

# Создание окружения на Linux Debian



# Linux. Debian.

Linux — это ядро Unix-подобной операционной системы. Исходно оно разрабатывалось для машин с процессорами 386 (и более новых), а сейчас может работать и на десятке других архитектур. Ядро Linux было написано Линусом Торвальдсом (Linus Torvalds) и многими компьютерщиками со всего мира.

Кроме ядра, в «Linux»-систему обычно входит:

- файловая система, соответствующая стандарту иерархии файловой системы Linux (Linux Filesystem Hierarchy Standard),
- разнообразные Unix-утилиты, многие из которых были разработаны проектом GNU и Free Software Foundation.

Debian GNU/Linux — это один из дистрибутивов операционной системы Linux с большим количеством пакетов.

В настоящее время существует три версии Debian GNU/Linux:

- *Выпуск 11*, также известен как «*стабильный (stable)*» дистрибутив или *bullseye*. Это стабильное и хорошо протестированное ПО, которое изменяется, если в него включаются существенные исправления, касающиеся безопасности или удобства использования.
- «*Тестируемый*» (*testing*) дистрибутив, в настоящий момент называется *bookworm*. Он включает пакеты, которые войдут в следующую «стабильную» версию; эти пакеты прошли некоторое тестирование в нестабильной версии, но для выпуска пока не готовы. Данная версия обновляется намного чаще, чем «стабильная», но не так часто, как «нестабильная».
- *Нестабильный дистрибутив*. Это версия, находящаяся в разработке и обновляемая в течение всего времени разработки. Вы можете установить эту систему и ее пакеты на свой страх и риск: не ожидайте, что она будет работать стабильно.

Чтобы узнать информацию о текущем дистрибутиве, можно воспользоваться командой `lsb_release -a`.

Результат должен быть подобным следующему:

```
No LSB modules are available.
```

```
Distributor ID: Debian
```

```
Description:    Debian GNU/Linux 11 (bullseye)
```

```
Release:        11
```

```
Codename:       bullseye
```

# Суперпользователь root

Права обычного пользователя в Linux крайне ограничены. Он может управлять только своим каталогом и открывать для чтения определенные файлы из корня. Доступ для их изменения или установки программ отсутствует, что делает привилегии суперпользователя крайне важными при настройке ОС и решении разных проблем. Обычный пользователь ограничивается следующим набором прав:

- чтение, запись и изменение любых атрибутов пользовательского каталога;
- то же самое и для каталога /tmp;
- выполнение программ в любом месте, где нет ограничений;
- чтение файлов с соответствующим атрибутом для всех пользователей.

При наличии рут-прав у пользователя появляется гораздо больше возможностей и расширяются границы взаимодействия с операционной системой. Становятся доступными любые действия со всеми папками и файлами.

**Важно: выполнение команд от root может привести к печальным последствиям!**



# Команда `sudo` и примеры её использования

Команда `sudo` тесно связана с `root` в Linux, поскольку отвечает за передачу прав суперпользователя и позволяет от его имени выполнять команды в терминале. Существует несколько её вариаций, использующихся при разных обстоятельствах. Подходит эта команда как для выполнения всего одного действия, так и для передачи прав на всю текущую сессию.

Команда начинается со служебного слова `sudo` и далее следует команда и её аргументы (опционально).

## Пример:

**Команда** `cp /home/myuser/nginx/nginx.conf/etc/nginx/nginx.conf` вызовет ошибку, если она запущена от обычного пользователя. Но если добавить в начале `sudo`, то команда будет выполнена от имени суперпользователя.

**Важно:** у текущего пользователя должны быть права на выполнение команды `sudo`.

**Важно:** при выполнении команды `sudo` будет запрошен пароль текущего пользователя.



# Установка программ

Установка программ осуществляется 4 способами:

- через магазин приложений;
- скачав файл .deb с сайта разработчиков;
- с помощью командной строки и репозиториев с программами;
- с помощью компиляции исходного кода.

# Через магазин приложений

Если у вас установлена графическая оболочка (например, GNOME), то установить программу можно через магазин приложений:

- GNOME Software используется в дистрибутивах с графической оболочкой GNOME.
- Discover установлен в дистрибутивах KDE. Как и подобает приложению KDE, очень красив и при этом довольно удобен.
- «Менеджер программ» — в Linux Mint.
- AppCenter — в elementary OS.
- Deepin Software Center — в Deepin Linux.

