, а также предоставлять дополнительную DNS информацию - MX запись, TXT записи и т.д. В самом простом варианте после названия команды мы должны подставить доменное имя - в результате получим все IP-адреса (A запись и AAAA запись), связанные с этим именем:

```
nslookup google.com
```

- Если мы хотим получить другие виды DNS записей, нам следует применить параметр **-q** после которого через равно надо указать один из типов записей - mx, txt, ns, a, aaaa и т.д. Для примера узнаем сервер электронной почты для ***google.com***:

```
nslookup -q=mx google.com
```

Работа с сетевыми интерфейсами

Введение в сетевые интерфейсы

- Чтобы наш компьютер взаимодействовал с другими устройствами в сети, нам необходимы аппаратные или программные устройства, которое называются сетевыми интерфейсами. К аппаратным устройствам можно причислить непосредственно физическую сетевую карту (Ethernet Adapter) или беспроводной Wi-Fi адаптер. К программным интерфейсам относится, во-первых, т.н. ***loopback***, который позволяет устройству отправлять и принимать данные в контексте самого себя и, во-вторых, сетевые адаптеры для виртуальных машин - данные интерфейсы позволяют этим устройствам работать в сети в качестве условно независимых узлов (тем не менее они используют настоящую сетевую карту).

- С каждым интерфейсом соотносится ряд адресов. Во-первых, это MAC-адрес - уникальный идентификатор сетевой карты (у программных интерфейсов он виртуальный). Во-вторых, это IP-адрес (обеих версий) - используется для идентификации устройства к контексте стека TCP/IP. В-третьих, IP-адрес маршрутизатора (роутера) в локальной сети. В-четвертых, адрес широковещательного запроса (если компьютер впервые попадает в сеть, то для определения роутера и других узлов он может отправить как раз такой запрос, который смогут увидеть вообще все устройства в этой локальной сети).

Команда **ip** - просмотр информации о сетевых интерфейсах

- В Ubuntu для работы с сетевыми интерфейсами используется команда **ip** - она пришла на смену устаревшей **ifconfig**. Данная команда позволяет просматривать информацию об интерфейсах, включать их и выключать, а также назначать им различные адреса (как мы помним, по умолчанию это делает роутер). Часто используемой вариацией команды **ip** является ее применение с параметром **a** (**addr**) - в результате мы увидим сетевые интерфейсы со всеми адресами:

```
ip addr
```

Если мы хотим увидеть информацию по конкретному интерфейсу, то следует дополнить команду параметром **show** и кодом интерфейса (например, ***enp0s3***):

```
ip addr show enp0s3
```

- Для просмотра информации канального уровня применяется параметр **link**:

```
ip link
```

- Для просмотра информации о маршрутизации применяется параметр **route**:

```
ip route
```

Команда **ip** - включение и выключение сетевых интерфейсов

- Мы можем выключить сетевой интерфейс при помощи параметров **link set**, после которых идет название интерфейса (например, ***enp0s3***) и слово **down**:

```
sudo ip link set enp0s3 down
```

- Для включения интерфейса следует применить такую же команду, только вместо **down** следует писать **up**:

```
sudo ip link set enp0s3 up
```

Команда **ip** - добавление и удаление адресов для сетевых интерфейсов

- Для добавления IP-адреса для сетевого интерфейса используется параметр **add** - применим его чтобы добавить адрес ***192.168.1.99/24*** для интерфейса ***enp0s3***:

```
sudo ip addr add 192.168.1.99/24 enp0s3
```

- Для удаления IP-адреса интерфейса используется параметр **del**:

```
sudo ip addr del 192.168.1.99/24 enp0s3
```

- Для удаления всех IP-адресов интерфейса следует применить параметр **flush**:

```
sudo ip addr flush dev enp0s3
```

Команда **ip** - изменение маршрутизаторов для сетевых интерфейсов

- Если нас не устраивает наш маршрутизатор сетевого интерфейса или этот маршрутизатор не может быть обнаружен автоматически. Для решения этой проблемы мы сначала должны удалить старый маршрутизатор (например, **192.168.1.1**), а потом добавить новый (например, **192.168.1.2**):

```
# удаление адреса старого маршрутизатора
sudo ip route del default via 192.168.1.1 dev enp0s3

# добавление адреса нового маршрутизатора
sudo ip route add default via 192.168.1.2 dev enp0s3
```

Команда **netplan** - установка постоянных параметров сетевого интерфейса

- Все предыдущие изменения, которые мы производили с помощью команды **ip**, будут оставаться в силе до первой перезагрузки операционной системы. После этого все вернется к настройкам по умолчанию. Если же мы хотим установить постоянные параметры, нам следует воспользоваться командой **netplan**.

- Команда **netplan** пользуется файлами, которые находятся в директории ***/etc/netplan***. Если мы ходим задать параметры для сетевого интерфейса, то следует создать в этой директории файл в формате yaml с любым названием (например, **config.yaml**). Именно в этом файле и будут прописаны все параметры (примеры будут представлены на следующей странице).

