Урок 8

Прикладной уровень

Прикладной уровень. SMTP, HTTP. Перспективные прикладные протоколы

Прикладной уровень

Прикладной уровень решает задачи передачи передачи данных между приложениями через сеть. В стеке TCP/IP прикладные протоколы непосредственно реализуют сами приложения, получая доступ к услугам транспортного и сетевого уровня с использованием механизма сокетов.

Основные задачи решаемые на прикладном уровне.

- Передача запросов от клиента к серверу.
- Передача ответов от сервера к клиенту.
- Шифрование данных и идентификация абонента.
- Обеспечение работы сетевых служб.

Прикладные протоколы могут использовать один из транспортных протоколов (TCP для гарантированной, UDP для негарантированной) или даже оба в зависимости от задач (DNS использует UDP для DNS-запросов, TCP для передачи файлов зон и DNSSEC). Прикладные протоколы могут как использовать стандартные средства для управления сеансом (TCP) и реализацией безопасности (TLS), так и реализовать самостоятельно (протокол QUIC, использующий HTTP/2 API на верхнем уровне и самостоятельно реализующий управление соединением и безопасность, используя UDP в качестве транспорта).

Многие прикладные протоколы изначально небезопасны. Ряд небезопасных протоколов был приспособлен для работе в безопасном режиме благодаря механизму TLS (HTTPS, FTPS), STARTTLS (SMTP, XMPP) либо использованию самостоятельных решений (SSH и его надстройки).

Протоколы прикладного уровня

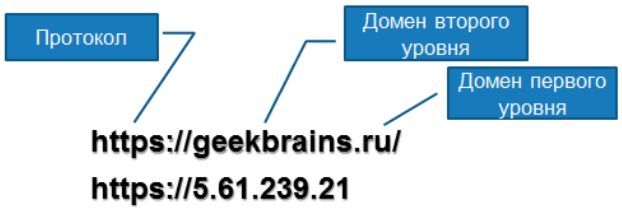
Рассмотрим список основных прикладных протоколов, которые используются приложениями:

HTTP, DNS, POP3, IMAP, SMTP, SNMP, Telnet, SSH, FTP, TFTP, RDP, iSCSI, RTP, NTP.

Служебные сетевые протоколы: DNS/DHCP/NTP/SNMP

Служебные сетевые протоколы обеспечивают работоспособность сетевых сервисов. Они не передают пользовательскую информацию, но обеспечивают работу сетевых узлов. Рассмотрим подробнее наиболее популярные сетевые протоколы: DNS/DHCP/NTP.

DNS

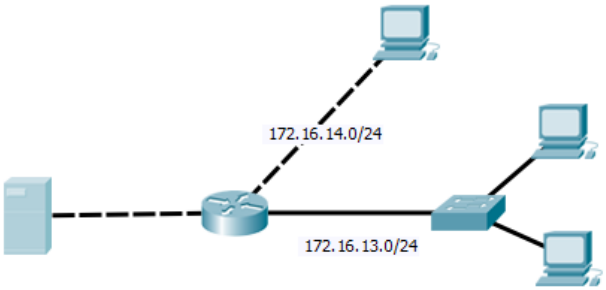


DNS, или система доменных имён — это сетевая служба, используемая практически всеми сетевыми устройствами для получения данных о доменах. Чаще всего служба используется для обращения к серверу с целью получить данные об IP-адресе определённому домена для обращения к серверу

домена. Запись в таблице доменной службы, отвечающая за сопоставление домена и адреса, называется A-запись.

Доменная система является распределенной базой данных, работающая с помощью клиент-серверной архитектуры. Мы уже рассматривали, что доменная служба имеет иерархию серверов. Обращение к серверам производится через протокол DNS. Порт/ID: 53/TCP, 53/UDP

DHCP



Dynamic Host Configuration Protocol, или протокол динамической конфигурации сетевых узлов — протокол, позволяющий узлам в компьютерной сети в автоматическом режиме получить IP-адрес и дополнительные параметры (маска сети, основной шлюз, доменный сервер и другие), нужные для работы в компьютерной сети. Протокол построен на клиент-серверной архитектуре. В качестве сервера может выступать компьютер, маршрутизатор или коммутатор 3-го уровня. Во время инициализации сетевого интерфейса с включенным режимом автоматической конфигурации клиент производит широковещательный запрос в сеть с целью обращения к DHCP-серверу. Сервер производит ответ, сообщая информацию о необходимых сетевых параметрах. Клиент отвечает серверу, подтверждая, что он готов принять сетевые параметры и нужно зарегистрировать IP-адрес из пула за ним. Сервер подтверждает регистрацию адреса, и клиент начинает использовать назначенный ему адрес. Администратор конфигурирует диапазон адресов (сетевой пул), которые будут назначены клиентам. Данный протокол ускоряет конфигурирование сетевых устройств, кроме того, существует возможность привязки адресов по MAC-адресам к каждому устройству. Протокол DHCP всегда используется в беспроводных сетях.

DHCP разработан на основе протокола BOOTP, который использовался для загрузки бездисковых терминалов и назначения им сетевых адресов. DHCP обратно совместим с BOOTP.

Порт/ID: 67, 68/UDP.

NTP

Network Time Protocol или протокол сетевого временя — сетевой протокол времени, служит для синхронизации часов сетевых устройств. Корректно установленное время на всех сетевых устройствах позволяет отслеживать события по логам, которые происходили в сети. Протокол также применяется в мобильных сетях для передачи данных о текущем времени на сотовые телефоны. Протокол разработан в 1985 году и используется по настоящее время в обновленной редакции. Актуальная рабочая версия NTPv4.

Протокол работает поверх протокола UDP и учитывает время передачи сообщения (для Unicast). Система NTP построена с учетом задержек среды передачи, таким образом, независимо от времени доставки сообщения клиент сможет вычислить текущее время. Версия 4 позволяет достичь точности в 10 мс при работе в Интернет и до 0,2 мс внутри локальной сети.

Протокол используется для синхронизации серверного времени машин, которые необходимы для синхронизации внутри локальной сети. Порт/ID: 123/UDP.

Имеется упрощенная версия SNTP (Simple NTP), также использует 123/UDP.

SNMP

Simple Network Management Protocol, или простой протокол сетевого управления — протокол для получения информации и управления сетевыми устройствами в IP-сетях.

Протокол SNMP может быть поддержан следующими устройствами: маршрутизатор, коммутатор, сервер, рабочая станция, принтер и другие. Порт/ID: 161/UDP,162/UDP.

Современные системы мониторинга такие, как Zabbix, также умеют работать с SNMP.

Файловые протоколы

Рассмотрим протоколы, которые используются для передачи файлов в компьютерных сетях.

- Просмотр каталогов, загрузка файлов с сервера и на сервер: FTP, TFTP.
- Безопасная, защищенная передача файлов: SFTP, FTPS.
- Доступ к файловым системам в локальных сетях: NFS/SMB/iSCSI.

FTP/TFTP

File Transfer Protocol, или файловый транспортный протокол — протокол для передачи файлов с сервера с установленным программным обеспечением на клиентское устройство. FTP позволяет пользователю передавать двоичные или текстовые файлы между компьютерами в сети. После подключения к файловому серверу клиент может произвести загрузку файлов с сервера или на сервер. Порт/ID: 21/TCP для команд, 20/TCP для данных.

Для запроса назначаются динамические порты из диапазона 49152-65534/TCP.

Trivial File Transfer Protocol, или простой протокол передачи файлов — это протокол, используемый для загрузки бездисковых рабочих станций, передачи логов работы или прошивок на телекоммуникационное оборудование. Главным отличием TFTP от FTP является отсутствие функций авторизации клиента, но можно настроить фильтрацию клиентов по их сетевым адресам.

Порт/ID: 69/UDP

FTPS/SFTP

File Transfer Protocol + SSL, или FTP/SSL данное расширение файлового протокола, использующее криптографический протокол SSL, выполняющий шифрование данных для обеспечения безопасности транспортной подсистемы.

FTPS отличается от протокола SFTP (протокола FTP, передающего данные через SSH-соединение).

Протокол обратно совместим с клиентами, не поддерживающими работу по FTPS.

Протокол использует для передачи служебных сообщений порт 990/TCP, а для передачи информации - 989/TCP. Данное отличие позволяет использовать стандартные порты 21 и 20/TCP для обратной совместимости протокола FTP.

SFTP, или SSH File Transfer Protocol — это файловый протокол, используемый для операций управления и копирования файлов через безопасное соединение протокола SSH.

Порт/ID: TCP/22.

Несмотря на то, что FTPS и SFTP - разные протоколы, обычно FTP-клиенты поддерживают оба протокола, делая работу неотличимой. Настраивая FTPS/SFTP, озаботьтесь изолированием домашней директории (chroot). По умолчанию даже имя доступа на чтение, пользователь, подключенный по FTP/FTPS/SFTP, может просматривать все содержимое диска, в т.ч. содержимое файла /etc/passwd (для Linux, что дает полное представление о пользователях и сервисах в системе).

Протоколы сетевого доступа, используемые в локальных сетях: NFS/SMB

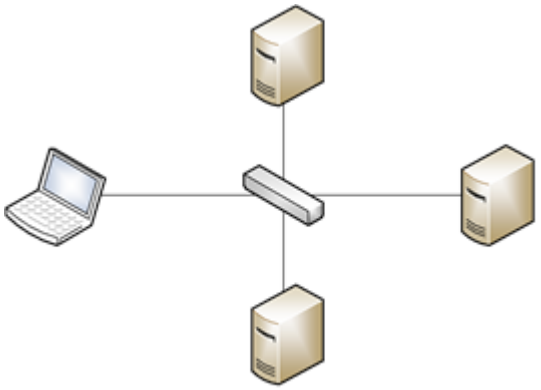
Network File System, или сетевая файловая система предоставляет возможность подключить (примонтировать) удаленную файловую систему через сеть (подключить сетевой диск). Протокол обеспечивает клиентам простой доступ к файлам и каталогам системы сервера. Отличием от протокола FTP является то, что NFS производит обращение не ко всему файлу сразу, а только к используемой процессом части. Такое отличие позволяет работать любому приложению клиента с сетевыми файлами, так же как и с локальной файловой системой.

Порт/ID: UDP/2049.

Server Message Block — это протокол, использующий клиент-серверную архитектуру, который используется клиентами для чтения, редактирования и записи файлов, а также обращения к службам у серверных программ в различных типах сетевого окружения.

Порт/ID: TCP/445.

iSCSI



iSCSI Internet Small Computer System Interface - это сетевое развитие технологии SCSI. Протокол используется для подключения внешних систем хранения данных к серверам или клиентам. Таким образом, благодаря этому протоколу вы можете подключить внешний сетевой диск как локальное устройство, полностью управляя его работой.

Основная задача протокола - это управление работой системы хранения данных и взаимодействие с ней. Протокол поддерживает работу сетей хранения данных SAN (Storage Area Network). Протокол используется в кластерных архитектурах совместно с высокоскоростными сетями передачи данных для организации SAN.

Порт/ID: TCP/3260.

WebDAV(Web Distributed Authoring and Versioning)

Изначально расширение для протокола HTTP для совместной работы, теперь это протокол, позволяющий использовать удаленные файловые системы как локальные. Поддерживается файловыми хранилищами такими, как Яндекс.Диск, поддерживается в Windows и Linux

Порт: TCP/80,443.

WEB протоколы: HTTP/HTTPS

Hyper Text Transfer Protocol — это протокол передачи гипертекстовых документов. Один из наиболее распространенных протоколов, используемых в сети. Протокол обеспечивает взаимодействие браузеров (клиентов) и веб-серверов. Передачу запросов и ответов (веб-страниц) сгенерированных веб-сервером. На данный момент разработана вторая версия протокола, о которой мы поговорим отдельно в следующей лекции.

Порт/ID: TCP/80.

HyperText Transfer Protocol Secure — это безопасная версия протокола HTTP с расширением, использующая протоколы шифрования SSL или TLS для безопасной передачи данных. Протокол использует систему шифрования и сертификатов для установления шифрованного канала передачи данных между клиентом и сервером поверх TCP-протокола. Большинство крупных социальных сетей, а также поисковых машин, сейчас использует данный протокол, он защищает персональные данные

пользователей, подключенных к локальной сети, через небезопасное сетевое соединение (например, общественную сеть Wi-Fi).

Порт/ID: TCP/443.

Web Distributed Authoring and Versioning или просто DAV — это расширение к протоколу HTTP, обеспечивающее работу клиентов над файлами, а также управление файловой структурой. Фактически расширение выполняет функции протокола FTP, отличие в том, что используется протокол HTTP для передачи данных.

WebDAV поддерживает : HTTP и HTTPS (SSL).

Почтовые протоколы POP3 и SMTP

POP3 (Post Office Protocol) — это протокол прикладного уровня, использующийся для почтовых соединений. Клиенты подключаются к серверу POP, который обрабатывает почту, а протокол POP используется для отправки и обработки обращений клиентов на получение почтовых сообщений. Ключевой особенностью протокола является то, что клиенты скачивают к себе почтовые сообщения и потом работают с ними локально.

Порт/ID: TCP/110.

SMTP (Simple Mail Transfer Protocol) — это протокол прикладного уровня, использующийся для передачи почтовых сообщений. Сервер SMTP принимает почтовое отправление и уведомляет клиента либо сообщением об ошибке, либо подтверждением удачной передачи.

Протокол SMTP обычно используется в связке с протоколом POP3 на клиентах для отправки и получения почты соответственно.

Порт/ID: 25/TCP, 587/TCP, 465/TCP (SMTP over SSL).

Internet Message Access Protocol — это протокол прикладного уровня, использующийся для взаимодействия с почтовым сервером. Протокол реализует интерфейс взаимодействия с хранилищем почты на сервере, как если бы почтовые сообщения находились локально. Клиент может управлять почтовыми сообщениями без загрузки файлов с сервера и обратно.

Протокол IMAP представляет собой альтернативу протоколу POP3 с небольшим функционалом по передаче сообщений. Порт/ID: TCP/143 TCP/993 (IMAP over SSL)

Протоколы управления: SSH/TELNET/RDP/RFB

TELNET — это протокол прикладного уровня, используемый для удаленного доступа в режиме терминала. TELNET предоставляет доступ удаленному пользователю управлять работой компьютера или сетевого устройства как при локальном подключении.

Порт/ID: 23/TCP.

SSH, или Secure Shell (безопасная оболочка) — протокол прикладного уровня, обеспечивающий возможность передачи данных по защищенному соединению. Протокол предоставляет возможность подключиться в режиме терминала к удаленному сетевому устройству, аналогично TELNET, а также

создать туннелированное TCP-соединение (например, для выхода в удаленную сеть или передачи файла).

Порт/ID: 22/TCP.

RDP, или Remote Desktop Protocol — протокол прикладного уровня для управления работой ОС Windows. Выполняет удалённое подключение к рабочему столу в графическом режиме. Используется для удаленной настройки компьютеров, а также для работы на ПК в терминальном режиме.

Порт/ID: 3389/TCP.

RFB, или Remote Framebuffe — платформонезависимый протокол прикладного уровня для удалённого доступа к рабочему столу компьютера, используется утилитой VNC. Протокол работает на основе кадрового буфера, поэтому его возможно применять для систем графическим оконным интерфейсом, например X Window System, Windows.

Порт/ID: 5900 до 5906/TCP для Java клиентов (5800 до 5806), подключение к клиенту 5500.

Потоковая передача RTP

RTP, или Real-time Transport Protocol - протокол прикладного уровня, используется как основной транспортный протокол для передачи голосовых и видеосообщений в IP-сетях и совместно с кодеками. Для протокола RTP не задан стандартный номер порт. Особенностью является то, что соединение устанавливается с применением чётного номера, а следующий нечётный номер применяется для управления передачей с помощью протокола RTCP.

UDP-порты (16k-32k).

Используется для IP телефонии и IPTV.

Протоколы телефонии SIP/H.323.

Session Initiation Protocol — протокол прикладного уровня для установления сеанса связи. Протокол используется для установления и завершения сеансов связи, для обмена информацией между пользователями. Обмен информацией включает в себя мультимедийное содержимое (IP-телефония, видео- и аудио конференции, чат-сообщения, трафик онлайн-игр). Наиболее популярный на данный момент протокол для управления голосовыми соединения - VoIP (Voice Over IP).

H.323 — это стек протоколов, пришедший из телефонии, реализующий сигнализацию VoIP. В настоящий момент H.323 заменяется протоколом SIP.

RTMP (Real Time Messaging Protocol).

Разработан Adobe и может использоваться в приложения FlashPlayer.

Протокол, используемый многими реализациями вебинаров. Как ни странно, протокол работает не поверх UDP, а поверх TCP, используя порт 1935. Может туннелироваться через HTTP, используя TCP порты 80 и 443 (RTMPT). Имеется ряд развитий протокола, среди которых RTMPS (как вы догадались, RTMP+TLS), RTMFP (Real-Time Media Flow Protocol), использующий UDP.

Работа прикладных протоколов и NAT

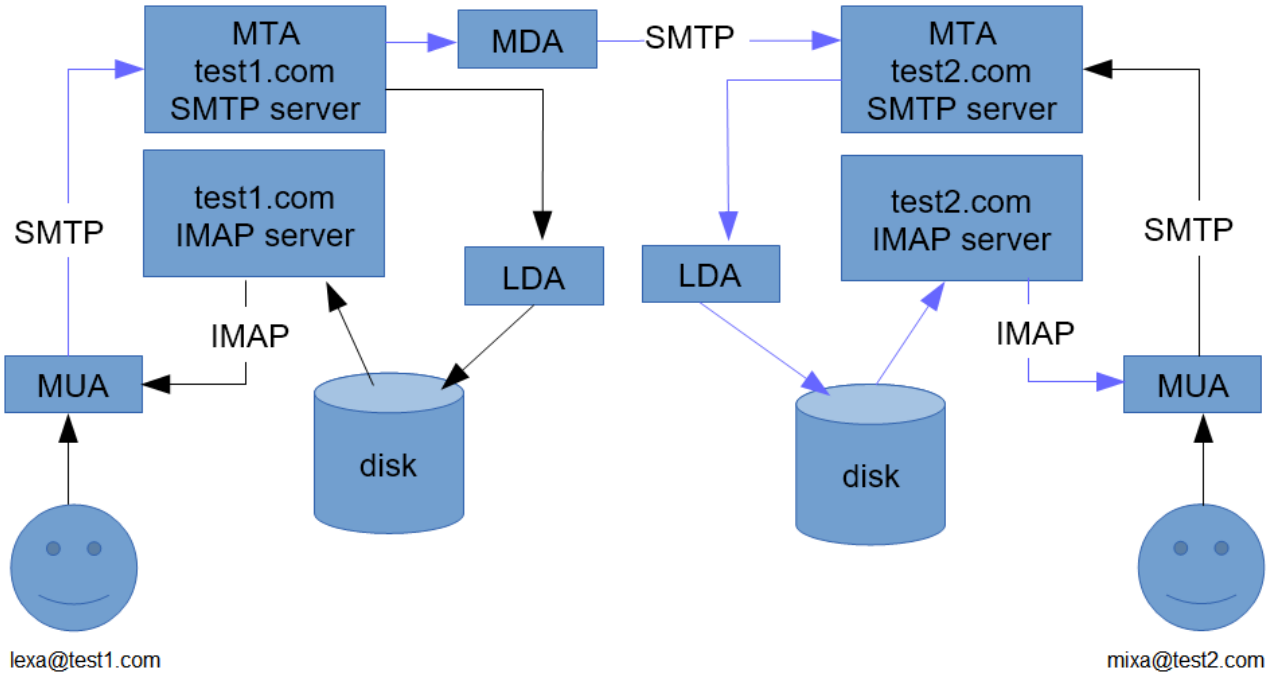
При преобразовании пакетов, направленных на хорошо известные порты, NAT проверяет полезную информацию пакетов, преобразовывает встроенные IP-адреса и создает полностью расширенное преобразование. Это происходит в статических и динамических конфигурациях NAT. Этот набор функций выполняется на пути с коммутацией процессов и является стандартным поведением для всех протоколов, для которых необходимо преобразование встроенных IP-адресов, включая FTP, DNS, IRC, SNMP, H.323 и SIP.

Сервисы электронной почты: SMTP/POP3/IMAP

Процесс доставки электронной почты

В процессе доставки электронное сообщение обычно проходит через несколько узлов электронной почты и ряд программ, выполняющих разные функции. Опишем главных участников процесса доставки в общепринятой для электронной почты терминологии.

- MTA (Mail Transfer Agent) — программа, которая принимает, маршрутизирует и отправляет другим серверам электронную почту. Как правило, для этого используется протокол SMTP. В качестве MTA в Linux используют такие программы, как postfix, exim. Исторически использовался sendmail.
- MDA (Mail Delivery Agent) — программа, которую вызывает MTA для действительной доставки сообщения, например, по протоколу SMTP на другую систему.
- LDA (Local Delivery Agent) — разновидность MDA для локальной доставки писем. LDA помещает письмо в почтовый ящик пользователя. Иногда не разделяют MTA и LDA, называя все программы доставки MDA. В качестве примера LDA может привести Dovecot, который умеет принимать от MTA письма по протоколу SMTP, сохранять на диск для последующей отдачи пользователю по протоколу PO3/IMAP4.
- MUA (Mail User Agent) — почтовая программа пользователя такая, как Outlook или Mozilla Thunderbird.



Рассмотрим процесс доставки сообщения, которое пользователь lexa@test1.com отправил пользователю mixa@test2.com.

1. Пользователь lexa@test1.com в своей почтовой программе нажимает кнопку «Отправить», после чего почтовая программа (MUA) соединяется с локальным почтовым сервером test1.com по протоколу SMTP и выполняет передачу сообщения. Далее обработка сообщения выполняется MTA на сервере test1.com.
2. MTA на test1.com сверяется с локальной конфигурацией и выясняет, он не принимает локально письма для домена test2.com.
3. Поэтому MTA на test1.com выполняет запрос к DNS и получает MX-записи для домена test2.com. Почтовым сервером с наивысшим приоритетом для домена test2.com является сервер test2.com.
4. MTA на test1.com вызывает SMTP MDA для доставки сообщения на сервер test2.com по протоколу SMTP.
5. MTA на test2.com получив сообщение, сверяется с локальной конфигурацией и выясняет, он принимает локально письма для домена test2.com.
6. Пользователь mixa в своей почтовой программе нажимает кнопку «Получить почту», после чего его почтовая программа (MUA) обращается к серверу test2.com по протоколу IMAP или POP3 и загружает доставленное ему сообщение от пользователя lexa@test1.com.

