## Основы компьютерных сетей.

разница между TCP/IP и моделью OSI.

* TCP/IP - это модель клиент-сервер, т.е. Когда клиент запрашивает сервис, он предоставляется сервером. Принимая во внимание, что OSI является концептуальной моделью.
* TCP/IP - это стандартный протокол, используемый для каждой сети, включая Интернет, тогда как OSI - это не протокол, а эталонная модель, используемая для понимания и проектирования архитектуры системы.
* TCP/IP следует вертикальному подходу. С другой стороны, модель OSI поддерживает горизонтальный подход.
* TCP/IP является материальным, а OSI - нет.
* TCP/IP следует принципу «сверху вниз», а модель OSI - подходу «снизу вверх»

## [Тема №1. Основные сетевые термины и сетевые модели](https://habr.com/ru/articles/307252/)

**Компоненты сети:**

* Оконечные узлы
* Промежуточные устройства
* Сетевые среды

**Для чего мы используем сети:**

* Приложения:

Загрузчики.

Протоколы:

***FTP****-* это стандартный протокол передачи данных с установлением соединения. Работает по протоколу. Стандартный номер порта 21

***TFTP****-* это упрощенная версия протокола FTP, которая работает без установления соединения, по протоколу UDP. Применяется для загрузки образа бездисковыми рабочими станциями.

Интерактивные приложения

Приложения в реальном времени

* Сетевые ресурсы
* Хранилище
* VoIP

**Топология сети**

* физическая
* логическая
* Топология с общей шиной
* Кольцевая топология
* Топология звезда
* Полносвязная топология
* Неполносвязная топология
* Смешанная топология

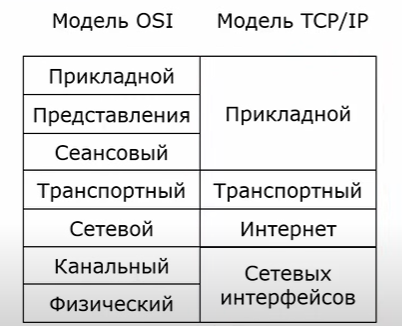
**Модель OSI**

* **Физический уровень (Physical Layer):** метод передачи данных, какая среда используется. **Ethernet**
* **Канальный уровень (Data Link Layer):** **MAC**-адреса и протокол «**Ethernet**», **ARP**
* **Сетевой уровень (Network Layer):** маршрутизация
* **Транспортный уровень (Transport Layer):** протоколы **TCP** (с установлением соединения) и **UDP** (без установления соединения)
* **Сеансовый уровень (Session Layer):** установление, управление и разрыв соединения между двумя хостами
* **Уровень представления (Presentation Layer):**структурирует информацию в читабельный вид для прикладного уровня. [**SSL**](https://ru.wikipedia.org/wiki/SSL)**,** [**TLS**](https://ru.wikipedia.org/wiki/TLS)
* **Прикладной уровень (Application Layer):**e-mail, браузеры по протоколу **HTTP**, **FTP** и остальное

[**Основные протоколы TCP/IP по уровням модели OSI**](https://ru.wikipedia.org/wiki/Шаблон:IPstack)

**Стек протоколов TCP/IP (**DoD (Department of Defence)**)**

* Прикладной
* Транспортный
* Интернет
* Уровень сетевого доступа



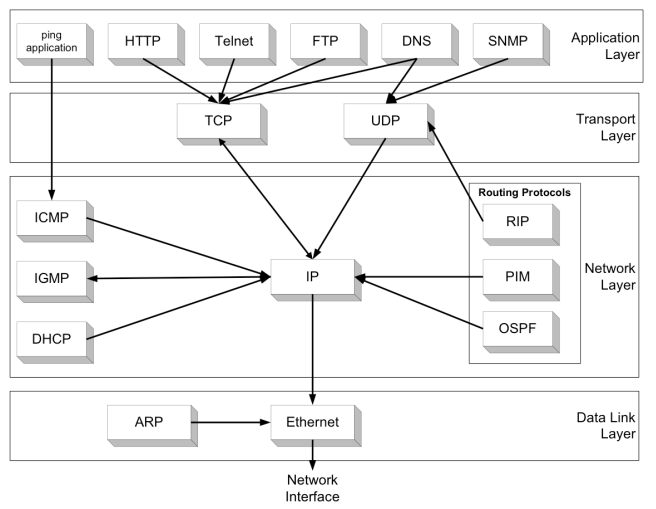
## [Тема №2. Протоколы верхнего уровня](https://habr.com/ru/articles/307714/)

**Протокол HTTP (англ. HyperText Transport Protocol)**

**URL (англ. Uniform Resource Locator)**

**CLI (Command Line Interface)**

* **HTTPS** (HyperText Transfer Protocol Secure) **443**
* **DNS** (Domain Name System) **53**
* **DHCP** (Dynamic Host Configuration Protocol)
* **POP3** (англ. Post Office Protocol Version 3) port **110/995(**POP3S)
* **IMAP** (англ. Internet Message Access Protocol) port **143**
* **SMTP** (англ. Simple Mail Transfer Protocol) port **25**
* **Telnet** (от англ. terminal network) port **23**
* **SSH** (англ. Secure Shell) port **22**
* **FTP** (англ. File Transfer Protocol) port **21**
* **TFTP** (англ. Trivial File Transfer Protocol)



## [Основы компьютерных сетей. Тема №3. Протоколы нижних уровней (транспортного, сетевого и канального)](https://habr.com/ru/articles/308636/)

**Физический уровень**

**Канальный уровень**

**Ethernet-кадр:**

* Преамбула
* MAC-адрес получателя
* MAC-адрес отправителя
* Тип (длина)
* Поле SNAP/LLC + данные
* FCS (от англ. Frame Check Sequence — контрольная сумма кадра)



**Протокол ARP:**

* Тип протокола канального уровня (Hardware type)
* Тип протокола сетевого уровня (Protocol type)
* Длина физического адреса в байтах (Hardware length)
* Длина логического адреса в байтах (Protocol length)
* Код операции (Operation)
* Физический адрес отправителя (Sender hardware address)
* Логический адрес отправителя (Sender protocol address)
* Физический адрес получателя (Target hardware address)
* Логический адрес получателя (Target protocol address)

Прежде, чем подключиться к одному из устройств, IP-протокол проверяет, есть ли в его [ARP](https://habr.com/ru/articles/80364/)-таблице маршрутизации запись о соответствующем устройстве. Если такая запись имеется, то происходит непосредственно подключение и передача пакетов. Если же нет, то посылается широковещательный ARP-запрос, который выясняет, какому из устройств принадлежит IP-адрес. Идентифицировав себя, устройство посылает в ответ свой MAC-адрес, а в ARP-таблицу отправителя заносится соответствующая запись.

1. Если адреса находятся в одной подсети, то вызывается протокол ARP и определяется физический адрес получателя, после чего IP-пакет инкапсулируется в кадр канального уровня и отправляется по указанному физическому адресу, соответствующему IP-адресу назначения.
2. Если нет – начинается просмотр таблицы в поисках прямого маршрута.
3. Если маршрут найден, то вызывается протокол ARP и определяется физический адрес соответствующего маршрутизатора, после чего пакет инкапсулируется в кадр канального уровня и отправляется по указанному физическому адресу.
4. В противном случае, вызывается протокол ARP и определяется физический адрес маршрутизатора по умолчанию, после чего пакет инкапсулируется в кадр канального уровня и отправляется по указанному физическому адресу.

**Сетевой уровень**

**IP (от англ. Internet Protocol)**

* **IPv4** (32-битные адреса)
* **IPv6** (128-битные адреса)

**IP-пакет:**

* Версия
* IHL (от англ. Internet Header Length — размер заголовка)
* Тип обслуживания
* Длина пакета
* Время жизни (TTL- Time To Live)
* Протокол
* Контрольная сумма заголовка
* Опции
* Смещение
* Данные
* Идентификатор
* Флаги
* Смещение фрагмента
* IP-адрес отправителя и IP-адрес получателя

**Транспортный уровень**

**UDP (**DHCP, TFTP**):**

* Порт источника
* Порт назначения
* Длина UDP
* Контрольная сумма UDP
* Данные

**TCP:**

* Порт источника и порт назначения
* Порядковый номер
* Номер подтверждения
* Длина заголовка
* Зарезервированный флаг
* Флаги
* Размер окна
* Контрольная сумма TCP
* Указатель важности
* Опции
* Данные

Установление TCP-сессии. Этот процесс называют «трехстороннее рукопожатие» или «**handshake**». В чем суть. Клиент отправляет TCP-сегмент с флагом «**SYN**». Получив сегмент, сервер принимает решение. Если он согласен установить соединение, то он **создает сокет для соединения Портов 65535** и отправляет ответный сегмент с флагом **«SYN+ACK».** Если не согласен, то отправляет сегмент с флагом «**RST**». Далее клиент смотрит на ответный сегмент. Если там стоит флаг «**SYN+ACK**», то он в ответ отправляет сегмент с флагом «**ACK**» и устанавливается соединение. Если же там стоит флаг «RST», то он прекращает попытки соединения. После того, как потребуется разорвать установившееся соединение, клиент формирует и отправляет TCP-сегмент с флагом «**FIN+ACK**». Сервер на этот сегмент отвечает аналогичным флагом «**FIN+ACK**». И наконец, клиент отправляет последний TCP-сегмент с флагом «**ACK**»

Количество **портов** ограничено с учётом 16-битной адресации (216=**65536**, начало — «0»). Все порты разделены на три диапазона — общеизвестные (или системные, **0—1023**), зарегистрированные (или пользовательские, **1024—49151**) и динамические (или частные, **49152—65535**).

**SYN cookie** - Cпособ противостоять **SYN**-флуду путем особого реагирования сервера [TCP](https://stormwall.pro/knowledge-base/protocol/tcp) на запрос клиента в ходе установки соединения с ним. Может снизить нагрузку от [SYN-атаки](https://stormwall.pro/knowledge-base/attack/syn-attack), однако в настоящее время не слишком эффективен против реальных **SYN**-атак. В ответ на клиентский SYN пакет мы НЕ сохраняем никакой информации о соединении, а пакуем ее в ответный **SYN+ACK** пакет и отправляем клиенту. И лишь в случае, когда клиент ответит нам своим **ACK** передав эту информацию - мы ее проверим, распакуем информацию, содержащуюся в нем, и создадим запись о таком соединении и откроем обмен данными.

## [Основы компьютерных сетей. Тема №4. Сетевые устройства и виды применяемых кабелей](https://habr.com/ru/articles/312340/)

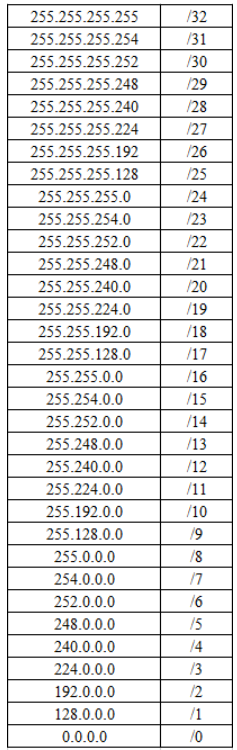
* Cетевая карта
* Повторитель (repeater)
* Концентратор (hub)
* Мост (bridge)
* Коммутатор
* Маршрутизатор (router)
* Dial-up модем



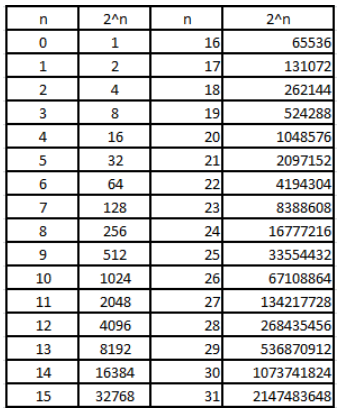
Принципиальное различие состоит в том, что коммутация реализована на втором уровне модели OSI, а маршрутизация - на третьем. Такое принципиальное отличие означает, что маршрутизация и коммутация используют разную информацию для организации передачи данных от отправителя получателю.

Наиболее существенное различие между маршрутизаторами и коммутаторами заключается в том, что маршрутизаторы соединяют сети, а коммутаторы соединяют устройства внутри сети. Маршрутизаторы могут подключаться к Интернету и локальным устройствам, в то время как коммутаторы подключают локальные устройства.

## [Основы компьютерных сетей. Тема №5. Понятие IP адресации, масок подсетей и их расчет](https://habr.com/ru/articles/314484/)







**N + S + H = 32**, где **N** — кол-во битов сети (класс A — 8 бит, B — 16 бит, C — 24 бита), **S** — кол-во заимствованных битов на подсеть (это то, что мы делали выше, когда заимствовали 1 бит из хостовой части), **H** — кол-во бит отводимых хостам.

 «Количество IP-адресов в сети?» и «Сколько доступных IP-адресов в сети для назначения хостам?». Два разных вопроса, которые могут поставить в тупик. Ответом на первый будет — все адреса, включая адрес сети и широковещательный адрес, а на второй вопрос — все адреса, кроме адреса сети и широковещательного адреса.

[Основы компьютерных сетей. Тема №6. Понятие VLAN, Trunk и протоколы VTP и DTP](https://habr.com/ru/articles/319080/)

**Тегированный Ethernet-кадр**

**Тег:**

* TPID (англ. Tag Protocol ID) или Идентификатор тегированного протокола
* PCP (англ. Priority Code Point) или значение приоритета
* CFI (англ. Canonical Format Indicator) или индикатор каноничного формата
* VID (англ. VLAN ID) или идентификатор VLAN
* **Access port или порт доступа**— порт, находящийся в определенном VLAN и передающий не тегированные кадры. Как правило, это порт, смотрящий на пользовательское устройство.
* **Trunk port или магистральный порт** — порт, передающий тегированный трафик. Как правило, этот порт поднимается между сетевыми устройствами.

**DTP (англ. Dynamic Trunking Protocol)**

**VTP (англ. VLAN Trunking Protocol)**

## [Основы компьютерных сетей. Тема №7. Протокол связующего дерева: STP](https://habr.com/ru/articles/321132/)

## [Основы компьютерных сетей. Тема №8. Протокол агрегирования каналов: Etherchannel](https://habr.com/ru/articles/334778/)

## [Основы компьютерных сетей. Тема №9. Маршрутизация: статическая и динамическая на примере RIP, OSPF и EIGRP](https://habr.com/ru/articles/335090/)

* Routing protocols (протоколы маршрутизации)
* Routed protocols (маршрутизируемые протоколы) (IPv4, IPv6)

**Протоколы динамической маршрутизации:**

* IGP (interior gateway protocols) — внутренние протоколы маршрутизации (RIP, **OSPF**, EIGRP). Внутри одной сети (компании)
* EGP (external gateway protocols) — внешние протоколы маршрутизации (на сегодня **BGP**).

OSPF используется для определения самого быстрого маршрута, в то время как BGP делает акцент на определении лучшего маршрута

## [NAT](https://habr.com/ru/articles/583172/)

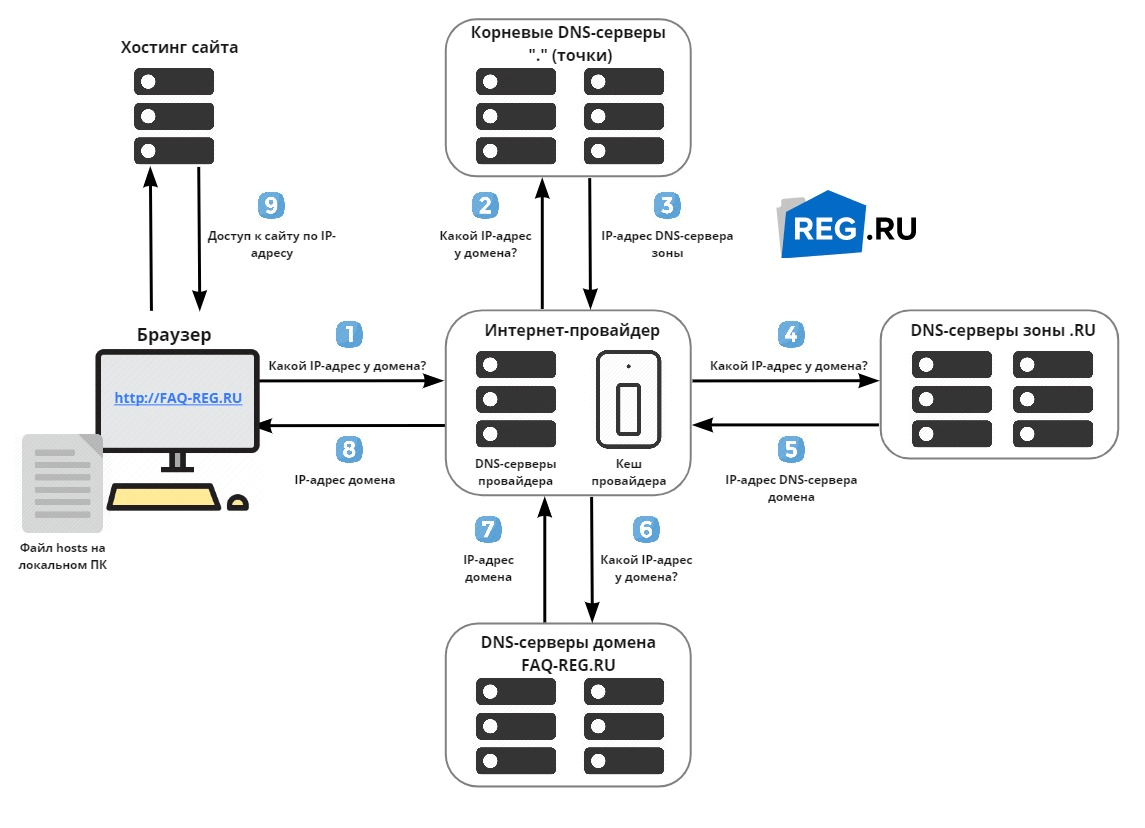
* это механизм в [сетях](https://ru.wikipedia.org/wiki/Компьютерная_сеть) [TCP/IP](https://ru.wikipedia.org/wiki/Стек_протоколов_TCP/IP), позволяющий преобразовывать [IP-адреса](https://ru.wikipedia.org/wiki/IP-адрес) транзитных [пакетов](https://ru.wikipedia.org/wiki/Пакет_(сетевые_технологии)). Также имеет названия *IP Masquerading*, *Network Masquerading* и *Native Address Translation*.
* при
* **Статический NAT** — отображение незарегистрированного IP-адреса на зарегистрированный IP-адрес на основании один к одному. Особенно полезно, когда устройство должно быть доступным снаружи сети.
* **Динамический NAT** — отображает незарегистрированный IP-адрес на зарегистрированный адрес из группы зарегистрированных IP-адресов. Динамический NAT также устанавливает непосредственное отображение между незарегистрированными и зарегистрированными адресами, но отображение может меняться в зависимости от зарегистрированного адреса, доступного в пуле адресов, во время коммуникации.
* **Перегруженный NAT** (NAPT, NAT Overload, PAT, маскарадинг) — форма динамического NAT, который отображает несколько незарегистрированных адресов в единственный зарегистрированный IP-адрес, используя различные порты. Известен также как [PAT](https://ru.wikipedia.org/wiki/Трансляция_порт-адрес) (Port Address Translation). При перегрузке каждый компьютер в частной сети транслируется в тот же самый адрес, но с различным номером порта

## [DNS](https://ru.wikipedia.org/wiki/DNS)

**DNS** ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *Domain Name System* «система доменных имён») — компьютерная [распределённая система](https://ru.wikipedia.org/wiki/Распределённая_база_данных) для получения информации о [доменах](https://ru.wikipedia.org/wiki/Доменное_имя). Чаще всего используется для получения [IP-адреса](https://ru.wikipedia.org/wiki/IP-адрес) по имени [хоста](https://ru.wikipedia.org/wiki/Хост) (компьютера или устройства), получения информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене ([SRV-запись](https://ru.wikipedia.org/wiki/SRV-запись)).