# Скачав файл .deb с сайта разработчиков

В этом случае необходимо выполнить команду `dpkg -i <path_to_file>.deb`



# С помощью командной строки и репозиториев с программами

`apt update` - обновляет список доступных пакетов

`apt install <package_name>`



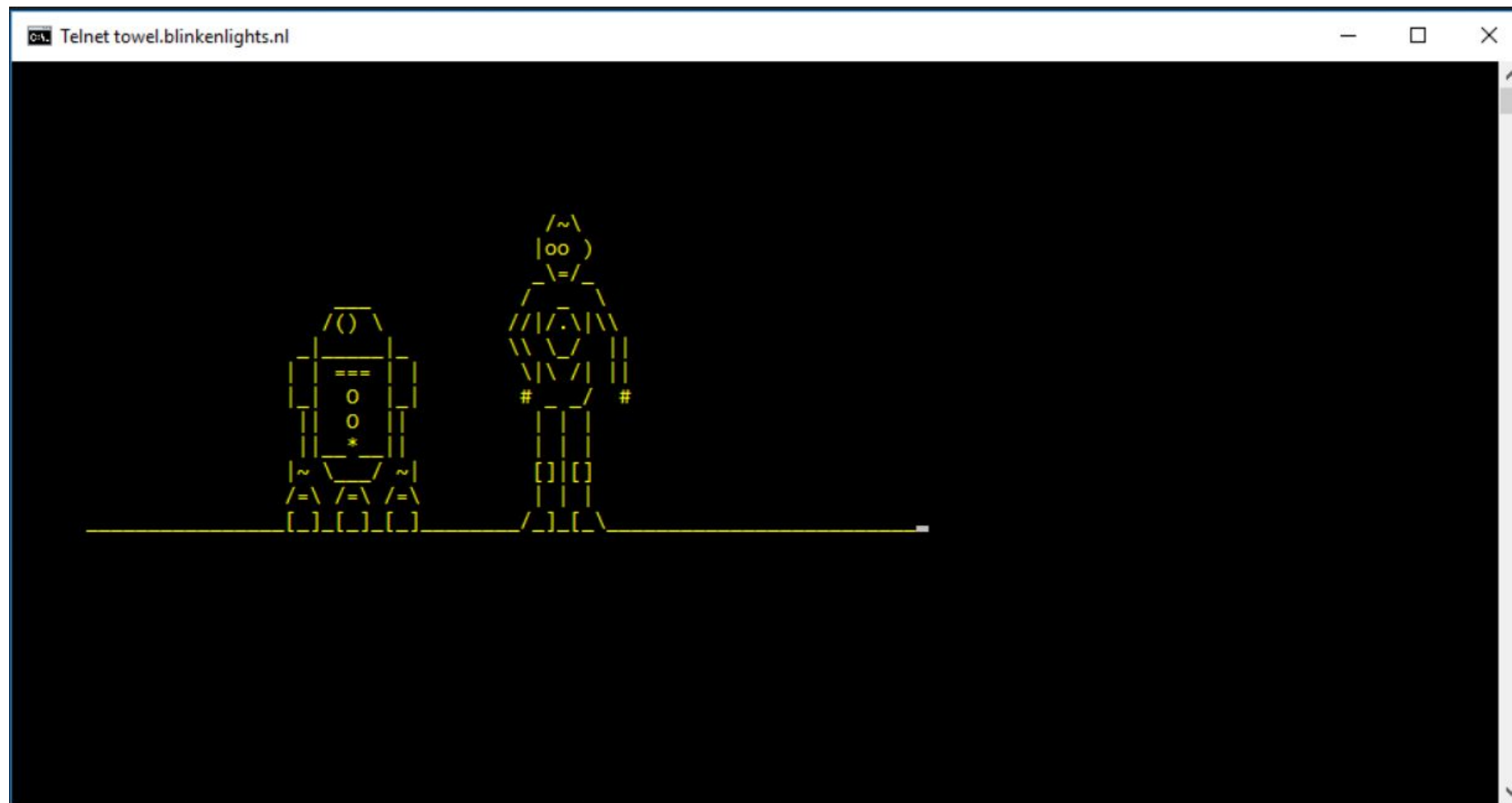
# С помощью компиляции исходного кода

В этом случае необходимо следовать инструкциям, доступным на сайте разработчиков.





# Работа с командной строкой



# Пользователи:

- **id** – выводит подробную информацию о пользователе (uid, gid и группа);
- **last** – выводит список информации о последних входах в систему, включая время, имя пользователя, ip-адрес и длительность сеанса;
- **who** - просмотр авторизованных пользователей;
- **groupadd "testgroup"** - создаёт группу с именем "testgroup";
- **adduser NewUser** - добавляет пользователя с именем "NewUser";
- **userdel NewUser** - удаляет пользователя с именем "NewUser";
- **usermod NewUser** - изменяет информацию о пользователе "New-User".



# Навигация по каталогам:

- **cd /. -** переход в основной каталог;
- **cd -** переход в домашний каталог (переменная \$HOME);
- **cd /root -** переход в каталог /root;
- **cd .. -** переход на один уровень ниже;
- **cd /root/.ssh -** переход в скрытую папку .ssh.

# Работа с файлами:

- **ls -al** – показывает файлы и каталоги в текущей папке;
- **pwd** - отображает текущий рабочий каталог;
- **mkdir NewFolder** - создаёт новый каталог с именем "NewFolder";
- **rm NewFile** - удаляет файл с именем "NewFile";
- **rm -f NewFile** - принудительно удаляет файл с именем "NewFile";
- **rm -r NewFolder** - рекурсивно удаляет каталог с именем "NewFolder";
- **rm -rf NewFolder** - принудительно рекурсивно удаляет каталог с именем "NewFolder";
- **cp oldfile1 newfile2** – копирует содержимое oldfile1 в newfile2;
- **cp -r olddir1 newdir2** - рекурсивно копирует каталог "olddir1" в "new-dir2". Dir2 будет создан, если он не существует;

- **mv oldfile1 newfile2** - переименовывает "oldfile1" в "newfile2";
- **ln -s /etc/log/file logfile** - создаёт ярлык (soft link) на файл;
- **touch newfile** - создаёт пустой файл с именем newfile;
- **cat > newfile** - перенаправляет STDIN (стандартный вывод) в файл newfile;
- **more newfile** - выводит содержимое newfile по частям;
- **head newfile** - выводит первые 10 строк файла newfile;
- **tail newfile** - вывод последних 10 строк newfile. Флаг -f позволяет выводить содержимое в infinity режиме. Полезно для отладки с использованием логирования в файл. С помощью флага -n можно управлять количеством выводимых строк;
- **wc newfile** - выводит количество байт, слов и строк нового файла.

# Права доступа к файлам/каталогам:

- **chmod 777 /root/ssh** - устанавливает права rwx (чтение, запись, выполнение) на файл ssh для всех, кто имеет доступ к серверу (владелец, группа, другие);
- **chmod 755 /root/ssh** - настраивает разрешения rwx для владельца и r\_x для группы и других пользователей;
- **chmod 766 /root/ssh** - устанавливает права rwx для владельца и rw для группы и других пользователей;
- **chown newuser newfile** - меняет владельца newfile на newuser;
- **chown newuser:newgroup newfile** - изменяет владельца и группу-владельца для newfile на newuser и newgroup;
- **chown newuser:newgroup newfolder** - меняет владельца и группу-владельца каталога newfolder на newuser и newgroup;
- **stat -c "%U %G" newfile** - отображает владельцев пользователей и групп newfile.

# Поиск:

- **grep searchargument newfile** - поиск аргумента searchargument в newfile;
- **grep -r searchargument newfolder** - рекурсивно просматривает все файлы в папке newfolder на наличие поискового аргумента;
- **locate newfile** - показывает все местоположения нового файла;
- **find /etc/ -name "searchargument"** - находит файлы с именем, начинающимся с searchargument, в каталоге /etc;
- **find /etc/ -size +50000k** – находит все файлы размером более 50000k в каталоге /etc.