Удобство использования MX-записей в том, что веб-сервер, выдающий страницу по адресу test.com, и почтовый сервер, принимающий письма на емейл enail@test.com, могут не совпадать. Более того, можно настроить так, чтобы почту для домена обслуживали Яндекс.Почта или Gmail.

Протокол SMTP

Протокол SMTP, как мы уже видели, используется при передаче сообщений от MUA к MTA или между MTA на различных узлах электронной почты. SMTP представляет собой текстовый протокол, когда клиент с помощью определенного набора команд задает параметры отправляемого сообщения, а

сервер подтверждает его прием или выдает сообщения об ошибках. Ниже представлен пример SMTP сессии.

```
root@ubutubu:~# telnet localhost 25 Trying ::1...
Trying 127.0.0.1...
Connected to localhost. Escape character is '^]'.
220 ubutubu.example.com ESMTP Sendmail 8.14.4/8.14.4/Debian-4.1ubuntu1; Thu, 10
Jul 2014 14:06:50
+0400; (No UCE/UBE) logging access from: localhost(OK)-localhost [127.0.0.1]
ehlo localhost
250-ubutubu.example.com Hello localhost [127.0.0.1], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN 250-VERB
250-8BITMIME
250-SIZE 250-DSN
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5
250-DELIVERBY 250 HELP
mail from: <oga@ubutubu.example.com>
250 2.1.0 <oga@ubutubu.example.com>... Sender ok
vkrcpt to: <oavdeev@prog-school.ru>
250 2.1.5 <oavdeev@prog-school.ru>... Recipient ok data
354 Enter mail, end with "." on a line by itself
Subject: Test test test mail
Test
1 2 3 .
250 2.0.0 s6AA6oi5004854 Message accepted for delivery quit
221 2.0.0 ubutubu.example.com closing connection
Connection closed by foreign host.
```

В данном примере соединение с SMTP сервером было установлено с помощью команды telnet и все команды вводились вручную. Первой командой SMTP клиент представляется серверу: EHLO localhost. Сервер может выполнить на этом этапе ряд проверок: проверить корректность доменного имени после EHLO, проверить, имеет ли имя DNS-записи типа A или MX и так далее. Уже на этом этапе можно ограничить часть спама, идущего через почтовый сервер. В нашем случае проверки прошли успешно, сервер представился сам и выдал список поддерживаемых расширенных команд протокола ESMTP. Далее клиент с помощью команды MAIL FROM сообщает серверу email, который будет использоваться сервером для возврата сообщения в случае неудачной доставки. Команда RCPT TO устанавливает получателя данного сообщения, команда может быть использована повторно, если письмо имеет несколько получателей, используется по одной команде на каждого получателя. Сообщения доставляются именно по значению RCPT TO, а не по полю To: служебного заголовка письма, в котором может быть вообще что угодно. После ввода команд MAIL FROM и RCPT TO сервер выполняет проверку корректности адресов и подтверждает проверку сообщениями Recipient ok и Sender ok. На данном этапе сервер проверяет возможность доставки сообщения с данным отправителем данному получателю. Как правило, один из них должен принадлежать почтовому домену, обслуживаемому данным почтовым сервером, или иметь IP в доверенной сети. Если это не так, возможно, кто-то пытается использовать почтовый сервер для отправки спама.

Это обеспечивается запросом PTR записи для обратного разрешения домена для IP-адреса отправителя, использования расширения SPF (Sender Policy Framework), позволяющего указать для домена, от имени которого отправляется письмо указать список серверов, имеющих право отправлять. SPF также базируется на системе DNS, используя записи вида TXT (позволяющие сопоставить доменному имени произвольное текстовое описание).

Пример SPF-записи, которая должна быть указана при использовании Яндекс.почты для домена на DNS-сервер, обслуживающим его зону.

```
@      TXT    v=spf1 redirect=_spf.yandex.ru
```

Пример, когда требуется указать еще отдельные сервера, которые тоже могут отправлять письма.

```
@      TXT    v=spf1  ip4:1.2.3.4  ip4:1.2.3.5  ip4:1.2.3.6  include:_spf.yandex.ru
~all
```

При проверке писем сервер также может проверить IP-адрес отправителя в черных списках.

Одной из реализаций черных списков является DNSBL, который представляет собой DNS-сервер. Чтобы проверить, заблокирован ли IP-адрес каким-либо DNSBL-списком, надо указать проверяемый IP в нотации DNS PTR (в обратном порядке октетов) и добавить имя домена DNSBL-сервера. Если ответ будет получен, то данный адрес заблокирован (находится в списке).

```
$ host -tA 71.60.206.220.bl.spamcop.net

220.206.60.71.bl.spamcop.net has address 127.0.0.2
```

Полученный IP-адрес может оказаться любым, важен лишь факт его наличия (отсутствия) в ответе на запрос. Именно поэтому сам IP можно использовать для описания, скажем, типа источника, по которому искомый адрес был добавлен в список. К примеру, 127.0.0.1 — открытые релеи, 127.0.0.2 — источники portscan'ов, и т. п. По той же причине полезно использовать host с опцией -t any, в ответ на которую вы можете получить дополнительный комментарий в поле типа «текст» (TXT RR).

```
$ host -t any 116.47.136.151.vote.drbl.sandy.ru

151.136.47.116.vote.drbl.sandy.ru descriptive text "070715:Spam in progress"

151.136.47.116.vote.drbl.sandy.ru has address 127.0.0.2
```

Пример адреса, которого нет в DNSBL-списке.

```
$ host -tA 72.205.229.181.bl.spamcop.net
```

```
Host 181.229.205.72.bl.spamcop.net not found: 3(NXDOMAIN)
```

Если почтовый сервер допускает отправку через него чужой почты, такой сервер называют «open relay» и обычно через некоторое время его адрес попадает в списки блокировок и другие серверы перестают принимать от него почту.

После того, как со стороны клиента указана команда DATA, происходит непосредственно ввод сообщения. Сообщение состоит из двух частей: служебных заголовков и тела письма. В нашем примере из заголовков указан только Subject: с темой письма. Если как в нашем случае поля служебных заголовков From: и To: не указаны, MTA самостоятельно формирует эти заголовки на основе введенных ранее клиентом команд MAIL FROM и RCPT TO (как правило, To: в этом случае не показывается в письме).

Кроме того в служебных заголовках отмечается весь маршрут, который проходит сообщение. На каждом узле, через который проходит сообщение, добавляется заголовок Received, в котором указывается, с какого IP поступило сообщение, каким сервером оно было принято, дату/время приема и получатель сообщения. Служебные сообщения отделяются от тела сообщения пустой строкой.

Традиционно считается, что сообщение может проходить через узлы, которые поддерживают только семибитную кодировку, поэтому тело и заголовки, в которых встречается кириллица, перед отправкой кодируются в семибитной кодировке base64. Эта кодировка прозрачна для почтовых клиентов, поэтому пользователи сразу получают перекодированный текст и не замечают процесса перекодировки. Раньше вместо base64 для бинарных вложений в письмах использовалась программа uudecode, переводившая бинарные вложения в текст; на стороне получателя программа uudecode восстанавливала исходный файл.
Признаком окончания сообщения служит строка, состоящая из одной точки. Встретив такую строчку, MTA говорит, что сообщение принято для дальнейшей передачи. Затем для окончания сессии клиент использует команду quit.

Дополнительно необходимо заметить, что в настоящее время используются расширения SMTP протокола, которые поддерживают аутентификацию (команда AUTH LOGIN) и шифрование сессии (команда STARTTLS). Эти возможности обычно используют, чтобы защитить соединение между клиентом и сервером от перехвата злоумышленником.
Пример письма, отправленного через sendmail из виртуальной машины описанным выше способом.

```
Received: from mxfront8j.mail.yandex.net ([127.0.0.1])
      by mxfront8j.mail.yandex.net with LMTP id A9txhLMh
      for <siblis@yandex.ua>; Mon, 24 Apr 2017 21:47:00 +0300
Received:  from  37.70.32.95.dsl-dynamic.vsi.ru  (37.70.32.95.dsl-dynamic.vsi.ru
[95.32.70.37])
      by    mxfront8j.mail.yandex.net    (nwsmtp/Yandex)    with    ESMTPS    id
u58pcgfyYV-kxd4gq85;
      Mon, 24 Apr 2017 21:47:00 +0300
      (using TLSv1.2 with cipher ECDHE-RSA-AES128-GCM-SHA256 (128/128 bits))
      (Client certificate not present)
X-Yandex-Front: mxfront8j.mail.yandex.net
X-Yandex-TimeMark: 1493059620
To: undisclosed-recipients:;
X-Yandex-Spam: 1
Received: from localhost (localhost [127.0.0.1])
      by ubuntu (8.14.4/8.14.4/Debian-4.1ubuntu1) with ESMTP id v3OIbGSO023496
      for <siblis@yandex.ua>; Mon, 24 Apr 2017 11:46:17 -0700
Date: Mon, 24 Apr 2017 11:46:17 -0700
From: medvedev@kremlin.ru
Message-Id: <201704241846.v3OIbGSO023496@ubuntu>
Subject: Good weather
Return-Path: medvedev@kremlin.ru
X-Yandex-Forward: 6cd83209625f7ffcf9c2f47a0f822f8c

Something
something mail
```

Письмо попало во вкладку Входящие.

Но уже второе письмо с большой долей вероятности уйдет в спам.

Один из заголовков, который можно наблюдать в таком случае.

```
X-Yandex-Spam: 4
```

Самостоятельно ответьте на вопрос почему.

Еще один инструмент для подтверждения подписей отправителя — DKIM.

Технология DomainKeys Identified Mail (DKIM) объединяет несколько существующих методов антифишинга и антиспама с целью повышения качества классификации и идентификации легитимной электронной почты. Вместо традиционного IP-адреса для определения отправителя сообщения DKIM добавляет в него цифровую подпись, связанную с именем домена организации. Подпись автоматически проверяется на стороне получателя, после чего для определения репутации отправителя применяются «белые списки» и «чёрные списки».

В технологии DomainKeys для аутентификации отправителей используются доменные имена. DomainKeys использует существующую систему доменных имен (DNS) для передачи открытых ключей шифрования.

Протокол POP3

POP3 — протокол прикладного уровня, используется почтовыми программами (MUA) для получения электронных сообщений с почтового сервера. Для POP3 на сервере зарезервирован 110-й порт TCP. Интерфейс, который используется в POP3, чем-то напоминает SMTP: клиент использует набор текстовых команд для выполнения различных действий, сервер обрабатывает команды и выдает на них подтверждения или сообщения об ошибках. Изначально в POP3 не предполагалось никакой защиты: пара логин/пароль передавались по сети в открытом виде, точно так же, как и текст сообщения. Со временем внедрили возможность шифрования пароля, а также появилась возможность шифрования сессии с помощью TLS/SSL. Рассмотрим пример сессии POP3. В данном примере строки, отправленные клиентом, начинаются с «C:», ответы сервера «S:».

```
S: <Сервер ожидает входящие соединения на порту 110>
C: <подключается к серверу>
S:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <сервер передаёт сообщение 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <сервер передаёт сообщение 2>
S:    .
C:    DELE 2
S:    +OK message 2 deleted
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C:  <закрывает соединение>
S:  <продолжает ждать входящие соединения>
```

В примере используется вариант с шифрованным паролем. Клиент передает его на сервер вместе с именем пользователя в команде APOP. Затем клиент с помощью команды STAT запрашивает

состояние своего почтового ящика. Сервер отвечает +OK 2 320, что означает, что в ящике пользователя 2 сообщения объемом 320 байт. Далее клиент командой LIST запрашивает развернутый листинг почтового ящика, по которому видно, что первое сообщение занимает 120 байт, второе — 100. После чего клиент командой RETR 1 запрашивает передачу 1-го сообщения. После успешной передачи сообщения клиент удаляет принятое письмо командой DELE 1. Потом те же команды используются для передачи и удаления второго сообщения. После чего клиент завершает сессию командой QUIT. Обратите внимание, что для POP3 удаление сообщений из ящика выполняется отдельной командой и является не обязательным условием для сессии. Поведение почтового клиента, решающее удалять ли сообщения на сервере после их скачивания, определяется настройками клиента.

Протокол IMAP

IMAP - еще один протокол для получения почтовых сообщений с сервера электронной почты. Так же, как и POP3 имеет свой набор команд, с помощью которого клиент взаимодействует с сервером. Сервер IMAP принимает клиентские соединения TCP на порту 143. IMAP задумывался как замена POP3, обладающая более продвинутым набором функций. Перечислим достоинства IMAP по сравнению с POP3.

- Письма хранятся на сервере, а не на клиенте. Возможен доступ к одному и тому же почтовому ящику с разных клиентов. Поддерживается также одновременный доступ нескольких клиентов. В протоколе есть механизмы, с помощью которых клиент может быть проинформирован об изменениях, сделанных другими клиентами.
- Поддержка нескольких почтовых ящиков (или папок). Клиент может создавать, удалять и переименовывать почтовые ящики на сервере, а также перемещать письма из одного почтового ящика в другой.
- Возможно создание общих папок, к которым могут иметь доступ несколько пользователей.
- Информация о состоянии писем хранится на сервере и доступна всем клиентам. Письма могут быть помечены как прочитанные, важные и т. п.
- Поддержка поиска на сервере. Нет необходимости скачивать с сервера множество сообщений для того, чтобы найти одно нужное.
- Поддержка онлайн-работы. Клиент может поддерживать с сервером постоянное соединение, при этом сервер в реальном времени информирует клиента об изменениях в почтовых ящиках, в том числе о новых письмах.
- Предусмотрен механизм расширения возможностей протокола.

Таким образом, мы видим, что в сравнении с POP3, IMAP позволяет переложить значительную часть функций почтового клиента на сервер и сделать возможным доступ к ящикам с различных клиентов.

Далее приводятся примеры использования команд IMAP протокола. Обратите внимание, что каждая команда, выдаваемая клиентом, предваряется уникальным идентификатором. Потом сервер может использовать этот ID в своих ответах, что позволяет клиенту легче разобраться, к какой команде относится ответ. Обычно идентификатор представляет собой короткую алфавитно-цифровую, например, a001. При вводе следующих команд клиент увеличивает значение идентификатора на единицу: a001, a002 и так далее.

Команда LOGIN. После того как по транспортному протоколу (например, TCP) было установлено соединение и от сервера пришла строка приветствия, клиент должен зарегистрироваться в системе.

Для этого чаще всего используется команда LOGIN. Аргументом команды является строка с идентификатором и паролем клиента.

```
S:   * OK IMAP4 revl Service Ready
C:   aOO1 login ali sesam
S:   aOO1 OK LOGIN completed
```

Команда LOGIN передает пароль и идентификатор пользователя по сети в открытом виде. Если пользователю необходима защита информации своей почты, он может пользоваться командой AUTHENTICATE. Аргументом команды является строка, указывающая механизм аутентификации, которым желает воспользоваться данный пользователь. В зависимости от выбранного типа аутентификации строится дальнейший обмен между сервером и клиентом. Например, при использовании механизма шифрования KERBEROS аутентификация выглядит следующим образом.

```
S: * OK KerberosV4 IMAP4revl Server
C: AO 01 AUTHENTICATE KERBEROS_V4
S:  + AmFYig==
 C: BAcAQrJ5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
+nZIiriJjnTNHJUtxAA+oOKPKfHEcAFs9a3CL50ebe/ydHJUwYFd
WwuQlMWiy6IesKvjL5rL9WjXUb9MwT9bpObYLGOKilQh
S: + or//EoAADZI=
C: DiAF5MgA+oOIALuBkAAmw==
S: A001 OK Kerberos V4 authentication successful
```

После регистрации в системе клиент должен выбрать каталог (папку) сообщений, с которым он будет работать. Выбор каталога осуществляется командой SELECT.

Аргументом команды является имя почтового каталога.

```
C A142 SELECT INBOX
S * 172 EXISTS
S * 1 RECENT
S * OK [UNSEEN 12) Message 12 is first unseen
S * OK [UIDVALIDITY 3857529045] UIDs valid
S * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S A142 OK [READ-WRITE] SELECT completed
```

Сервер IMAP4, прежде чем подтвердить завершение обработки команды, передает клиенту атрибуты данного каталога. В показанном выше примере.

1. В папке "INBOX" - 172 сообщения (строка "* 172 EXISTS").
2. Из них одно только что поступившее (строка "* 1 RECENT").

3. В папке есть непрочитанные сообщения, минимальный порядковый номер непрочитанного сообщения - 12 (строка "* OK [UNSEEN 12] Message 12 is first unseen").
4. Уникальный временный идентификатор папки INBOX в данной сессии - 3857529045 (строка "* OK [UIDVAL1DITY 3857529045] UIDs valid").
5. Сообщения в данной папке могут иметь флаги, указанные в строке FLAGS (строка "* FLAGS (\Answered \Flageed VDeleted N^" \Draft)").
6. Клиент может менять у сообщений флаги "\Deleted" и "\Seen" (строка "* OK [PERMANENTFLAGS (\Deleted \Seen *)] Limited ").
7. Клиент имеет права на запись и чтение сообщений из INBOX (строка "A142 OK [READ-WRITE] SELECT completed").

Команда SELECT устанавливает текущий каталог для работы клиента. Если пользователю необходимо получить информацию о состоянии какого-либо каталога, достаточно воспользоваться командой EXAMINE с именем каталога в качестве аргумента команды, например.

```
C: A932 EXAMINE bloop
S: * 17 EXISTS
...
```

Команда EXAMINE возвращает те же параметры, что и команда SELECT, а отличается от команды SELECT только тем, что открывает заданный почтовый ящик исключительно на чтение.

Если необходимо запросить статус какой-либо папки, не меняя текущий каталог, можно воспользоваться командой STATUS. В качестве параметров данной команде придаются: имя папки и тип запрашиваемой информации. В зависимости от указанного типа, команда может возвращать: количество сообщений в папке, количество новых сообщении количество непрочитанных сообщений, UIDVALIDITY каталога, UID следующего сообщения, которое будет добавлено в данную папку, например.

```
<>ttC: Д042 STATUS blob (MESSAGES UNSEEN)
S: * STATUS blob (MESSAGES 231 UNSEEN 12)
S: A042 OK STATUS completed
```

Чтобы получить список папок (подкаталогов), находящихся в определенной папке и доступных клиенту, можно воспользоваться командой LIST. Аргументами команды являются: имя каталога, список подкаталогов который хотим получить (пустая строка - "" означает текущий каталог) и маска имен подкаталогов. Имена каталогов и маски имен подкаталогов могут интерпретироваться по-разному в зависимости от реализации почтовой системы и структуры описания иерархии папок. Например, список папок, находящихся в корне, можно получить так.

```
C: A004 LIST "/" *
S: * LIST (\Noinferiors ) "/" INBOX
S: * LIST <\Noinferiors ) "/"•• OUTBOX
S: * LIST <\Noinferiors ) "/".. WasteBox
```

```
S: A004 OK LIST completed
```

Ответ сервера содержит список папок в соответствии с их положением в иерархии и флаги данных папок (флаг "\Noinferiors" означает, что внутри данной папки нет и не может быть построенной иерархии).

После получения информации на каталог пользователь может прочитать любое сообщение или определенную группу сообщении, часть сообщения или определенные атрибуты сообщения. Для этого используется команда FETCH.

Аргументами данной команды являются порядковый номер сообщения и критерии запроса. Критерии содержат описание вида возвращаемой информации. Например, можно запросить части заголовков или UID-сообщений в папке, или сообщения, имеющие или не имеющие определенные флаги. Так запрос заголовков сообщений, находящихся в INBOX с порядковыми номерами от 10 до 12, будет выглядеть так.

```
C: A654 FETCH 10:12 BODY [HEADER]
S: * 10 FETCH (BODY [HEADER] {350}
S: Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDTl
S: From: raan@globe.com
S: Subject: Hi
S: To: imap@world.edu
S: Message-Id:
S^ mime-Vresion: 1.0
S: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S: )
S: *11 FETCH ....
S: *12 FETCH ....
S: A654 OK FETCH completed
```

После просмотра сообщения, пользователь может сохранить его с другими флагами, добавить или удалить флаги сообщения (пометить данное сообщение на удаление). Для этого используется команда STORE. Аргументами команды являются: номера сообщений, идентификатор операции и перечень флагов. Например, операция добавления флага удаления ("\Deleted") трем сообщениям выглядит следующим образом.

```
C: AOO3 STORE 2:4 +FLAGS (\DELETED)
S: *2 FETCH FLAGS (\Deleted \ Seen)
S: *3 FETCH FLAGS (\Deleted )
S: *4 FETCH FLAGS (\Deleted \Flagged \Seen)
S: A003 OK STORE completed
```

S: A003 OK STORE completed.

Ответом на выполнение команды будут переданы строки новых флагов указанных сообщений.

* Пользователь также может организовать поиск сообщений по определенным критериям. Для этого используется команда SEARCH. Критерий поиска состоит из комбинации нескольких условий поиска, а результатом поиска будет множество сообщений, находящихся в пересеченных условий. Условия могут налагаться на состав, структуру тела или заголовка сообщений, а также на флаги, размер, идентификатор периоды дат сообщений. Результатом работы команды является строка, состоящая из последовательных номеров сообщений, удовлетворяющих критерию поиска. Например, поиск всех непрочитанных сообщений, поступивших от "smith" с 1-03-96, будет выглядеть так.

```
C: A282 SEARCH UNSEEN FROM 'Smith" SINCE 1-Mar-1996
S: * SEARCH 2 84 882
S: A282 OK SEARCH completed
```

Результатом поиска будут сообщения с последовательными номерами 2, 84 и 882. * IMAP4 позволяет не только искать и читать сообщения в каталогах, этот протокол позволяет добавлять, копировать и перемещать сообщения в каталоги. Добавление сообщения в папку можно осуществить командой APPEND.

```
C: A003 APPENDSAVED-MESSAGES (\Seen) {310}
C: Date: Mon, 7 Feb 1997 21:52:25 - 0800 {PST}
C: From: Fred Foobar
C:     Subject:
aftenoon
meeteng C: TO:
mooch@owatagu.
siam.edu
C: Message-Id:
C: Mime-Version: 1.0
C: Content-Type: Text/PLAIN; CHARSET=US-ASCII
C:
C: Hello Joe, do you think we can meet at 3:30 tomorrow?
C:
S: A003 OK APPEND completed
```

Команда COPY копирует сообщения с заданными порядковыми номерами в указанный каталог, например.

```
C: A003 COPY 2:4 MEETENG
S: A003 OK COPY completed
```

Основы работы веб-сервера и протокол HTTP

Для того чтобы разобраться, как устроен веб-сервер, нужно рассмотреть основные его компоненты. Кроме того, веб-сервер обычно размещается на хостинге. Что такое хостинг и какие виды существуют, мы с вами разберем в этой теме. Будут рассмотрены: проблемы масштабирования веб-приложений, основные понятия HTTP, особенности работы технологии AJAX и протокола HTTP, отличия в работе HTTP2.