*Resolve.conf*

[*Как работает DNS*](file:///C:\Users\dimmm\AppData\Roaming\Users\dimmm\AppData\Roaming\Microsoft\Word\Rfr%20hf,jnftn%20ВТЫ)



в DNS есть файлы, в которых хранятся все необходимые связи между доменным именем и IP-адресами. Написанное в этом файле называется описанием DNS-зоны, или просто DNS-зоной.

Есть несколько видов записей, которые могут быть файле. Основные записи:

* А — IP-адрес веб-ресурса, который соответствует определённому имени домена,
* MX — адрес почтового сервера,
* CNAME — указывает аналог основному доменному имени. Эта запись чаще всего используется для прикрепления поддомена,
* NS — адрес DNS-сервера, где хранятся все ресурсные записи,
* TXT — любая текстовая информация о домене,
* SPF — список серверов, которым позволено отправлять письма от имени указанного домена,
* SOA — исходная запись зоны, в которой указаны сведения о сервере.

## [HTTP](https://ru.wikipedia.org/wiki/HTTP)

[URI](https://ru.wikipedia.org/wiki/URI) (Uniform Resource Identifier)

**Методы**

* **OPTIONS** - для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. можно также начать какой-либо процесс
* **GET** - для запроса содержимого указанного ресурса
* **HEAD** - Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело
* **POST** - для передачи пользовательских данных заданному ресурсу
* **PUT** - для загрузки содержимого запроса на указанный в запросе URI
* **PATCH** - Аналогично PUT, но применяется только к фрагменту ресурса
* **DELETE** - Удаляет указанный ресурс
* **TRACE** - Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе
* **CONNECT** - Преобразует соединение запроса в прозрачный [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP)-туннель

[**Коды состоя**](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#403)**ния**

* 1xx - [**Информационный**](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP#1xx)(англ. **informational**)
* 2xx - [**Успех**](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP#2xx) (англ. **Success**)
  + [200 OK](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#200) («хорошо»)
  + [206 Partial Content](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#206) («частичное содержимое»)
* 3xx - [**Перенаправление**](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP#3xx) (англ. **Redirection**)
  + [301 Moved Permanently](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#301) («перемещено навсегда»)
  + [302 Found](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#302) («найдено»)
  + [304 Not Modified](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#304) («не изменялось»)
* 4xx **-** [**Ошибка клиента**](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP#4xx) (англ. **Client Error**)
  + 400 Bad Request — сервер обнаружил в запросе клиента синтаксическую ошибку. Появился в HTTP/1.0.
  + 401 Unauthorized — для доступа к запрашиваемому ресурсу требуется [аутентификация](https://ru.wikipedia.org/wiki/Аутентификация).
  + [403 Forbidden](https://ru.wikipedia.org/wiki/Ошибка_403) («запрещено (не уполномочен)») В отличие от 401 Unauthorized, личность клиента известна серверу.
  + [404 Not Found](https://ru.wikipedia.org/wiki/Ошибка_404) («не найдено»)
  + 405 Method Not Allowed — указанный клиентом метод нельзя применить к текущему ресурсу
  + 407 Proxy Authentication Required — ответ аналогичен коду [401](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#401) за исключением того, что аутентификация производится для прокси-сервера. Механизм аналогичен идентификации на исходном сервере. Появился в HTTP/1.1.
  + 408 Request Timeout — время ожидания сервером передачи от клиента истекло.
* 5xx - [**Ошибка сервера**](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP#5xx) (англ. **Server Error**)
  + [500 Internal Server Error](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#500) («внутренняя ошибка сервера»)
  + 503 Service Unavailable — сервер временно не имеет возможности обрабатывать запросы по техническим причинам (обслуживание, перегрузка и прочее).
  + [502 Bad Gateway](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#502) («плохой, ошибочный шлюз») сервер, выступая в роли шлюза или прокси-сервера, получил недействительное ответное сообщение от вышестоящего сервера.
  + [503 Service Unavailable](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#503) («сервис недоступен») сервер временно не имеет возможности обрабатывать запросы по техническим причинам (обслуживание, перегрузка и прочее).
  + 504 Gateway Timeout — сервер в роли шлюза или прокси-сервера не дождался ответа от вышестоящего сервера для завершения текущего запроса

**Заголовки**

* [**General Headers**](https://ru.wikipedia.org/wiki/Список_заголовков_HTTP#%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8)(«Основные заголовки») — могут включаться в любое сообщение клиента и сервера;
* [**Request Headers**](https://ru.wikipedia.org/wiki/Список_заголовков_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0)(«Заголовки запроса») — используются только в запросах клиента;
* [**Response Headers**](https://ru.wikipedia.org/wiki/Список_заголовков_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0)(«Заголовки ответа») — только для ответов от сервера;
* [**Entity Headers**](https://ru.wikipedia.org/wiki/Список_заголовков_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D1%81%D1%83%D1%89%D0%BD%D0%BE%D1%81%D1%82%D0%B8) («Заголовки сущности») — сопровождают каждую сущность сообщения.

[GET](https://ru.wikipedia.org/wiki/HTTP#GET) /about.html HTTP/1.1 - метод, УРЛ, версия протокола

Host: example.org - адрес хоста

User-Agent: MyLonelyBrowser/5.0

## [Что такое протокол HTTPS и как он работает](https://timeweb.com/ru/community/articles/chto-takoe-protokol-https-i-princip-ego-raboty)

**HTTP** (от англ. Hyper Text Transfer Protocol)

**SSL** (от англ. Secure Socket Layer)

**TLS** (от англ. Transport Layer Security):

* отвечает за конфиденциальность передаваемых данных;
* проводит аутентификацию;
* следит за целостностью передаваемой информации.

**Основные виды сертификатов:**

* Самоподписанные SSL-сертификаты
* Групповой SSL-сертификат
* SSL-сертификаты, подписанные доверенным центром сертификации (CA)
* Доменные сертификаты
* Организационные сертификаты
* Расширенные проверенные сертификаты

**Настройка HTTPS:**

* Запрос на получение сертификата (CSR)
* Получение подписанных SSL ключей на 90-дневной период
* Настройка Nginx для работы с SSL

Удостоверяющие центры [Thawte](https://www.nic.ru/catalog/ssl/thawte/), [GeoTrust](https://www.nic.ru/catalog/ssl/geotrust/), [DigiCert](https://www.nic.ru/catalog/ssl/symantec/), [GlobalSign](https://www.nic.ru/catalog/ssl/globalsign/) и [Comodo](https://www.nic.ru/catalog/ssl/comodo/). (Kaspersky Lab)

Шифрование – синхронное (шифровка и расшифровка одним ключом) и асинхнонное (открытый и закрытый ключи)

## [Каковы этапы TLS-рукопожатия](https://www.cloudflare.com/ru-ru/learning/ssl/what-happens-in-a-tls-handshake/)

Точные шаги в рамках TLS-рукопожатия зависят от используемого алгоритма обмена ключами и наборов шифров, поддерживаемых обеими сторонами. Алгоритм обмена ключами RSA, который в настоящее время считается небезопасным, использовался в версиях TLS до версии 1.3. Это происходит примерно следующим образом:

1. **Сообщение "hello" клиента:** Клиент инициирует рукопожатие, посылая серверу сообщение "hello". В сообщении будет указано, какую версию TLS поддерживает клиент, поддерживаемые наборы шифров и строка случайных байтов, известная как "random клиента".
2. **Сообщение "hello" сервера:** В ответ на сообщение "hello" клиента сервер отправляет сообщение, содержащее [SSL-сертификат](https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/) сервера, выбранный сервером набор шифров и "random сервера" – еще одну строку случайных байтов, генерируемую сервером.
3. **Аутентификация:** Клиент проверяет SSL-сертификат сервера в центре сертификации, который его выдал. Он подтверждает, что сервер является тем, за кого себя выдает, и что клиент взаимодействует с реальным владельцем домена.
4. **Premaster secret:** Клиент посылает еще одну строку случайных байтов, которая называется "premaster secret". Premaster secret шифруется открытым ключом и может быть расшифрован сервером только с помощью закрытого ключа. (Клиент берет [открытый ключ](https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/) из SSL-сертификата сервера).
5. **Используемый закрытый ключ:** Сервер расшифровывает "premaster secret".
6. **Создание сеансовых ключей:** Клиент и сервер генерируют сеансовые ключи из random'а клиента, random'а сервера и premaster secret. Они должны прийти к одинаковым результатам.
7. **Клиент готов:** Клиент отправляет сообщение "готово", зашифрованное сеансовым ключом.
8. **Сервер готов:** Сервер отправляет сообщение "готово", зашифрованное сеансовым ключом.
9. **Безопасность симметричного шифрования достигнута:** Рукопожатие завершено, и связь продолжается с использованием сеансовых ключей.

Все TLS-рукопожатия используют асимметричную криптографию (открытый и закрытый ключ), но не все будут использовать закрытый ключ в процессе генерации сеансовых ключей. Например, кратковременное рукопожатие Диффи-Хеллмана происходит следующим образом:

1. **Сообщение "hello" клиента:** Клиент посылает сообщение "hello", содержащее версию протокола, random клиента и список наборов шифров.
2. **Сообщение "hello" сервера:** Сервер отвечает своим SSL-сертификатом, выбранным набором шифров и random'ом сервера. В отличие от описанного выше RSA-рукопожатия, в это сообщение сервер также включает следующее (шаг 3):
3. **Цифровая подпись сервера:** Сервер вычисляет цифровую подпись всех сообщений до этого момента.
4. **Цифровая подпись подтверждена:** Клиент проверяет цифровую подпись сервера, подтверждая, что сервер является тем, за кого он себя выдает.
5. **Параметр DH клиента:**Клиент отправляет серверу свой параметр DH.
6. **Клиент и сервер вычисляют premaster secret:** Вместо того чтобы клиент генерировал premaster secret и отправлял его серверу, как в RSA-рукопожатии, клиент и сервер по отдельности используют параметры DH, которыми они обменялись, для вычисления подходящего premaster secret'а.
7. **Создание сеансовых ключей:** Теперь клиент и сервер вычисляют сеансовые ключи из premaster secret'а, random'а клиента и random'а сервера, как и при RSA-рукопожатии.
8. **Клиент готов:** То же самое, что и рукопожатие RSA.
9. **Сервер готов**
10. **Безопасность симметричного шифрования достигнута**

## [SSH алгоритм установления соединения](https://habr.com/ru/articles/425637/)

**1. Установка TCP-соединения**

На этом этапе происходит сетевое подключение клиента к серверу на TCP-порт, указанный в опции Port (по умолчанию: 22) в файле конфигурации сервера **/etc/ssh/sshd\_config**.

**2. Открытие защищенного канала**

**2.1 Обмен идентификационными данными**

После установки TCP-соединения, клиент и сервер (далее по тексту – стороны) обмениваются версиями SSH-протокола и другими вспомогательными данными, необходимыми для выяснения совместимости протоколов и для выбора алгоритмов работы.

**2.2 Выбор алгоритмов: обмена ключами, шифрования, сжатия и т.п**.

При работе SSH используется довольно много алгоритмов, одни из них используются для шифрования, вторые для обмена ключами, третьи для сжатия передаваемых данных и т.п. На этом шаге стороны отсылают друг другу списки поддерживаемых алгоритмов, наибольший приоритет имеют алгоритмы в начале каждого списка. Затем сравнивают алгоритмы в полученных списках с алгоритмами, имеющимися в системе, и выбирают первый совпавший в каждом списке.

**2.3 Получение сессионного ключа шифрования**

Процесс получения сессионного ключа может отличаться в зависимости от версии алгоритма, но в общих чертах сводится к следующему:

**Сервер отсылает клиенту свой ключ** (DSA, RSA или т.п. согласно договорённости между сторонами, произведёнными в п.2.2).

Если клиент производит соединение с данным сервером впервые (о чем говорит отсутствие записи в файле /home/username/**.ssh/known\_hosts** у клиента), то пользователю будет задан вопрос о доверии ключу сервера. Если же соединение с данным сервером уже устанавливалось ранее, то клиент сравнивает присланный ключ с ключом, записанным в /home/username/.ssh/known\_hosts. Если ключи не совпадают, то пользователь получит предупреждение о возможной попытке взлома.

Сеансовый ключ создается исключительно на период жизни канала и уничтожается при закрытии соединения.

**3. Аутентификация клиента**

И только теперь, когда клиент и сервер установили канал для зашифрованной передачи данных, они могут произвести аутентификацию по паролю или ключам.

В общих чертах, аутентификация посредством ключей происходит следующим образом:

Клиент отсылает серверу имя пользователя (username) и свой публичный ключ.

Сервер проверяет в файле /home/username/**.ssh/authorized\_keys** наличие присланного клиентом открытого ключа. Если открытый ключ найден, то сервер генерирует случайное число и шифрует его открытым ключом клиента, после чего результат отправляется клиенту.

Клиент расшифровывает сообщение своим приватным ключом и отправляет результат серверу.

Сервер проверяет полученный результат на совпадение с тем числом, которое он изначально зашифровал открытым ключом клиента, и в случае совпадения считает аутентификацию успешной.

**4. Уровень подключения**

После проведения всех вышеперечисленных процедур, пользователь получает возможность передавать команды серверу или копировать файлы.

На этом уровне обеспечивается: мультиплицирование каналов (возможность работы множества каналов к одному серверу за счет объединения их в один канал), туннелирование и т. п.

## [CORS и принцип одинакового источника](https://cloud.yandex.ru/ru/docs/glossary/cors)

Access-Control-Allow-Origin: \*

**CORS** (Cross-Origin Resource Sharing, англ.  «совместное использование ресурсов разных источников»)

**Структура Cross-Origin Resource Sharing**

Методы CORS предназначены для управления доступом к дескрипторам (тегам) на веб-страницах в сети. Управляемые типы доступа подразделяются на три основных категории по работе с информацией сторонних ресурсов:

***Доступ на запись*** — это доступ к ссылкам, заполнению веб-форм и переадресации на сторонние веб-страницы, т.е. на передачу информации в сторонний источник (веб-ресурс).

***Доступ на вставку*** относится к категории доступа на считывание информации из стороннего источника. К этому типу принадлежат вставки в код дескрипторов audio, video, img, embed, object, link, script, iframe и другие элементы оформления веб-страниц. Структура подобных дескрипторов подразумевает самостоятельную инициацию перекрестных (cross-origin) запросов из сторонних источников. Все дескрипторы этой категории представляют низкий уровень угрозы безопасности, поэтому разрешены в веб-браузере по умолчанию.

***Доступ на считывание*** — это дескрипторы, загружаемые с использованием фоновых методов вызова, таких как fetch(), технологии обмена данными Ajax и пр. Поскольку подобные дескрипторы могут содержать в теле любые участки кода (в том числе вредоносного), они запрещены в веб-браузерах по умолчанию.

При настройке веб-сайта механизм CORS позволяет выборочно блокировать различные категории доступа пользователя к ресурсам (запись, вставку или считывание).

## LINUX

kill(pid, snum);

|  |  |  |
| --- | --- | --- |
| Номер | Название | Описание |
| 01 | SIGHUP | Освобождение линии (hangup). |
| 02 | SIGINT | Прерывание (interrupt). |
| 03 | SIGQUIT | Выход (quit). |
| 04 | SIGILL | Некорректная команда (illegal instruction). Не переустанавливается при перехвате. |
| 05 | SIGTRAP | Трассировочное прерывание (trace trap). Не переустанавливается при перехвате. |
| 06 | SIGIOT или SIGABRT | Машинная команда IOT. |
| 07 | SIGEMT | Машинная команда EMT. |
| 08 | SIGFPE | Исключительная ситуация при выполнении операции с вещественными числами (floating-point exception) |
| 09 | SIGKILL | Уничтожение процесса (kill). Не перехватывается и не игнорируется. |
| 10 | SIGBUS | Ошибка шины (bus error). |
| 11 | SIGSEGV | Некорректное обращение к сегменту памяти (segmentation violation). |
| 12 | SIGSYS | Некорректный параметр системного вызова (bad argument to system call). |
| 13 | SIGPIPE | Запись в канал, из которого некому читать (write on a pipe with no one to read it). |
| 14 | SIGALRM | Будильник |
| 15 | SIGTERM | Программный сигнал завершения |
| 16 | SIGUSR1 | Определяемый пользователем сигнал 1 |
| 17 | SIGUSR2 | Определяемый пользователем сигнал 2 |
| 18 | SIGCLD | Завершение порожденного процесса (death of a child). |
| 19 | SIGPWR | Ошибка питания |
| 22 |  | Регистрация выборочного события |

Kill -0

**Команда ps** - вывод информации о выполняемых процессах

|  |  |
| --- | --- |
| Параметр | Описание |
| -a | отобразить все процессы, связанных с терминалом (отображаются процессы всех пользователей) |
| -e | отобразить все процессы |
| -t список терминалов | отобразить процессы, связанные с терминалами |
| -u идентификаторы пользователей | отобразить процессы, связанные с данными идентификаторыми |
| -g идентификаторы групп | отобразить процессы, связанные с данными идентификаторыми групп |
| -x | отобразить все процессы, не связанные с терминалом |

**top** - вывод информации о процессах в реальном времени

**free** – информация об оперативной памяти

**df** - информация о дисковой памяти

**nice** – изменение приоритета команды. Понижение приоритета 1..19, с sudo – повышение (--10)

**nohup** - выполняет запуск команды в режиме игнорирования сигналов. Не игнорируются только сигналы SIGHUP и SIGQUIT

**nohup** - игнорирование сигналов прерывания

**kill** - принудительное завершение процесса

**jobs, fg, bg** - команды выполнения процессов в фоновом режиме

**load average** – среднее количество процессов **И** операций ввода\вывода в процессорной очереди

Есть логин пароль, IP. как осмотреться на машине?