# Архивирование:

- **tar -cf archive.tar newfile** - создаёт архив 'archive.tar' из файла 'newfile'
- **tar -xf archive.tar** - распаковывает файл 'archive.tar';
- **tar -zcvf archive.tar.gz /var/log/** - создаёт архив из каталога /var/log;
- **gzip newfile** - сжимает новый файл (он будет иметь расширение .gz).

# Процессы:

- **ps** - выводит текущие запущенные процессы;
- **ps aux | grep 'bash'** - ищет идентификатор процесса 'bash';
- **top** - показывает все запущенные процессы;
- **kill pid** - завершает процесс по pid;
- **killall process** - завершает все процессы с именем "process";
- **pkill process-name** - посылает сигнал процессу;
- **bg** - отправляет приостановленный процесс на фоновое выполнение;
- **fg** - выводит запущенный процесс из фонового режима;
- **fg process** - выводит процесс с именем "process" из фонового режима;
- **lsuf** – показывает списки файлов, которые используют процессы;
- **pgrep bash** - ищет идентификатор процесса bash;
- **pstree** - показывает древовидное представление процессов.



# Система:

- **uname** - показывает информацию о системе;
- **uname -r** - показывает информацию о ядре Linux;
- **uptime** - показывает продолжительность работы системы и среднюю загрузку;
- **hostname** - показывает имя хоста;
- **hostname -i** - показывает IP-адрес хоста;
- **date** - показывает дату и время;
- **w** - отображает пользователей, работающих в системе;
- **whoami** - отображает ваше имя пользователя.

# Использование диска:

- **df -h** - показывает свободное пространство на смонтированных разделах (в байтах)
- **df -i** - показывает свободные inodes в файловой системе
- **fdisk -l** - показывает информацию о диске, разделах и файловой системе
- **du -sh** - отображает нераспределенное пространство на смонтированных разделах в MB, GB, TB
- **findmnt** - отображает все точки монтирования
- **mount /dev/sdb1 /mnt** - монтирует раздел 1 диска sdb в /mnt

# Сеть:

- **ip addr show** - показывает IP-адреса всех доступных сетевых интерфейсов;
- **ip address add 192.168.0.1/24 dev eth0** - присваивает адрес 192.168.0.1 интерфейсу eth0;
- **ifconfig** - показывает IP-адреса всех доступных сетевых интерфейсов;
- **ping 192.168.0.1** - отправляет запрос по протоколу ICMP для подключения к узлу 192.168.0.1.
- **whois domain** - показывает информацию о доменном имени;
- **dig domain** - получает информацию DNS о домене;
- **dig -x 192.168.0.1** - инвертирует разрешение имен;

- **host sevsu.ru** – резолвит адрес хоста (отправляет запрос и ожидает ответа);
- **hostname -I** - показывает локальные адреса;
- **wget имя\_файла(ссылка на файл)** - загружает файл;
- **netstat -pnltu** - показывает все порты, прослушиваемые на хосте (требуется "apt-get install net-tools").

# Удалённое подключение:

- **ssh root@host** - подключает к удалённому хосту по ssh от имени root;
- **ssh -p port\_number user@host** - подключает к удалённому хосту, если используется порт ssh, отличный от 22;
- **ssh -i <path\_to\_rsa\_private\_key> user@host** - подключает к удалённому хосту с аутентификацией по ключу, если ключи расположены не в месте по умолчанию (/home/<username>/.ssh/id\_rsa);
- **ssh-keygen** - генерирует пару public-private ключей для доступа к серверу с аутентификацией по ключу;
- **ssh host** - использует соединение по умолчанию в качестве текущего пользователя;
- **telnet host <port>** - использует соединение telnet (если <port> не задан, по умолчанию применяется порт 23).

# Установка и настройка веб-сервера



# Установка nginx

[http://nginx.org/en/linux\\_packages.html#Ubuntu](http://nginx.org/en/linux_packages.html#Ubuntu)

Установите пакеты, необходимые для подключения apt-репозитория:

```
sudo apt install curl gnupg2 ca-certificates lsb-release  
ubuntu-keyring
```

Нужно импортировать официальный ключ, используемый apt для проверки подлинности пакетов:

```
curl https://nginx.org/keys/nginx_signing.key | gpg  
--dearmor | sudo tee  
/usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null
```



Для подключения apt-репозитория для стабильной версии nginx, выполните следующую команду:

```
echo "deb  
[signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg  
] https://nginx.org/packages/ubuntu `lsb_release -cs`  
nginx" | sudo tee/etc/apt/sources.list.d/nginx.list
```

Для использования пакетов из репозитория nginx вместо распространяемых в дистрибутиве, настройте закрепление:

```
echo -e "Package: *\nPin: origin nginx.org\nPin: release  
o=nginx\nPin-Priority: 900\n" | sudo tee  
/etc/apt/preferences.d/99nginx
```

Осталось обновить пакеты и установить nginx:

```
sudo apt update
```

```
sudo apt install nginx
```

Убедимся, что nginx установлен и выведем его версию:

```
nginx -v
```

```
nginx version: nginx/1.18.0
```

Запустим nginx в фоновом режиме (daemon):

```
sudo systemctl start nginx
```

Добавим nginx в автозагрузку при старте сервера:

```
sudo systemctl enable nginx
```

Открыв браузер и введя `http://localhost`, вы должны увидеть на экране следующее:

## **Welcome to nginx!**

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

Рисунок – Стартовая страница  
nginx

# Установка php-fpm

Для установки последней версии введем следующие команды:

```
sudo apt install software-properties-common
```

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

```
sudo apt update
```

```
sudo apt install php-fpm php-mysql php-mbstring php-xml  
php-curl php-gd
```

Убедимся, что php нужной версии установлен:


```
php -v
```

```
PHP 8.1.11 (cli) (built: Sep 29 2022 22:28:49) (NTS)
```

```
Copyright (c) The PHP Group
```

```
Zend Engine v4.1.11, Copyright (c) Zend Technologies
```

```
with Zend OPcache v8.1.11, Copyright (c), by Zend  
Technologies
```