HTTP — простой текстовый протокол, позволяющий запросить у веб-сервера информацию. HTTP-протокол позволяет передавать заголовки (информация о user-agent-е, в роли которого, как правило, выступает браузер, но не всегда. curl и wget тоже user-agent-ы, таким образом не каждый HTTP-клиент — браузер, веб-сервер также сообщает информацию о себе, метаинформацию о документе, кодировку и тип содержимого: MIME-type, что позволит клиенту корректно раскодировать информацию).

Существует несколько типов запросов, но наиболее часто применяются два: GET и POST.

GET служит для запроса страницы. Как правило, в качестве адреса ресурса в GET передаётся путь к файлу относительно веб-директории. Например, /articles/article.html.

Либо, если запрашивается веб-приложение, ему могут быть переданы параметры /articles.php?page=12.

Возможно настроить веб-сервер так, чтобы веб-приложение было замаскировано, а параметры передавались через имена «папок» и «файлов». То есть /articles/pages/12.html на самом деле может означать как файл 12.html, хранящийся в директории pages, которая находится в директории articles, так и некий скрипт, у которого запрашивается статья с номером 12.

Такие запросы благодаря параметрам или пути к ресурсу могут передавать информацию на сервер. Но такие значения кешируются и сохраняются в истории браузера, поэтому для передачи информации на сервер используются запросы POST.

Как правило, с помощью POST отправляются новые файлы на веб-сервер (upload), новые записи на форумах и блогах, логин и пароль для входа в ту или иную систему. Запрос POST, как правило, передается скрипту на PHP (или другом языке), и тот его обрабатывает.

В ответ сервер может вернуть HTML-код запрошенного ресурса или двоичный файл (например, картинку). В ответе веб-сервер сообщает код.

- 200 Ok — все хорошо, ресурс найден и отправлен пользователю.
- 201 Created — новая страница на сайте создана (например, скриптом PHP, который получил запрос POST).
- 301 — ресурс переехал.

- 404 — страница не найдена (либо в файловой системе, либо скрипт не сумел найти соответствующую запись в БД).
- 403 — доступ запрещен (например, при HTTP-аутентификации пользователь не сумел ввести правильный пароль).
- 500 — ошибка сервера (ошибка в конфигурационном файле).

Веб-сервер

Веб-сервер – это понятие может подразумевать аппаратную или программную составляющие.

Это компьютер, на котором установлено программное обеспечение, выполняющие ответы на запросы пользователей.

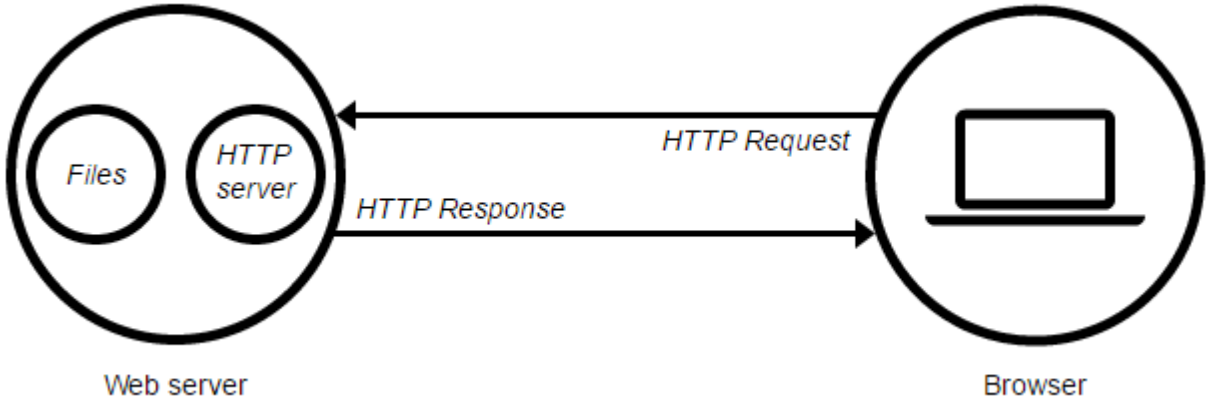
Веб-сервер – это программа, принимающая запросы от клиентских программ (браузеров), контролирующая доступ к ресурсам и генерирующая ответы на запросы.

Понятие "Веб-сервер" может относится как к железу, так и к программному обеспечению.

С точки зрения железа, `Веб-сервер` - это компьютер, который хранит ресурсы сайта (HTML-документы, CSS-стили, JavaScript-файлы и другое) и доставляет их на устройство конечного пользователя (веб-браузер и т.д.). Обычно подключен к сети Интернет и может быть доступен через доменное имя, например, mozilla.org. mozilla.org.

С точки зрения ПО, веб-сервер включает в себя некоторые вещи, которые контролируют доступ веб-пользователей к размещенным на сервере файлам, это минимум HTTP-сервера. HTTP-сервер - это часть ПО, которая понимает URL'ы (веб-адреса) и HTTP (протокол, который использует ваш браузер для просмотра веб-страниц).

Когда браузер производит обращение к странице, находящейся на веб-сервере, то сервер производит считывание файла или генерацию запрошенного контента и производит передачу контента клиенту.



Простыми словами, когда браузеру нужен файл, размещенный на веб-сервере, браузер запрашивает его через HTTP. Когда запрос достигает нужного веб-сервера (железо), сервер HTTP (ПО) передает запрашиваемый документ обратно, также через HTTP.

Мы будем рассматривать веб-сервер как программу, т.е. сервис, принимающий HTTP-запросы и выдающий ответы, как правило, содержащие HTML-код (но могут присутствовать и двоичные файлы).

Apache2 — популярный и, пожалуй, лидирующий веб-сервер. Есть и другие веб-сервера.

- lighthttpd — используется во встраиваемом оборудовании с линуксом на борту.
- Microsoft IIS — в основном на Windows Server-ах.
- nginx — еще один популярный веб-сервер, более того, часто работающий в связке с Apache.
- Google Web Server — вариация Apache2 от Google.

Технически веб-приложение не обязательно должно использовать отдельный веб-сервер. Возможна реализация протокола HTTP внутри самого приложения, но такие решения могут применяться для встраиваемых систем, и по большей части, применяется тот или иной веб-сервер.

LAMP — Linux+Apache2+MySQL+PHP — наиболее популярная конфигурация сервера, ориентированного на выдачу веб-содержимого. Linux мы уже умеем поставить и настроить, осталось изучить остальные компоненты LAMP.

Стоит отметить, что Apache2 может использоваться и в Windows как для сервера, так и для локальной разработки (также такая конфигурация называется WAMP).

Виды веб-серверов

Существует два типа веб серверов.

Статический – передает запрошенный с него контент в исходном виде.

Динамический – сервер, включающий в себя статический сервер и программный интерпретатор, обрабатывающий файлы перед их передачей клиенту. Динамический веб-сервер также называют сервером приложений и баз данных.

Статический веб-сервер или стек состоит из компьютера (железо) с сервером HTTP (ПО). Мы называем это “статикой”, потому что сервер посылает размещенные на нем файлы в браузер “как есть”.

Динамических веб-сервер состоит из статического веб-сервера плюс дополнительного программного обеспечения, наиболее часто сервером приложений и базы данных. Мы называем его “динамический”, потому что сервер приложений изменяет исходные файлы перед отправкой в ваш браузер по HTTP.

Установка веб-сервера

Для того чтобы работать с веб-страницей, веб-сервер не нужен, но если мы планируем работать с динамическими страницами, потребуется программа, выполняющая обработку скриптовых языков. Мы с вами будем работать с одним из самых распространенных веб серверов: apache.

Далее отдельно рассмотрены процессу установки для linux и windows.

Обратите внимание: не открывайте внешний доступ к вашему сайту.

- Веб-сервер по умолчанию рассчитан на локальное использование.
- Может быть использован для проникновения в вашу систему (небезопасно).

- Не оптимизирован для больших нагрузок.

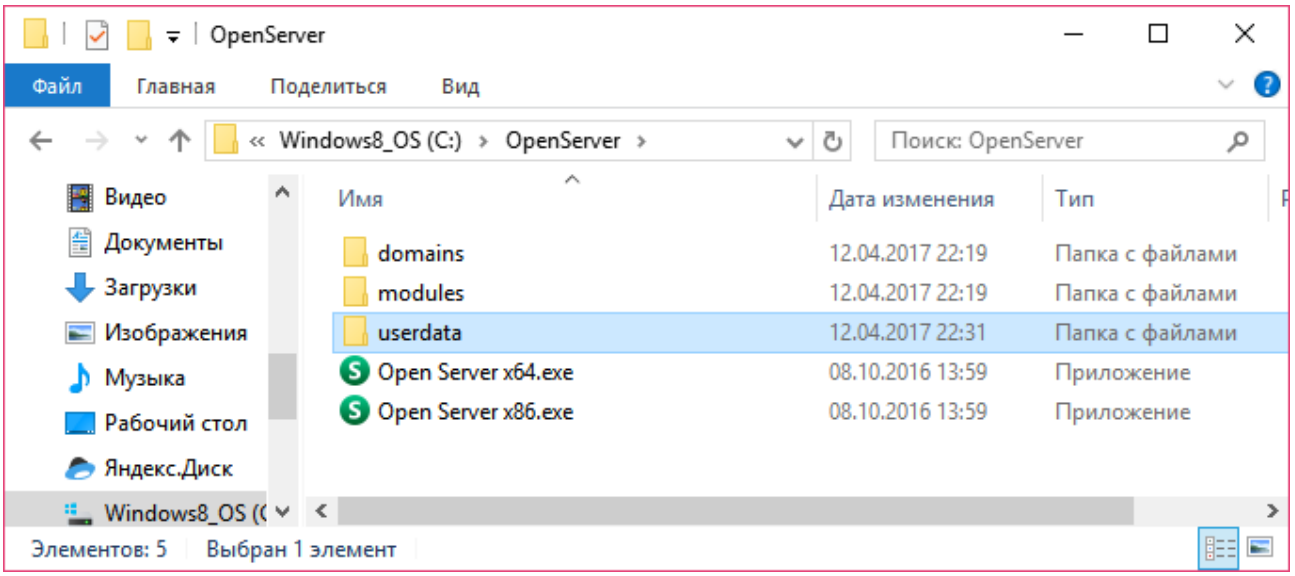
Установка веб-сервера для локальной разработки на ПК с Windows

Если сделать и отладить веб-страничку на html/css/js можно без каких-либо дополнительных инструментов, просто размещая проект в некой папке, для отладки веб-приложений, например, php, уже нужна реализация сервера. Можно воспользоваться полноценным сервером, но для небольших задач можно поставить локальный сервер, который будет отзываться на 127.0.0.1 и выполнять скрипты, отдавая результат по протоколу HTTP.

Одним из известных, но устаревших на данный момент таких локальных серверов, является проект Дмитрия Котерова, локальный сервер (Apache, PHP, MySQL, Perl и т.д.) и программная оболочка, для разработки сайтов на «домашней» (локальной) Windows-машине без необходимости выхода в Интернет.

Так как Денвер долгое время не обновлялся, у него появился независимый духовный преемник — OpenServer.

Также как и «Denwer» он имеет портативный формат (просто директория со всем содержимым на диске или флешке), точно также достаточно создать в директории с сайтами test.ru и перезапустить сервер, после чего сайт будет доступен локально через test.ru прямо в браузере.



Точно также, как и Денвер, OpenServer правит файл C:\Windows\System32\Drivers\etc\host, добавляя туда строчку.

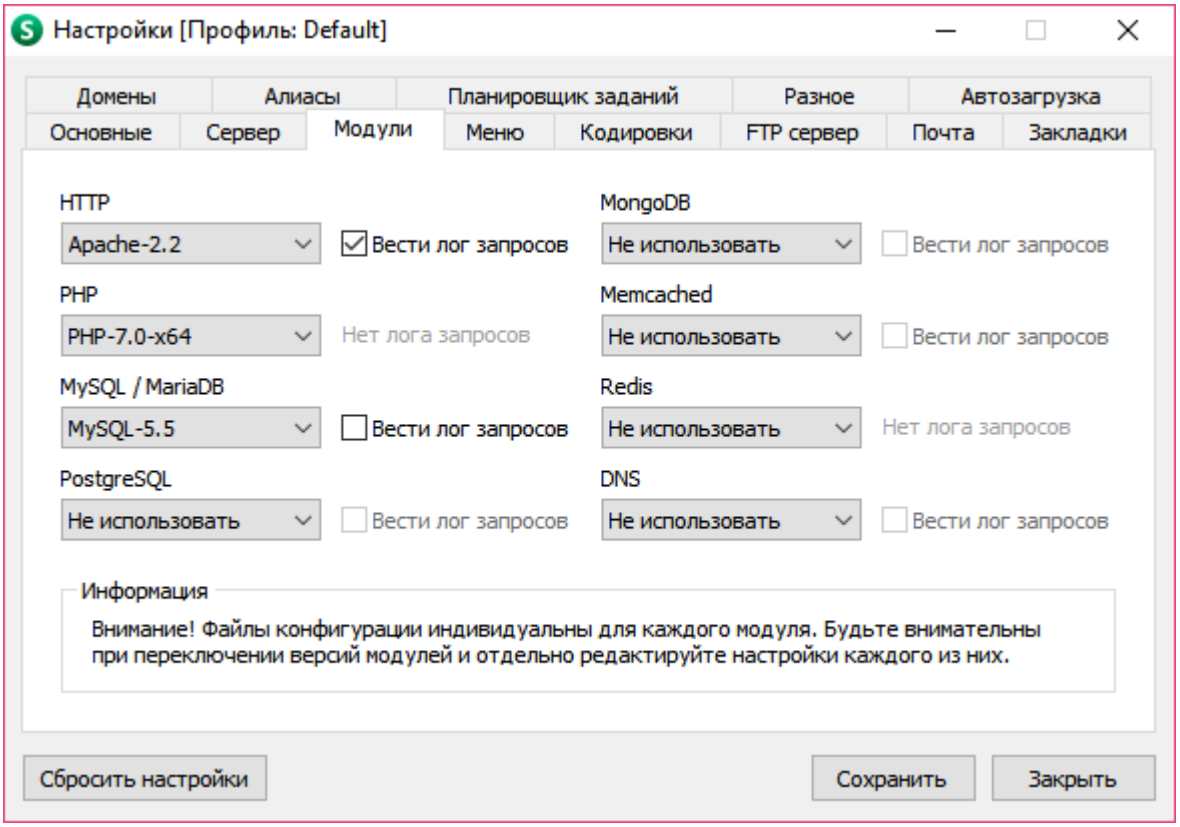
```
127.0.0.1 test.ru
```

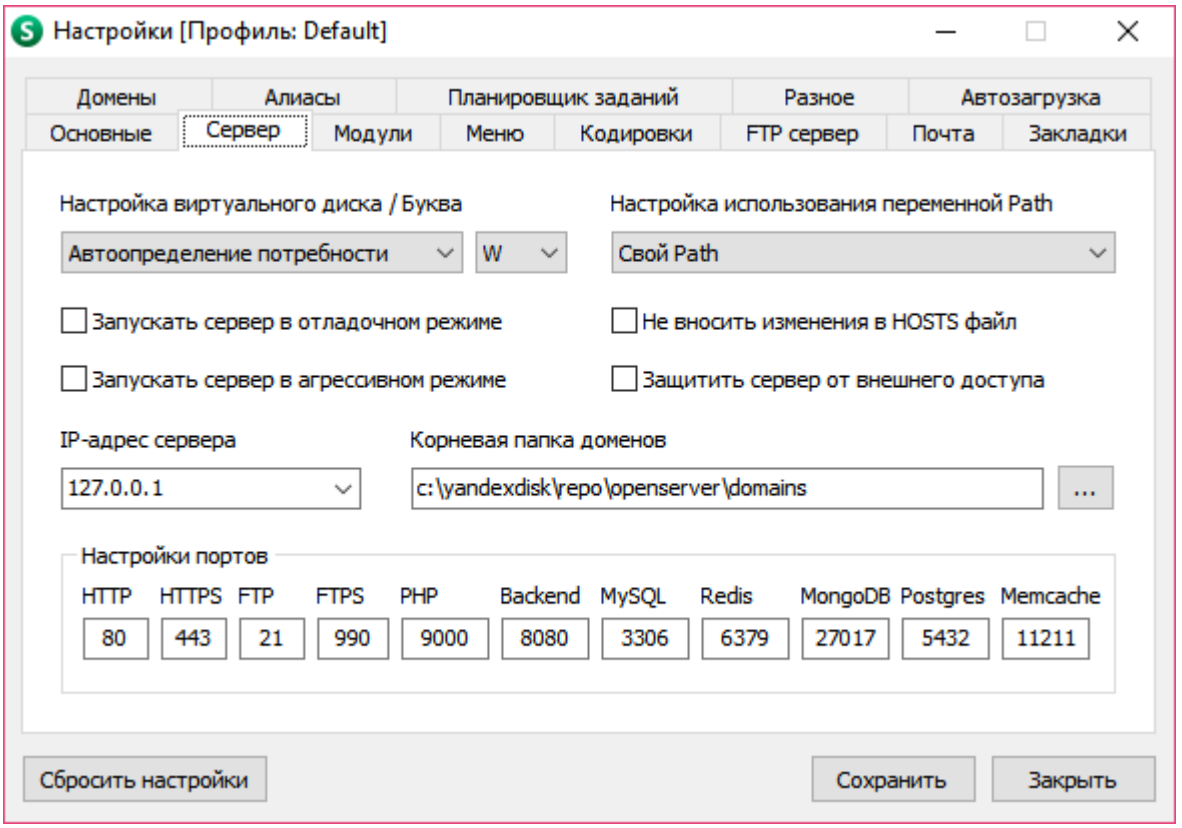
В отличие от Денвера OpenServer позволяет разместить проект в произвольной директории (что удобно в работе с git), а index.html не обязательно должен находиться в test.ru. Вы можете задать

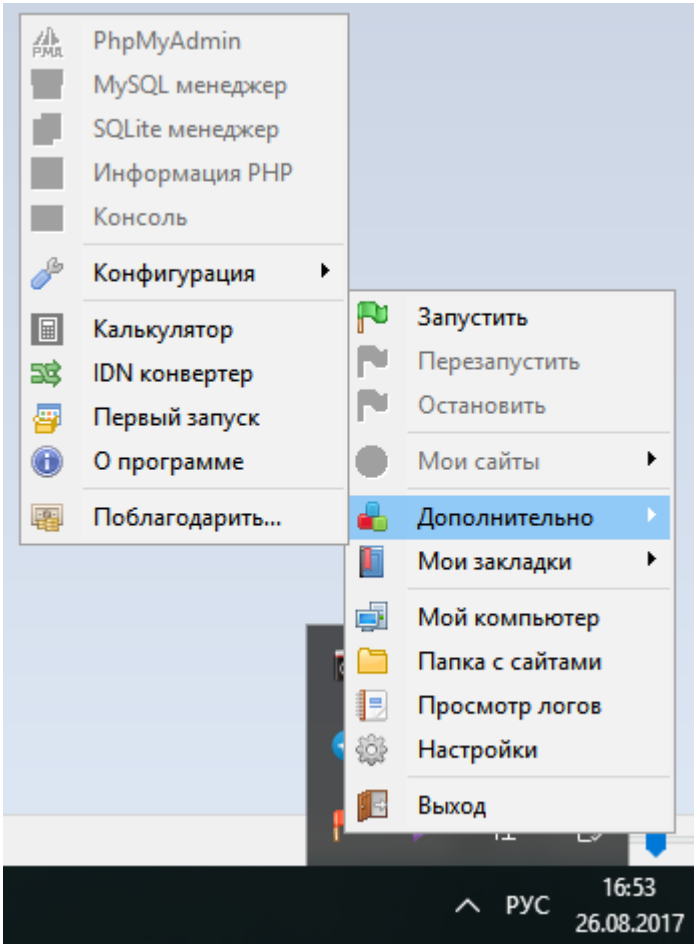
произвольную директорию с index.html (например, public), а скрипты хранить вне ее (что удобно для роутинга в фреймворках YII2 и Laravel).

Помимо апача, mysql и заглушки sendmail, сохраняющей почу в виде текстовых файлах, в OpenServer есть много всего нужного и полезного (Nginx, Redis, Memcache, MongoDB, Postgres, SQLIte и т.д.).

OpenServer можно бесплатно скачать с сайта https://ospanel.io/ (но при желании можно пожертвовать разработчикам, чтобы они не подумали повторить судьбу денвера).







OpenServer в системном Трее.

Есть и другие реализации. XAMPP, MAMP (для Mac OS X) и т.д. Но все же разработка под Windows того, что будет работать в Linux, не совсем точное воспроизведение среды разработки. Потому для серьезной разработки лучше установить виртуальную машину (VirtualBox или VMWare Player – оба доступны бесплатно), установить Ubuntu 18, и/или купить VDS с Ubuntu 18 (или другой версией Linux) и полностью наслаждаться абсолютно идентичным окружением (использовать Docker, устанавливать любые ПО, которые понадобятся).

Установка на ПК с Linux (Ubuntu)

Для работы с веб-серверов в Linux нам понадобится LAMP. LAMP — акроним, обозначающий набор (комплекс) серверного программного обеспечения, широко используемый во Всемирной паутине. LAMP назван по первым буквам входящих в его состав компонентов.

Linux — операционная система Linux.

Apache — веб-сервер.

MySQL — СУБД.

PHP — язык программирования, используемый для создания веб-приложений (помимо PHP могут подразумеваться другие языки такие, как Perl и Python). Вы можете установить его, введя команды.