* /var/log/
* dmesg - логи ядра,
* логи контейнеров на хосте
* количество inod
* iostat
* top
* lsblk
* fstab
* ss -p

Осмотреться на сервисе через ssh

* посмотреть открытые порты:  
  lsof -i  
  netcat -tunapl  
  ss -tunapl
* посмотреть открытые сокеты: lsof -i -U
* посмотреть трафик:  
  tcpdump  
  iftop

[Дистрибутивы Linux](https://ru.wikipedia.org/wiki/Шаблон:Дистрибутивы_Linux)

**Ubuntu/Debian/Kali/Mint Linux:**

apt-get install - скачать и установить программу

apt-get remove - удалить программу

dpkg –i - установить программу из файла .deb

dpkg –r - удалить программу

**RedHat/CentOS/Amazon Linux:**

yum install - скачать и установить программу

yum remove - удалить программу

rpm –i - установить программу из файла .rpm

rpm –e - удалить программу

**/etc/apt/sources.list** – список репозиториев

**sudo add-apt-repository; /etc/apt/sources.list.d/** – добавить репозиторий deb

**wget -q -O - https://dl-ssl.google.com/linux/linux\_signing\_key.pub | sudo apt-key add** – добавить ключ репы

файл в каталоге **/etc/yum.repos.d/** с расширением **\*.repo** - добавить репозиторий rpm

## [Logrotate](https://www.dmosk.ru/miniinstruktions.php?mini=logrotate-linux)

Для приложение, ротация логов настраивается в отдельных файлах, расположенных по пути **/etc/logrotate.d/**

Задание на автоматический запуск создается по умолчанию в файле **/etc/cron.daily/logrotate.**

## [Немного о директории /proc в Linux](https://mnorin.com/nemnogo-o-direktorii-proc-v-linux.html)

**/proc** — это не настоящая файловая система. Она виртуальная. **Ее основная задача — получение состояния системы и частично выполнение управляющих действий**

Информация о процессах хранится в директориях **/proc/N**, где **N** — числовой идентификатор процесса.

**/proc/N/cmdline** — Содержимое командной строки, которой был запущен процесс.

**/proc/N/environ** — Описание окружения, в котором работает процесс.

**/proc/N/exe** — Символическая ссылка на выполнимый файл запущенной программы.

**/proc/N/limits** — Лимиты на использование системных ресурсов, актуальные для работающего процесса.

**/proc/N/mounts** — Список смонтированных ресурсов, которые доступны процессу

**/proc/N/status** — Статус работающей программы.

**/proc/N/cwd** — Текущая директория для процесса

**/proc/N/fd** — Файловые дескрипторы, которые используются процессом.

**/proc/N/fdinfo** — Информация о файловых дескрипторах.

**/proc/N/root** — Символическая ссылка на директорию, которая для данного процесса является корневой

**/proc/N/net** — Сетевые системные ресурсы и их параметры, действующие для конкретного процесса.

**Общесистемные псевдо-файлы и псевдо-директории**

**/proc/acpi** — директория, связанная с управлением питанием и различным устройствами.

**/proc/asound** — директория, связанная со звуковыми устройствами.

**/proc/bus** — информация о системных шинах и устройствах, которые к ним подключены.

**/proc/fs** — Информация о файловых системах. free

**/proc/sys** — Псевдо-директория, содержащая массу информации о системе.

**/proc/sys/fs** — Псевдо-директория, содержащая информацию о подсистеме, связанной с файловыми системами.

**/proc/sys/net** — Общесистемные сетевые параметры.

**/proc/sys/vm** — Параметры виртуальной памяти.

## [mount point](https://ru.wikipedia.org/wiki/Точка_монтирования)

Список файловых систем, которые поддерживаются ядром, находится в файле **/proc/filesystems**

**/etc/fstab** - файл конфигурации

**mount /dev/cdrom /mnt/cdrom**

**umount /dev/cdrom**

**mount -t ntfs -o noatime,users,rw,fmask=111,dmask=000,locale=ru\_RU.UTF-8 /dev/sda1 /mnt/win\_xp**

**mount 172.22.2.1:/mnt/iso/ /mnt/iso/** - Монтирование сетевых дисков NFS

**sshfs user@udalenniy\_server:/tmp ~/udalennaya\_papka** - Монтирование каталога файловой системы с другого компьютера через ssh

## [О sysctl](https://habr.com/ru/companies/otus/articles/340870/)

**net** — сеть; **kern** — ядро; **vm** — память и кэши

**vm.swappiness** — настройка агрессивности “высвапливания” памяти. 0-100 (default 60)

**vm.min\_free\_kbytes**— Определяет минимальный размер свободной памяти который необходимо поддерживать

**vm.overcommit\_memory** — разрешает/запрещает “аллоцировать” памяти больше чем есть

**net.ipv4.ip\_forward** — разрешение или запрет на маршрутизацию пакетов

**net.ipv4.{all,default,interface}.rp\_filter** — контролирует опцию Reverse Path Filtering (проверка пакетов)

**net.ipv4.ip\_local\_port\_range** — определяет минимальный и максимальный порты, который используется для создания локального клиентского сокета

**net.ipv4.ip\_default\_ttl** — время жизни пакета (TTL) по умолчанию

**net.core.netdev\_max\_backlog** — регулирует размер очереди пакетов между сетевой картой и ядром

Для понимания надо знать как работают соединения в tcp:

1. Программа открывает слушающий сокет: socket() -> listen(). В итоге получает, например \*:80 (80-й порт на всех интерфейсах).
2. Клиент устанавливает соединение: a) посылает серверу **syn**-пакет (вот в этом месте и работает **tcp\_max\_syn\_backlog**); b) получает от сервера **syn-ack**; c) посылает серверу **ack** (а вот тут уже работает **somaxconn**)
3. Через вызов accept соединение обрабатывается и передается процессу для работы с конкретным клиентом.

**net.core.somaxconn** — размер очереди установленных соединений ожидающих обработки accept()

**listen.backlog** это параметр **backlog** функции **TCP listen** того сокета, на котором висит fpm

параметр backlog отвечает за размер очереди одновременно \_**ожидающих**\_ подключений к сокету, то есть инициированных (SYN - SYN,ACK - ACK), но еще не принятых сервером (established)

-1 использует текущий **hard limit net.core.somaxconn**,. Значение по умолчанию в линуксах равно 128 и этого более, чем достаточно для любого php-fpm.

**net.ipv4.tcp\_max\_syn\_backlog** — размер очереди **не установленных соединений**

**net.ipv4.tcp\_window\_scaling** — масштабирование окна. **tcp\_window -** это количество байт, которые мы можем отправить без подтверждения.

**net.ipv4.tcp\_wmem,net.ipv4.tcp\_rmem** — это три числа в БАЙТАХ, гарантированный размер буфера, размер по умолчанию и максимальный размер

**net.ipv4.tcp\_mem** — настройки управления памятью tcp-стека. 3 числа в СТРАНИЦАХ

**kernel.core\_pattern** — позволяет задавать формат имени (и пути), которое будет использоваться для сохранения core dump.

[существует](https://www.opennet.ru/base/sys/io_scheduler_linux.txt.html) ровно четыре основных планировщика I/O:

\* **BFQ** – основан на CFQ, использует Round Robin – последовательный перебор процессов для доступа к записи на диск

\* **CFQ** - Completely Fair Queuing - полностью справедливая очередь

\* **Deadline**

\* **NOOP**

\* **Anticipatory** - упреждающий конвейер

## [POSIX](https://ru.wikipedia.org/wiki/POSIX)

**POSIX** ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) ***P****ortable****O****perating****S****ystem****I****nterface* - переносимый интерфейс [операционных систем](https://ru.wikipedia.org/wiki/Операционная_система)) — набор стандартов, описывающих интерфейсы между [операционной системой](https://ru.wikipedia.org/wiki/Операционная_система) и [прикладной программой](https://ru.wikipedia.org/wiki/Прикладное_программное_обеспечение) (системный [API](https://ru.wikipedia.org/wiki/Интерфейс_программирования_приложений)), библиотеку языка C и набор приложений и их интерфейсов.

## [Стандартные потоки ввода/вывода](http://xgu.ru/wiki/Стандартные_потоки_ввода/вывода)

* Стандартный поток ввода (**stdin**) — 0;
* Стандартный поток вывода (**stdout**) — 1;
* Стандартный поток ошибок (**stderr**) — 2.
* **/dev/stdin** — стандартный поток ввода;
* **/dev/stdout** — стандартный поток вывода;
* **/dev/stderr** — стандартный поток ошибок.

**< файл** Использовать файл как источник данных для стандартного потока ввода.

**> файл** Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

**2> файл** Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

**>>файл** Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

**2>>файл** Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

**&>файл** или **>&файл** Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: **>файл 2>&1**.

**>&-** Закрыть поток вывода перед вызовом команды;

**2>&-** Закрыть поток ошибок перед вызовом команды;

**cat <<EOF** Весь текст между блоками EOF (в общем случае вместо EOF можно использовать любое слово) будет выведен на экран. Важно: перед последним EOF не должно быть пробелов! (heredoc синтаксис).

**EOF<<<string** Аналогично, но только для одной строки (для bash версии 3 и выше)

| № п/п | команда | как будет интерпретировано |
| --- | --- | --- |
| 1 | command1 && command2 | Выполнить command2 только если command1 вернула 0 |
| 2 | command1 & command2 | Выполнение command1 отправить в фон (bg) и сразу выполнить command2 в (fg) |
| 3 | command1 ; command2 | Выполнить команды последовательно вне зависимости от результата первой |
| 4 | command1 || command2 | Выполнить command1, если она вернет не 0, выполнить command2 |
| 5 | command1 | command2 | Пайп. Передать stdout command1 в stadin command2 |
| 6 | command1 command2 | Выполнить command1 с аргументом 'command2' |

**tee** [*опции*] *файл* - позволяет сохранять данные, передающиеся в канале

**find** – поиск файлов

## [Systemd за пять минут](https://habr.com/ru/companies/slurm/articles/255845/)

— менеджер системы и служб для Linux

**/usr/lib/systemd/system/** – юниты из установленных пакетов RPM — всякие nginx, apache, mysql и прочее  
**/run/systemd/system/** — юниты, созданные в рантайме — тоже, наверное, нужная штука  
**/etc/systemd/system/** — юниты, созданные системным администратором — а вот сюда мы и положим свой юнит.

Юниты содержат информацию о системных сервисах, прослушиваемых сокетах, сохраненных снапшотах состояний системы и других объектах, относящихся к системе инициализации.

Типы юнитов **systemd**:

.**service** – системный сервис,

.**target** — группа юнитов **systemd**,

.**automount** – точка автомонтирования файловой системы,

.**device** – файл устройства, распознанного ядром,

.**mount** – точка монтирования файловой системы,

.**path** – файл или директория в файловой системе,

.**scope** – процесс, созданный извне,

.**slice** – группа иерархически организованных юнитов, управляющая системными процессами,

.**snapshot** – сохраненное состояние менеджера **systemd**,

.**socket** – сокет межпроцессного взаимодействия,

.**swap** – свап-устройство или свап-файл (файл подкачки),

.**timer** – таймер **systemd**.

Для просмотра, старта, остановки, перезагрузки, включения или выключения системных сервисов используется команда **systemctl**

**systemctl enable myunit  
systemctl -l status myunit**

**systemctl start myunit**

* **systemctl halt** – останавливает систему,
* **systemctl poweroff** – выключает систему,
* **systemctl reboot** – перезагружает систему,
* **systemctl suspend** – переводит систему в режим ожидания,
* **systemctl hibernate** – переводит систему в спящий режим,
* **systemctl hybrid-sleep** – переводит систему в режим гибридного сна.

[**journalctl**](https://habr.com/ru/companies/ruvds/articles/533918/) – журнал сервисов **systemd**

# **journalctl -p 0**  
  
Для уровней важности, приняты следующие обозначения:

* 0: emergency (неработоспособность системы)
* 1: alerts (предупреждения, требующие немедленного вмешательства)
* 2: critical (критическое состояние)
* 3: errors (ошибки)
* 4: warning (предупреждения)
* 5: notice (уведомления)
* 6: info (информационные сообщения)
* 7: debug (отладочные сообщения)

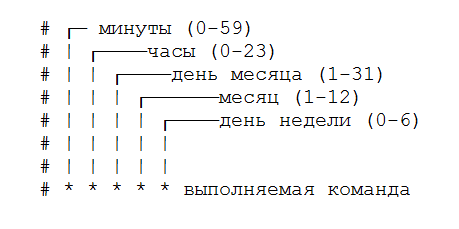
**# journalctl -u** – просмотр журналов определенного сервиса

## [Таймеры systemd](https://habr.com/ru/companies/ruvds/articles/512868/)

**systemctl status \*timer** - сведения о таймерах

## [Cron Jobs](https://habr.com/ru/companies/badoo/articles/468061/)

**Шаблон:**



**Минуты(0-59) Часы(0-24) День(1-31) Месяц(1-12) День недели(0-6) Команда**

**crontab -e** # редактировать таблицу задач

**crontab -l** # показать таблицу задач

**crontab -r** # удалить таблицу задач

**crontab path/to/file.crontab** # загрузить таблицу задач из файла

# строки-комментарии игнорируются

#

# задача, выполняемая ежеминутно

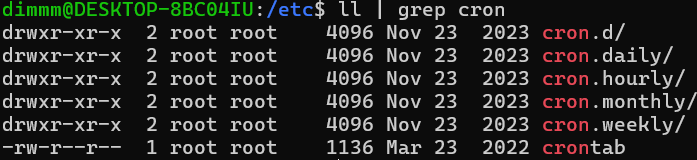
\* \* \* \* \* /path/to/exec -a -b -c

# задача, выполняемая на 10-й минуте каждого часа

10 \* \* \* \* /path/to/exec -a -b -c

# задача, выполняемая на 10-й минуте второго часа каждого дня и использующая перенаправление стандартного потока вывода

10 2 \* \* \* /path/to/exec -a -b -c > /tmp/cron-job-output.log

Таблицы задач обычных пользователей располагаются в **/var/cron/tabs/username**

## [Уровни логирования](https://habr.com/ru/companies/yandex_praktikum/articles/739058/)

* **FATAL**: является наивысшим уровнем критичности логов и указывает на самые критические ошибки и проблемы, которые могут привести к немедленному завершению программы или системы. Логи с уровнем FATAL обычно означают серьезные сбои, которые требуют немедленного вмешательства и исправления.
* **ERROR**: этот уровень используется для записи ошибок и проблем, которые могут привести к некорректной работе приложения. Логи с уровнем ERROR указывают на проблемы, которые требуют вмешательства и исправления.
* **WARN:** уровень WARN указывает на предупреждения и потенциальные проблемы, которые не являются критическими ошибками. Логи с уровнем WARN могут включать сообщения о неправильном использовании приложения, некорректных данных или других ситуациях, требующих внимания.
* **INFO:** этот уровень предоставляет информацию о ходе работы приложения и важных событиях. Логи с уровнем INFO содержат сообщения, которые помогают отслеживать основные операции и состояние приложения. Например, они могут сообщать о начале и окончании определенных операций, загрузке ресурсов, отправке и получении запросов, изменении состояния приложения и других событиях, которые могут быть полезны для отслеживания хода выполнения программы.
* **DEBUG**: содержат подробности о ходе выполнения приложения, значимые переменные и другие данные, которые могут быть полезными при обнаружении и исправлении ошибок.
* **TRACE**: это наиболее подробный уровень логирования. Логи с уровнем TRACE содержат очень подробную информацию о состоянии приложения, включая значения переменных, шаги выполнения и другие детали. Они обычно используются во время отладки и разработки для более глубокого анализа приложения.

## [Как настроить сеть в Ubuntu](https://help.reg.ru/support/servery-vps/oblachnyye-servery/ustanovka-programmnogo-obespecheniya/kak-nastroit-set-v-ubuntu-16-04)

**ifconfig -a** - список установленных сетевых интерфейсов

**ip link show** – то же

Чтобы настроить сеть, нужно изменить конфигурационный файл **/etc/network/interfaces**.