# Установка mariadb (fork mysql)

<https://downloads.mariadb.org/mariadb/repositories/>

## Установка MariaDB 10.9

```
sudo apt install apt-transport-https curl
sudo curl -o
/etc/apt/trusted.gpg.d/mariadb_release_signing_key.asc
'https://mariadb.org/mariadb_release_signing_key.asc'
sudo sh -c "echo 'deb
https://mirror.docker.ru/mariadb/repo/10.9/ubuntu focal
main' >>/etc/apt/sources.list"
sudo apt update
sudo apt install mariadb-server
```

Убедимся, что установлена MariaDB 10.9

```
mysql --version
```

```
mysql Ver 15.1 Distrib 10.9.3-MariaDB, for  
debian-linux-gnu (x86_64) using readline 5.2
```



# Первоначальная настройка базы данных

Начнём настройку mysql:

```
sudo mysql_secure_installation
```

Используем нативную аутентификацию:

```
Switch to unix_socket authentication [Y/n] n
```

Сменим пароль рута mysql:

```
Change the root password? [Y/n] Y
```

Удалим остальных анонимных пользователей:

```
Remove anonymous users? [Y/n] Y
```

**Запретим подключаться удаленно от рута:**

`Disallow root login remotely? [Y/n] Y`

**Удалим тестовую БД, созданную при установке:**

`Remove test database and access to it? [Y/n] Y`

**Применим настройки:**

`Reload privilege tables now? [Y/n] Y`

# Настройка nginx

По умолчанию nginx при установке создает файл конфигурации для одного хоста. Он находится в папке sites-available:  
`/etc/nginx/sites-available/default`

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server;  
    root /var/www/html;  
    server_name _;  
    location / {  
        try_files $uri $uri/ =404;  
    }  
}
```

Наиболее важные директивы:

- **server\_name** - это хост, на который поступает запрос. В качестве `server_name` по-умолчанию выступает `localhost`;
- **listen** - обычно здесь указывается порт, который “слушает” веб-сервер;
- **root** - корневая директория проекта;
- **index** - файл, который является корневым, при обращении к корневой директории в запросе.

Например, если указать в качестве index файл index.html, то при запросе `http://localhost` будет произведен поиск файла `/var/www/html/index.html` (root/index). В случае его отсутствия, сервер выдаст ошибку.

Nginx может обрабатывать не один, а несколько хостов. Для этого они все должны иметь уникальное в рамках сервера сочетание `server_name` и `port`.

Не может быть двух сайтов на `localhost:80`.

## Создадим новый виртуальный хост:

```
sudo nano /etc/nginx/sites-available/sevgu.conf
```

```
server {  
    listen 80;  
    server_name sevgu.local;  
    root /var/www/sevgu/data;  
    access_log /var/www/sevgu/logs/access.log;  
    error_log /var/www/sevgu/logs/error.log;  
    index index.php;  
    charset utf-8;  
  
    location / {  
        try_files $uri $uri/  
/index.php?$query_string;  
    }
```

```
    location = /favicon.ico {  
        access_log off;  
        log_not_found off;  
    }  
  
    location = /robots.txt {  
        access_log off;  
        log_not_found off;  
    }  
  
    error_page 404 /index.php;
```

```
    location ~ /\.php$ {  
        fastcgi_pass unix:  
/var/run/php/php8.1-fpm.sock;  
        fastcgi_param  
SCRIPT_FILENAME $realpath_root$  
fastcgi_script_name;  
        include fastcgi_params;  
    }  
  
    location ~ /\.(!well-known).* {  
        deny all;  
    }  
}
```

Чтобы конфигурация была рабочей, необходимо создать заданные папки, как указано в конфигурации:

```
sudo mkdir -p /var/www/sevgu/data
```

```
sudo mkdir -p /var/www/sevgu/logs
```

Теперь для папки sevgu нужно дать права текущему пользователю, а точнее, сделать его владельцем папки:

```
sudo chown -R username:username /var/www/sevgu.
```

Далее создадим файл index.php внутри папки /var/www/sevgu/data с содержимым:

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

Теперь мы можем проверить конфигурацию:

```
sudo nginx -t
```

Данная команда выведет информацию об успешности тестирования конфигурации.



Далее необходимо создать ярлык в папке sites-enabled для того, чтобы конфиг активировать.

```
sudo ln -s /etc/nginx/sites-available/sevgu.conf  
/etc/nginx/sites-enabled/sevgu.conf
```

И перезапустим nginx:

```
sudo systemctl restart nginx
```

И последнее, что нужно сделать, добавить в локальный DNS наш server\_name. Для этого отредактируем файл /etc/hosts, добавив в конец строку:

```
127.0.0.1 sevgu.local
```

Проверим работу нашего сайта: <http://sevgu.local>, где должны увидеть информацию об установке PHP.

# PHP расширения

Расширения php используют, когда требуется расширить набор функций для работы PHP-скриптов на сервере. Каждое расширение имеет узкую функциональность и способно серьезно облегчить и ускорить выполнение конкретных задач, связанных с php-кодом.

Например, для выполнения запросов из нашего PHP-скрипта на другой backend можно использовать расширение cURL, а для выполнения запросов к базе данных - PDO.

Список практически всех доступных расширений есть на официальном сайте php.

Установленных версий php на сервере может быть несколько. Нативная версия PHP — версия из официального репозитория ОС. Дополнительно можно установить альтернативные версии. Это иногда необходимо, если для работы проекта требуется определенная версия php, которой нет в официальном репозитории.

Далее рассмотрим несколько вариантов установки расширений php:

- через панель управления, например, ISPmanager. Большинство расширений уже установлено и подключено, но есть те, которые можно дополнительно установить или просто подключить.
- через стандартные пакетные менеджеры yum (CentOS) и apt (Ubuntu/Debian) для нативной версии php.
- через репозиторий модулей pecl — подойдёт как для нативной, так и для альтернативной версии php.

# Коротко о pecl

Во время установки расширений с помощью пакетных менеджеров и альтернативного репозитория модулей, есть вероятность столкнуться с некоторыми проблемами во время работы с pecl.

## Пример 1.

Представим, что у нас две версии php:

Нативная (native) — `/usr/bin/php`

Альтернативная (alt) — `/opt/php81/bin/php` — альтернативная версия php 8.

Чтобы установить расширения php, для нативной и альтернативной версий необходимо использовать разные команды.

Для нативной версии:

```
pecl install Название_расширения
```

Для альтернативной, php 8.1, так:

```
/opt/php81/bin/pecl install Название_расширения
```

**Важно: необходимо указывать полный путь(он может отличаться!) к бинарному файлу pecl.**

## Пример 2

Если вы используете не самую свежую версию php, то во время установки расширения можете получить сообщение о её несоответствии требованиям — по умолчанию выбираются расширения последних версий, которые часто требуют свежих версий php. Пример уведомления:

```
requires PHP (version >= 7.0.0, version <= 7.1.0),  
installed version is 5.6.40
```

В уведомлении указано, что версия php должна быть не ниже 7.0.0 и не выше 7.1.0, а мы пытаемся установить расширение на версию 5.6.40.

Чтобы понять, какая версия расширения нужна для установки на имеющуюся версию php (в данном случае 5.6.40), обратимся за помощью на официальный сайт.

**Важно: после установки любого пакета необходимо перезапустить php-fpm daemon.**



# Установка расширений через apt, yum и pecl

Далее описаны установки расширений для всех версий php — от установки зависимостей до подключения. Описание разделено на версию ОС и версию php: нативную (native) или альтернативную (alt). Будьте внимательны при установке расширений для альтернативной версии через pecl.

# mcrypt

## **native:**

```
apt install php-dev libmcrypt-dev  
pecl install mcrypt  
echo extension=mcrypt.so >> Путь_к_конфигу
```

## **alt:**

```
apt install php-dev libmcrypt-dev  
Полный_путь/pecl install mcrypt  
echo extension=mcrypt.so >> Путь_к_конфигу
```

# Подключение расширения через консоль

После установки нужного расширения php, необходимо его подключить. А точнее, прописать путь к конфигурационному файлу — если этого не сделать, то php не будет знать, что добавлено новое расширение.

Необходимо узнать путь к общему конфигурационному файлу php, указывая полный путь к бинарному файлу, например, для альтернативной версии php 8.1:

```
/opt/php81/bin/php -i | grep 'Configuration File'
```

**Вывод будет следующим:**