- sudo apt-get install tasksel.
- sudo tasksel install lamp-server.

Файлы, размещаемые на сайте, необходимо загружать в каталог: /etc/www/html.

Файлы будут доступны по локальному адресу вашего компьютера.

Если же вы хотите не просто установить LAMP, но досконально разобраться в установке и настройке Apache-сервера и Nginx-сервера для Linux, см. соответствующий раздел ниже.

Хостинг

Хостинг – услуга по предоставлению информационных ресурсов на внешнем сервере в сети. Используется для публикации сайтов в интернете. Для публикации сайта необходимо также приобрести доменное имя и настроить запись на обращение к серверам хостинга.

Хостинг (англ. hosting) — услуга по предоставлению ресурсов для размещения информации на сервере, постоянно находящемся в сети (обычно Интернет).

Физический хостинг включает в себя два типа хостингов: Dedicated и Colocation.

Первый вариант (Dedicated) наиболее дорогой. Вы арендуете аппаратный сервер, самостоятельно или при помощи технической поддержки провайдера осуществляете настройку и мониторинг работы сервера. Плюсом этого подхода является то, что вы можете разместить свой сервер у любого провайдера, предоставляющего данную услугу, и использовать его инфраструктуру (широкополосный доступ в интернет, отказоустойчивые линии питания, серверное оборудование). Вы снижаете стартовые затраты на закупку оборудования путем увеличения эксплуатационных расходов.

Минусами данного подхода является то, что если вы не используете мощности сервера – оборудование у вас простаивает. Второй минус - это высокая цена аренды.

Второй вариант (Colocation) физического хостинга заключается в размещение вашего оборудования в ЦОДе провайдера. Данный вариант обладает всеми плюсами первого, несколько меньшей ценой аренды места для сервера. К минусам можно отнести то, что если оборудование вам больше не требуется, вам необходимо будет его продавать.

Следующая группа хостингов – виртуальные хостинги. Данный вид значительно отличается от физического ценой в меньшую сторону. Разберем особенности каждого типа виртуальных хостингов.

Наиболее популярны из-за цены виртуальные разделяемые хостинги. Данное решение подойдет для небольшого сайта. Цена на такой хостинг начинается от нескольких долларов. Можно найти бесплатные варианты, где провайдеры размещают свою рекламу. Основной минус данного хостинга связан с тем, что на одном сервере размещаются десятки сайтов, и в случае проблем у одного из клиентов, будет перегружен весь сервер. Также в случае увеличения нагрузки на сервер из-за вашего сайта провайдер обратится к вам с просьбой решить эту проблему или перейти на другой более дорогой тариф, например VPS.

Virtual Private Server/Virtual Dedicated Server – это хостинг на базе виртуальной машины, может быть настроен провайдером, либо вы можете его настроить полностью самостоятельно (в зависимости от договора с провайдером). На базе данного типа хостинга функционируют облачные сервисы.

Виртуальный сервер идеально подходит для сайтов, превысивших возможности обычного хостинга, высоконагруженных сетевых служб, а также для проектирования, разработки и тестирования программного обеспечения. Закрытые корпоративные проекты с повышенными требованиями к безопасности и конфиденциальности данных в массе своей тоже базируются на виртуальных выделенных серверах — особенно это касается небольших компаний, для которых недоступны покупка или аренда физического сервера.

Облачный хостинг подразумевает возможности для быстрого масштабирования возможностей вашей системы путем наращивания ресурсов из пула.

Облачные хостинги делаются на классы в зависимости от типа предоставляемого сервиса.

IaaS = Infrastructure as a Service. Вам не потребуется покупать оборудование, не потребуется строить собственный дата-центр, не потребуется нанимать системных инженеров, которые отвечают за обслуживание техники на физическом уровне. Данную часть вы отдаете на обслуживание облачному провайдеру. В вашей зоне ответственности остается управление операционной системой, установкой и настройкой приложений.

DaaS = Data as a Service. Если вам нужно разместить большие объемы файлов, не заботясь о покупке и обслуживании систем хранения данных, то вы должны воспользоваться DaaS. Обслуживание техники на физическом уровне аналогично IaaS остается на стороне облачного провайдера. В вашей зоне ответственности остается управление операционной системой, установкой и настройкой приложений.

При переходе от модели IaaS к модели PaaS (Platform as a Service) дополнительно на сторону облачного провайдера передается управление операционными системами и базами данных.

Для варианта SaaS = Software as a Service на сторону облачного провайдера дополнительно передаются вопросы установки и настройки приложений, мониторинга, резервного копирования,

защищенной публикации в интернет, то есть полный перечень всех вопросов. При потреблении услуг в данной модели для вашей компании не обязательно содержать в штате технического специалиста. Роль менеджера ИТ может выполнять сотрудник даже с минимальным техническим бэкграундом для управления внешним контрактом.

Проблемы масштабирования веб приложений

С ростом популярности web-приложения его поддержка неизбежно начинает требовать всё больших и больших ресурсов. Для оптимизации работы необходимо учитывать следующие пункты.

- Оптимизация настроек веб-сервера.
- Использование кэширования.
- Наращивание аппаратных возможностей/оптимизация скриптов.
- Использование распределенных файловых систем.
- Использование облачных технологий.
- Использование нереляционных баз данных (NoSQL).
- Мониторинг.

Правильно сконфигурированный веб-сервер позволит обслуживать более большое количество пользователей, чем с конфигом по умолчанию.

Использование кэширования. В данном случае можно установить дополнительный веб-сервер, который будет отдавать статические страницы из кэша, тем самым снизив нагрузку на динамический веб-сервер. В качестве кэширующего сервера часто используют nginx.

Следующий пункт является наиболее спорным, поскольку не всегда оптимизация работы самого приложения даст значительный прирост, принимать решение о покупке нового оборудования или оптимизации работы кода нужно исходя из мониторинга. Кроме того, необходимо учитывать материальную сторону, не всегда оптимизировать сложное веб-приложение дешевле, чем развернуть еще один сервер.

Распределенные файловые системы и распределенные серверные системы позволяют обрабатывать колоссальное количество запросов. Сложно представить, какое количество серверов отвечает за работу поисковых систем.

Развертывание вашей системы в облаке позволит быстро масштабировать аппаратные возможности и на основе данных мониторинга после снижения нагрузки освобождать мощности, тем самым снижая расходы.

Использование нереляционных баз данных - это еще одна возможность для увеличения производительности вашего приложения. Они применяются чаще не для хранения всех данных приложения, а лишь для решения специфических задач (журналирование, кэширование, очереди заданий) и поэтому менее распространены в простых проектах.

Мониторинг веб-сервиса не позволяет увеличить его производительность, но дает информацию о том, когда это увеличение потребуется.

Облачные хостинги

Облачные технологии – это одновременное использование ресурсов нескольких серверов. Главная особенность облачного хостинга – это возможность покупки ресурсов по потребностям и оплата за

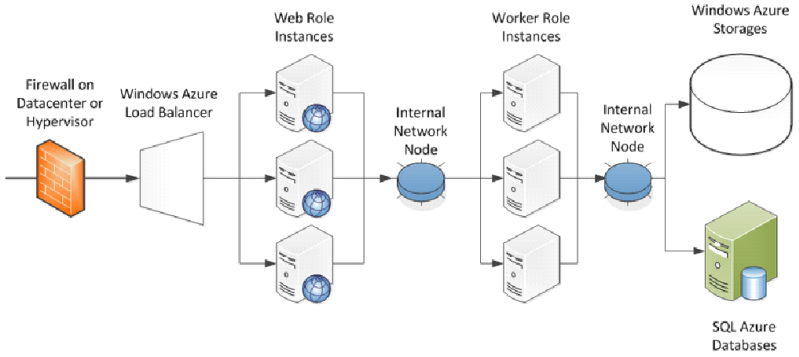
услуги в зависимости от нагрузки на сервер. При заказе облачного хостинга нужно лишь выбрать, какой размер дискового пространства вам понадобится для сайта. А оплата хостинга будет зависеть от того, сколько процессорного времени было затрачено на работу с вашими данными. Соответственно, чем больше будет расти посещаемость сайтов, тем больше будет использовано ресурсов серверов.

Доступность – ваши сайты будут доступные 24 часа 7 дней в неделю. Хостинг не размещён на одном сервере, а использует ресурсы целой сети серверов, и такая сеть может объединять более 100 единиц. Для оптимальной работы всех сайтов в такой сети используется распределение нагрузки (балансер нагрузки) и разделение дискового пространства.

Гибкость – для облачного хостинга можно подобрать оптимальную конфигурацию ресурсов под потребности определенного сайт.

Надежность – благодаря использованию сети серверов ваши сайты будут работать наиболее стабильно. Один или несколько серверов будет предоставлять вычислительные ресурсы, еще сервера будут делать бэкап данных, а другие в случае надобности будут восстанавливать эти резервные копии. Данная схема работы просто исключает заторможенность или недоступность сайта.

Оплата – вы платите только за те ресурсы, которые вам необходимы, если их окажется мало, вы всегда можете улучшить конфигурацию облачного хостинга или наоборот. Так что нет необходимости покупать ресурсы с «запасом». Никогда не получите письма от владельцев серверов о «превышении нагрузки» и просьбой перейти на более дорогой тарифный план или купить VDS.



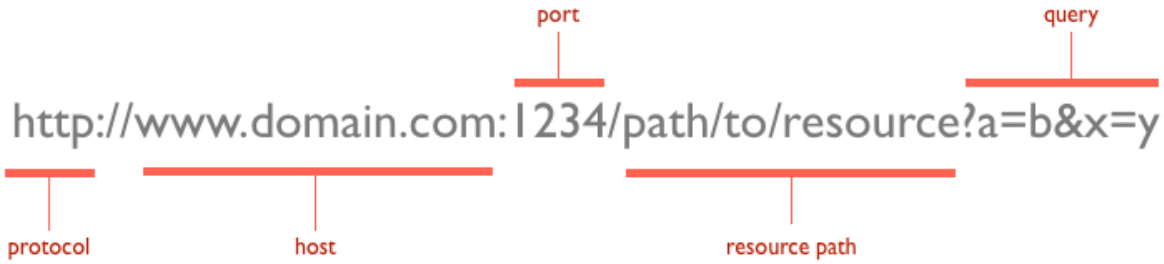
Основные понятия HTTP

HyperText Transfer Protocol (протокол передачи гипертекста) - протокол прикладного уровня, осуществляющий передачу структурированных данных в формате HTML. Также протокол позволяет передавать произвольные данные (документы/картинки/видео/музыку).

Технология является клиент серверной и использует.

- Веб-сервер.

- Браузер/клиентское приложение.



Методы HTTP

Существующие методы.

GET: получить доступ к существующему ресурсу. В URL перечислена вся необходимая информация, чтобы сервер смог найти и вернуть в качестве ответа искомый ресурс.

POST: используется для создания нового ресурса. POST запрос обычно содержит в себе всю нужную информацию для создания нового ресурса.

PUT: обновить текущий ресурс. PUT запрос содержит обновляемые данные.

DELETE: служит для удаления существующего ресурса.

Данные методы самые популярные и чаще всего используются различными инструментами и фреймворками. В некоторых случаях PUT и DELETE-запросы отправляются посредством отправки POST, в содержании которого указано действие, которое нужно совершить с ресурсом: создать, обновить или удалить.

Также HTTP поддерживает и другие методы.

HEAD: аналогичен GET. Разница в том, что при данном виде запроса не передаётся сообщение. Сервер получает только заголовки. Используется, к примеру, для того, чтобы определить, был ли изменён ресурс.

TRACE: во время передачи запрос проходит через множество точек доступа и прокси серверов, каждый из которых вносит свою информацию: IP, DNS. С помощью данного метода можно увидеть всю промежуточную информацию.

OPTIONS: используется для определения возможностей сервера, его параметров и конфигурации для конкретного ресурса.

Коды состояния

В ответ на запрос от клиента сервер отправляет ответ, который содержит, в том числе, и код состояния. Данный код несёт в себе особый смысл для того, чтобы клиент мог лучше понять, как интерпретировать ответ:

1xx: Информационные сообщения.

Набор этих кодов был введён в HTTP/1.1. Сервер может отправить запрос вида: Expect: 100-continue, что означает, что клиент ещё отправляет оставшуюся часть запроса. Клиенты, работающие с HTTP/1.0, игнорируют данные заголовки.

2xx: Сообщения об успехе.

Если клиент получил код из серии 2xx, то запрос ушёл успешно. Самый распространённый вариант - это 200 OK. При GET запросе сервер отправляет ответ в теле сообщения. Также существуют и другие возможные ответы.

- 201 Created: (в ответ на PUT и POST) запрашиваемый ресурс создан. Заголовок Location должен вернуть адрес созданного ресурса.
- 202 Accepted: запрос принят, но может не содержать ресурс в ответе. Это полезно для асинхронных запросов на стороне сервера. Сервер определяет, отправить ресурс или нет.
- 204 No Content: в теле ответа нет сообщения.
- 205 Reset Content: указание серверу о сбросе представления документа.
- 206 Partial Content: ответ содержит только часть контента. В дополнительных заголовках определяется общая длина контента и другая инфа.

3xx: Перенаправление.

Своеобразное сообщение клиенту о необходимости совершить ещё одно действие. Самый распространённый вариант применения: перенаправить клиент на другой адрес.

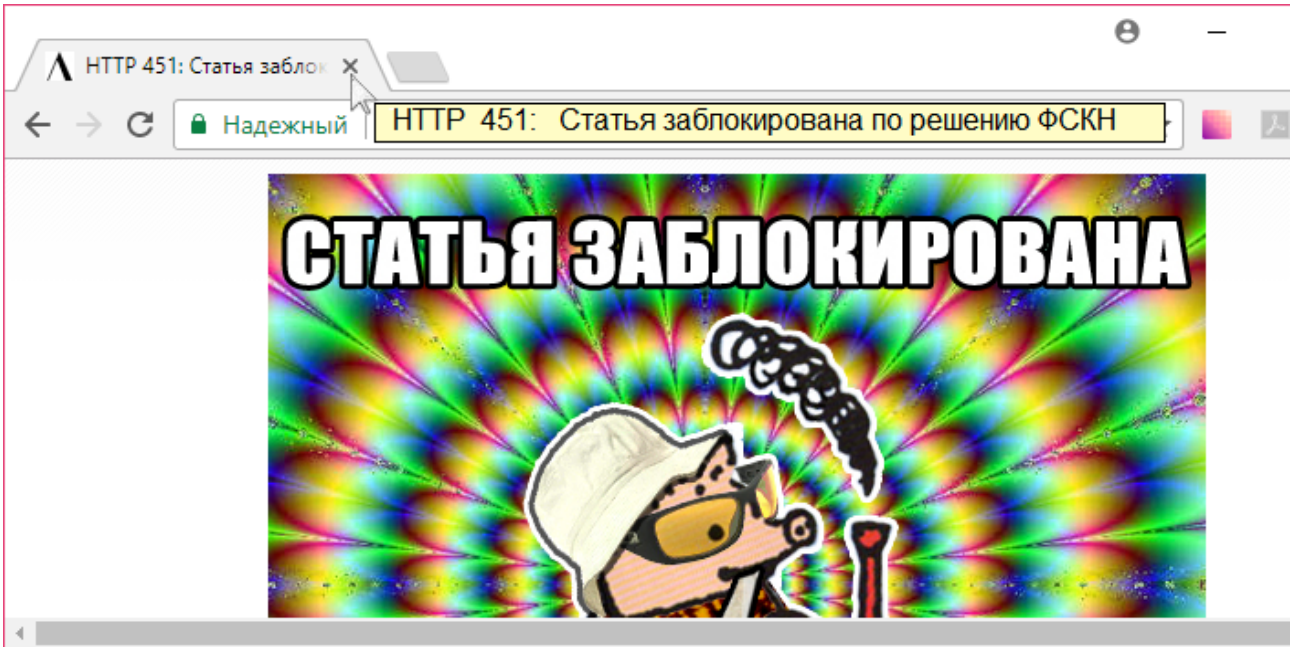
- 301 Moved Permanently: ресурс теперь можно найти по другому URL-адресу.
- 303 See Other: ресурс временно можно найти по другому URL-адресу. Заголовок Location содержит временный URL.
- 304 Not Modified: сервер определяет, что ресурс не был изменён и клиенту нужно задействовать закэшированную версию ответа. Для проверки идентичности информации используется ETag (хэш Сущности - Enttity Tag).

4xx: Клиентские ошибки.

Данный класс сообщений используется сервером, если он решил, что запрос был отправлен с ошибкой. Наиболее распространённый код: 404 Not Found. Это означает, что ресурс не найден на сервере. Другие возможные коды.

- 400 Bad Request: вопрос был сформирован неверно.
- 401 Unauthorized: для совершения запроса нужна аутентификация. Информация передаётся через заголовок Authorization.
- 403 Forbidden: сервер не открыл доступ к ресурсу. (Пользователь не ввел верные имя пользователя и пароль в случае base или digest авторизации, либо ресурс ограничен по IP-адресам, может быть доступен только локально).

- 405 Method Not Allowed: неверный HTTP-метод был задействован для того, чтобы получить доступ к ресурсу.
- 409 Conflict: сервер не может до конца обработать запрос, т.к. пытается изменить более новую версию ресурса. Это часто происходит при PUT-запросах.
- 451 Unavailable For Legal Reasons — ресурс заблокирован Роскомнадзором. Название намекает на роман Рэя Брэдбери «451 градус по Фаренгейту» (там пожарники книжки сжигали).



5xx: Ошибки сервера.

Ряд кодов, которые используются для определения ошибки сервера при обработке запроса. Самый распространённый: 500 Internal Server Error. Пример: ошибка в файле .htaccess.

Другие варианты.

- 501 Not Implemented: сервер не поддерживает запрашиваемую функциональность.
- 502 Bad Gateway: вы обратились к реверс-прокси (например, Nginx, обычно такие машины называют фронтендом), но он так и не получил данные от бэкенда (например, Apache, истек таймаут).
- 503 Service Unavailable: это может случиться, если на сервере произошла ошибка или он перегружен. Обычно в этом случае сервер не отвечает, а время, данное на ответ, истекает.

Заголовки HTTP

Заголовки HTTP (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы.

General Headers (рус. Основные заголовки) — должны включаться в любое сообщение клиента и сервера.

Request Headers (рус. Заголовки запроса) — используются только в запросах клиента.

Response Headers (рус. Заголовки ответа) — только для ответов от сервера.

Entity Headers (рус. Заголовки сущности) — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

Заголовки в HTML

Язык разметки HTML позволяет задавать необходимые значения заголовков HTTP внутри <HEAD> с помощью тега <META>. При этом название заголовка указывается в атрибуте http-equiv, а значение — в content. Почти всегда выставляется значение заголовка Content-Type с указанием кодировки, чтобы избежать проблем с отображением текста браузером.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html;charset=utf8">

<meta http-equiv="Content-Language" content="ru">
```

User Agent

User Agent — это клиентское приложение, использующее определённый сетевой протокол. Термин обычно используется для приложений, осуществляющих доступ к веб-сайтам таким, как браузеры, поисковые роботы (и другие «пауки»), мобильные телефоны и другие устройства.

Веб-сайты для мобильных телефонов часто вынуждены жёстко полагаться на определение User-Agent, так как браузеры на разных мобильных телефонах слишком различны. Поэтому мобильные веб-порталы обычно генерируют разные страницы в зависимости от модели мобильного телефона. Эти различия могут быть как небольшими (изменение размера изображений специально для меньших экранов), так и весьма существенными (формат WML вместо XHTML).

Строка User-agent также используется веб-мастерами для предотвращения индексирования «поисковыми пауками» некоторых страниц сайта, например, когда индексирование определённых страниц не имеет смысла или конкретный «паук» создаёт большую нагрузку на сервер. Веб-мастер может использовать специальный файл robots.txt для рекомендаций «пауку» или просто настроить веб-сайт не отдавать «пауку» эти страницы.

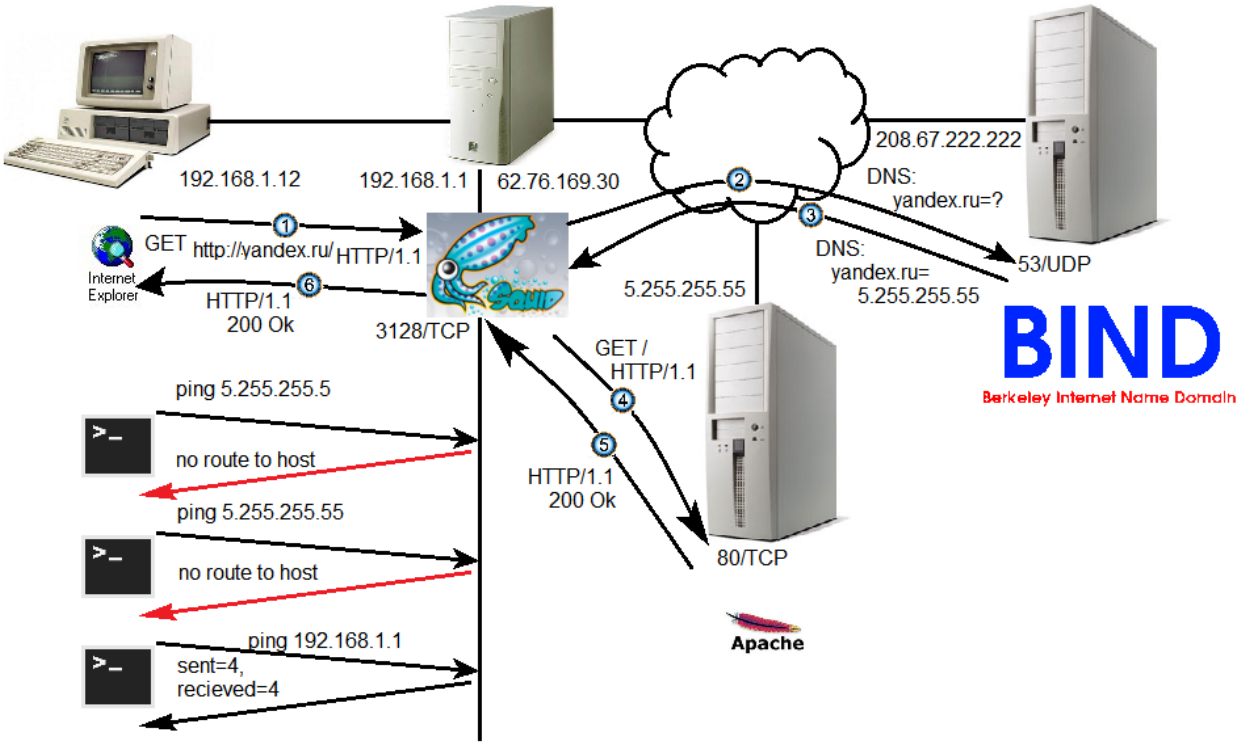
HTTP-Прокси

Кроме веб-серверов и user-agent’ов существует еще класс программ, которые могут выступать и в качестве клиентов, и в качестве серверов. Речь идет о HTTP-прокси.

Технология HTTP-прокси более древняя, чем NAT и позволяла предоставить доступ в Интернет узлам, у которых есть сеть, но нет доступа в Интернет. Нет пингов, не работает ftp и “асечки”, а

интернет есть. Как же так? В настройках TCP/IP таких узлов не было адреса шлюза по умолчанию (сейчас Windows определяет такие подключения как «Без доступа в Интернет», но вот в настройках браузера был пункт: прокси-сервер, с указанием IP-адреса, номера порта, логина и пароля. IP-адрес использовался локальный, а сам компьютер, на котором был установлен прокси-сервер, имел доступ и в Интернет, и в локальную сеть. Но маршрутизация между ними не была включена).

Когда клиент хотел получить доступ в интернет, он обращался не к удаленной машине, а устанавливал TCP-сессию на адрес прокси-сервера и у него просил все то, что мог бы попросить у настоящего сервера, будь подключение к Интернет. Прокси-сервер получал запрос и от своего имени обращался к веб-серверу, а получив ответ, возвращал его клиенту. Либо мог закешировать, и если страница не менялась, отдать ее еще потом несколько раз разным клиентам. Такая вот технология доступа в интернет 7-уровня.



В случае HTTP-прокси веб-сервер устанавливает соединение (1) на адрес прокси-сервера (192.168.1.1:3128) и запрашивает ресуср yandex.ru. Как он это делает? В строке запроса GET для прокси будет находиться не адрес ресурса, а полностью URL.