**Настройка dhcp:**

auto enp0s3

iface enp0s3 inet dhcp

**Настройка static ip:**

auto enp0s3

iface enp0s3 inet static

address 192.168.0.0

netmask 255.255.255.0

gateway 192.168.0.1

dns-nameservers 8.8.8.8 192.168.0.10

service networking restart

**настроить DNS только для этого сеанса:**

**sudo nano /etc/resolv.conf**

nameserver 8.8.8.8

## [Способы освобождения дискового пространства](https://ip-calculator.ru/blog/ask/ways-to-free-up-disk-space-on-ubuntu-linux-servers/)

**Проверка использования**

sudo du -h --max-depth=1 | sort -h

du -cksh \*

**Удаляет библиотеки и пакеты, которые были установлены автоматически и все старые ядра Linux**

sudo yum autoremove

**Очистите кеш apt. Проверка:**

sudo du -h /var/cache/apt/archives

# Remove only the outdated packages

sudo apt-get autoclean

# Remove entire cache

sudo apt-get clean

**Очистить журналы Systemd**

# Check usage

journalctl --disk-usage

# Clear logs older than 3 days

sudo journalctl —vacuum-time=3d

**Удалите старые версии снимков приложений**

# Check disk space taken up by snaps

sudo du -h /var/lib/snapd/snaps | sort -h

**Запустите скрипт Bash, чтобы удалить неиспользуемые снимки.**

#!/bin/bash

# Removes old revisions of snaps

# CLOSE ALL SNAPS BEFORE RUNNING THIS

set -eu

snap list --all | awk '/disabled/{print $1, $3}' |

while read snapname revision; do

snap remove "$snapname" --revision="$revision"

done

**Освободить пространство, используемое объектами докера**

docker prune

## [Переполнение inodes](https://timeweb.cloud/docs/unix-guides/troubleshooting-unix/inodes-overflow)

количестве использованных и занятых дескрипторов

df -hTi

Найти директории, содержащие в себе большое количество файлов

echo "Detailed Inode usage for: $(pwd)" ; for d in `find -maxdepth 1 -type d |cut -d\/ -f2 |grep -xv . |sort`; do c=$(find $d |wc -l) ; printf "$c\t\t- $d\n" ; done ; printf "Total: \t\t$(find $(pwd) | wc -l)\n"

## Исправляем ошибку “Too many open files“ в Linux

Максимально количество файловых дескрипторов, которые могут быть открыты в вашей файловой системе всеми процессами можно узнать так:

# cat /proc/sys/fs/file-max

Чтобы узнать, сколько файлов открыто сейчас, выполните:

$ cat /proc/sys/fs/file-nr

Чтобы вывести текущее ограничение на количество открытых файлов в ядре Linux, выполните:

# sysctl fs.file-max

Выведем ограничение на количество открытых файлов для одного процесса текущего пользователя:

# ulimit -n

Выведем максимальное количество для одного пользователя (max user processes):

# ulimit –u

Для вывода Soft-ограничения:

# ulimit -Sn

Для вывода Hard-ограничения:

# ulimit -Hn

Например, вы получили ошибку too many open files для nginx. Проверьте, сколько файлов разрешено открывать процессу этому пользователю:

$ sudo -u nginx bash -c 'ulimit -n'

В старых ядрах Linux значение fs.file-max может быть равно 10000. Поэтому проверьте это значение и увеличьте его, чтобы оно было больше чем ограничение в limits.conf:

# sysctl -w fs.file-max=500000

После изменений, перезапустите терминал и проверьте значение лимита max\_open\_files:

# ulimit -n

Вы можете изменить лимит на количество открытых файловых дескрипторов для конкретного сервиса, а не для всей системы. Рассмотрим на примере apache. Чтобы изменить значения, откройте настройки службы через [systemctl](https://winitpro.ru/index.php/2019/10/11/avtozagruzka-servisov-i-skriptov-v-linux/):

# systemctl edit httpd.service

После изменения, нужно обновить конфигурацию сервиса и перезапустить его:

# systemctl daemon-reload  
# systemctl restart httpd.service

Например, вы определил PID сервиса 32724:

# cat /proc/32724/limits | grep "Max open files”

После того, как вы увеличил ограничения на количество открытых файлов для сервера, нужно также поправить конфигурационный файл службы. Например, для веб-сервера Nginx в файле конфигурации **/etc/nginx/nginx.conf** нужно задать значение в директиве:

worker\_rlimit\_nofile 16000

При настройке Nginx на [высоконагруженном](https://winitpro.ru/index.php/2019/11/12/proizvoditelniy-web-server-nginx-php-fpm/) 8-ядерном сервере с worker\_connections 8192 нужно в worker\_rlimit\_nofile указать 8192\*2\*8 ([vCPU](https://winitpro.ru/index.php/2019/11/15/virtualnaya-mashina-vcpu-cores/)) = 131072.

После чего выполнить рестарт Nginx:

# nginx -t && service nginx -s reload

Для apache, нужно создать директорию:

# mkdir /lib/systemd/system/httpd.service.d/

После этого создайте файл limit\_nofile.conf:

# nano /lib/systemd/system/httpd.service.d/limit\_nofile.conf

И добавьте в него:

[Service]

LimitNOFILE=16000

Не забудьте перезапустить сервис httpd.

Чтобы изменить лимиты на открытые файлы в рамках текущей сессии пользователя, выполните команду:

# ulimit -n 3000

## [Этапы жизненного цикла ПО](https://www.sravni.ru/kursy/info/zhiznennyj-cikl-po/)

1. **Определение потребностей бизнеса**.
2. **(только для аутсорса) Предпродажа**.
3. **Инициация**. С этого шага начинаются этапы SDLC (английская аббревиатура для жизненного цикла ПО), в которых принимает участие непосредственно команда разработки. ТОПы команды получают на руки ТЗ, анализируют его и называют +/- точные сроки и стоимость разработки. Если заказчика это не устраивает – начинаются обсуждения, если заказчика все устроило – ПО уходит в разработку.
4. **Проектирование/инженерный дизайн**.
5. **Имплементация**. Непосредственно разработка
6. **Тестирование**.
7. **Внедрение**.
8. **Сопровождение**.
9. **Смерть**.

**Парадигмы разработки ПО**

**Каскадная**

Идея: делаем сразу весь функционал по ТЗ, смотрим на результат.

**Гибкая (итеративная)**

Идея: делаем минимально жизнеспособный продукт, выпускаем на рынок, пошагово дорабатываем.

**Парадигма на стыке**

смешение принципов каскадной и гибкой парадигмы в разных пропорциях

**Scrum** отталкивается от спринтов – коротких (2-8 недель) промежутков, на которые команда ставит себе определенные задачи.

**Kanban** строится вокруг досок (Trello, Jira) и изолированных задач. Здесь тоже есть бэклог, из которого достаются фичи для реализации. Каждая фича затем делится на простые задачи, которые выкладываются на доску.

## [Основные команды для работы с Git и GitHub](https://skillbox.ru/media/code/osnovnye-komandy-dlya-raboty-s-git-i-github/)

git --version

git config --global user.name "Name Surname"

git config --global user.email "your@email"

git init

# Добавляем в индекс один файл

git add file\_name

# Добавляем в индекс несколько файлов

git add file\_name\_1 file\_name\_2 file\_name\_3

# Добавляем в индекс все изменённые файлы

git add .

git help command\_name

$ git status # Запрашиваем текущее состояние репозитория

git commit -m "Commit message"

# cделать новый коммит, минуя индекс

git commit -am "Commit message"

# перезаписать сообщение последнего коммита

git commit --amend -m "New commit message"

git log # Запрос на просмотр журнала коммитов

git log --oneline # Запрос на вывод истории коммитов в одну строку

git show abc12345 # Запрос на просмотр коммита с хешем abc12345

# Смотрим разницу между последним коммитом и текущим состоянием репозитория

git diff

# Вернуть неотслеживаемый файл к состоянию последнего коммита

git restore file\_name

# Вернуть все файлы из индекса к состоянию последнего коммита

git restore --staged

# Удалить файл из индекса и рабочей директории

git rm file\_name

# Откатываемся и переводим последующие коммиты в индекс

git reset --soft[mixed/hard] commit\_hash

git branch branch\_name

git branch -m old\_branch\_name new\_branch\_name

# Запрашиваем список всех доступных веток

git branch

git checkout branch\_name (-b – с созданием новой ветки)

git switch branch\_name (переключение с проверкой)

# Переключаемся на основную ветку, которая будет принимать изменения

git checkout main\_branch

# Сливаем изменения из второстепенной ветки в основную

git merge secondary\_branch

# Удаляем второстепенную ветку

git branch -d secondary\_branch

# Запрашиваем список удалённых репозиториев, которые связаны с локальным

git remote

# Команда для первой загрузки изменений в удалённый репозиторий: текущая ветка будет связана с веткой main в удалённом репозитории origin

git push -u origin main

# Скачиваем изменения из удалённого репозитория и добавляем их в локальную ветку

git pull

## [Мониторинг ИТ-систем](https://u-connect.ru/blog/posts/it-resheniya/organizatsiya-sistemy-monitoringa-it/)

1. **Мониторинг ИТ-инфраструктуры**

* сокращение продолжительности простоев элементов IT-структуры
* рост доступности ПО для бизнеса
* повышение качества проактивного анализа ошибок и сбоев
* увеличиваются показатели работоспособности информационных активов

1. **Мониторинг ИТ-сервисов**

* повышение уровня доступности IT-сервисов
* сокращение затрат на их обслуживание
* рост эффективности работы ИТ-службы и качества технической поддержки

**Функции мониторинга ИТ**

* своевременная регистрация проблем в работе ИТ-элементов
* обнаружение локации и характера неисправности
* выявление степени влияния неполадки на дальнейшую производительность ИТ-сервисов
* отслеживание изменений в работе IT-инфраструктуры
* предотвращение возможных сбоев

**Подходы к внедрению ИТ мониторинга**

1. От ИТ-структуры (снизу вверх)
2. От IT-сервисов (сверху вниз)

## [Лог файлы Linux по порядку](https://habr.com/ru/articles/332502/)

**Основные лог файлы**

* приложения;
* события;
* службы;
* системный.
* **/var/log/syslog** или **/var/log/messages** содержит глобальный системный журнал, в котором пишутся сообщения с момента запуска системы, от ядра Linux, различных служб, обнаруженных устройствах, сетевых интерфейсов и много другого.
* **/var/log/auth.log** или **/var/log/secure** — информация об авторизации пользователей, включая удачные и неудачные попытки входа в систему, а также задействованные механизмы аутентификации.
* **/var/log/dmesg** — драйвера устройств. Одноименной командой можно просмотреть вывод содержимого файла. Размер журнала ограничен, когда файл достигнет своего предела, старые сообщения будут перезаписаны более новыми. Задав ключ --level= можно отфильтровать вывод по критерию значимости.
* **/var/log/alternatives.log** — Вывод программы update-alternatives, в котором находятся символические ссылки на команды или библиотеки по умолчанию.
* **/var/log/anaconda.log** — Записи, зарегистрированные во время установки системы.
* **/var/log/audit** — Записи, созданные службой аудита auditd.
* **/var/log/boot.log** — Информация, которая пишется при загрузке операционной системы.
* **/var/log/cron** — Отчет службы crond об исполняемых командах и сообщения от самих команд.
* **/var/log/cups** — Все, что связано с печатью и принтерами.
* **/var/log/faillog** — Неудачные попытки входа в систему. Очень полезно при проверке угроз в системе безопасности, хакерских атаках, попыток взлома методом перебора. Прочитать содержимое можно с помощью команды faillog.
* **var/log/kern.log** — Журнал содержит сообщения от ядра и предупреждения, которые могут быть полезны при устранении ошибок пользовательских модулей, встроенных в ядро.
* **/var/log/maillog/** или **/var/log/mail.log** — Журнал почтового сервера, используемого на ОС.
* **/var/log/pm-powersave.log** — Сообщения службы экономии заряда батареи.
* **/var/log/samba/** — Логи файлового сервера Samba, который используется для доступа к общим папкам Windows и предоставления доступа пользователям Windows к общим папкам Linux.
* **/var/log/spooler** — Для представителей старой школы, содержит сообщения USENET. Чаще всего бывает пустым и заброшенным.
* **/var/log/Xorg.0.log** — Логи X сервера. Чаще всего бесполезны, но если в них есть строки начинающиеся с EE, то следует обратить на них внимание.
* **/var/log/yum.log** — Для программ, установленных с помощью Yum в RedHat Linux.
* **/var/log/emerge.log** — Для ebuild-ов установленных из Portage с помощью emerge в Gentoo Linux.
* **/var/log/dpkg.log** — Для программ, установленных с помощью dpkg в Debian Linux и всем семействе родственных дистрибутивах.
* **/var/log/lastlog** — Последняя сессия пользователей. Прочитать можно командой last.
* **/var/log/tallylog** — Аудит неудачных попыток входа в систему. Вывод на экран с помощью утилиты pam\_tally2.
* **/var/log/btmp** — Еже один журнал записи неудачных попыток входа в систему. Просто так, на всякий случай, если вы еще не догадались, где следует искать следы активности взломщиков.
* **/var/log/utmp** — Список входов пользователей в систему на данный момент.
* **/var/log/wtmp** — Еще один журнал записи входа пользователей в систему. Вывод на экран командой utmpdump.
* **/var/log/mysql/** — Лог базы данных MySQL.
* **/var/log/httpd/** или **/var/log/apache2/** — Лог веб сервера Apache, журнал доступа находится в access\_log, а ошибки — в error\_log.
* **/var/log/lighthttpd/** — Лог веб сервера lighttpd.

## [Bash-скрипты](https://habr.com/ru/companies/ruvds/articles/325522/)

Вы можете запустить скрипт любым из указанных способов:

* sh show\_all.sh
* bash show\_all.sh
* ./show\_all.sh

#!/bin/bash

**Переменные среды**

$HOME

**Пользовательские переменные**

$grade

**Подстановка команд**

mydir=`pwd`

mydir=$(pwd)

**Математические операции**

$((a+b))

**Управляющая конструкция if-then**

if команда

then

команды

fi

**Управляющая конструкция if-then-else**

if команда

then

команды

else

команды

fi

**Сравнение чисел**

n1 **-eq** n2 Возвращает истинное значение, если n1 равно n2.  
n1 **-ge** n2 Возвращает истинное значение, если n1больше или равно n2.  
n1 **-gt** n2 Возвращает истинное значение, если n1 больше n2.  
n1 **-le** n2 Возвращает истинное значение, если n1меньше или равно n2.  
n1 **-lt** n2 Возвращает истинное значение, если n1 меньше n2.  
n1 **-ne** n2 Возвращает истинное значение, если n1не равно n2.

#!/bin/bash

val1=6

if [ $val1 -gt 5 ]

then

echo "The test value $val1 is greater than 5"

else

echo "The test value $val1 is not greater than 5"

fi

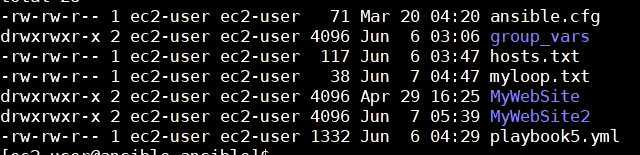
**Сравнение строк**

**str1 = str2** Проверяет строки на равенство, возвращает истину, если строки идентичны.  
**str1 != str2** Возвращает истину, если строки не идентичны.  
**str1 < str2** Возвращает истину, если str1 меньше, чем str2.  
**str1 > str2** Возвращает истину, если str1 больше, чем str2.  
**-n str1** Возвращает истину, если длина str1 больше нуля.  
**-z str1** Возвращает истину, если длина str1 равна нулю.

**Проверки файлов**

**-d file** Проверяет, существует ли файл, и является ли он директорией.  
**-e file** Проверяет, существует ли файл.  
**-f file** Проверяет, существует ли файл, и является ли он файлом.  
**-r file** Проверяет, существует ли файл, и доступен ли он для чтения.  
**-s file** Проверяет, существует ли файл, и не является ли он пустым.  
**-w file** Проверяет, существует ли файл, и доступен ли он для записи.  
**-x file** Проверяет, существует ли файл, и является ли он исполняемым.  
**file1 -nt file2**Проверяет, новее ли file1, чем file2.  
**file1 -ot file2**Проверяет, старше ли file1, чем file2.  
**-O file** Проверяет, существует ли файл, и является ли его владельцем текущий пользователь.  
**-G file** Проверяет, существует ли файл, и соответствует ли его идентификатор группы идентификатору группы текущего пользователя.