- Для проверки наших настроек следует выполнить команду **sudo netplan try** - она покажет возможные ошибки в нашем конфигурационном файле:

```
sudo netplan try
```

Для окончательного внедрения изменений применяется команда **sudo netplan apply** - новые настройки будут применены немедленно (если они правильные):

```
sudo netplan apply
```

Команда **netplan** - примеры конфигурационных yaml файлов

Пример конфигурации проводного интерфейса

```
network:
  version: 2
  renderer: networkd
  ethernets:
   enp0s3:
     dhcp4: false
     dhcp6: false
     addresses:
       - 192.168.1.111/24
     routes:
       - to: 0.0.0.0/0
         via: 192.168.1.1
     nameservers:
       addresses: [8.8.8.8, 8.8.4.4]
```

Пример конфигурации Wi-Fi интерфейса

```
network:
  version: 2
  renderer: networkd
  wifis:
    wlp0s20f3:
      dhcp4: false
      dhcp6: false
      access-points:
        "название сети":
          password: "пароль"
      addresses:
        - 192.168.1.222/24
      routes:
       - to: 0.0.0.0/0
         via: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

- Иногда программа, которая осуществляет применение настроек (**networkd**), может быть выключена. Для включения надо применить следующую команду:

```
sudo systemctl start systemd-networkd
```

Взаимодействие с DHCP

Команда **dhclient** - запросы на получение локального IP-адреса

- Мы уже знаем, что при подключении устройства к локальной сети роутер (он же маршрутизатор) выдает этому устройству локальный IP-адрес (в данном случае используется протокол DHCP). Мы можем проанализировать этот процесс и даже вмешаться в него с помощью утилиты **dhclient**.

- Для того, чтобы сбросить IP-адрес для сетевого интерфейса, например, ***enp0s3***, следует дополнить команду dhclient параметром **-r**, а затем указать имя интерфейса:

```
sudo dhclient -r enp0s3
```

- Для запроса на получение нового IP-адреса следует применить команду **dhclient** вообще без всяких параметров, но в конце нужно просто указать название сетевого интерфейса:

```
sudo dhclient enp0s3
```

- Если мы хотим получать дополнительную информацию при сбросе и получении IP-адреса, то полезно будет использовать дополнительный параметр **-v**:

```
sudo dhclient -v enp0s3
```

Настройка firewall

Введение

- Чаще всего устройства общаются между собой не с целью обмениваться адресами и обозначать свое присутствие в сети, а для того, чтобы переслать данные от одной программы к другой. В контексте каждого компьютера программы могут иметь свой уникальный номер, называемый портом (число в интервале от 1 до 65535). Именно к этому порту обычно и адресован запрос.

- Однако если мы запустили какое-либо веб-приложение, например, на 5000 порту, а затем попытаемся зайти на него с другого компьютера при помощи адреса локальной сети, то мы убедимся, что приложение недоступно. Здесь нет никакой ошибки, просто по умолчанию операционная система запрещает внешние запросы к портам.

- Чтобы разрешить доступ к портам, можно применить много способов, но самым удобным (в контексте Ubuntu) из них является использование утилиты **ufw** (Uncomplicated Firewall). Если по какой-то причине она отсутствует на вашем компьютере, то ее можно установить следующей командой:

```
sudo apt install -y ufw
```

Команда **ufw** - подготовка к работе

- Чтобы **ufw** начала работать (в этом случае именно **ufw** будет определять что можно, а что нельзя - даже если до этого некоторые порты были включены другими утилитами), надо сначала ее запустить, т.к. по умолчанию она может быть выключена. Для этого нужно применить команду **ufw enable** от имени суперпользователя:

```
sudo ufw enable
```

- Если мы хотим посмотреть, работает ли вообще в данный момент **ufw**, а также увидеть все разрешенные и запрещенные порты, то можно применить параметр **status**:

```
sudo ufw status
```

Команда **ufw** - открытие портов

- Открытие порта для tcp и udp соединение производится при помощи параметра **allow**, после которого надо указать номер открываемого порта (например, 5000):

```
sudo ufw allow 5000
```

- Если мы хотим открыть порт только для одного конкретного IP-адреса, например, для 192.168.1.192, то команда будет выглядеть несколько сложнее:

```
sudo ufw allow from 192.168.1.192 to any port 5000
```

- По умолчанию мы открываем порт как для tcp, так и для udp соединений. Если мы хотим открыть порт только для какого-то конкретного вида транспортного уровня мы после названия порта через слеш должны прописать или tcp или udp:

```
sudo ufw allow 5000/tcp
```

- Открытие порта только для определенного порта и только для определенного типа сетевого уровня также имеет отличие от простого открытия порта:

```
sudo ufw allow from 192.168.1.192 to any port 5000 proto tcp
```

Команда **ufw** - закрытие портов

- Когда мы запускаем **ufw** (**sudo ufw enable**), мы разрешаем на компьютере только те порты, которые прописаны в рамках самого **ufw**. Однако тем не менее мы можем явно запрещать порты в рамках самого **ufw** с помощью параметра **deny**:

```
sudo ufw deny 5000
```

- Гораздо интереснее запрещать доступ к портам с определенных IP-адресов. Запретим доступ для адреса 192.168.1.192:

```
sudo ufw deny from 192.168.1.192 to any port 5000
```

- Также можно запрещать доступ для целых подсетей - для этого после IP-адреса надо через слеш указать маску подсети:

```
sudo ufw deny from 192.168.1.0/24 to any port 5000
```

- В ufw порядок имеет значение, т.к. выполняется первое подходящее правило - для того, чтобы поставить правило на 1 место, надо применить параметр **insert**:

```
sudo ufw insert 1 deny from 192.168.1.0/24 to any port 5000
```

Команда **ufw** - удаление правил

- Для того, чтобы удалить правило, надо знать его порядковый номер. Номера правил можно увидеть если запустить в командной строке следующую команду:

```
sudo ufw status numbered
```

- Затем можно применить параметр **delete**, после которого следует написать номер правила (например, 2):

```
sudo ufw delete 2
```

- Если мы хотим отключить не правила, а выключить сам ufw, то применим команду, представленную ниже:

```
sudo ufw disable
```