```
GET http://yandex.ru/ HTTP/1.1
```

Если у прокси этот ресурс уже есть в кеше, он может сразу отдать.

Если нет, то скорее всего прокси запросит значение доменного имени yandex.ru (2) у DNS-сервера (208.67.222.222). DNS-сервер (3) ответит, что yandex.ru это 5.255.255.55. Прокси-сервер установит соединение на 5.255.255.55:80 и сделает HTTP-запрос (4).

```
GET / HTTP/1.1
```

Веб-сервер отправить веб-содержимое на адрес 62.76.169.30 (5).

Прокси-сервер может закешировать страницу, а также передаст уже в рамках сессии установленной на 192.168.1.1 с 192.168.1.12 в адрес клиента 192.168.1.12 веб-страницу.

Как видимо, proxy – это практически Man In The Middle.

А как же работает прокси, если работа идет с HTTPS. В этом случае клиент обращается к серверу с просьбой установить соединение к некоему ресурсу. Прокси-сервер создаст тоннель и внутрь уже заглянуть не сможет. Для этого есть метод CONNECT.

```
CONNECT http://yandex.ru/ HTTP/1.1
```

Уже внутри сам клиент спросит.

```
GET / HTTP/1.1
```

Но с точки клиента TCP-соединение все равно установлено на адрес 192.168.1.1.

Но любые попытки пробраться в Интернет, кроме HTTP (и HTTP напрямую), успеха иметь не будут. Нет маршрута на сетевом уровне.

Когда-то единственный способ выпустить пользователей компьютеров с локальными адресами в Интернет (до NAT) и способ ускорить загрузку ресурсов. Скорости были небольшими (а то и вообще dial-up), контент был статический, java script использовался для рисования снежинок и листиков на странице. Сейчас кое-где прокси-серверы используются как способ контроля (веб 2.0 существенно ограничивает возможности кеширования, так как контент динамичный, а благодаря асинхронным запросом еще и собирается на ходу. Кеширование таких данных смысла не имеет. Скорости тоже уже большие. Из полезных вещей остаются возможность аутентификации, логирования и возможность ограничить скорость. С другой стороны требуется распределение нагрузки и кеширования на стороне веб-сервисов (такой прокси называется обратный прокси или реверс-прокси, на стороне сервера).

Выделяют следующие виды прокси.

- Непрозрачный прокси. Браузер знает, что работает через прокси, запросы отправляются на прокси.
- Gрозрачный прокси. Трафик завернут на соответствующую машину, клиент не знает, что работает через прокси.

Пример ПО прокси: squid (может работать как непрозрачный, прозрачный (только для http), обратный прокси).

- Реверс-прокси (обратный прокси).

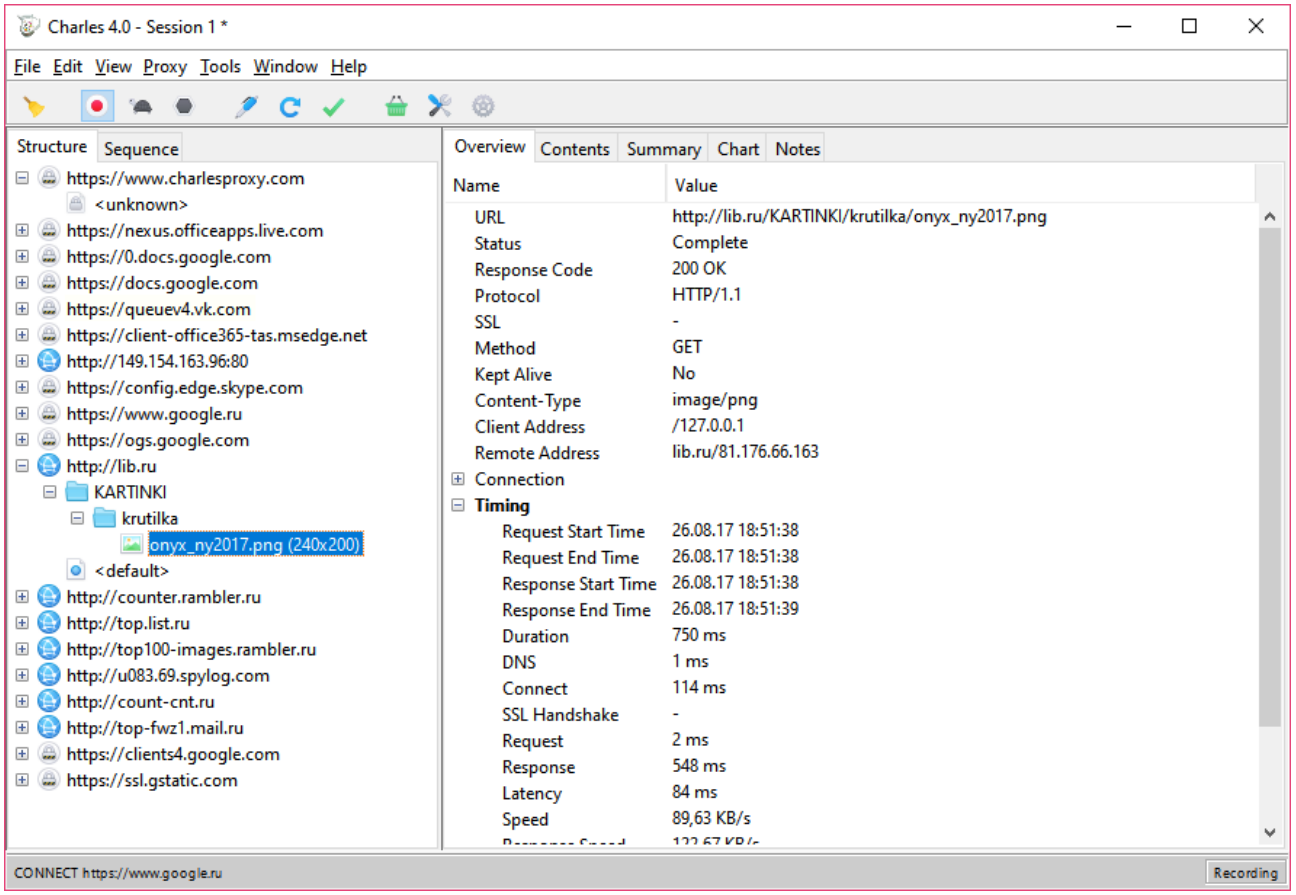
Раньше страницы были статичными, каналы - низкоскоростными. Проблему решало кеширование прокси-сервером информации. Сейчас информация динамическая, а каналы достаточно скоростные.

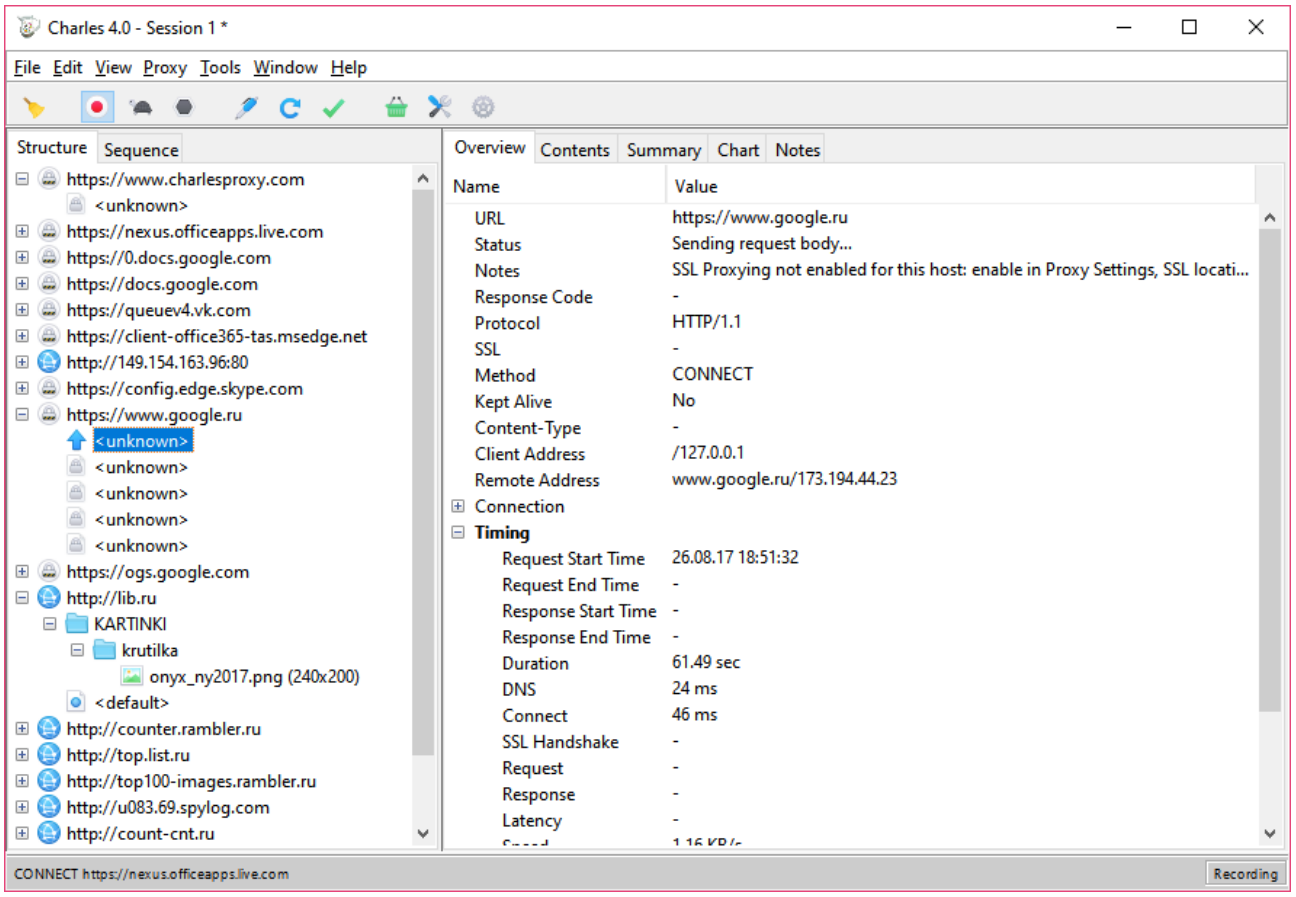
Потому возникает иная задача: распределение нагрузки по воркерам в высоконагруженных проектах и кеширование часто отдаваемых документов.

Примеры: squid, nginx (последний – скорее веб-сервер).

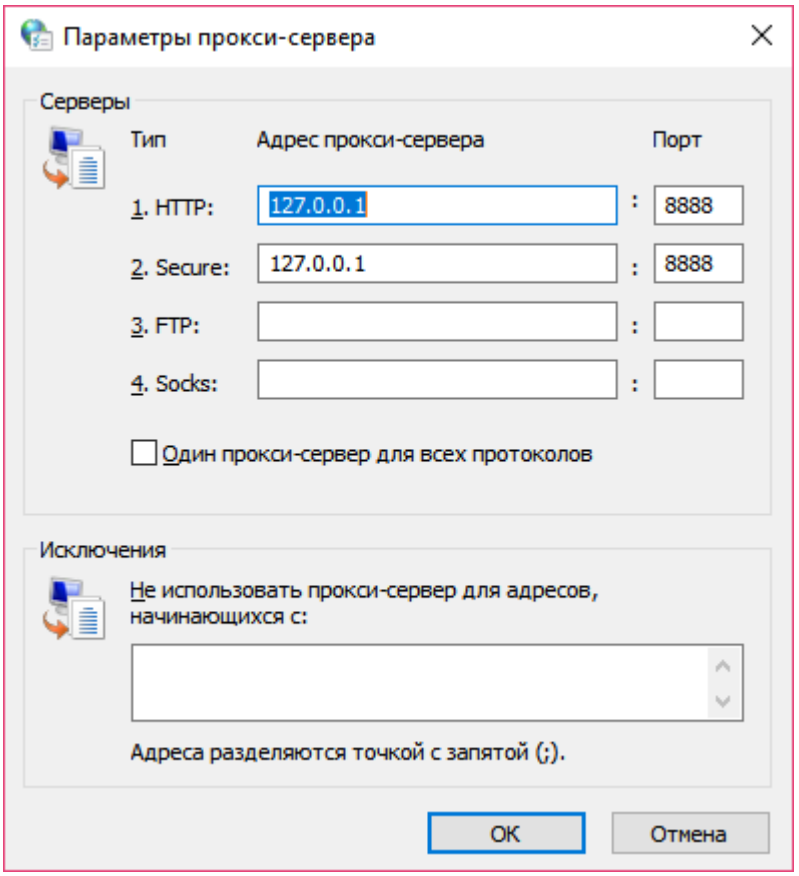
Непрозрачный прокси не сможет обрабатывать защищенный трафик (сертификаты как раз направлены на борьбу с атакой MITM). Теоретически возможна подмена сертификата, но браузер будет об этом информировать.

Также прокси-сервер может использоваться для отладки. В этом случае он будет запущен локально (127.0.0.1). Так Charles proxy (программа платная, но есть бесплатный пробный период, доступна для WIndows и MAC) позволяет анализировать HTTP-трафик, заголовки, умеет подменять сертификаты. Похоже на Wireshark только на прикладном уровне для HTTP.

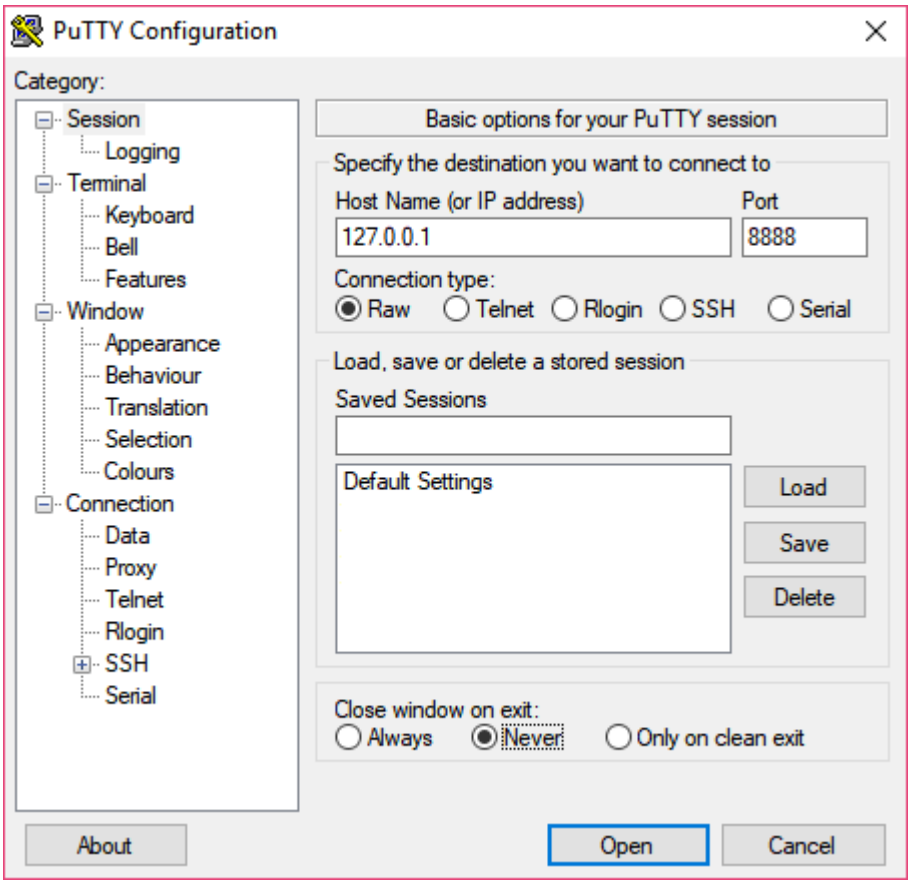


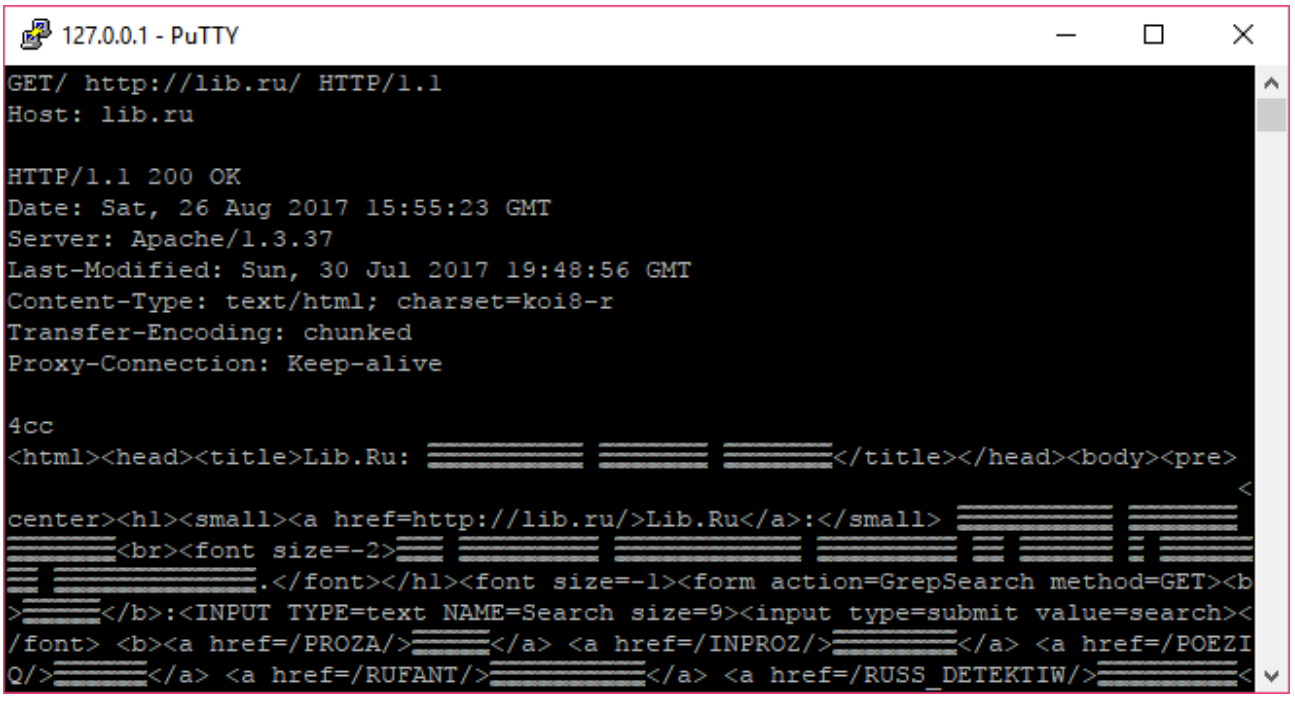


Charles-прокси сам изменяет настройки прокси при запуске.



Подключимся и мы туда же.


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
127.0.0.1
8888
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel


127.0.0.1 - PuTTY
GET/ http://lib.ru/ HTTP/1.1
Host: lib.ru
HTTP/1.1 200 OK
Date: Sat, 26 Aug 2017 15:55:23 GMT
Server: Apache/1.3.37
Last-Modified: Sun, 30 Jul 2017 19:48:56 GMT
Content-Type: text/html; charset=koi8-r
Transfer-Encoding: chunked
Proxy-Connection: Keep-alive
4cc
<html><head><title>Lib.Ru:
</title></head><body><pre>
<
center><hl><small>Lib.Ru:</small>

.</hl><font size=-1><form action=GrepSearch method=GET><b
>:<INPUT TYPE=text NAME=Search size=9><input type=submit value=search><
/font>   <a href=/POEZI
Q/> <a href=/RUSS_DETEKTIW/>

Cookie

Куки - данные, переданные веб-сервером клиенту, используемые каждый раз при отправке на сервер.

Сферы применения.

- Аутентификация пользователей.
- Хранения персональных настроек пользователей.
- Отслеживания сеансов пользователей.
- Мониторинг пользователей.

Ниже приведен пример установки куки на языке php.

```
<?php
$text = 'пример';
setcookie("Test_Cookie", $text, time()+3600);  /* срок действия 60 минут */
?>
```

С помощью следующего скрипта можно вывести содержимое установленных cookie.