## Ansible



ansible-inventory –list - список хостов с переменными

[**7-Ansible - Запуск Ad-Hoc Команд**](https://www.youtube.com/watch?v=3eRQDA7Y-lA&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=7)

**ansible all -m setup** - вывод инфы о хостах в группе

**ansible all -m shell -a “<bash command>”** - запуск shell команд для группы

**ansible all -m command -a “<command>”** - запуск команд (без переменных и операторов перенаправления потока)

**ansible all -m copy -a “src=privet.txt dest=/home mod=777” -b** - копирование файла на хосты группы с изменением разрешений, с правами sudo (-b)

**ansible all -m file -a “path=/home/privet.txt state=absent” -b** - удаление файла

**ansible all -m get\_url -a “url= https://\*\*\* dist=home” -b** - скачать файл

**ansible all -m yum -a “name=stress state=installed(later)” -b** - установка приложения

**ansible all -m yum -a “name=stress state=removed” -b** - удалить приложение

**ansible all -m uri -a “” url=\*\*\*”** - проверить доступность узла

**ansible all -m uri -a “” url=\*\*\* return\_content=yes”** - проверить доступность узла с контентом

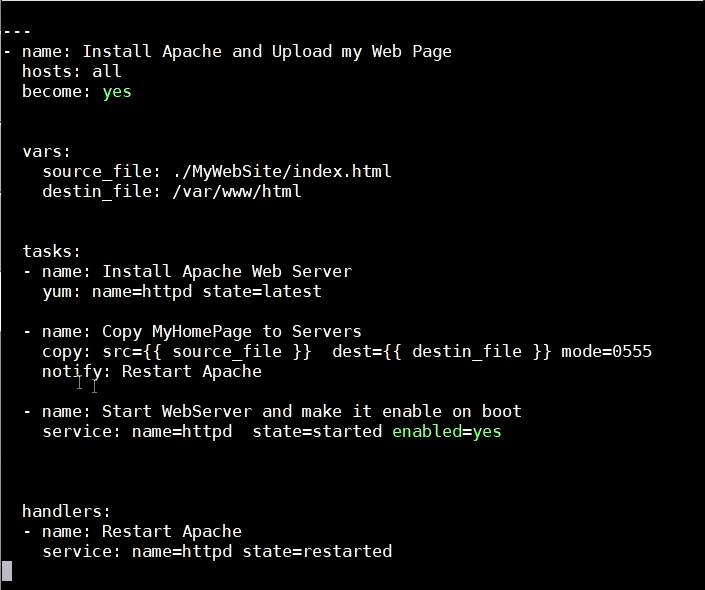
**ansible all -m service -a "name=\*\*\* state=started enabled=yes" -b** - запустить приложение и добавить в автозагрузку

**ansible all -m all -m shell -a “<command>”** [**-v**/-vv/-vvv/-vvvv/-vvvvv] - verbose

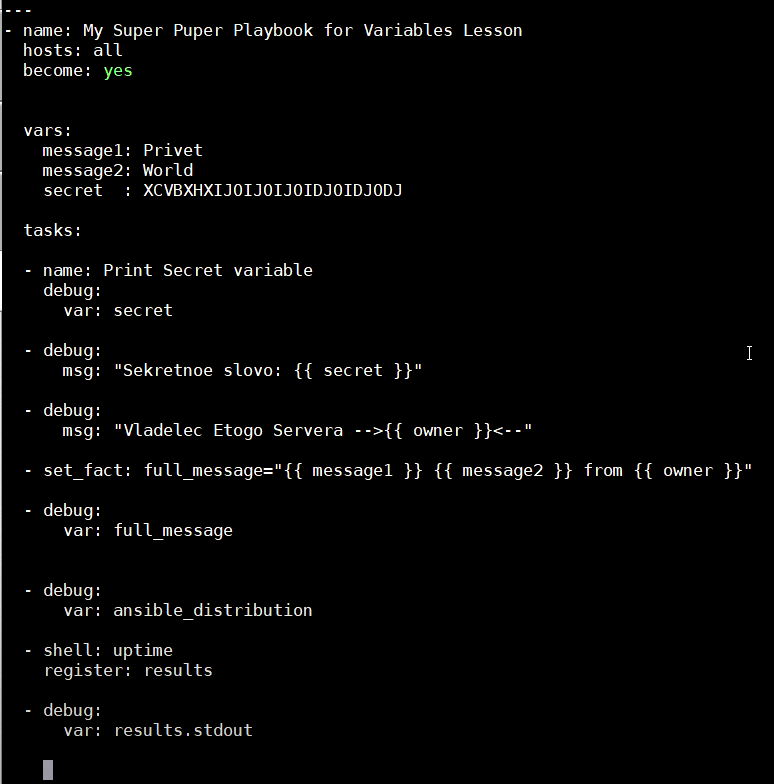
**ansible-doc -l | grep \*\*\*** - поиск по доке

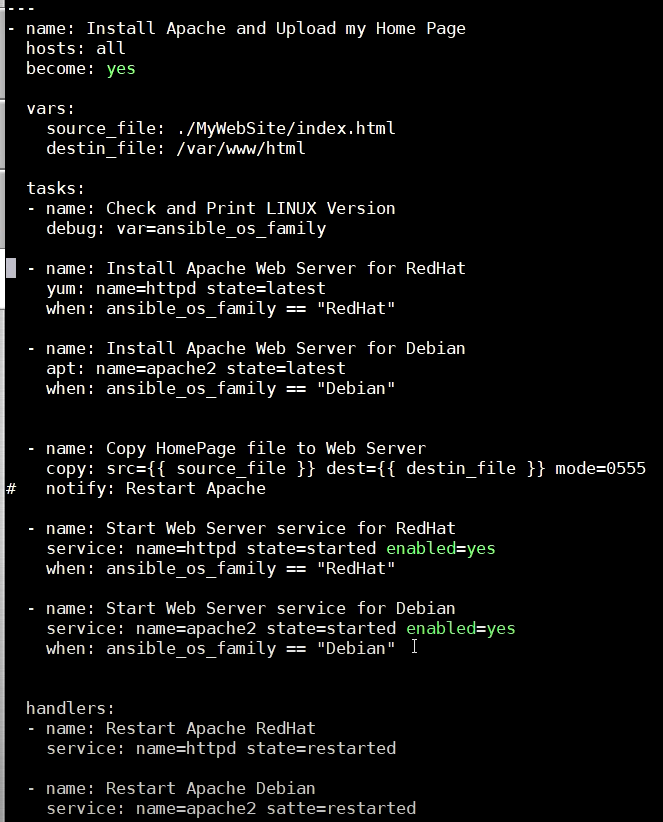
**10-Ansible - Первые** [**Pla****ybook**](https://www.youtube.com/watch?v=5VjcJNQ7nlI&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=10)

**ansible-playbook playbook.yml** - запуск плейбука

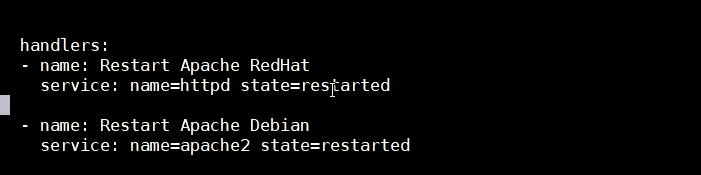
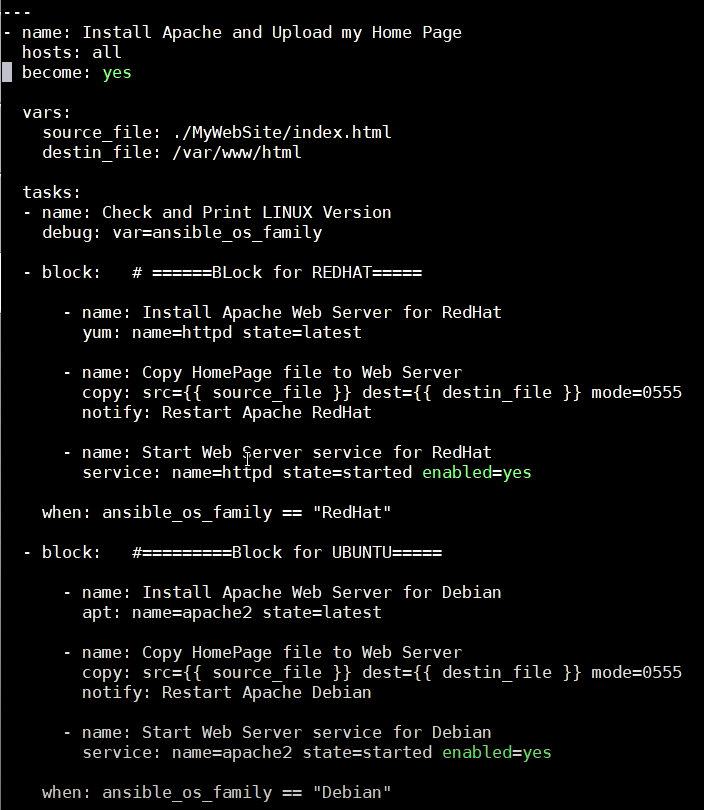


[**11-Ansible - Переменные - Debug, Set\_****f****act, Register**](https://www.youtube.com/watch?v=-vuZdaMdX4I&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=11)

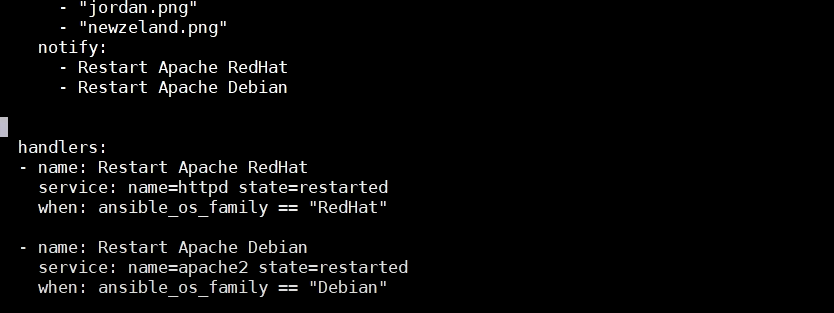
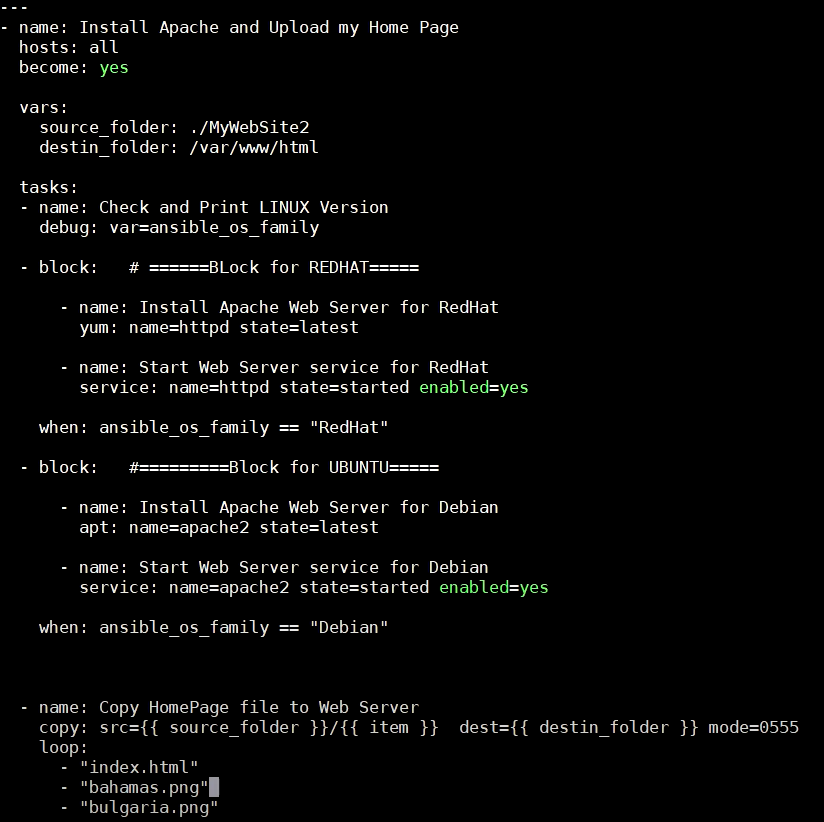




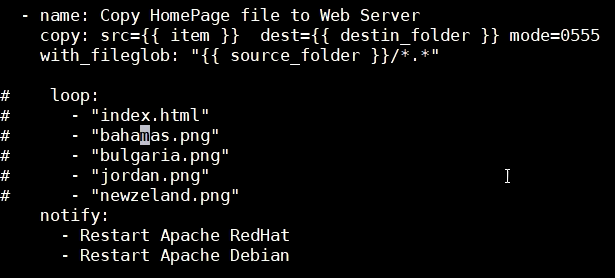
[**12-Ansible - Блоки и Условия – Block-When**](https://www.youtube.com/watch?v=QMsFDxfUYKQ&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=12)



[**13-Ansible - Циклы – Loop, With\_Items, Until, With\_fileglob**](https://www.youtube.com/watch?v=O6ZK7qkUwVE&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=13)

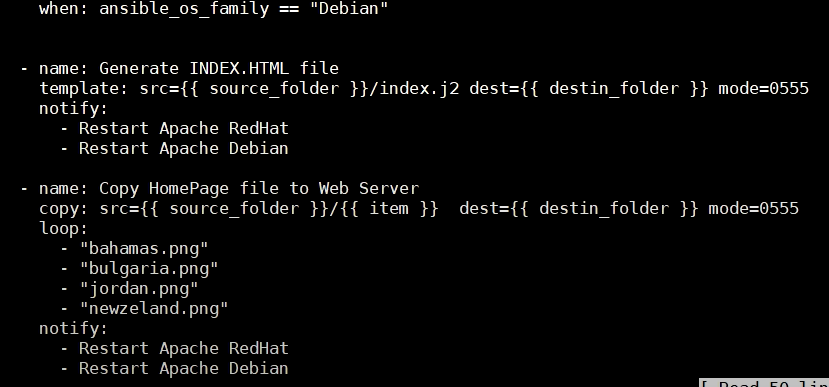
-

or



[**14-Ansible - Шаблоны - Jinja Template**](https://www.youtube.com/watch?v=DmY6v3Ccw3w&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=14)

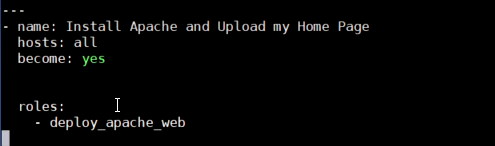
**template**  – замена переменных в файле



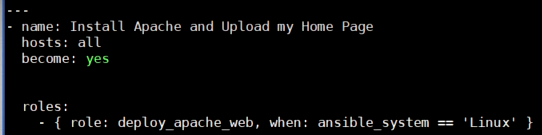
[**15-Ansible - Создание Ролей - Roles**](https://www.youtube.com/watch?v=9pHMZnb3JDQ&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=15)

**ansible-galaxy init <role\_name>** - создание роли (необходимо предварительно создать папку **roles**)

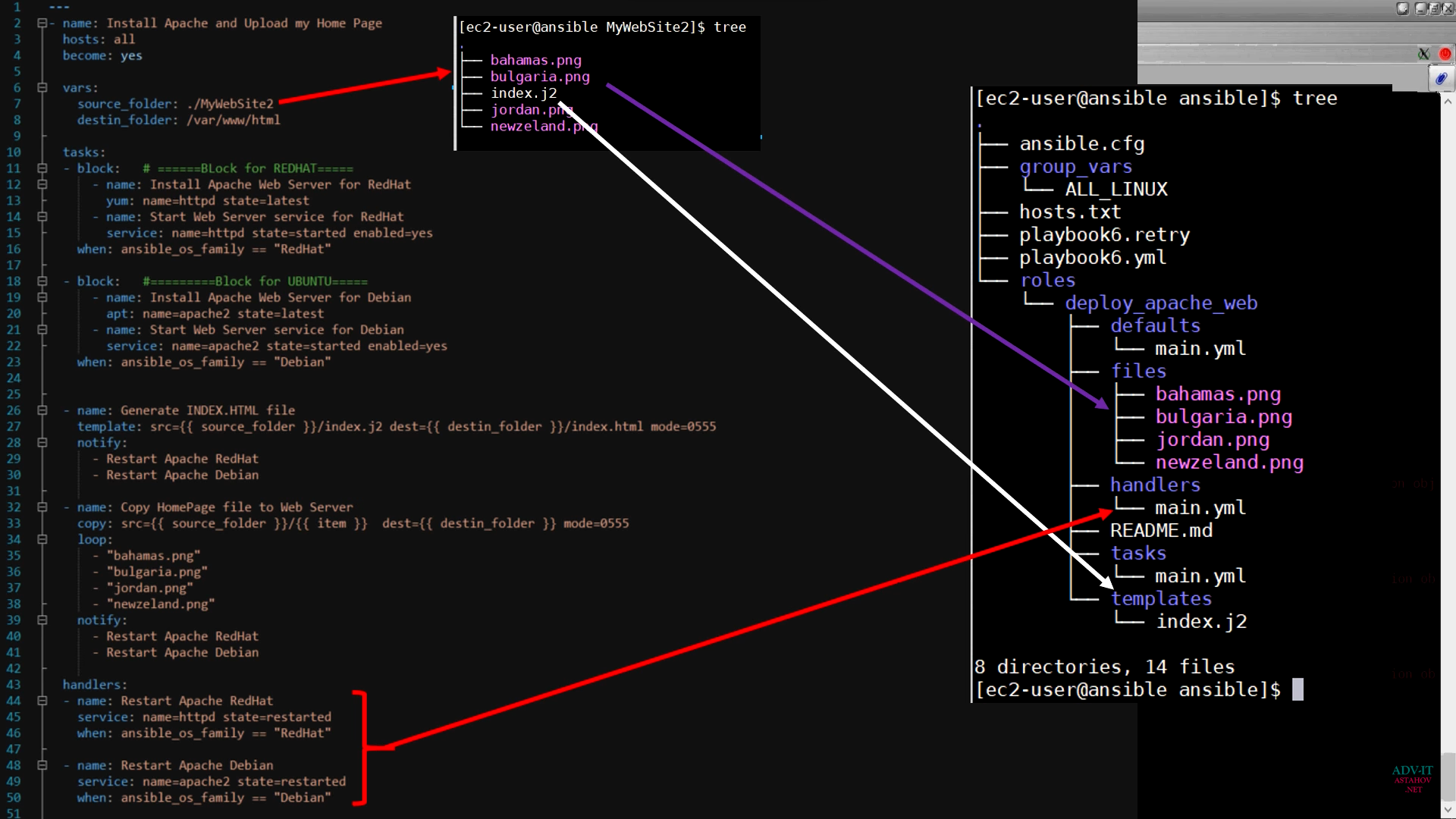
файлы раскидываются по созданным папкам, плейбук изменяется до



или



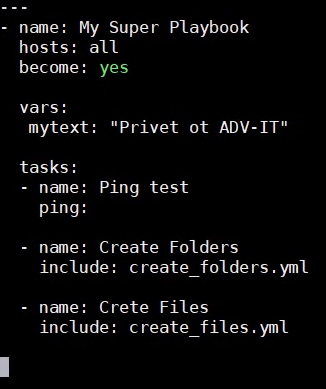
если нужно условие запуска



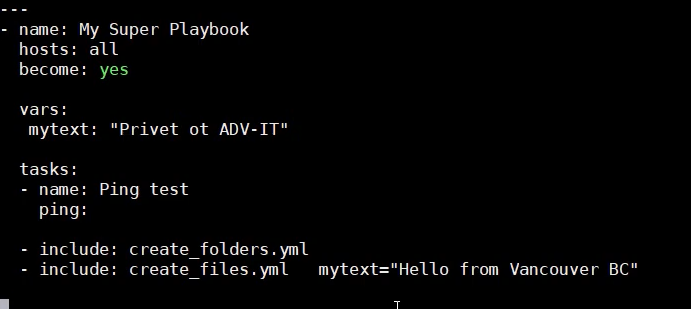
[**16-Ansible - Внешние переменные - extra-vars**](https://www.youtube.com/watch?v=HGkb5hXDwvc&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=16)

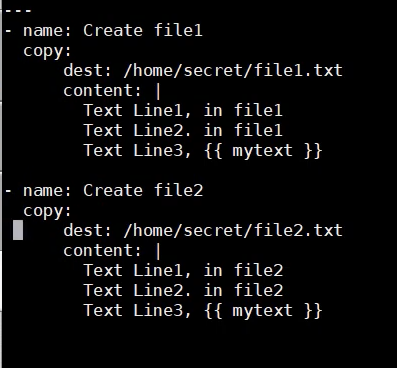
**ansible-playbook playbook.yml –extra-vars “<ключ>=<значение>”** - запуск плейбука с переменными