```
Configuration File (php.ini) Path => /opt/php81/etc  
Loaded Configuration File => /opt/php81/etc/php.ini
```

**Необходимо значение** /opt/php81/etc/php.ini

# Режимы работы РНР



# Модуль Apache

В данной схеме используется связка nginx + apache, где php выступает в качестве скриптового языка, а весь бекэнд обрабатывает apache, и отдает nginx уже сгенерированный HTML.

Nginx в свою очередь обрабатывает запросы к статическому содержимому, такому как сам HTML, картинки, аудио, видеофайлы, JavaScript и CSS - файлы.

При этом nginx проксирует запросы от клиента на бекэнд и обратно.

**Важно: все запросы на бекэнд Apache обрабатывает сам, внутри своих процессов. При этом расходуется значительная часть вычислительных ресурсов.**

Если из данной цепочки исключить nginx, то Apache будет тратить дополнительные вычислительные ресурсы и на обработку статического контента.

А это, сравнительно с количеством вычислительных ресурсов, которое тратит на это nginx (предназначенный для этого), непозволительная роскошь для небольших проектов.

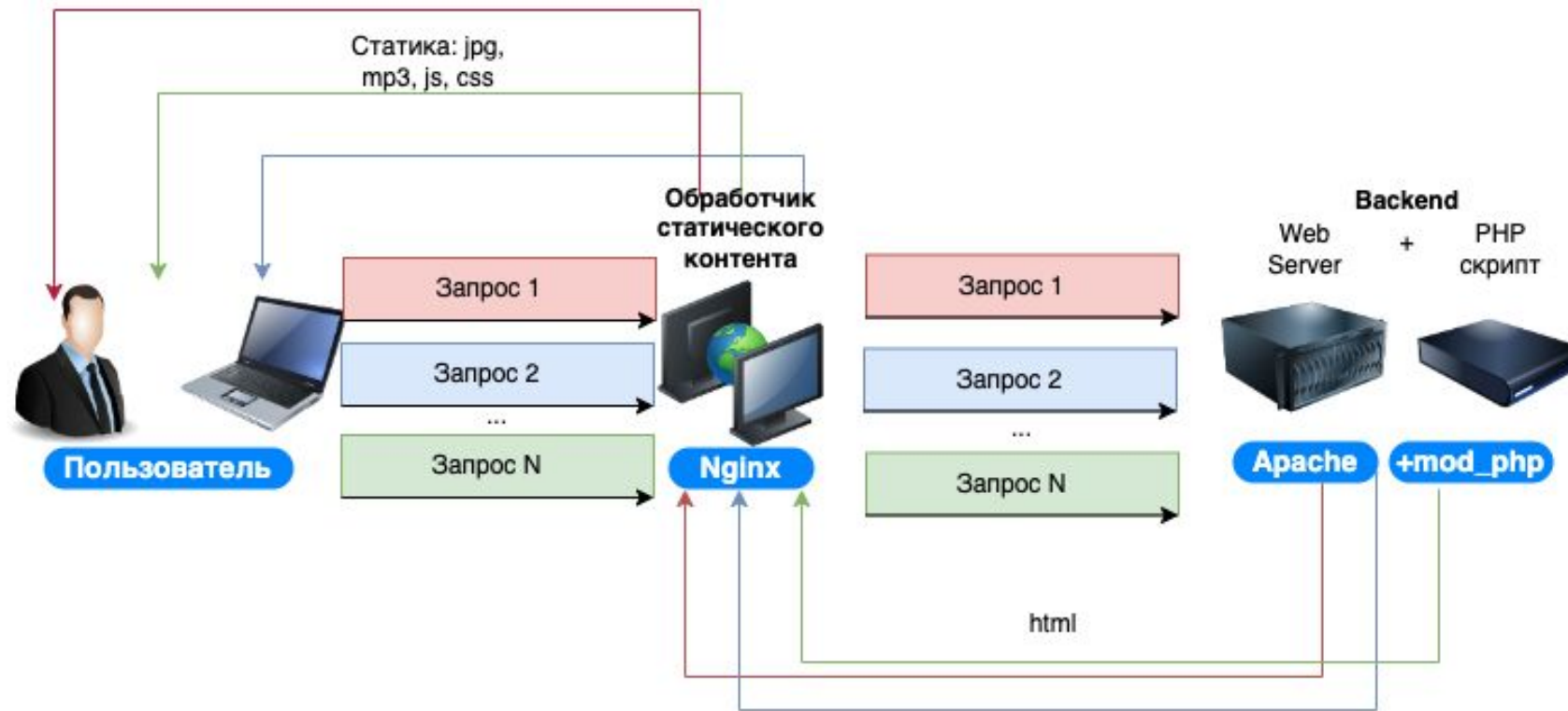


Рисунок – Обработка запросов с помощью Apache



# PHP-CGI и PHP-FastCGI

Как и в схеме с Apache + mod\_php в качестве обработчика статики выступает nginx, а бекэнд контролирует Apache. Важным отличием является то, что сам php код исполняет не сам Apache, а он лишь создает N процессов PHP-CGI, где N - количество запросов на бэкенд. Скрипты CGI обрабатывают запросы, выполняют код и отдают ответ обратно Apache, который через nginx транслирует их клиенту.

В данной схеме Apache является неким контроллером, который запускает нужные процессы и отвечает за их выполнение.

**Важно: каждый запрос на бекэнд порождает соответствующий ему процесс, который данный запрос обрабатывает.**

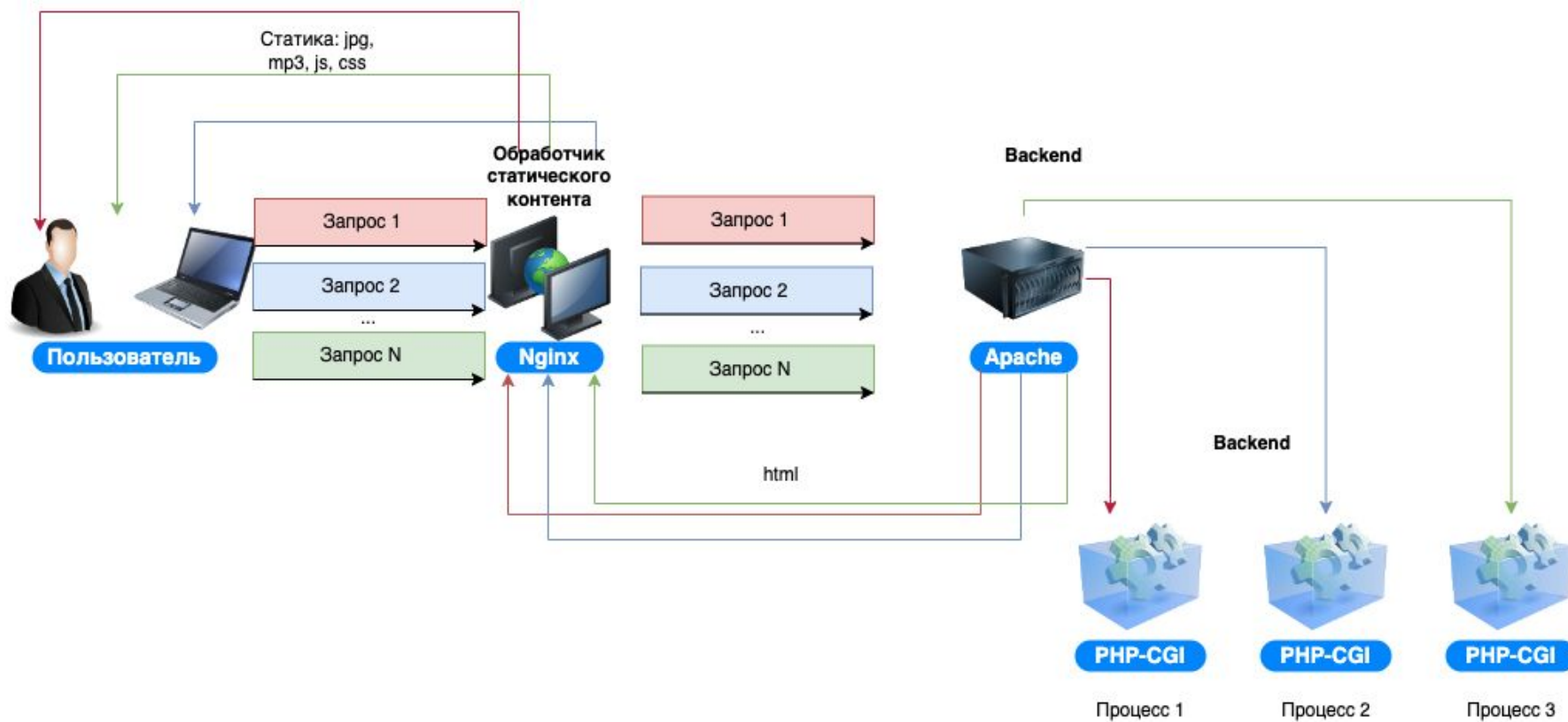


Рисунок - Обработка запросов с помощью CGI.

FastCGI отличается от CGI тем, что каждый запрос на бекэнд не порождает тождественное количество процессов, а запускает один процесс PHP-FastCGI, который обрабатывает все запросы внутри себя. Это несколько оптимальнее CGI, т.к. создание нового процесса - весьма ресурсоемкая задача для операционной системы.

Важно: эта схема также предполагает использование Apache: PHP-FastCGI не может самостоятельно запустить свой процесс, ему все также нужен контроллер.

# PHP-FPM

php-fpm — это FastCGI process manager. Он представляет из себя отдельную службу, которая работает независимо от какого-либо веб-сервера. Он может сам принимать запросы от веб-сервера через unix-сокеты или через сетевое соединение. Другими словами, можно держать сайты на одном сервере, а php-скрипты исполнять на другом.