```
<?php
echo$_COOKIE["Test_Cookie "];

?>
```

Sessions

Сессии и cookies предназначены для хранения сведений о пользователях при переходах между несколькими страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя и по запросу отсылаются браузером серверу.

Сессия во многом походит на cookie и представляет собой текстовой файл, хранящий пары ключ/значение, но уже не на машине клиента, а на сервере. Во многих случаях сессии являются более предпочтительным вариантом, чем cookies.

Уникальный идентификатор сессии SID передается через cookie, т.е. посредством HTTP-заголовка Set-Cookie, либо может быть передан POST/GET запросом.

За это отвечают две настройки в php.ini.

- session.use_cookies - если равно 1, то PHP передает идентификатор в куках, если 0 - то нет.
- session.use_trans_sid если равно 1, то PHP передает его, добавляя к URL и формам, если 0 - то нет.

Ниже приведен пример установки сессии на языке php.

```php
<?php

session_start();

$_SESSION['test']='Hello world!';

?>
```

С помощью следующего скрипта можно вывести содержимое установленных сессий.

```php
<?php

echo$_SESSION['test'];

?>
```

Еще один пример кода с использованием сессий, который фиксирует количество обновлений страницы пользователем.

```php
<?

session_start();

if (!isset($_SESSION['reload'])) $_SESSION['reload']=0;

echo "Страница обновлена ".$_SESSION['reload']++." раз. <br>";

echo   "<a   href=".$_SERVER['PHP_SELF'].">обновить";     //   данная   гиперссылка ссылается на страницу, на которой выполнен этот скрипт.

?>
```

Форматы сообщений запроса/ответа

Ниже приведена структура передаваемого сообщения с помощью HTTP.

```
message = <start-line>
          *(<message-header>)
          CRLF
          [<message-body>]
<start-line> = Request-Line | Status-Line
<message-header> = Field-Name ':' Field-Value
```

Между заголовком и телом сообщения должна обязательно присутствовать пустая строка. Заголовков может быть несколько.

- Общие заголовки.
- Заголовки запроса.
- Заголовки ответа.
- Заголовки сущностей.

Тело ответа может содержать полную информацию или её часть, если активирована соответствующая возможность (Transfer-Encoding: chunked). HTTP/1.1 также поддерживает заголовок Transfer-Encoding.

Типы заголовков мы с вами рассмотрели ранее.

HTTPS

HyperText Transfer Protocol Secure – веб-протокол, использующий шифрование для передачи запросов и ответов. Работает на TCP порту 443. Использует сертификаты для обеспечения конфиденциальности передачи. Основная задача протокола защита от сетевой атаки man-in-the-middle.

Название этого расширения — HTTPS (HyperText Transfer Protocol Secure). Для HTTPS-соединений обычно используется TCP-порт 443. HTTPS широко используется для защиты информации от перехвата, а также, как правило, обеспечивает защиту от атак вида man-in-the-middle — в том случае, если сертификат проверяется на клиенте, и при этом приватный ключ сертификата не был скомпрометирован, пользователь не подтверждал использование неподписанного сертификата и на компьютере пользователя не были внедрены сертификаты центра сертификации злоумышленника.

Области применения HTTPS.

- Почтовые сервисы.
- Файловые сервисы.
- Банковские системы.
- Интернет магазины.
- Социальные сети.

На данный момент HTTPS поддерживается всеми популярными веб-браузерами.

HTML

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML).

Отметим, что HTML5 является скриптовым языком и в отличие от прошлых версий является языком программирования.

Ниже представлен пример страницы на HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Это заголовок страницы</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf8">
<link href="styles.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1>Здравствуйте!</h1>
<!-- Это комментарий -->
<p>Это моя первая страница HTML. <b>Этот текст выводится жирным
шрифтом.</b></p>
</body>
</html>
```

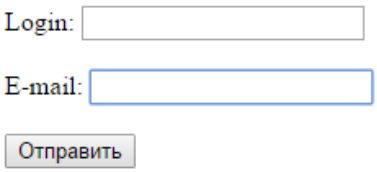
HTTP и HTML

Протокол HTTP предусматривает различные методы взаимодействия, которые были рассмотрены нами ранее. Взаимодействие с пользователем производится с помощью форм, которые могут содержать различные элементы ввода информации (кнопки, чек боксы, поля ввода и т.д.).

Ниже представлена форма ввода, использующая метод GET и содержащая 3 элемента (2 поля ввода и кнопку).

```
<form method="GET" action="srcipt.php">
 Login: <input type="text" name="login"><br><br>
 E-mail: <input type="text" name="email"><br><br>
<input type="submit" value="Отправить"> </form>
```

На рисунке ниже представлен результат обработки HTML-кода браузером.



HTML+ Скриптовые языки

Скриптовые языки значительно расширяют возможности отображения контента страниц. Создание динамичных страниц, изменяющих контент, возможно благодаря использованию скриптовых языков. Скриптовые языки могут исполняться на стороне сервера при генерации контента передаваемого браузеру или на стороне клиента. Наиболее широко используемыми являются следующие языки.

- JavaScript.
- PHP.
- Python.
- Ruby.
- Perl.

Создадим файл index.php, который выводит надпись hello world! Сравним вывод содержимого в блокноте/браузере/браузере при ответе веб-сервера.

```
<? Echo'hello world!';?>
```

Второй пример приведен для JavaScript, данный пример исполняется в браузере. Выполним аналогичные действия для данного скрипта.

```
<!DOCTYPE HTML> <html> <head>

 <!-- Тег meta для указания кодировки --> <meta charset="utf-8">

</head>

<body>

<p>Начало документа...</p>

<script> alert( 'Привет, Мир!' ); </script>

<p>...Конец документа</p>

</body>

</html>
```

GET и POST

Сравнение методов работы GET и POST. Как нами было рассмотрено ранее, метод GET отличается передачей информации в адресной строке. Проверим это на практике.

```
<html>

 <body> <form method="GET">

<!--указание метода GET--> Login: <input type="text" name="login"><br>

 E-mail: <input type="text" name="email"><br>

 <input type="submit" value="Отправить"> </form>

<?php

//С помощью суперглобального массива $_GET

 //выводим принятые значения:

 echo "<br/>login = ". $_GET['login'];

 echo "<br/>email = ". $_GET['email'];?>

 </body>

</html>
```

На уроке мы выполнили аналогичные действия для метода POST, изменив обработчик и форму.

Загрузка данных через PHP

Для наглядности переделаем прошлый пример и разобьем его на 2 файла form.php и post.php.

```
<html>
<body>
<form method="POST"> <!--указание метода GET-->
 Login: <input type="text" name="login">
<br> E-mail: <input type="text" name="email">
<br> <input type="submit" value="Отправить">
</form>
</body>
 </html>
```

Файл, принимающий переданные данные post.php.

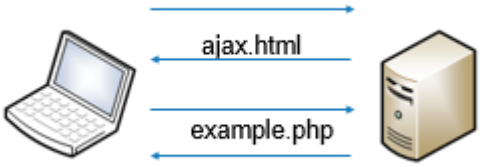
Обратите внимание, что при нажатии на кнопку передачи переменные не используют адресную строку для передачи.

```
<html>

<body>

<?php // С помощью суперглобального массива $_POST // выводим принятые значения:

echo "<br/>login = ". $_POST ['login'];

echo "<br/>email = ". $_POST ['email'];?>

</body>

 </html>
```

Особенности работы технологии AJAX и протокола HTTP

Содержание блока изменится согласно проработке файла example.php

Передать данные скрипту



```
<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<script                                                    type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js">

$(document).ready(function(){    $("#send").click(function()

{$("#status").load("example.php","y=Yes&n=No");    })});

</script>

</head>

<body>

   <p   id="status">Содержание   блока   изменится   согласно   проработке   файла
example.php</p>

<input id="send" type="button" value="Передать данные скрипту" />

</body>

</html>
```

Содержимое файла example.php приведено ниже.

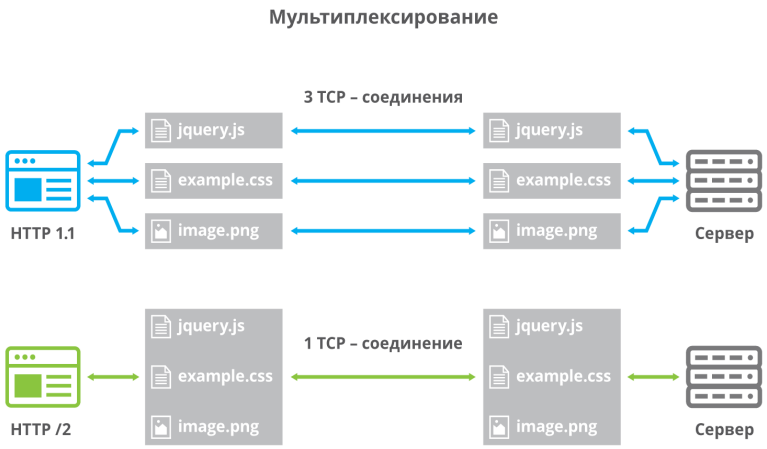
```
<?php

header("Content-type: text/plain; charset=utf-8");

header("Cache-Control: no-store, no-cache, must-revalidate");

header("Cache-Control: post-check=0, pre-check=0", false);// Получаем и проверяем

if (isset($_REQUEST['y'])) { $y = stripslashes($_REQUEST['y']); if ($y == '') {
unset($y);} }

if (isset($_REQUEST['n'])) { $n = stripslashes($_REQUEST['n']); if ($n == '') {
unset($n);} }

echo  "Получили  переменные  <strong>$y</strong>  и  <strong>$n</strong>  .  Всё
работает!";?>
```

SPDY и HTTP/2

В 2015 была утверждена новая версия HTTP — HTTP/2.

HTTP/2 уже поддерживается в популярных веб-серверах: Apache и Nginx.

Протокол SPDY - это реализованный Google в 2009 проект по ускорению работы HTTP. Оба протокола подразумевают необходимость поддержки и клиентом и браузером новой версии.



Протокол HTTP/2 является бинарным. По сравнению с предыдущим стандартом изменены способы разбиения данных на фрагменты и транспортирования их между сервером и клиентом.

В HTTP/2 сервер имеет право послать то содержимое, которое еще не было запрошено клиентом. Это позволит серверу сразу выслать дополнительные файлы, которые потребуются браузеру для отображения страниц без необходимости анализа браузером основной страницы и запрашивания необходимых дополнений.

Также часть улучшений получена (в первом черновике HTTP/2, который представлял собой копию спецификации SPDY) за счет мультиплексирования запросов и ответов для преодоления проблемы «head-of-line blocking» протоколов HTTP 1; сжатия передаваемых заголовков и введения явной приоритизации запросов.

Примеры API

Интернет-инфраструктура во многом заточена под WWW. 80 и 443 порты закрыты в последнюю очередь на файрволлах, http-трафик проходит через самое дремучее оборудование. Иногда http даже используется для туннелирования не http-трафика (rtmpt). Поэтому возникает мысль, а почему бы вместо разработки нового протокола не воспользоваться готовым. Проходит через NAT и прокси. Можно воспользоваться шифрованием. Поэтому часто для взаимодействия ПО (клиент-сервер и т.д.) используют HTTP в качестве транспорта. Такой подход нарушает модель OSI (но нас после 6 занятия

этим не удивишь!), иногда с точки зрения OSI такие протоколы относят к сеансовому уровню. По большей части это протоколы RPC (Remote Procedure Call).

SOAP (Simple Object Access Protocol — простой протокол доступа к объектам)

Один из наиболее известных вариантов — SOAP. Представляет собой библиотеку, которая инкапсулирует в HTTP XML-заголовки. Удобство: подключил библиотеку и готово, недостатки: избыточность и нечитаемость XML и не использование достоинств HTTP.

Пример SOAP-запроса (вкладывается в HTTP).

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>12345</productID>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

Пример SOAP-ответа.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productID>12345</productID>
         <productName>Стакан граненый</productName>
         <description>Стакан граненый. 250 мл.</description>
         <price>9.95</price>
         <currency>
             <code>840</code>
             <alpha3>USD</alpha3>
             <sign>$</sign>
             <name>US dollar</name>
             <accuracy>2</accuracy>
         </currency>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
```

```
</soap:Envelope>
```

По материалам: https://ru.wikipedia.org/wiki/SOAP.

С другой стороны, если в SOAP используется громоздкий XML, почему бы не изобрести свой формат обмена с лаконичностью и эстетическим удовлетворением? Сказано-сделано. Так появился XML-RPC.

XML-RPC

Удаленный вызов процедур через XML, так расшифровывается название протокола.

Пример запроса.

```
<?xml version="1.0"?>
 <methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
     <param>
         <value><i0>41</i0></value>
     </param>
   </params>
 </methodCall>
```

Пример ответа.

```
<?xml version="1.0"?>
 <methodResponse>
   <params>
     <param>
         <value><string>South Dakota</string></value>
     </param>
   </params>
 </methodResponse>
```

По материалам: https://ru.wikipedia.org/wiki/XML-RPC.

Это уже лучше, но все равно слишком громоздко. Почему бы еще не упростить. Например, использовать вместо xml – json.

JSON-RPC

Удаленный вызов процедур через json, так расшифровывается название протокола.

Пример запроса.

```
{"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
```

Пример ответа.

```
{"jsonrpc": "2.0", "result": -19, "id": 2}
```

По материалам: https://ru.wikipedia.org/wiki/JSON-RPC

Вот это уже гораздо-гораздо лучше. Но все равно не то. Дело в том, что достоинства HTTP не используются. Он по-прежнему работает как транспорт. Но в HTTP есть коды состояний, методы, ресурс, заголовки. Так если мы запрашиваем некий ресурс и в json получаем “не найден”, код возврата HTTP будет все равно 200 Ok.

REST-API

Representational State Transfer (REST) — «передача состояния представления», альтернатива для SOAP и RPC.

В отличие от вышестоящих протоколов это не протокол и не библиотека, это философия, согласно которой сами методы HTTP, коды состояний, запросы будут нести смысловую нагрузку. Соответственно мы будем использовать GET для того, чтобы получить информацию, PUT — отправить, DELETE — удалить, в адресе ресурса мы укажем объект, с которым хотим работать, плюс у нас еще есть заголовки и возможность получать json с данными в ответе.

Хороший пример REST API — API Яндекс.Диска. Документация по API доступна по адресу https://tech.yandex.ru/disk/api/concepts/about-docpage/.

Запросы можно сразу оттестировать на тестовом полигоне https://tech.yandex.ru/disk/poligon/.

Подключаемся по https на cloud-api.yandex.net:443.

```
GET /v1/disk HTTP/1.1
```

Получаем 200 Ok и тело HTTP-ответа.

```
{
  "max_file_size": 10737418240,
  "total_space": 35433480192,
  "trash_size": 0,
  "is_paid": false,
  "used_space": 26186303125,
  "system_folders": {
    "odnoklassniki": "disk:/Социальные сети/Одноклассники",
    "google": "disk:/Социальные сети/Google+",
    "instagram": "disk:/Социальные сети/Instagram",
    "vkontakte": "disk:/Социальные сети/ВКонтакте",
    "mailru": "disk:/Социальные сети/Мой Мир",
    "downloads": "disk:/Загрузки/",
    "applications": "disk:/Приложения",
    "facebook": "disk:/Социальные сети/Facebook",
    "social": "disk:/Социальные сети/",
    "screenshots": "disk:/Скриншоты/",
    "photostream": "disk:/Фотокамера/"
  },
  "revision": 1503705721964162
}
```


Удалить файл test.

```
DELETE /v1/disk/trash/resources?path=test HTTP/1.1
```

Ответ 404 Not Found, тело.

```
/v1/disk/trash/resources?path=tes
```

Помимо REST-API в описании API имеется WebDav API для подключения диска в Linux и Windows как сетевого диска и API для уведомлений вашей ПО на основе XMPP.

WebDAV

WebDav описано по адресу: https://tech.yandex.ru/disk/doc/dg/concepts/about-docpage/.

Можем посмотреть, что в WebDav есть новые методы по сравнению с HTTP.

Пример: приложение создает каталог /b/ внутри каталога /a/, находящегося в корневом каталоге Диска.

```
MKCOL /a/b/ HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 023747ff12123cdhfhy773457
```

Если каталог /a/ существует и каталог /b/ был создан успешно, возвращается следующий ответ.

```
HTTP/1.1 201 Created
Content-Length: 0
```

По материалам https://tech.yandex.ru/disk/doc/dg/reference/move-docpage/

Пример: файл lion.png копируется из папки pictures в папку animals.

```
COPY /pictures/lion.png HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 023747ff12123cdhfhy773457
Destination: /animals/lion.png
Overwrite: F
```

Заголовок Overwrite можно задать, чтобы запретить перезапись уже существующего файла с таким именем. Значение T по умолчанию разрешает перезапись, значение F — запрещает. Если в каталоге /animals/ уже есть файл lion.png, то запрос из примера не будет выполнен.

Если копирование прошло успешно, возвращается следующий ответ.

```
HTTP/1.1 201 Created
Content-Length: 0
```

По материалам https://tech.yandex.ru/disk/doc/dg/reference/move-docpage/

XMPP

XMPP (Extensible Messaging and Presence Protocol — расширяемый протокол обмена сообщениями и информацией о присутствии, ранее известный как Jabber — открытый, основанный на XML, свободный для использования протокол для мгновенного обмена сообщениями и информацией о присутствии в режиме, близком к режиму реального времени. Изначально спроектированный легко

расширяемым, протокол, помимо передачи текстовых сообщений, поддерживает передачу голоса, видео и файлов по сети.

Клиентское приложение может получать от Яндекс.Диска оповещения об изменениях, происходящих с Диском пользователя: загрузка и удаление файлов, создание и удаление каталогов, изменение объема доступного пространства.

Оповещения реализованы с помощью протокола XMPP: приложение связывается с сервером Яндекс.Диска и получает сообщения определенного формата.

Приложение должно подключиться к 5222 порту сервера push.xmpp.yandex.ru.

Если порт 5222 недоступен, можно использовать порт 443: для этого порта возможно только шифрованное соединение, поэтому подключение к серверу должно начинаться с TLS-приветствия (пункт 3).

Последовательность подключения к XMPP-серверу.

Приветствие.

```
Приложение запрашивает создание потока:

<?xml version="1.0"?>

<stream:stream   xmlns:stream="http://etherx.jabber.org/streams"   version="1.0"
xmlns="jabber:client"                  to="ya.ru"                  xml:lang="en"
xmlns:xml="http://www.w3.org/XML/1998/namespace">
```

Сервер возвращает идентификатор потока и сообщает о доступных возможностях (шифрование, сжатие ZLIB и авторизация с помощью SASL).

```
<?xml version='1.0'?>

<stream:stream                                          xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'   id='4235063168'   from='ya.ru'
version='1.0' xml:lang='en'>

<stream:features>

  <starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>

  <compression xmlns="http://jabber.org/features/compress">

    <method>zlib</method>

  </compression>

  <mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
```

```
</stream:features>
```

Установка шифрованного соединения (TLS).

Приложение отправляет строфу запроса шифрования.

```
<starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>
```

Сервер подтверждает установку шифрованного соединения.

```
<proceed xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>
```

TLS-приветствие. Приложение заново запрашивает создание потока (см. пункт 1), используя шифрованное соединение.

При необходимости приложение может запрашивать сжатие потока в формате ZLIB. Сжатие в протоколе XMPP описано как расширение стандартного протокола: XEP-0138.

Авторизация. Приложение передает на сервер токен авторизации в кодировке base64. Токен нужно составлять из логина пользователя и OAuth-токена, разделяя их нулевым байтом, например: test\00c4181a7c2cf4521964a72ff57a34a07.

```
<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl"
mechanism="X-YANDEX-OAUTH">dGVzdABjNDE4MWE3YzJjZjQ1MjE5NjRhNzJmZjU3YTM0YTA3</aut
h>
```

При успешной авторизации сервер отвечает следующим образом.

```
<success xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
```

Авторизованное приветствие. Приложение запрашивает создание потока (см. пункт 1). Сервер перечисляет возможности, доступные после авторизации.

Установка ресурса. Приложение передает название ресурса в следующей строке.

```
<iq type="set" id="bind_1">
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">
    <resource>YaDisk-client</resource>
```

```
  </bind>
</iq>
```

Сервер возвращает присвоенный приложению JID, который следует использовать в последующих запросах.

```
<iq xmlns="jabber:client" type="result" id="bind_1">
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">
    <jid>test@ya.ru/YaDisk-client</jid>
  </bind>
</iq>
```

Открытие сессии. Приложение запрашивает открытие сессии.

```
<iq type="set" id="session_1">
  <session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>
</iq>
```

Сервер отвечает результатом открытия сессии.

```
<iq type="result" id="session_1">
<session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>
</iq>
```

Запрос версии клиента (XEP-0092).

После открытия сессии сервер запрашивает версию клиентского приложения.

```
<iq from="ya.ru" type="get" to="test@ya.ru/YaDisk-client" id="ask_version">
<query xmlns="jabber:iq:version"/>
</iq>
```

Приложение должно вернуть собственное название, номер версии приложения и опционально версию операционной системы.

```
<iq type="result" to="ya.ru" id="ask_version">
  <query xmlns="jabber:iq:version">
    <name>YaDiskClient</name>
    <version>1.1</version>
    <os>Mac OS X</os>
  </query>
</iq>
```

После того, как соединение установлено, сервер периодически проверяет связь с приложением, присылая запросы следующего вида.

```
<iq from="ya.ru" type="get" to="test@ya.ru/YaDisk-client" id="ping_1">
  <ping xmlns="urn:xmpp:ping"/>
</iq>
```

Приложению следует отвечать следующим образом, копируя в ответ значение атрибута id из запроса сервера.

```
<iq type="result" to="ya.ru" id="ping_1"/>
```

После этого можно подписаться на уведомления и получать оповещения об изменениях на Яндекс.Диске https://tech.yandex.ru/disk/doc/dg/concepts/xmpp_xmpp-notification-docpage/.

Установка Apache2 и Nginx в Ubuntu

Дальше мы рассмотрим, как установить и настроить полноценный веб-сервер, работающий не только на локалхосте, но и доступный в Интернет. Для этого понадобится купить доменное имя и арендовать VDS с Ubuntu, адрес которого прописать в DNS-записях DNS-хостинга (от DNS-регистратора или другого).

Установка и настройка Apache2

Для установки apache2 выполним команду.