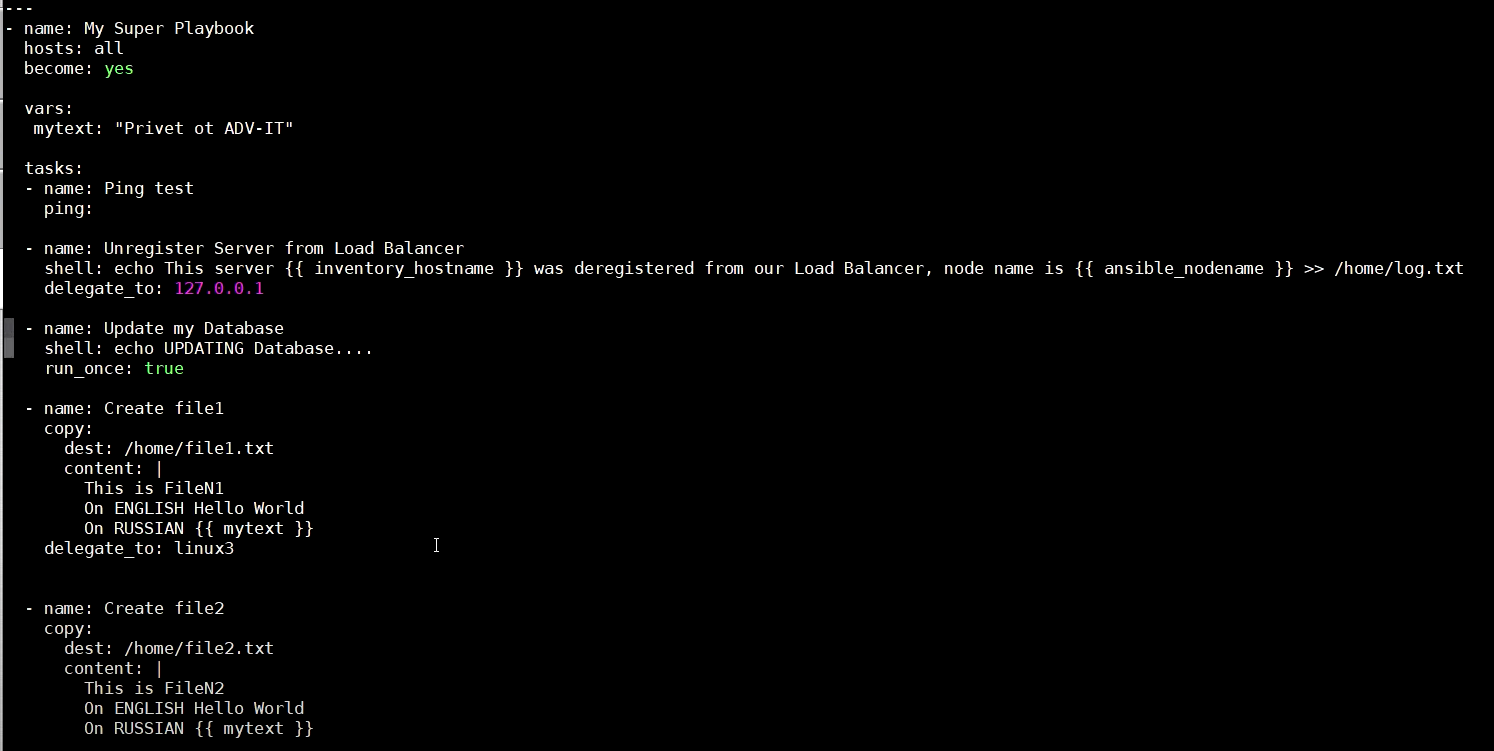
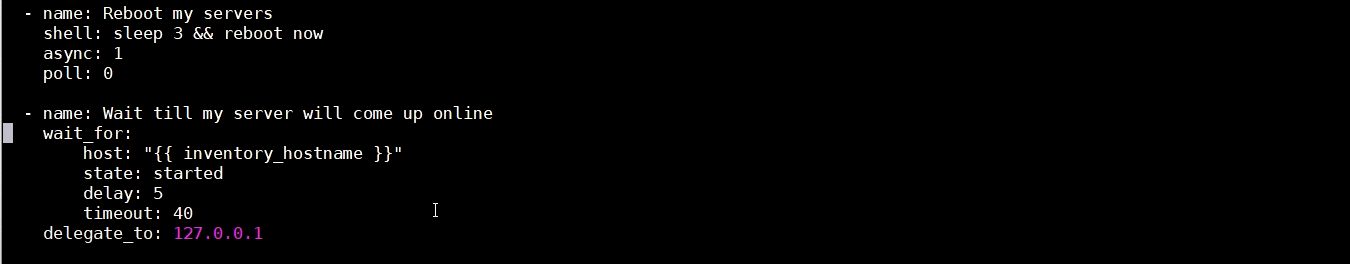
[**17-Ansible - Использование Import, Include**](https://www.youtube.com/watch?v=Yk623MKAw5I&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=17)



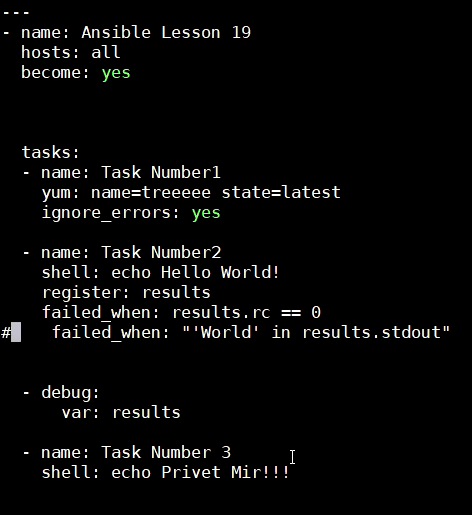
или

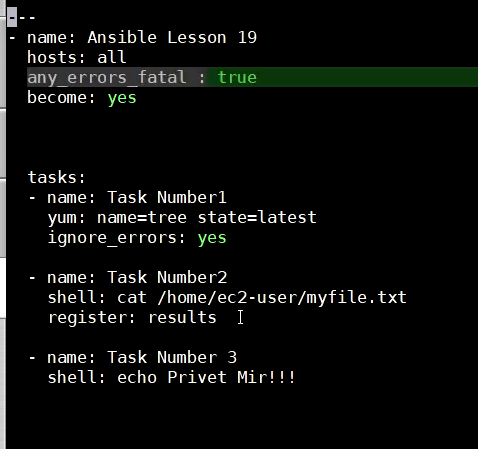




[18-Ansible - Перенаправление выполнения Task из Playbook на определённый сервер - delegate\_to](https://www.youtube.com/watch?v=yjwcbeL9S10&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=18) 

[19-Ansible - Перехват и Контроль ошибок](https://www.youtube.com/watch?v=lO_pMa_Gbhs&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=19)





[20-Ansible - Хранение Секретов - ansible-vault](https://www.youtube.com/watch?v=20g9BNilDvg&list=PLg5SS_4L6LYufspdPupdynbMQTBnZd31N&index=20)

**ansible-vault create** <название\_файла> - создание зашифрованного файла

**ansible-vault view** <название\_файла> - просмотр файла

**ansible-vault edit** <название\_файла> - редактирование файла

**ansible-vault rekey** <название\_файла> - смена пароля

**ansible-vault encrypt** <название\_файла> - зашифровать файл

**ansible-vault decrypt** <название\_файла> - расшифровать файл

**ansible-playbook** <плейбук**> --ask-vault-pass** - запуск плейбука

**ansible-playbook** <плейбук> **-- vault-password-file** <название\_файла> - запуск плейбука с паролем в файле

**ansible-vault encrypt-string**  - зашифровать строку

**ansible-vault encrypt-string –** **stdin-name** <переменная>- зашифровать строку и вывести в переменную

## [Процессы и потоки в Linux](https://fileenergy.com/linux/protsessy-i-potoki-v-linux-obyasnenie-kak-linux-upravlyaet-protsessami-i-potokami)

Каждый **процесс** имеет свой уникальный идентификатор (**PID**), свое адресное пространство, свои ресурсы и свои права доступа.

**Поток** — это часть процесса, которая может выполняться параллельно с другими потоками того же процесса. Каждый **поток** имеет свой уникальный идентификатор (**TID**), свой счетчик команд, свой стек и свои регистры. Но все потоки одного процесса разделяют общее адресное пространство и ресурсы процесса.

Linux поддерживает два типа **многозадачности**: вытесняющую и кооперативную. **Вытесняющая** многозадачность означает, что операционная система сама определяет, когда и как долго каждый процесс или поток может использовать процессор. **Кооперативная** многозадачность означает, что процессы или потоки сами отдают управление процессору другим задачам.

**Создание процессов. Корневой процесс с PID(1) – init()**

через системный вызов **fork**() или через системный вызов **exec**()

Системный вызов **fork**() создает новый процесс, который является точной копией родительского процесса.

Системный вызов **exec**() заменяет текущий процесс новым процессом, который загружает и выполняет другой исполняемый файл.

Например, когда мы вводим в терминале команду **ls**, чтобы просмотреть список файлов в текущей директории, происходит следующее:

Терминал (**shell**) создает новый дочерний процесс с помощью системного вызова **fork**().  
Дочерний процесс заменяет себя программой **ls** с помощью системного вызова **exec**().  
Программа **ls** выполняет свою задачу и выводит список файлов на экран.  
Программа **ls** завершается и возвращает управление родительскому процессу (**терминалу**).

**Выполнение процессов**

* Запуск
* Готовность
* Ожидание
* Завершение

**Управление процессами**

* Системный вызов wait()
* Системный вызов kill() (SIGTERM (запрос на завершение), SIGKILL (принудительное завершение), SIGSTOP (приостановка) и SIGCONT (продолжение).)
* ps
* top
* htop

**Потоки в Linux**

**Создание потоков**

через библиотеку **POSIX** Threads (Pthreads) или через системный вызов **clone**()

**Выполнение потоков**

* Запуск
* Готовность
* Ожидание

**Управление потоками**

* **pthread\_join**() Эта функция позволяет одному потоку ждать завершения другого потока и получать его значение возврата.
* **pthread\_cancel**(). Эта функция позволяет одному потоку отправлять запрос на завершение другому потоку.
* **pthread\_kill**(). Эта функция позволяет одному потоку отправлять сигнал другому потоку

## [Управление памятью в Linux](https://habr.com/ru/articles/793232/)

Ключевые понятия управления памятью в Linux:

* **Виртуальная память (Virtual Memory)**  
  Linux использует концепцию виртуальной памяти, которая создает иллюзию наличия у каждого процесса своего личного пространства памяти. Виртуальная память позволяет системе исполнять код приложений, используя больший объем памяти, чем физически доступно. Это достигается путем сброса неиспользуемых блоков памяти приложений на диск
* **Система страница (Paging)**  
  Физическая и виртуальная память разделены на блоки фиксированного размера, которые называются страницами. Система страниц позволяет эффективно управлять памятью и активирует механизм обмена данными между ОЗУ и диском (swap)
* **Выделение памяти (Memory Allocation)**При выполнении команд процессам требуется память. За выделение процессам подходящих блоков памяти отвечает соответствующий диспетчер. Память выделяется из свободной физической памяти. При необходимости физическая память освобождается сбросом неактивных страниц на диск
* **Пространство ядра и пользовательское пространство (Kernel Space and User Space)**Память в Linux подразделяется на пространство ядра и пользовательское пространство. Пространство ядра зарезервировано для исполнения кода ядра, расширений ядра и большинства драйверов устройств. Пользовательское пространство – это область памяти, с которой работают все пользовательские приложения
* **Кеширование (Caching)**  
  Linux использует несколько механизмов кеширования для улучшения производительности системы. Так, например, кеш страниц (page cache) используется для кеширования файлов, читаемых с диска, а кеш буфера (buffer cache) используется для управления операциями записи на диск
* **Чрезмерное выделение памяти (Memory Overcommit)**Linux позволяет выделить процессам больше памяти, чем реально доступно. Эта концепция известна как memory overcommit. Она позволяет большему количество процессов выполняться одновременно при условии, что процессы не используют всю выделенную им память

Как работает виртуальная память

* **Сегментация памяти (Memory Segmentation)**Когда приложение запускается, ему выделяется диапазон адресов виртуальной памяти. Эта память разделена на блоки (chunks), называемые страницами.
* **Таблица страниц (Page Tables)**Операционная система сохраняет для каждого процесса структуру данных, которая называется таблицей страниц. Таблица страниц сопоставляет адреса страниц виртуальной памяти с адресами страниц физической памяти.
* **Доступ к памяти (Memory Access)**Когда приложение выполняет запрос на чтение или запись к памяти, ЦПУ использует таблицу страниц для трансляции адреса виртуальной памяти в адрес физической.
* **Подкачка и запрос страниц (Swapping and Demand Paging)**Если вся физическая память использована, а приложению требуется загрузить новую страницу в нее, то ОС может выбрать страницу физической памяти для сброса-«подкачки» (swap out) на диск. В таблицу страниц вносится пометка, что данная страница отсутствует в физической памяти. Если позднее приложение попытается получить доступ к адресу выгруженной на диск страницы, то это приведет к ошибке, и операционная система загрузит-«подкачает» (**swap in**) страницу с диска в физическую память (возможно сбросив при этом на диск другую страницу), после чего приложение сможет получить к ней доступ. Механизм загрузки страниц в физическую память при обращении к ним называется **demand paging** и позволяет общему объему виртуальной памяти для всех процессов превышать объем фактически доступной физической памяти

**Translation Lookaside Buffer (TLB)**. TLB по сути является небольшим кешем. В случае отсутствия в нем информации для трансляции адреса виртуального адреса в физический, ЦПУ обращается к таблице страниц процесса и получает физический адрес страницы, после чего добавляет в TLB эту информацию

**64**-разрядная версия Linux позволяет использовать до **128 ТБ** **виртуальной** памяти для каждого процесса. Это теоретический максимум объема виртуальных адресов памяти, к которым можно получить доступ. Для **физической** памяти, которая соответствует установленной ОЗУ в вашем компьютере, 64-разрядный Linux может обработать в общей сложности около **64 ТБ**.

Память разделена на пространство ядра и пользовательское пространство, причем пользовательское пространство занимает половину объема, что приводит к максимуму в 128 ТБ памяти на каждый процесс.

Данные, связанные с процессом, также являются частью образа процесса. Некоторые из них хранятся в регистрах, обычно представленных регистрами процессора. И существуют динамические области хранения данных ([куча](https://wiki.colobridge.net/linux/управление_памятью_в_linux_и_openvz)), выделяемые процессу по ходу работы при необходимости.

Еще у процесса есть [стек](https://wiki.colobridge.net/linux/управление_памятью_в_linux_и_openvz), содержащийся в памяти и используемый для хранения локальных переменных программы и передачи параметров. Когда процесс выполняет обращение к функции или подпрограмме, в стек отправляется новый фрейм. Одной из частей каждого фрейма является указатель на базу предыдущего фрейма, который позволяет легко вернуться из вызова функции.

В архитектуре x86\_64 возможно использовать страницы размером 4 килобайта (4096 байтов), 2 мегабайта, и (в некоторых AMD64) 1 гигабайт.

## Namespaces и Cgroups

[**Namespaces**](https://habr.com/ru/companies/selectel/articles/279281/) (пространства имен) — это абстракция (программная прослойка) над физическими ресурсами. Если раньше процессы обращались напрямую к ресурсам, то с появлением namespaces, все запросы проходят через этот дополнительный слой абстракции.

Посмотреть можно в **/proc/1/ns**

Пространство имён Что изолирует

**PID** PID процессов

**NETWORK** Сетевые устройства, стеки, порты и т.п.

**USER** ID пользователей и групп

**MOUNT** Точки монтирования

**IPC** SystemV IPC, очереди сообщений POSIX

**UTS** Имя хоста и доменное имя NIS

**CGROUP**

**TIME**

[**Cgroups**](https://habr.com/ru/companies/selectel/articles/303190/) (control group) — группа процессов Linux, на которые наложена изоляция и установлены ограничения на вычислительные ресурсы (процессорные, сетевые, ресурсы памяти, ресурсы ввода-вывода) со стороны ядра Linux.

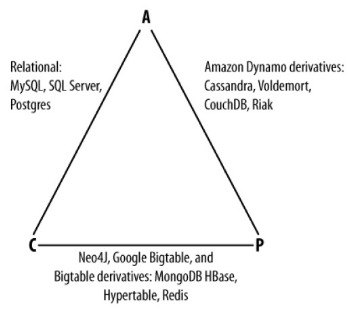
Механизм cgroups состоит из двух составных частей: ядра ([cgroup core](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/cgroup.c)) и так называемых подсистем. В ядре версии 4.4.0.21 таких подсистем 12:

* [**blkio**](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/block/blk-cgroup.c) — устанавливает лимиты на чтение и запись с блочных устройств;
* [cpuacct](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/sched/cpuacct.c) — генерирует отчёты об использовании ресурсов процессора;
* [**cpu**](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/sched/core.c) — обеспечивает доступ процессов в рамках контрольной группы к CPU;
* [cpuset](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/cpuset.c) — распределяет задачи в рамках контрольной группы между процессорными ядрами;
* [devices](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/security/device_cgroup.c) — разрешает или блокирует доступ к устройствам;
* [freezer](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/cgroup_freezer.c) — приостанавливает и возобновляет выполнение задач в рамках контрольной группы
* [hugetlb](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/mm/hugetlb_cgroup.c) — активирует поддержку больших страниц памяти для контрольных групп;
* [**memory**](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/mm/memcontrol.c) — управляет выделением памяти для групп процессов;
* [net\_cls](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/net/core/netclassid_cgroup.c) — помечает сетевые пакеты специальным тэгом, что позволяет идентифицировать пакеты, порождаемые определённой задачей в рамках контрольной группы;
* [netprio](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/net/core/netprio_cgroup.c) — используется для динамической установки приоритетов по трафику;
* [**pids**](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/cgroup_pids.c) — используется для ограничения количества процессов в рамках контрольной группы.