И в этом случае, Apache становится не нужен, потому что nginx может работать с php-fpm сам, напрямую. Apache тоже может работать в таком же режиме с php-fpm, но это не имеет смысла. php-fpm не может отдавать статические файлы или html, он может только исполнять php. получается такая схема: запрос пользователя > nginx > php-fpm > nginx, он собирает из ответа php и **статика** страницу > отдает пользователю. FPM в разы производительнее Apache, в каком бы режиме тот ни был запущен.

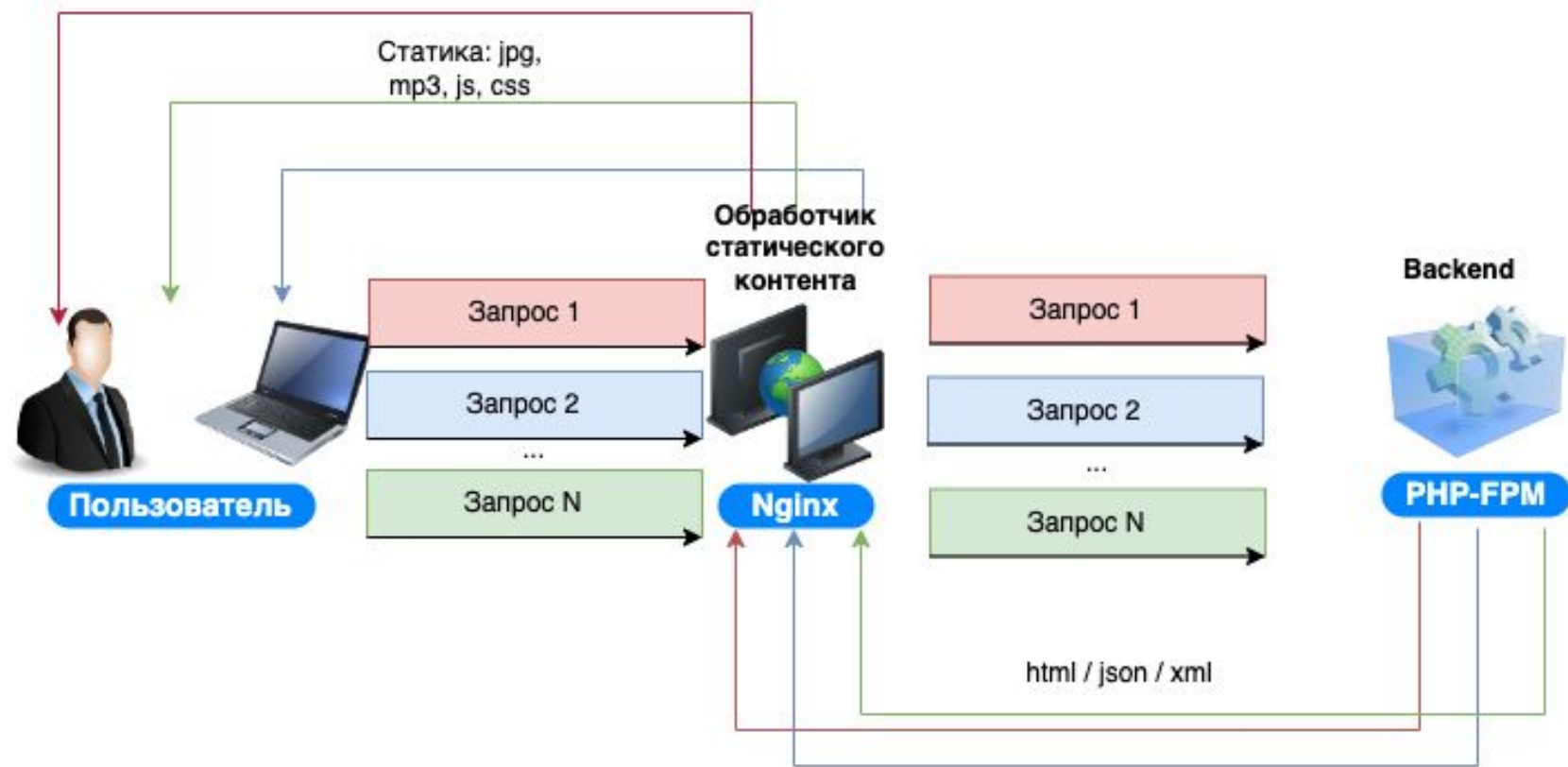


Рисунок - Обработка запросов с помощью php-fpm

# Docker

**Docker** — популярная технология контейнеризации, появившаяся в 2013 году. Тогда одноименная компания предложила способ *виртуализации ОС*, при котором код приложения, среда запуска, библиотеки и зависимости упаковываются в единую «капсулу» — *контейнер Docker*.

Благодаря использованию контейнеров (контейнеризации), в частности Docker, разработчики больше не задумываются о том, в какой среде будет функционировать их приложение и будут ли в этой среде необходимые для тестирования опции и зависимости. Достаточно упаковать приложение со всеми зависимостями и процессами в контейнер, чтобы запускать в любых системах: Linux, Windows и macOS. Платформа Docker позволяет отделить приложения от инфраструктуры. Контейнеры не зависят от базовой инфраструктуры, их можно легко перемещать между облачной и локальной инфраструктурами.

# Основные понятия

**Docker-платформа** - программа, обеспечивающая возможность упаковки и запуска приложения в контейнере на любом Linux сервере. Она собирает код и зависимости. Благодаря хорошей мобильности и воспроизводимости это упрощает масштабирование.

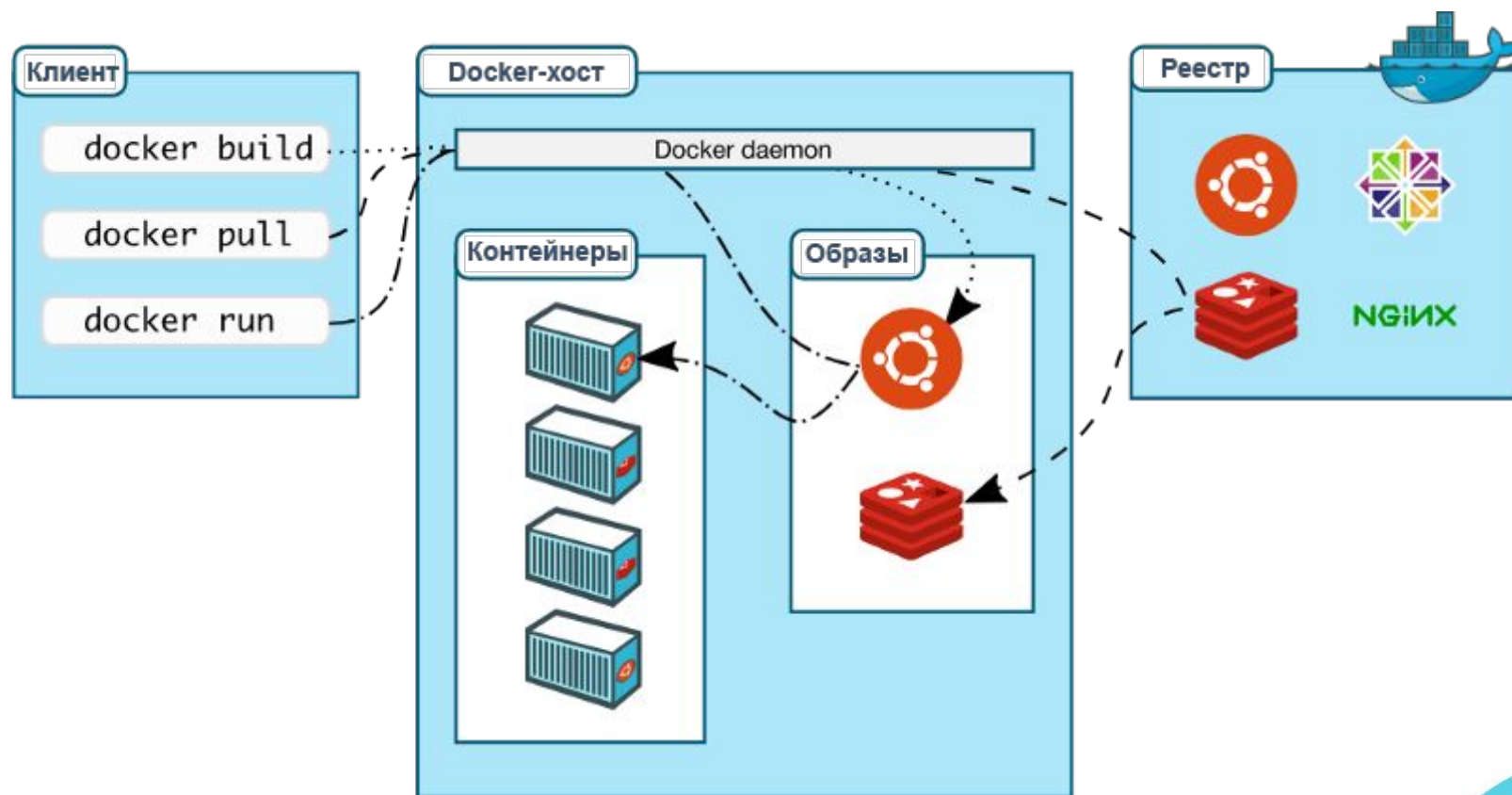
**Docker-движок** - клиент-серверное приложение. Docker Community Edition (CE) — бесплатная версия. Docker Enterprise — платный продукт, он поставляется с дополнительными функциями поддержки, управления и безопасности.

**Docker-клиент** - основной способ взаимодействия с Docker'ом. При использовании Docker Command Line Interface (CLI) вы просто вводите в терминал нужную команду, которая обычно начинается со слова **docker**. Затем Docker-клиент использует Docker API для отправки команды на Docker Daemon.



# Основные понятия

**Docker Daemon** - Docker-сервер, отвечающий на Docker API запросы. Он управляет образами, контейнерами, Docker-сетями и тома Docker.



# Docker образ (image)

Шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