```
$ sudo apt-get -y install apache2
```

Легко проверить, что Apache2 установился. Узнайте IP-адрес вашей машины и введите его в Вашем браузере. Вы должны увидеть стартовую страницу-заглушку веб-сервера Apache2.

Изучите структуру конфигурации в директории /etc/apache2.

Обратите внимание, что вы не найдёте httpd.conf. Если вы где-то нашли информацию о данном конфигурационном файле, то эта информация уже устарела. Вместо него применяется набор из apache2.conf и конфигурационных файлов, расположенных в соответствующих директориях.

В Debian и Ubuntu используется удобный подход на основе символических ссылок. Директория *-available хранит доступные конфигурационные файлы, а *-enabled — символические ссылки на активные настройки в данный момент.

Конфигурационные файлы. conf-available/conf-enabled

Большинство настроек хранятся в директориях conf-available и conf-enabled. Папка conf-available содержит файлы *.conf — все доступные конфигурационные файлы. Папка conf-enabled содержит символические ссылки на конфиги из conf-available. Именно эти конфигурационные файлы активны, остальные «выключены». Утилиты a2enconf и a2disconf включают/выключают соответствующий конфиг. Фактически они создают/удаляют символическую ссылку через ln -s и предлагают послать серверу apache2 сигнал restart или reload.

Модули Apache2. mods-available и mods-enabled

Директории mods-available и mods-enabled хранят модули. Если модуль не был установлен, его следует установить через apt-get install.

Активация/деактивация модулей выполняется аналогично конфигурационным файлам через утилиты a2enmod / a2dismod.

Виртуальные хосты. sites-available и sites-enabled

Конфигурационные файлы для виртуальных хостов содержатся в директориях sites-available и sites-enabled. Исходно файлы находятся в sites-available и они не активны. При активации сайта создается символическая ссылка в директории sites-enabled на соответствующий конфигурационный файл из директории sites-available. При деактивации символическая ссылка из директории sites-enabled удаляется, а сам конфигурационный файл в директории sites-available остаётся, что очень удобно, так как можно подключать/отключать сайты.

Активация/деактивация хоста(ну вдруг ваш клиент не оплатил хостинг ;) через команды a2ensite и a2dissite.

Обратите внимание на два файла, присутствующих по умолчанию: 000-default.conf и default-ssl.conf.

Это пример сайта (отвечает для всех доменных имен по IP-адресу Вашего сервера), первый для протокола HTTP (порт 80), второй для протокола HTTPS (порт 443).

По умолчанию включен только HTTP.

Создаём простейший конфигурационный файл

Обратите внимание, что при попытке запуска веб-сервер выдает предупреждение.

```
Could not reliably determine the server's fully qualified domain name, using
127.0.1.1 for ServerName
```

Можно сделать соответствующую запись в файле /etc/apache2.conf, но мы пойдем другим путем, создадим fqdn.conf в директории conf-available и активируем конфиг.

```
$ echo "ServerName localhost" | sudo tee /etc/apache2/conf-available/fqdn.conf
$ sudo a2enconf fqdn
```

Работаем с модулями

Пример 1. Модуль активен

Предположим, IP-адрес вашей виртуальной машины 192.168.131.136. Тогда с помощью модуля Apache2 status мы можем посмотреть состояние сервера. Но если вы попытаетесь просмотреть в браузере http://192.168.131.136/server-status, увидете ошибку 403. Доступ запрещен.

Что это означает? Если мы посмотрим в mods-enable увидим, что модуль status там присутствует, стало быть активен. Значит ситуация не в том, что модуль не активен, а в настройках прав доступа. Можем посмотреть данный ресурс локально, например, в текстовом браузере lynx.

```
$ lynx  http://192.168.131.136/server-status
```

Или перейдя через Alt-F7 в X-Windows и запустив этот адрес в FIrefox.

Работает. Но можно настроить так, чтобы из сети вашей хостовой машины тоже можно было посмотреть данный файл. Откройте файл /etc/apache2/mods-available/status.conf и после строки

```
Require local
```

добавьте,

```
Require ip 192.168.131.0/24
```

исправьте на адрес вашей сети.

Необходимо, чтобы сервер перечитал конфигурацию.

```
$ sudo systemctl reload apache2
```

Пример 2. Модуль установлен, но не активен

Следующий пример. Кроме модуля status есть модуль info.

Откройте http://192.168.131.136/server-info.

Независимо от того, с хоста или гостевой машины вы смотрите. Ошибка 404 (not found).

Если мы посмотрим в mods-enabled, мы не найдем файл info, но в mods-available он присутствует. Стало быть он установлен, но не активен.

Поэтому выполняем

```
$ sudo a2enmod info
$ sudo systemctl restart apache2
```

и проверяем.

Также можете поправить IP-адреса, чтобы Info можно было видеть с машины-хоста (аналогично выше проделанным операциям).

Пример 3. Модуль не установлен

Например, php7. Для этого установим его и некоторые дополнения (для работы с mysql, curl — http-клиент для работы с удаленными файлами, json — часто-применяемый транспорт между бэкендом и фронтендом).

Т.к. в дальнейшем нам понадобится PHP самой свежей версии, добавим репозиторий:

```
sudo add-apt-repository ppa:ondrej/php
```

Жмём Enter, затем обновляем локальную базу данных доступных пакетов:

```
sudo apt-get update
```

Теперь устанавливаем сам PHP и дополнения:

```
$sudo apt-get -y install php7.4 libapache2-mod-php7.4 php7.4-mysql php7.4-curl
php7.4-json
```

Перезапускаем веб-сервер.

```
$ sudo systemctl reload apache2
```

Проверяем.

```
$ php -v
```

Создадим phpinfo.

```
# cat >/var/www/html/info.php
<?php
phpinfo();
?>
Ctrl-D
```

Проверяем http://192.168.131.136/info.php.

Чтобы каждый раз не писать IP-адрес, можно в файле (если ваша машина хост — Windows) c:\windows\system32\drivers\etc\hosts прописать с помощью редактор FAR или Notepad++.

```
192.168.131.136 test.tst
```

Защита директорий через HTTP-аутентификацию

Аутентификация — процедура проверки пользователя.

Авторизация — предоставление пользователю прав на те или иные действия.

Это разные понятия, но часто процедура аутентификации и авторизации выполняется совместно.

Базовая аутентификация

Рассмотрим простейший способ защитить директорию с помощью HTTP-аутентификации.

Для начала потребуется установить apache2-utils.

```
$ sudo apt-get apache2-utils
```

Создаём файл паролей /etc/apache2/ht_passwd с пользователем admin.

```
# htpasswd -c /etc/apache2/ht_passwd admin
```

Вводим пароль для админа. Добавим пользователя user.

```
# htpasswd /etc/apache2/ht_passwd user
```

Введем пароль для пользователя user. Создадим директорию.

```
#mkdir /var/www/private
```

Создадим два файла.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <title>Super Secret Zone</title>
</head>
<body>
      <h1>Super Secret Zone</h1>
      <p>Only for valid users</p>
```

```
</body>
</html>
```

.htaccess

```
AuthType Basic
AuthName "Valid user required:"
AuthUserFile "/etc/apache2/ht_passwd"
Require valid-user
```

Обратите внимание, при ошибке в данном файле при попытке обратиться к защищенной директории будет возникать 500 ошибка — ошибка сервера.

Также стоит подробнее самостоятельно изучить возможности .htaccess, в частности с помощью регулярных выражений можно перенаправлять разные запросы на один index.php (так называемый роутинг в PHP-фреймворках).

Проверяем, http://test.tst/private не работает. Что не хватает? В apache2.conf находим строку.

```
<Directory /var/www>
 Options Indexes FollowSymLinks
 AllowOverride None
 Require all granted
</Directory>
```

Меняем на.

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

Просим сервер перечитать конфигурацию.

```
# systemctl reload apache2
```

Проверяем, http://test.tst/private теперь работает.

Дайджест-аутентификация

Для digest аутентификации используется утилита htdigest вместо htpasswd.

Создаём файл и пользователя.

```
# htdigest c /etc/httpd/digest_passwd "Restricted Directory" test
```

.htaccess

```
AuthType Digest
AuthName "Valid user required:"
AuthUserFile "/etc/apache2/digest_passwd"
Require valid-user
```

Многосайтовость

Если мы в файле (если ваша машина хост — Windows) c:\windows\system32\drivers\etc\hosts, пропишем.

```
192.168.131.136 test.tst
192.168.131.136 test.a
192.168.131.136 test.b
```

Всё равно каждый раз будет отдаваться одна и та же страница-заглушка.

Как поддерживать несколько сайтов?

- Использовать разделение по портам. test.tst ведёт на один сайт. test.tst:8080 на другой. Так иногда осуществляют доступ, например, к админке, но чаще всего так поступают, если на сервере работает несколько приложений-серверов HTTP. Например, на test.tst на 80 порту (по умолчанию работает) apache2, а на test.tst:8080 — сервер онлайн-радиостанции icecast2. В последнее время так поступать также не принято, так как с помощью nginx можно осуществлять проброс трафика, например, test.tst/audio/ будет передавать трафик icescat2, слушающему трафик на 127.0.0.1:8080, а остальные пути на 127.0.0.1:80, который слушает apache2.
- Использовать несколько IP-адресов. В этом случае многосайтовость будет осуществляться благодаря механизму DNS, у одного доменного имени — один IP-адрес, у другого — другой. Веб-сервер будет определять сайт именно по IP-адресу.
- Использовать механизм виртуальных-хостов. В этом случае в протоколе http передается заголовок host: который указывает доменное имя. В этом случае сайты будут различаться по хосту.

Многосайтовость по порту

Скопируйте 0000-default.conf в 8080.conf. Отредактируйте. Исправьте, чтобы получилось

```
<VirtualHost *:8080>
```

и

```
DocumentRoot /var/www/8080
```

Создайте соответствующую директорию и положите туда файл, указав, например, в title и h1, что это именно сайт с порта 8080.

В ports.conf добавьте.

```
Listen 8080
```

Активируйте сайт и перезагрузите сервер.

```
# a2ensite 8080
# systemctl reload apache2
```

Проверьте. Деактивируйте сайт.

Многосайтовость по IP-адресу

Поднимите новый IP-адрес на вашей машине.

```
# ifconfig ens33:second 172.16.0.1 netmask 255.255.0.0 up
```

В Windows (Win+X>Командная строка>административная) пропишите маршрут на этот адрес через IP-адрес вашей виртуальной машины.

```
c:\>route add 172.16.0.1 mask 255.255.0.0 192.168.131.128
```

192.168.131.128 — адрес гостевой Ubuntu.

Добавьте адрес 172.16.0.1 в hosts в Windows (точнее, исправьте).

```
172.16.0.1 test.b
```

Скопируйте 000-default.conf в test.b Отредактируйте test.b.

```
<VirtualHost 172.16.0.1:80>
```

Не забудьте создать директорию, указать её в test.b и создать индекс. Активируйте сайт, попросите веб-сервер перечитать конфигурацию, проверьте. Сравните в браузере сайты test.a и test.b.

Теперь деактивируйте сайт, удалите конфигурационный файл test.b и отключите сетевой интерфейс ens33:second.

Не забудьте удалить маршрут в Windows.

```
c:\>route delete 172.16.0.1 mask 255.255.0.0 192.168.131.128
```

Многосайтовость по домену, копируйте 000-default.conf в test.a.

Раскомментируйте в файле ServerName и впишите туда test.a.

```
ServerName test.a
```

Не забудьте создать директорию, указать её в test.a и создать индекс. Активируйте сайт, нужно попросить веб-сервер перечитать конфигурацию, проверьте.

Сравните. Удалите сделанные изменения или настройте свои сайты.

Безопасность HTTP

HTTP- небезопасный протокол. Это означает, что пароли, отправленные через POST или сообщенные вам, могут быть доступны абсолютно любому лицу, которое имеет доступ к одной из промежуточных машин. Провайдеру, соседям по хостингу и т.п. и т.д.

Поэтому в настоящее время всё чаще используется HTTPS. HTTP оправдан только в том случае, если сайт только отдаёт незашифрованное содержимое (например, простейшая домашняя страничка или информационный ресурс). Если на сайте присутствует форма авторизации, используйте HTTPS. Более того, в настоящее время наличие HTTPS является основным требованием Google и Яндекс для ранжирования, так что HTTPS оправдан даже для сайтов, не получающих никаких данных от пользователя.

Технически HTTPS - это протокол HTTP, использующий сессию TLS, который в свою очередь использует сессию TCP. (HTTP непосредственно использует сессию TCP). Для работы по HTTPS принято использовать порт 443.

TLS

TLS — новый протокол безопасности, основанный на SSL (тем не менее TLSv1 тоже успел устареть, это надо учитывать). Его идея состоит в использовании алгоритма RSA, сложность взлома которого определяется алгебраической сложностью задачи нахождения простых сомножителей натурального числа.

RSA — алгоритм асимметричного шифрования. Вместо общего симметричного ключа используется пара асимметричных ключей, публичный и приватный. То, что зашифровано одним, может быть

расшифровано парным ключем. При этом публичный ключ распространяется корреспондентам, а приватный остается только у выпустившего ключ. Публикация приватного ключа — ошибка, требующая перевыпуск ключа и сертификатов.

Ассиметричные алгоритмы могут решить только одну задачу одновременно из двух.

- Шифрование трафика (если шифруется публичным ключом, расшифруется приватным. Зашифровать может любой, а прочитать только владелец публичного ключа. Таким образом трафик не доступен для прослушивания. Но подтвердить подлинность отправителя невозможно).
- Аутентификация – проверка подлинности абонента (если шифрует приватным ключом отправитель, а получатели расшифровывают публичным. Если они смогли расшифровать, значит, отправить мог только отправитель, чей публичный ключ у них есть. Но расшифровать такой трафик может любой).

Так как на практике требуется решение обеих задач, применяется третья сторона — удостоверяющий центр. Публичные ключи удостоверяющих центров импортируются в браузеры производителем браузера, таким образом удостоверяющая способность таких центров (CA) не вызывает сомнений.

Владелец же сайта выпускает приватный и публичный ключи и подписывает публичный ключ у удостоверяющего центра. Для этого центру сертификации передаётся запрос на подпись по стандарту X.509, а в ответ получается сертификат. Таким образом браузер может проверить, является ли сайт тем, кто себя выдает.

Сертификат — публичный ключ и дополнительная информация, хэш от которой зашифрован приватным ключом удостоверяющего центра. Подпись ключа — услуга, предоставляемая удостоверяющими центрами. Можно подписать сайт собственно-сгенерированным удостоверяющим центром, но браузеры будут выводить сообщение о том, что валидность сайта проверить невозможно. Но для тестирования вариант самоподписанного сертификата подойдет.

На практике часто, говоря SSL, подразумевают TLS. Тем не менее SSL v3 и TLS v1 уже признаны небезопасными, это надо иметь в виду при настройке сервисов, использующих шифрование.

Проверка mod_ssl и доступ по HTTPS

Активирует default-ssl сайт.

```
# sudo a2ensite default-ssl
```

Перечитайте конфигурацию сервера и проверьте. Ничего не вышло. Не активирован модуль ssl.

Активируйте.

```
# sudo a2enmod ssl
```

Перезагрузите сервер. Проверьте test.tst. Теперь все работает. Если хотите посмотреть, какой сертификат выдан, увидите.

- Имя сертификата Ubuntu.
- Выдан Ubuntu.

Можно попробовать подготовить свой сертификат для проверки работы.

Если установлен openssl, сгенерировать самоподписанный ключ можно с помощью скрипта /usr/lib/ssl/apache2mod_ssl/gentestcrt.sh

Мы рассмотрим вариант генерации самоподписанного сертификата по шагам.

Настройка сайта с самоподписанным сертификатом

Для начала мы разберём способ создания самоподписанных сертификатов вручную. На практике используется подпись сертификатов настоящим удостоверяющим центром, генерация самоподписанных сертификатов может быть полезна для понимания механизма работа, а также для задач тестирования. Создание сертификатов, подписанных настоящим удостоверяющим центром с помощью Let's Encrypt, мы разберем ниже. А пока создаём самоподписанный сертификат.

Сначала создадим собственный центр сертификации (CA).

Для этого создадим частный ключ для CA.

```
# cd /etc/apache2/

# mkdir ssl

# chmod 700 ssl

# cd ssl

#openssl genrsa -des3 -out my-ca.key
```

Секретная фраза — пароль к нашему центру сертификации.

Создаём сертификат.

```
# openssl req -new -x509 -days 3650 -sha256 -key my-ca.key -out my-ca.crt
```

Придется ввести уже запомненный пароль и ответить на вопросы. Создаем ключ для apache2.

```
# openssl genrsa -des3 -out my-apache.key 1024
```

Потребуется ввести секретную фразу.

Ее мы удалим из сертификата сервера апач, чтобы она не запрашивалась при каждом старте сервера.

```
# openssl rsa -in my-apache.key -out new.my-apache.key

# mv new.my-apache.key my-apache.key
```

Создаём запрос на подпись сертификата Certificate Signature Request (csr), чтобы подписать в нашем центре сертификации.

```
# openssl req -new -key my-apache.key -out my-apache.csr
```

Важно при запросе Common name FQDN поместить доменное имя, например test.tst.

И подпишем наш сертификат Apache в нашем центре сертификации.

```
# openssl x509 -req -in my-apache.csr -out my-apache.crt -sha256 -CA my-ca.crt
-CAkey my-ca.key -CAcreateserial -days 36

# chmod 0400 *.key
```

Далее редактируем default-ssl.conf.

```
SSLCertificateFile /etc/apache2/ssl/my-apache.crt

SSLCertificateKeyFile /etc/apache2/ssl/my-apache.key

SSLCACertificateFile /etc/apache2/ssl/my-ca.crt
```

Перезагружаем сервер. Проверяем https://test.tst.

Смотрим информацию о сертификате, видим, что отображается созданный и подписанный нами.

Создание сертификата с помощью Let’s Encrypt

Будем устанавливать из официальных репозиториев.

```
$ sudo apt update
$ sudo apt install python-letsencrypt-apache
```

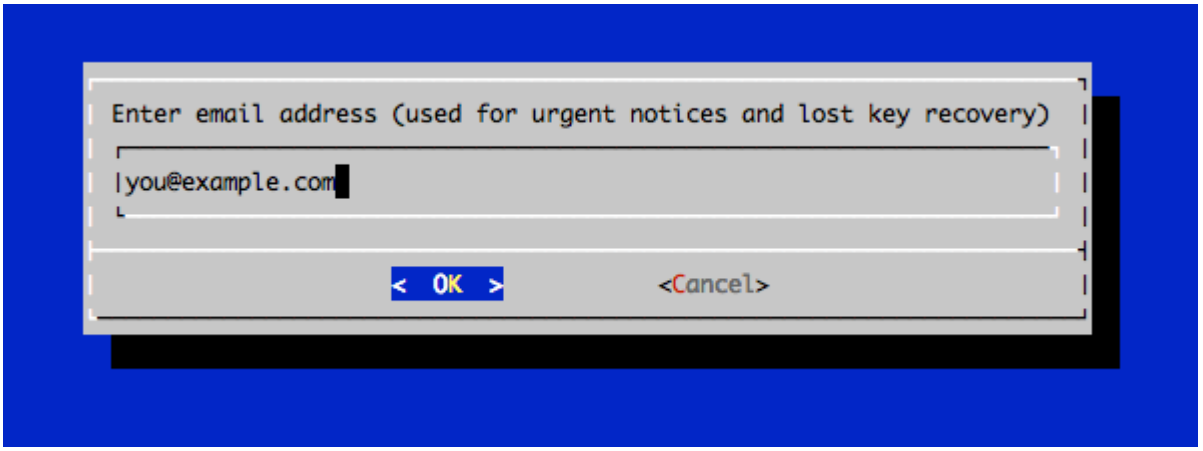
Получаем сертификат.

```
$ sudo letsencrypt --apache -d example.com
```

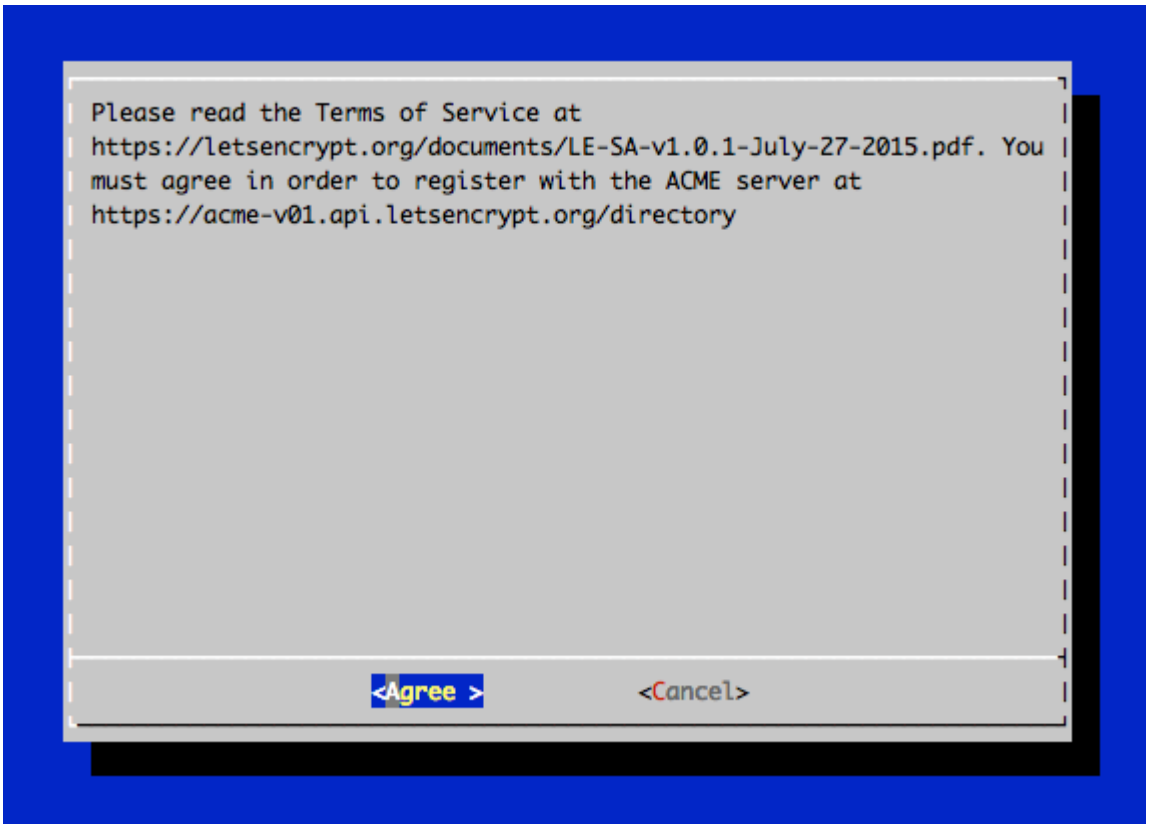
Если нужно сделать сертификат для нескольких доменов/поддоменов, их можно передать в дополнительных параметрах, но основным будет первое доменное имя, поэтому первым нужно указать основной домен, а потом поддомены.

```
$ sudo letsencrypt --apache -d example.com -d www.example.com
```

Потребуется указать адрес электронной почты.