## [Теорема](https://habr.com/ru/articles/328792/) CAP

В CAP говорится, что в распределенной системе возможно выбрать только 2 из 3-х свойств:

* **C (consistency)** — согласованность. Каждое чтение даст вам самую последнюю запись.
* **A (availability)** — доступность. Каждый узел (не упавший) всегда успешно выполняет запросы (на чтение и запись).
* **P (partition tolerance)** — устойчивость к распределению. Даже если между узлами нет связи, они продолжают работать независимо друг от друга.



## [Реляционные БД](https://aws.amazon.com/ru/relational-database/)

**Реляционная база данных** – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая строка таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется.

**Важные аспекты реляционных БД:**

**SQL (Structured Query Language)** – основной интерфейс работы с реляционными базами данных

**Целостность данных** – это полнота, точность и единообразие данных.

**Транзакция** в базе данных – это один или несколько операторов SQL, выполненных в виде последовательности операций, представляющих собой единую логическую задачу.

**Соответствие требованиям ACID**

Для соблюдения целостности данных все транзакции в БД должны соответствовать требованиям **ACID**, то есть быть атомарными, единообразными, изолированными и надежными.

**Атомарность** – это условие, при котором либо транзакция успешно выполняется целиком, либо, если какая-либо из ее частей не выполняется, вся транзакция отменяется. **Единообразие** – это условие, при котором данные, записываемые в базу данных в рамках транзакции, должны соответствовать всем правилам и ограничениям, включая ограничения целостности, каскады и триггеры. **Изолированность** необходима для контроля над согласованностью и гарантирует базовую независимость каждой транзакции. **Надежность** подразумевает, что все внесенные в базу данных изменения на момент успешного завершения транзакции считаются постоянными.

## [Первичный ключ (primary key)](https://ru.wikipedia.org/wiki/Первичный_ключ)

**Перви́чный ключ** ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *primary key*) — в [реляционной модели данных](https://ru.wikipedia.org/wiki/Реляционная_модель_данных) один из [потенциальных ключей](https://ru.wikipedia.org/wiki/Потенциальный_ключ) [отношения](https://ru.wikipedia.org/wiki/Отношение_(реляционная_модель)), выбранный в качестве основного ключа (или ключа по умолчанию).

Если в [отношении](https://ru.wikipedia.org/wiki/Отношение_(реляционная_модель)) имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют «альтернативными».

## [Связи между таблицами базы данных](https://habr.com/ru/articles/488054/)

1. Многие ко многим.
2. Один ко многим.  
   * с обязательной связью;
   * с необязательной связью;
3. Один к одному.  
   * с обязательной связью;
   * с необязательной связью;

## Виды репликации (архивирования) БД на примере Postgres

| вид | особенности |
| --- | --- |
| Логическая | pg\_dump, pg\_dumpall, pg\_resore |
|  | - клиент к базе. требуются права на копируемые таблицы. |
|  | - дамп базы можно загрузить в более новые версии или на серверы с другой архитектурой |
|  | - восстановление большой базы будет занимать много времени по сравнению с физической репликацией |
|  | - перед восстановлением саму базу, пользователей надо создать руками |
|  | - выгрузка в текстовом формате или в арив. Доступна параллельная выгрузка в архивном формате |
| Физическая | - Копируется каталог с данными после отстановки БД |
|  | - Или создать checkpoint, сделать снепшот и восстанавливать из снепшота. |
|  | База будет считать что в момент снятия снепшота она была аварийно выключена. |
|  | Будет накатываться журнал WAL |
|  | Файлы данных, табличные пространства, журналы должны находиться на обном разделе, т.к. снимок должен |
|  | быть одновременным |
|  | - rsync + rsync --checksum |
|  | - pg\_basebackup |
| Потоковая | - архивирование wal и восстановление из базового архива + из wal |
|  | - нужен wal\_level не ниже replica и archive\_mode=on и задать archive\_command |
|  | - архивирование WAL не учитывает изменения конфигурационных файлов. Их нужно архивировать отдельно |

## [NoSQL](https://aws.amazon.com/ru/nosql/)

[Базы данных «ключ-значение»](https://aws.amazon.com/ru/nosql/key-value/) поддерживают высокую разделяемость и допускают горизонтальное масштабирование на уровне, недоступном для других типов баз данных NoSQL. Redis, ETCD

У [баз данных документов](https://aws.amazon.com/ru/documentdb/) тот же формат модели документа, который разработчики используют в коде своих приложений.  MongoDB

[Графовые базы данных](https://aws.amazon.com/ru/neptune/) предназначены для упрощения разработки и запуска приложений, работающих с наборами тесно связанных данных.

В то время как другие нереляционные базы данных хранят данные на дисках или SSD, [хранилища данных в памяти](https://aws.amazon.com/ru/nosql/in-memory/) предназначены для устранения необходимости доступа к дискам.

[База данных поисковой системы](https://aws.amazon.com/ru/nosql/search/) – это тип нереляционной базы данных, предназначенной для поиска содержимого информации, например журналов выходных данных приложений, используемых разработчиками для устранения неполадок.

## Идеальный [пайплайн](https://habr.com/ru/companies/rabota/articles/560922/)

* сканить код;
* билдить код;
* тестить код;
* деплоить приложение;
* тестить приложение;
* делать Merge;
* просить других людей подтверждать MR через code review.

## [Инкапсуляция](https://blog.skillfactory.ru/glossary/inkapsulyacziya/)

Инкапсуляция в программировании — это принцип, согласно которому внутреннее устройство сущностей нужно объединять в специальной «оболочке» и скрывать от вмешательств извне. Доступ к объектам возможен через специальные открытые методы, а напрямую обратиться к их содержимому нельзя.

## [Kubernetes](https://habr.com/ru/articles/725640/)

Типовой состав ПО рабочего узла включает в себя (*Рисунок 1*):

1. Служебные компоненты Kubernetes: агент управления узлом [kubelet](https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/), узловой прокси [kube-proxy](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/)
2. [Сетевой плагин (Container Network Interface, CNI plugin)](https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/).
3. Контейнерный движок: cri-o, containerd или Docker + cri-dockerd plugin.
4. Рабочие нагрузки (workloads), то есть сами контейнеры, из-за которых все и затевалось. Однако, здесь важно уточнить один существенный момент — минимальной единицей управления рабочей нагрузкой в Kuberbetes является ["*под*" (pod)](https://kubernetes.io/docs/concepts/workloads/pods/), состоящий из одного (как правило) или нескольких контейнеров.

Состав ПО управляющих узлов (*Рисунок 2*) дополнительно включает в себя:

1. Управляющие компоненты Kubernetes: планировщик [kube-scheduler](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/), базовый демон управления [kube-controller-manager](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/), REST API сервер управления [kube-apiserver](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/).
2. Отказоустойчивое хранилище [etcd](https://etcd.io/).
3. Балансировщик нагрузки (для случаев использования нескольких управляющих узлов в кластере).

[**Основы Kubernetes**](https://habr.com/ru/articles/258443/)

**Концепции Kubernetes**

**Nodes** ([node.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/node.md)): Нода это машина в кластере Kubernetes.  
**Pods** ([pods.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/pods.md)): Pod это группа контейнеров с общими разделами, запускаемых как единое целое.  
**Replication Controllers** ([replication-controller.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/replication-controller.md)): replication controller гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.  
**Services** ([services.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/services.md)): Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.  
**Volumes** ([volumes.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/volumes.md)): Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.  
**Labels** ([labels.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/labels.md)): Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов.  
**Kubectl** Command Line Interface ([kubectl.md](https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/kubectl.md)): kubectl интерфейс командной строки для управления Kubernetes.

## [Работа с Docker](https://habr.com/ru/articles/659049/)

Собрать контейнер без докера – **kanico, podman, bildah**

Запустить контейнер – **cri-o, docker.d, <=container.d**

## [Создание собственного образа Docker](https://www.dmosk.ru/miniinstruktions.php?mini=docker-self-image)

[**8 best practices по использованию Docker в продакшине**](https://bool.dev/blog/detail/top8-docker-best-practices)

* 1. Используйте **официальный Docker image** в качестве базового image для приложения, если это возможно
  2. Используйте **конкретные версии Docker image**
  3. Используйте **официальные Image'ы небольшого размера**
  4. Оптимизация кэширования для image слоев при image build (стараться не изменять низлежащие слои)
  5. Используйте **.dockerignore file**
  6. Используйте **Multi-Stage билды**
  7. Используйте **наименее привилегированного пользователя**
  8. **Сканируйте свои Docker image** на наличие уязвимостей в системе безопасности

[**Дюжина инструкций Dockerfile**](https://habr.com/ru/companies/ruvds/articles/439980/)

**FROM** — задаёт базовый (родительский) образ.

**LABEL** — описывает метаданные. Например — сведения о том, кто создал и поддерживает образ.

**ENV** — устанавливает постоянные переменные среды.

**RUN** — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.

**COPY** — копирует в контейнер файлы и папки.

**ADD** — копирует файлы и папки в контейнер, может распаковывать локальные .tar-файлы.

**CMD** — описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.

**WORKDIR** — задаёт рабочую директорию для следующей инструкции.

**ARG** — задаёт переменные для передачи Docker во время сборки образа.

**ENTRYPOINT** — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.

**EXPOSE** — указывает на необходимость открыть порт. !!!Не открывает порт!!!

**VOLUME** — создаёт точку монтирования для работы с постоянным хранилищем.

## [Сетевые драйверы в Docker](https://habr.com/ru/companies/otus/articles/730798/)

Для того, чтобы работать с сетью Docker необходимы драйверы. Имеется несколько драйверов по умолчанию, которые обеспечивают основной функционал по работе с сетью:

* **none**: отключение всех сетевых ресурсов.
* **bridge**: сетевой драйвер по умолчанию. По сути, это мост между контейнером и хостовой машиной. Мостовые сети обычно используются, когда приложения выполняются в автономных контейнерах, которые должны взаимодействовать друг с другом.
* **host**: для автономных контейнеров устраняется сетевая изолированность между контейнером и хостом Docker и напрямую используются сетевые ресурсы хоста.
* **overlay**: наложенные сети соединяют несколько демонов Docker.
* **macvlan**: сети Macvlan позволяют присваивать контейнеру MAC-адрес, благодаря чему он выглядит как физическое устройство в сети.

**docker network ls**

**docker0** – дефолтный сетевой бридж для контейнеров

**tmpfs** - временное файловое хранилище. Это некая специально отведённая область в оперативной памяти компьютера **--tmpfs**

Основные команды

# справочная информация  
docker --help # список доступных команд  
docker <command> --help # информация по команде  
   
docker --version # версия Docker  
docker info # общая информация о системе  
   
# работа с образами  
docker search debian # поиск образов по ключевому слову debian  
   
docker pull ubuntu # скачивание последней версии (тег по умолчанию latest) официального образа ubuntu (издатель не указывается) из репозитория по умолчанию docker.io/library  
docker pull prom/prometheus # скачивание последней версии (latest) образа prometheus от издателя prom из репозитория docker.io/prom  
docker pull docker.io/library/ubuntu:18.04 # скачивание из репозитория docker.io официального образа ubuntu с тегом 18.04  
   
docker images # просмотр локальных образов  
   
docker rmi <image\_name>:<tag> # удаление образа. Вместо <image\_name>:<tag> можно указать <image\_id>. Для удаления образа все контейнеры на его основе должны быть как минимум остановлены  
docker rmi $(docker images -aq) # удаление всех образов  
   
# работа с контейнерами  
docker run hello-world # Hello, world! в мире контейнеров  
docker run -it ubuntu bash # запуск контейнера ubuntu и выполнение команды bash в интерактивном режиме  
docker run --name docker-getting-started --publish 8080:80 docker/getting-started # запуск контейнера gettind-started с отображением (маппингом) порта 8080 хоста на порт 80 внутрь контейнера  
docker run --detach --name mongodb docker.io/library/mongo:4.4.10 # запуск контейнера mongodb с именем mongodb в фоновом режиме. Данные будут удалены при удалении контейнера!  
   
docker ps # просмотр запущенных контейнеров  
docker ps -a # просмотр всех контейнеров (в том числе остановленных)  
docker stats --no-stream # просмотр статистики  
   
docker start alpine # создание контейнера из образа alpine  
   
docker start <container\_name> # запуск созданного контейнера. Вместо <container\_name> можно указать <container\_id>  
docker start $(docker ps -a -q) # запуск всех созданных контейнеров  
   
docker stop <container\_name> # остановка контейнера. Вместо <container\_name> можно указать <container\_id>  
docker stop $(docker ps -a -q) # остановка всех контейнеров  
   
docker rm <container\_name> # удаление контейнера. Вместо <container\_name> можно указать <container\_id>  
docker rm $(docker ps -a -q) # удаление всех контейнеров  
   
# система  
docker system info # общая информация о системе (соответствует docker info)  
docker system df # занятое место на диске  
docker system prune -af # удаление неиспользуемых данных и очистка диска

## Kubernetes

[**Ресурсы рабочей нагрузки**](https://kubernetes.io/ru/docs/concepts/workloads/controllers/)

[Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)

[StatefulSet](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/)

[DaemonSet](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/)

**DaemonSet –** сущностьдля запуска ровно одного инстанса приложения на каждой ноде кластера

**Kubernetes Master:**

Scheduler

Controllers (controller manager, cloud controller manager)

Etcd

**Api-Server**

**Worker Nodes**

Kubelet

 /var/log/containers/ and /var/log/pods/

Kube-proxy

pod

Доступ к поду:

**NodePort**

**LoadBalancer** (облако), для bare metal – [**MetalLB**](https://habr.com/ru/companies/slurm/articles/443110/)(реализация балансировщика нагрузки для кластеров Kubernetes на «голом железе» со стандартными протоколами маршрутизации)

**Probes**

Liveness Probe

Readiness Probe

Startup Probe

**Services**

ClusterIP

NodePort

LoadBalancer

ExternalName

ExternalIPs

Headless (для StatefullSet)

ResourceQuota

LimitRange

# Get-команды с основном выводом  
kubectl get services # Вывести все сервисы в пространстве имён  
kubectl get pods --all-namespaces # Вывести все поды во всех пространств имён  
kubectl get pods -o wide # Вывести все поды в текущем пространстве имён с подробностями  
kubectl get deployment my-dep # Вывести определённое развёртывание  
kubectl get pods # Вывести все поды в пространстве имён  
kubectl get pod my-pod -o yaml # Получить информацию по поду в формате YAML  
   
# Посмотреть дополнительные сведения команды с многословным выводом  
kubectl describe nodes my-node  
kubectl describe pods my-pod  
   
# Вывести сервисы, отсортированные по имени  
kubectl get services --sort-by=.metadata.name  
   
# Вывести поды, отсортированные по количеству перезагрузок  
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'  
   
# Вывести постоянные тома (PersistentVolumes), отсортированные по емкости  
kubectl get pv --sort-by=.spec.capacity.storage  
   
# Получить метку версии всех подов с меткой app=cassandra  
kubectl get pods --selector=app=cassandra -o \  
 jsonpath='{.items[\*].metadata.labels.version}'  
   
# Получить все рабочие узлы (с помощью селектора исключаем узлы с меткой 'node-role.kubernetes.io/master')  
kubectl get node --selector='!node-role.kubernetes.io/master'  
   
# Получить все запущенные поды в пространстве имён  
kubectl get pods --field-selector=status.phase=Running  
   
# Получить внешние IP-адреса (ExternalIP) всех узлов  
kubectl get nodes -o jsonpath='{.items[\*].status.addresses[?(@.type=="ExternalIP")].address}'  
   
# Вывести имена подов, принадлежащие к определённому RC  
# Использование команды "jq" помогает упросить поиск в jsonpath, подробнее смотрите на сайте https://stedolan.github.io/jq/  
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to\_entries | .[] | "\(.key)=\(.value),"')%?}  
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})  
   
# Показать метки всех подов (или любого другого объекта Kubernetes, которым можно прикреплять метки)  
kubectl get pods --show-labels  
   
# Получить готовые узлы  
JSONPATH='{range .items[]}{@.metadata.name}:{range @.status.conditions[]}{@.type}={@.status};{end}{end}' \  
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"  
   
# Вывод декодированных секретов без внешних инструментов  
kubectl get secret my-secret -o go-template='{{range $k,$v := .data}}{{"### "}}{{$k}}{{"\n"}}{{$v|base64decode}}{{"\n\n"}}{{end}}'  
   
# Вывести все секреты, используемые сейчас в поде.  
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq  
   
# Вывести все идентификаторы (containerID) контейнеров инициализации (initContainers) во всех подах.  
# Это полезно при очистке остановленных контейнеров, не удаляя при этом контейнеры инициализации.  
kubectl get pods --all-namespaces -o jsonpath='{range .items[].status.initContainerStatuses[]}{.containerID}{"\n"}{end}' | cut -d/ -f3  
   
# Вывести события, отсортированные по временной метке  
kubectl get events --sort-by=.metadata.creationTimestamp  
   
# Сравнить текущее состояние кластера с состоянием, в котором находился бы кластер в случае применения манифеста.  
kubectl diff -f ./my-manifest.yaml

**Информация о кластере**

Вывод информации о версии клиента и сервера

kubectl version

Вывод поддерживаемых ресурсов API на сервере

kubectl api-resources

Вывод поддерживаемых версий API на сервере, в виде «group/version»

kubectl api-versions

Получение информации о кластере

kubectl cluster-info

Получение списка узлов в кластере

kubectl get nodes

Получение информации о главном узле

kubectl get nodes master -o wide

Получение подробной информации о главных узлах

kubectl describe nodes master

**Информация о конфигурации**

Отображать всех настроек kubeconfig

kubectl config view

Просмотр текущего контекста

kubectl config current-context

Показать кластеры, определенные в kubeconfig

kubectl config get-clusters

Описать один или много контекстов

kubectl config get-contexts

**Пространства имен**

Получить все пространства имен

kubectl get namespaces

Получить информацию о пространстве имен в формате yaml

kubectl get namespaces -...o yaml

Описать пространство имен по умолчанию

kubectl **describe** namespace default

Создать новое пространство имен

kubectl create namespace my-namespace

Удалить пространство имен

kubectl delete namespace my-namespace

**Модули**

Получить модули из текущего пространства имен

kubectl get pods

Получать модули из всех пространств имен

kubectl get pods --all-namespaces

Получите pods из указанного пространства имен

kubectl get pods -namespace=my-namespace

Создать модуль

kubectl run my-pod-1 --image=nginx:latest --dry-run

Посмотреть, как будет обрабатываться pod

kubectl run my-pod-1 --image=nginx:latest --dry-run=client

Создать pod в указанном пространстве имен

kubectl run my-pod-2 --image=nginx:latest --namespace=my-namespace

Создать pod с меткой к нему

kubectl run nginx --image=nginx -l --labels=app=test

Получить все pod с выводом меток

kubectl get pods...-show-labels

Получить капсулы с расширенным/широким выводом

kubectl get pods -o wide

Перечислить капсулы в отсортированном порядке

kubectl get pods --sort-by=.metadata.name

Получение журналов стручка

kubectl logs my-pod-1

Получение стручков в указанном пространстве имен с расширенным/широким выводом

kubectl get pods my-pod-2 --namespace=my-namespace -o wide

Получите журналы pod в указанном пространстве имен

kubectl logs my-pod-2 --namespace=my-namespace

Опишите pod

kubectl describe pod my-pod-1

Описать pod в указанном пространстве имен

kubectl describe pods my-pod-1 --namespace=my-namespace

Удалить pod из текущего пространства имен

kubectl delete pod my-pod-1

Удалить pod из указанного пространства имен

kubectl delete pods my-pod-1 --namespace=my-namespace

**Развертывания**

Получение списка развертываний из текущего пространства имен

kubectl get deployments

Получение списка развертываний из указанного пространства имен

kubectl get deployments --namespace=my-пространство имен

Создать развертывание

kubectl create deployment my-deployment-1 --image=nginx

Получить указанное развертывание

kubectl get deployment my-deployment-1

Получить указанное развертывание с его метками

kubectl get deployment my-deployment-1 --show-labels

Опишите указанное развертывание

kubectl describe deployments my-deployment-1

Получите подробную информацию о развертывании в формате yaml

kubectl get deployment my-deployment-1 -o yaml

Измените образ в существующем развертывании

kubectl set image deployment my-deployment-1 nginx=nginx:1.16.1

Просмотр истории развертывания

kubectl rollout history deployment my-deployment-1

Отмена предыдущего развертывания

kubectl rollout undo deployment my-deployment-1

Возврат к определенной версии истории развертывания

kubectl rollout undo deployment my-deployment-1 --to-revision=2

Показать статус развертывания

kubectl rollout status deployment my-deployment-1

Перезапустить ресурс

kubectl rollout restart deployment my-deployment-1

Масштабировать развертывание до 3

kubectl scale --replicas=3 deployment my-deployment-1

Масштабировать с текущего количества до нужного

kubectl scale --current-replicas=3 --replicas=5 deployment my-deployment-1

Это создаст HPA (Horizontal Pod Aotuscaler)

kubectl autoscale deployment my-deployment-1 --min=2 --max=10

**Services**

Сначала создайте pod с меткой app=myapp.

Затем:

Создаем pod с меткой

kubectl run my-pod --image=nginx --labels=app=myapp

Создаем сервис типа NodePort, который будет использовать метки pod для селектора, но нам нужно указать тип, поэтому сначала создайте файл определения, а затем создайте сервис

kubectl expose pod my-pod --port=80 --name nginx-service --type=NodePort --dry-run=client -o yaml

Создайте сервис, который будет иметь тип NodePort, но у него не будет селектора как у my-app

kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml

Получить службы из текущего контекста

kubectl get service

Получить подробную информацию о службах

kubectl get service -o wide

Получить службы с метками на них

kubectl get service --show-labels

Получите сервисы из всех пространств имен

kubectl get services --all-namespaces

Опишите сервис, чтобы узнать о нем больше

kubectl describe service nginx-service

Получите конкретный сервис

kubectl get service nginx-service

Удалите сервис

kubectl delete service nginx-service

**Управление объектами из файлов .yaml/.yml**

Сначала создайте файл определения для pod

Создайте файл определения для pod

kubectl run mypod --image=nginx --dry-run=client -o yaml > my-pod.yml

Создайте объект

kubectl create -f my-pod.yml

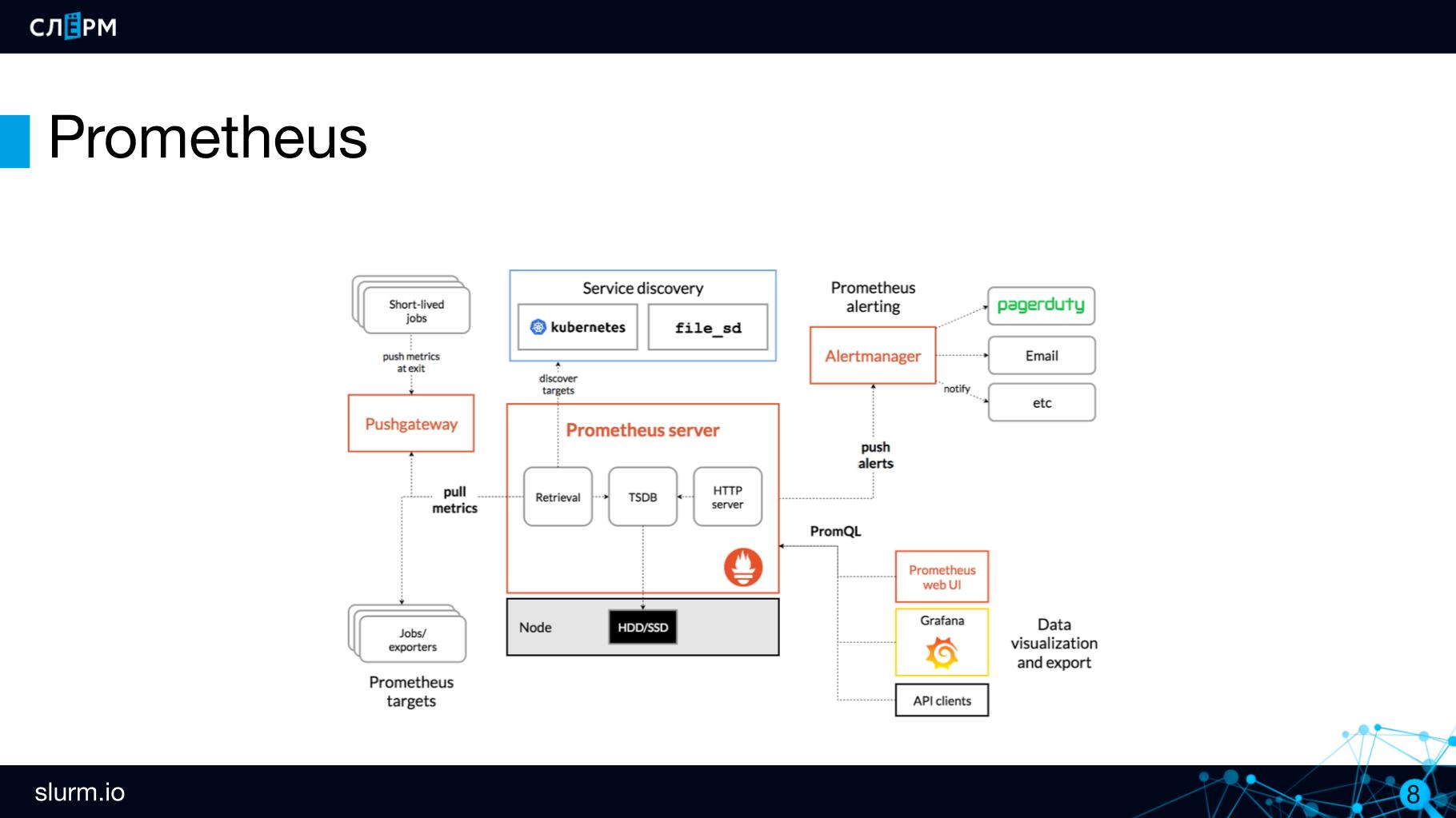
Удалите объект

kubectl delete -f my-pod.yml

Постоянные тома (Persistent Volumes, **PV**) — это единицы хранения, подготовленные админом. Они не зависят от подов и их скоротечной жизни. Persistent Volume Claim (**PVC**) — это запросы на хранилище, то есть **PV**. С **PVC** можно привязать хранилище к ноде, и эта нода будет его использовать.

## [Мониторинг кластера Kubernetes](https://habr.com/ru/companies/slurm/articles/516748/)

**Prometheus**

Метрики хранит time series database (**TSDB**). **TSDB** — это не отдельная база данных, а пакет на языке Go, который вшит в Prometheus. Грубо говоря, всё находится в одном бинаре.

Не храните данные в **TSDB** долго

Prometheus Server работает по модели **pull**: сам ходит за метриками в те эндпоинты, которые мы ему передали. Сказали: «ходи в API Server», и он раз в n-ое количество секунд туда ходит и забирает метрики.

Для объектов с короткой продолжительностью жизни (job или cron job), которые могут появляться между периодами скрапинга, есть компонент **Pushgateway**. В него **пушатся** метрики от краткосрочных объектов: job поднялся, выполнил действие, отправил метрики в Pushgateway и завершился. Через некоторое время Prometheus в своём ритме сходит и заберёт эти метрики из **Pushgateway**.

Ну и напоследок расскажу про киллер-фичу Prometheus — **Discovering**. При работе с Prometheus не нужно указывать конкретные адреса объектов для мониторинга, достаточно задать их тип. То есть не надо писать «вот IP-адрес, вот порт — мониторь», вместо этого нужно определить, по каким принципам находить эти объекты (targets — цели). Prometheus сам, в зависимости от того, какие объекты сейчас активны, подтягивает к себе нужные и добавляет на мониторинг.

## [Ingress](https://selectel.ru/blog/how-select-ingress/)

**Ingress** — сущность кластера Kubernetes, где создается описание конфигурации самого Ingress Сontroller.

**Ingress Controller** — обрабатывает трафик. Его конфигурация формируется из всех Ingress внутри одного кластера.

Ingress-контроллер не отменяет необходимости во внешнем балансировщике нагрузки, он лишь добавляет дополнительный уровень маршрутизации и большую гибкость в распределении трафика.

**Сервисы для трафика на Ingress-контроллер**

Reverse proxy в Ingress Controller слушает порты с HTTP/HTTPS-соединениями. Трафик может попадать на порты тремя способами:

**NodePort** — на случайных портах в диапазоне 30000-32767;

**HostPort** — можно повесить на порты 80 или 443;

**Host Network** — Pod повесит свои порты на публичном сетевом интерфейсе, в этом случае откроются все порты контейнера, что не всегда безопасно.

**NodePort**

Ставить Ingress Controller на NodePort без LoadBalancer — плохая практика, поскольку URL будет включать только порт, который указан в NodePort .

Для таких сценариев лучше использовать Deployments. Это позволит проще скейлить количество подов, ответственных за входящий трафик, прописывать им nodeAffinity и запускать сразу группу контроллеров.

**HostPort**

При использовании HostPort порт пробрасывается с хоста, где запущен под, в этот самый Pod. LoadBalancer на входе не пригодится, но для работы сайта в DNS придется указать, что адрес домена есть на всех узлах.

Пример конфигурации DNS для трех воркеров:

*ingress.example.org A 10.0.0.1*

*ingress.example.org A 10.0.0.2*

*ingress.example.org A 10.0.0.3*

*www.example.org CNAME ingress.example.org*

Для этой установки лучше использовать DaemonSet, поскольку он позволяет запустить не более одного Pod на хосте. Deployment возможен, но со своими особенностями: надо прописать affinity, чтобы не назначить пару подов на один хост. Это поможет избежать конфликта портов.

**Host Network**

При запуске Ingress Controller в общей сети с хостом пробрасывать порт не нужно. Но в этом случае все порты, которые открыты в Pod, будут доступны из сети. Для запуска стоит применить DaemonSet. Аналогично с HostPort, чтобы избежать конфликта портов.

**Что выбрать**

Если LoadBalancer есть на входе, выбирайте NodePort, если нет — HostPort и DNS Round Robin. В тестовой песочнице можно попробовать Host Network, но на боевых проектах лучше так не экспериментировать.

**Популярные контроллеры**

**Ingress от Nginx Inc**

Это стабильное решение с обратной совместимостью, высокой скоростью и без Lua-плагинов или других модулей. Бесплатная версия имеет ограниченную функциональность, а в полной реализованы realtime-метрики, JWT-валидация, health check и другие инструменты. Есть поддержка TCP/UDP-трафика.

**Traefik Kubernetes Ingress**

Решение создавалось для маршрутизации запросов микросервисов и их динамической среды. Благодаря этому Traefik получил поддержку разных методов балансировки, обновление конфигураций без перезагрузки, поддержку различных протоколов, REST API, web-интерфейс и много других полезных фич.

Traefik из коробки поддерживает сертификаты Let’s Encrypt, но для организации высокой доступности нуждается в создании собственного KV-хранилища.

**HAProxy**

Один из самых популярных инструментов, применяется не только как балансировщик, но и в качестве proxy. Решение позволяет обновлять конфигурацию без потери трафика, service discovery на основе DNS, поддерживает динамическое конфигурирование через API. Шаблоны конфига полностью можно полностью отредактировать через смену CM, что достаточно удобно.

**Istio Ingress**

Решение из разряда full service mesh, которое полностью управляет трафиком в кластере. Проект разработан на базе ***Envoy*** и пропагандирует максимальную управляемость, масштабируемость и безопасность. Решение позволяет контролировать маршрутизацию, мониторинг, авторизацию доступов и балансировку нагрузки.

## [ISTIO](https://habr.com/ru/companies/flant/articles/569612/)

**Istio** — частный случай «сервисной сетки» (Service Mesh)

**Service Mesh** дает набор «кирпичей», из которых мы можем собирать собственные паттерны управления сетью. По-другому: Service Mesh — это фреймворк для управления любым TCP-трафиком с собственным декларативном языком. А в качестве бонуса Service Mesh предлагает дополнительные возможности для мониторинга (observability).

## HELM

helm list - показать, что инсталлировано через helm

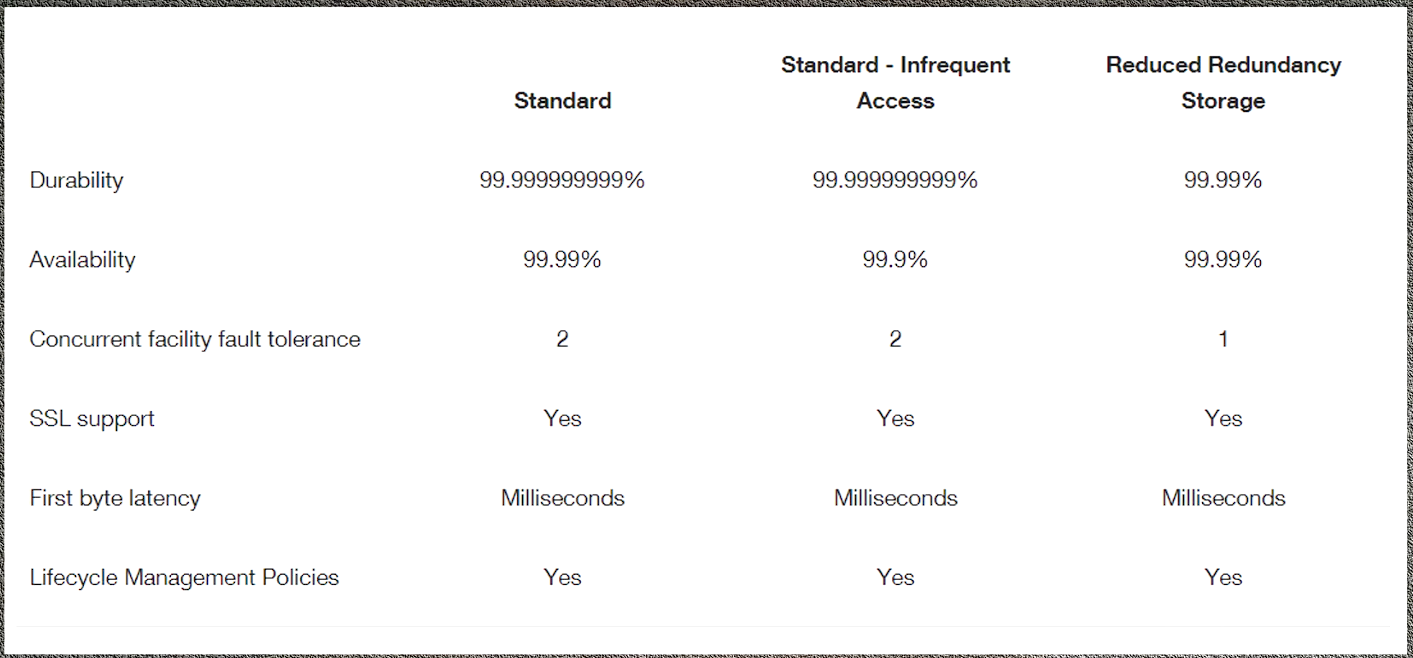
helm install <app> <directory> - инсталлировать чарт

helm package <directory> - архивировать в файл

helm delete <app> - удалить чарт

## AWS

S3 Buckets



Instance types

T – General Purpose

M – General Purpose

C – Compute Optimized

F – FPGA Optimized

G,P – GPU Optimized

X,R – RAM Optimized

D,I – Storage Optimized

EBS – Elastic Block Storage

Root Boot:

General Purpose SSD (gp2) 100 – 3000 iops / (gp3) 3000 - 16000 iops

Provisioned IOPS SSD (io1) from 400 iops / (io2) from 4000 iops

Magnetic (standart) up to 1 TB

Other:

Cold HDD (sc1)

Throughput Optimized HDD (st1)

[attaching-and-mounting-existing-ebs-volume-to-ec2-instance-filesystem](https://stackoverflow.com/questions/28792272/attaching-and-mounting-existing-ebs-volume-to-ec2-instance-filesystem-issue)