Принять лицензионное соглашение.



Также нужно выбрать, нужно ли разрешать http-трафик или сразу направлять его на https (что правильнее). Будет произведена настройка Apache2 в соответствии с выбранным ответом.

Когда установка SSL сертификата Apache ubuntu будет завершена, вы найдёте созданные файлы сертификатов в папке /etc/letsencrypt/live. В этой папке будут четыре файла.

- cert.pem - ваш сертификат домена
- chain.pem - сертификат цепочки Let's Encrypt
- fullchain.pem - cert.pem и chain.pem вместе.
- privkey.pem - секретный ключ вашего сертификата.

Теперь вы можете зайти на сайт по https. Чтобы проверить, как работает SSL и правильно ли выполнена установка ssl сертификата на сайт, вы можете открыть в браузере такую ссылку.

```
https://www.ssllabs.com/ssltest/analyze.html?d=example.com&latest
```

SSL настроен. Стоит иметь в виду, что сертификаты, полученные от Let’s Encrypt, действительны в течение 90 дней, поэтому рекомендуется продлевать их каждые 60 дней.

Клиент letsecnrypt имеет команду renew, которая позволяет проверить установленные сертификаты и обновить их если до истечения срока осталось меньше 30 дней.

Чтобы запустить процесс обновления для всех настроенных доменов, выполните.

```
$ sudo letsencrypt renew
```

Если сертификат был выдан недавно, то команда проверит его дату истечения и выдаст сообщение, что продление пока не требуется. Если вы создали сертификат для нескольких доменов, то в выводе будет показан только основной домен. Но обновление будет актуально для всех.

Самый простой способ автоматизировать этот процесс – добавить вызов утилиты в планировщик cron. Для этого выполним команду.

```
$ crontab -e
```

Добавим строку.

```
30 2 * * 1 /usr/bin/letsencrypt renew >> /var/log/le-renew.log
```

Команда обновления будет выполняться каждый понедельник, в 2:30 утра. Информация про результат выполнения будет сохранятся в файл /var/log/le-renewal.log.

Nginx

Для работы nginx нам понадобится свободный 80 порт, поэтому убедитесь что он не занят сервером Apache. Если же он им используется, смените порт на 8080. Не забудьте перезапустить Apache. Также можете удалить или переименовать стандартную страничку от Apache /var/www/html/index.html .

Теперь можно приступать к установке и дальнейшей настройке nginx. Для начала добавим репозиторий от производителя, чтобы получать самую свежую версию nginx:

```
add-apt-repository ppa:nginx/stable
```

Жмём Enter, затем обновляем локальную базу данных доступных пакетов и устанавливаем nginx:

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

Но этого, конечно, мало. Заглянем в директорию /etc/nginx.

Здесь есть файл nginx.conf (в котором есть подключение всех файлов в conf.d), здесь могут быть подключения (а могут и не быть, но можно сделать на sites-enabled).

В директории conf.d находится файл default.conf, который и обеспечивает нам работу нужной страницы.

Если мы хотим сделать новый файл, легко его скопировать и настроить.

Какие-то вещи здесь похожи на apache2, какие-то сильно отличаются.

Для начала давайте разберем пример default.conf по умолчанию.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    # SSL configuration
    #
    # listen 443 ssl default_server;
    # listen [::]:443 ssl default_server;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name _;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;

    }

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #       include snippets/fastcgi-php.conf;
    #
    #       # With php7.0-cgi alone:
    #       fastcgi_pass 127.0.0.1:9000;
    #       # With php7.0-fpm:
    #       fastcgi_pass unix:/run/php/php7.0-fpm.sock;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #       deny all;
    #}
}
```

Бросаются в глаза listen и servername.

Есть и root, но добавляется незнакомая сущность location. После работы с .htaccess довольно сложно привыкнуть к работе с nginx, ведь в Nginx нет отдельных правил для директорий в том виде, в каком

они были в апач. Апач смотрел текущую директорию на предмет наличия этого скрытого файла, затем родительскую и так далее. Поэтому все фокусы с rewrite здесь выглядят по-другому.

Как вы заметили, location может быть несколько. В location допустимы регулярные выражения, если они относятся к данному серверу (но если мы перенаправляем трафик на другой сервер apache2, придется обойтись без regexp). Разница в том, что apache2 смотрит в директории, Nginx анализирует запросы. location это набор конфигураций, который подпадает под тот или иной запрос.

Пока мы видим немного: все запросы, которые идут от / рассматривать как запрос статических файлов из директории /var/www/html.

Также закомментированы несколько примеров.

Это отправка запросов, содержащих .php веб-серверу апач (слушающему 80 порт, но по адресу 127.0.0.1), либо через Fast-CGI на php, слушающем 127.0.0.1 по TCP-порту 9000.

В любом случае .htaccess файлы не должны выдаваться запрашивающим (кстати, апач или php через cgi могут обрабатывать ту же самую директорию), поэтому также нужно раскомментировать запрет на .ht-файлы.

Как мы можем улучшить конфигурационные файлы для сайта?

Создать директории для всех и для активных сайтов, если они ещё не созданы:

```
mkdir /etc/nginx/sites-{available,enabled}
```

Проверить в файле /etc/nginx/nginx.conf. наличие строчки include /etc/nginx/conf.d/*.conf, если её нету, то добавить:

```
include /etc/nginx/sites-enabled/*.conf;
```

Можно перезапустить сервер и убедиться, что все работает.

Дальше создадим директории для доступных и активных сайтов, переместим наш default.conf в доступные сайты и скопируем в servername.conf, а также создадим на него символическую ссылку.

```
cd /etc/nginx/sites-available/
cp default servername.conf
cd ../sites-enabled/
ln -s ../sites-available/servername.conf .
```

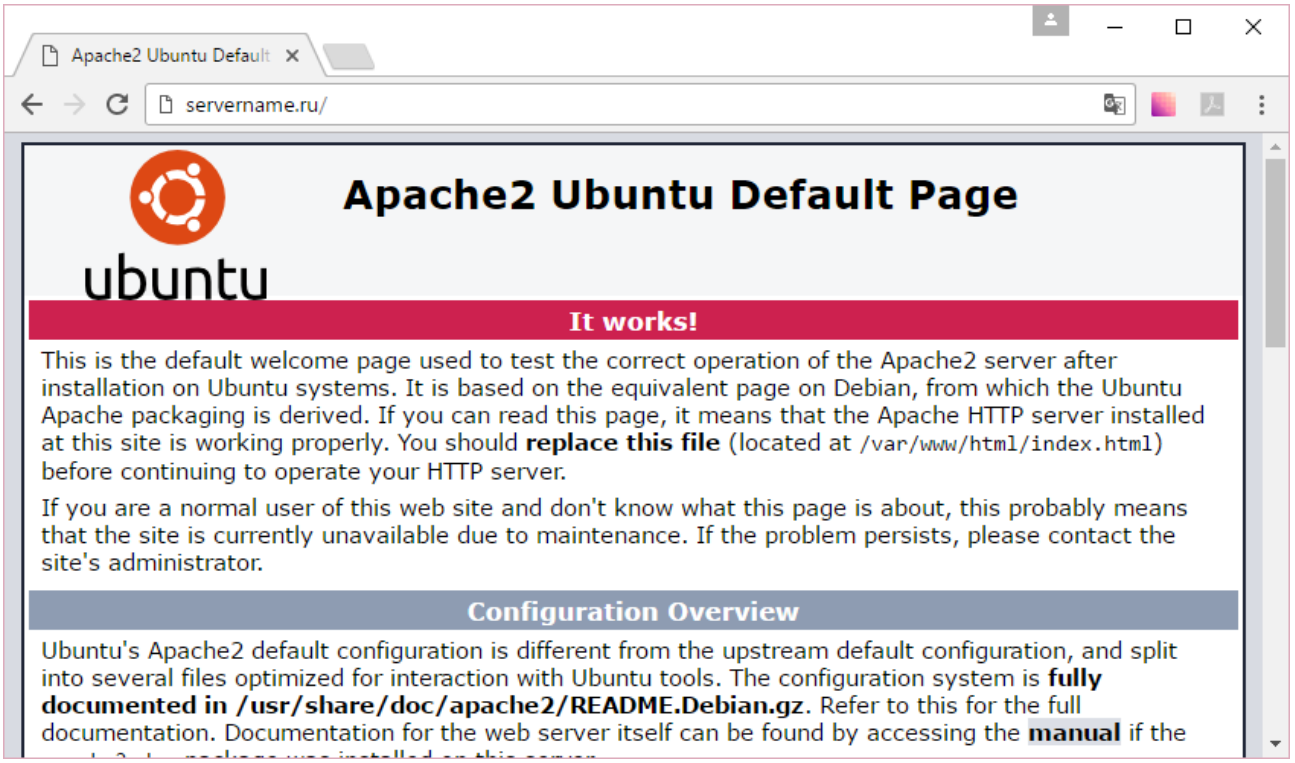
Поправим и наш servername.conf. Обратим все в http и добавим два описания для server, одно с root, другое с редиректом. Вот что у нас получилось.

```
server {
    listen 80;
```

```
    listen [::]:80;

    server_name www.servername.ru;
    return 301 $scheme://servername.ru$request_uri;
}

server {
    listen 80;
    listen [::]:80;

    root   /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name  servername.ru;

...
}
```

Давайте теперь заставим наш сервер все запросы передавать на другой сервер.

```
server {
    listen 80;
    listen [::]:80;

    server_name www.servername.ru;
    return 301 $scheme://servername.ru$request_uri;
}

server {
    listen 80;
    listen [::]:80;

    root   /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name  servername.ru;

    location / {
        proxy_pass http://192.168.131.27;
    }
}
```



Можно сделать несколько location, скажем, один обрабатывает /, другой /wiki, третий /shop — и направить все на разные сервера.

Но еще интереснее балансировать нагрузку. В Nginx есть реализация Round Robin, когда запросы по очереди будут раскидываться по указанным серверам. Для этого в Nginx имеется модуль upstream.

```
index  index.html index.htm;
upstream backend {
    server 1.2.3.4       weight=5;
    server 5.6.7.8:8080;

    server 192.168.131.60   backup;
    server server3.example.ru   backup;
}
server {
    server_name www.servername.ru;
    return 301 $scheme://servername.ru$request_uri;
}
server {
    root   /var/www/html;
    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;
    server_name  servername.ru;

    location / {
        proxy_pass http://backend;
    }
}
```

Обратите внимание, что если первые два сервера не ответят, а третий доступен, nginx вернет данные от apache2, хотя и с большой задержкой.

Устанавливаем php-fpm.

```
apt install php7.4-fpm
systemctl status php7.4-fpm
```

Создадим /var/www/html/phpinfo.php.

```
<?php
phpinfo();
?>
```

Настраиваем наш сайт.

```
server {
    listen 80;
    listen [::]:80;

    server_name www.servername.ru;
    return 301 $scheme://servername.ru$request_uri;
}
server {
    listen 80;
    listen [::]:80;

    root   /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.php index.html index.htm index.nginx-debian.html;

    server_name  servername.ru;

    location / {
       try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php7.4-fpm.sock;
    }

    location ~ /\.ht {
        deny  all;
    }
}
```

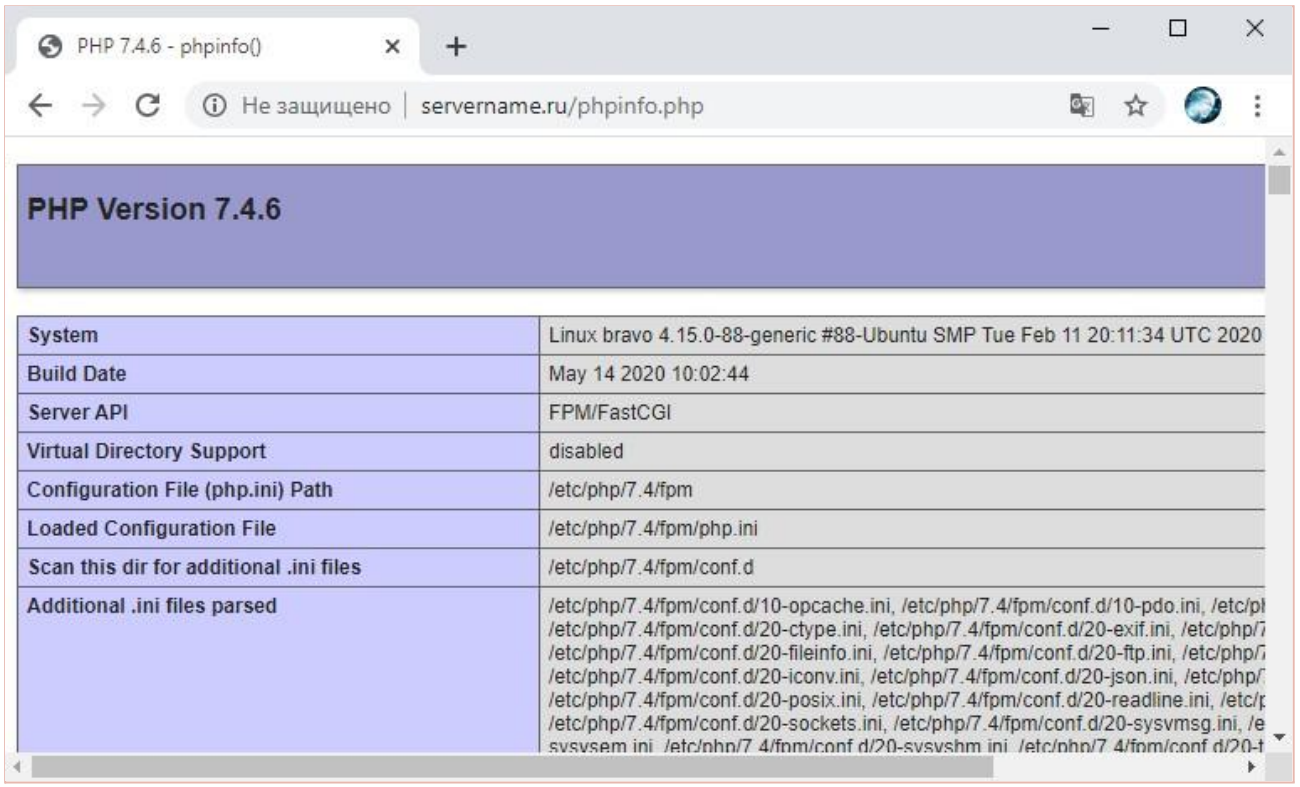
Проверяем. Если ошибка, у нас есть логи /var/log/php7.4-fpm.log, /var/log/nginx/error.log.

Если ошибка возникнет, то скорее всего из-за пользователя и группы, которым принадлежит файл fastcgi_pass unix:/run/php/php7.4-fpm.sock; настройте так, чтобы и nginx, и php7.4-fpm могли к нему обратиться. Сделать это можно, добавив пользователя Nginx, от имени которого работает веб-сервер в группу www-data, от имени которой работает php-fpm.

```
# usermod -a -G www-data nginx
```

Более того, все файлы в директории веб-сервера тоже лучше всего сделать принадлежащими пользователю www-data.

Если все сделали правильно.



MySQL

Чтобы реализовать полноценный LAMP (Linux, Apache2 и PHP), либо LEMP или LNMP (Linux+Nginx/Enginx*+MySQL+PHP) либо даже LNAMP =(Linux+Nginx+Apache+MySQL+PHP), нам также понадобится MySQL-сервер.

Устанавливаем.

```
sudo apt-get install mysql-server mysql-client mysql-common php7.4-mysql
```

При установке будет предложено создать пароль для рута (не линуксового, а майскьюэлового – это разные вещи).

Правим /etc/mysql/my.cnf.

```
bind-address=другой машины либо 0.0.0.0
```

Например, bind-address=192.168.131.1, если вы хотите с хостовой машины под виндовс обращаться к MySQL, либо оставьте.

```
bind-address=127.0.0.1
```

(Рекомендуется. Как туннелировать трафик на 127.1:3306 с помощью ssh мы уже знаем.)

```
[mysqld]

skip-character-set-client-handshake

character-set-server=utf8

init-connect='SET NAMES utf8'

collation-server=utf8_general_ci

...

[client]

default-character-set=utf8

..

[mysqldump]

default-character-set=utf8

..
```

Очистить mysql от мусора. Запустить.

```
mysql_secure_installation
```

Зайти рутом.

```
mysql -u root -p
```

Создаем базу mydatabase.

```
CREATE DATABASE mydatabase;
```

Создаём пользователя user с паролем 1234 и даем ему все права на эту базу.

```
GRANT ALL ON mydatabase.* TO user@localhost IDENTIFIED BY '1234';

Exit
```

Заходим пользователем user и вводим пароль.

```
mysql -u user -p

1234
```

Указываем какую базу будем использовать.

```
USE mydatabase;

Создаем таблицу

create table mytable (

id int not null auto_increment,

txt varchar(100),

n int,

primary key (id));
```

Добавляем значения.

```
insert into mytable (txt,n) values('Wall Street',7);

insert into mytable (txt,n) values('5th Avenu',1);

...

select * from mytable;
```

Получаем на выход содержимое таблицы. Если хочется работать в веб-интерфейсе, можно установить phpMyAdmin. Можно подготовить команды в отдельный файл, например, cmds.sql содержимого.

```
USE mydatabase;

insert into mytable (txt,n) values('Gorki park',7);

insert into mytable (txt,n) values('Kremlin',NULL);
```

Выполняем данные из файла.

```
mysql -u user -p mydatabase <cmds.sql
```

Можно зайти юзером и сделав USE и SELECT полюбоваться результатом.

Делаем бэкап.

```
mysqldump -u user -p mydatabase >backup.sql
```

С помощью скриптов и крона можно сделать это регулярным. Восстанавливаем (и используем один раз mysqladmin а не mysql).

Создаем рутом базу newbase.

```
mysqladmin -u root -p create newbase
```

Импортируем в newbase бэкап.

```
mysql -u root -p --default-character-set=utf8 newbase<backup.sql
mysql -u root -p
USE newbase;
select * from mytable;
exit
```

Кстати, сравниваем таблицу, понимаем, откуда какие строки.

Практическое задание

1. Скачать OpenServer (технически Denwer тоже подойдет, но он очень сильно устарел) или XAMPP/MAMP, либо используете виртуалку, либо VDS с Linux и веб-сервером.

Попробовать приложенные примеры на PHP.

Отследить их с помощью CharlesProxy (скачать, если работаете с OpenServer или denwer, он этот трафик тоже ловит).

Проанализировать запросы-ответы, заголовки, что происходит.

Если работаете с виртуалкой, то можно либо через CharlesProxy, но технически можно и с помощью Wireshark.

2. Попробовать с помощью putty, либо telnet, либо openssl s_client -connect подключиться к веб-серверу и самостоятельно сформировать заголовки и разобрать заголовки ответа. Какой заголовок какую информацию сообщает (клиент-серверу, а сервер-клиенту).
3. Подключиться с помощью браузера к какому-нибудь ресурсу, поддерживающему http и в Wireshark или CharlesProxy, проанализировать заголовки. Сколько TCP-сессий формирует браузер. Почему и для чего?
4. *При знании Linux. Настроить VDS, поставить веб-сервер Nginx, загрузить на сервер веб-проект.

Дополнительные материалы

1. https://www.nginx.com/resources/wiki/start/topics/tutorials/install/
2. https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/
3. http://nginx.org/ru/docs/beginners_guide.html
4. Таненбаум Э., Уэзеролл Д. Т18 Компьютерные сети. 5-е изд. — СПб.: Питер, 2012. — 960 с. (Глава 7)
5. https://tools.ietf.org/html/rfc2616
6. https://tech.yandex.ru/disk/
7. https://habrahabr.ru/company/yandex/blog/241223/

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы.

1. https://ru.wikipedia.org/wiki/DNSBL
2. https://ru.wikipedia.org/wiki/DomainKeys_Identified_Mail
3. Установка и настройк PHP7 http://help.ubuntu.ru/wiki/php
4. https://rtfm.co.ua/apache-mpm-worker-prefork-ili-event/
5. https://losst.ru/ustanovka-ssl-sertifikata-apache-ot-lets-encrypt
6. Установка и настройк PHP7 http://help.ubuntu.ru/wiki/php
7. https://rtfm.co.ua/apache-mpm-worker-prefork-ili-event/
8. https://losst.ru/ustanovka-ssl-sertifikata-apache-ot-lets-encrypt
9. https://interface31.ru/tech_it/2010/10/pochtovyj-server-dlya-nachinayushhix-nastraivaem-dns-zonu.html
10. https://losst.ru/ustanovka-postfix-ubuntu-s-dovecot
11. https://www.8host.com/blog/ustanovka-i-nastrojka-postfix-v-ubuntu-16-04/
12. http://forum.ubuntu-fr.org/viewtopic.php?id=1992106
13. http://stackoverflow.com/questions/168736/how-do-you-set-a-default-value-for-a-mysql-datetime-column
14. http://stackoverflow.com/questions/35565128/mysql-incorrect-datetime-value-0000-00-00-000000
15. https://www.digitalocean.com/community/tutorials/how-to-configure-a-mail-server-using-postfix-dovecot-mysql-and-spamassassin
16. https://bozza.ru/art-169.html
17. https://mnorin.com/dovecot-sieve-sortirovka-pochty-na-servere.html
18. http://itzx.ru/linux/derektiva-nginx-location-s-primerami
19. https://ru.wikipedia.org/wiki/SOAP
20. https://ru.wikipedia.org/wiki/XML-RPC
21. https://ru.wikipedia.org/wiki/JSON-RPC
22. https://tech.yandex.ru/disk/
23. https://ru.wikipedia.org/wiki/XMPP