Docker-образ состоит из слоев. Каждое изменение записывается в новый слой.

- При загрузке или скачивании Docker-образа, операции производятся только с теми слоями, которые были изменены.
- Слои исходного Docker-образа являются общими между всеми его версиями и не дублируются.
- На основе образа можно создавать новые, расширенные образы. При этом слои изначального образа не являются общими, и скачивание происходит заново.

# Docker контейнер (container)

Виртуализированная среда выполнения, в которой разработчики могут изолировать приложения от хостовой системы. Эти контейнеры представляют собой компактные портативные хосты, в которых можно быстро и легко запустить приложение.

Контейнер собирается из образа (image). Если проводить аналогию с ООП, *image* - это *класс*, описывающий приложение, а *контейнер* - это *объект*, который имеет установленные свойства.

Важной особенностью контейнера является стандартизация вычислительной среды, работающей внутри контейнера. Это не только гарантирует, что приложение работает в идентичных условиях, но и упрощает обмен данными с другими партнерами по команде.

# Docker контейнер (container)

Контейнеры работают автономно, изолированно от основной системы и других контейнеров, и потому ошибка в одном из них не влияет на другие работающие контейнеры, а также поддерживающий их сервер.

Docker утверждает, что эти блоки «обеспечивают самые сильные возможности изоляции в отрасли».

Поэтому при разработке приложения нет необходимости беспокоиться о безопасности host операционной системы.

**Б - Безопасность**

# Пример. Dockerfile

```
FROM php:8.1-fpm

# Установка необходимых PHP - расширений
RUN docker-php-ext-install pdo_mysql zip curl

# Задание рабочей директории внутри контейнера
WORKDIR /var/www

# Копирование файлов с host ОС в контейнер
COPY /var/www/mysite/data/* ./
COPY /var/www/mysite/entrypoint.sh /entrypoint.sh
```

Команда **RUN** запускает команду в консоли контейнера

**WORKDIR** задает директорию, относительно которой выполняются все последующие команды

**COPY** позволяет копировать необходимые файлы из основной (host) операционной системы внутрь контейнера. Т.к. выше была задана **WORKDIR**, то относительный путь назначения назначается с учетом **WORKDIR**.

# Пример. docker-compose.yml

```
version: "3.7"
```

```
services:
```

```
  nginx:
```

```
    container_name: nginx
```

```
    image: nginx:latest
```

```
    restart: unless-stopped
```

```
    networks:
```

```
      - my-network
```

```
    ports:
```

```
      - "333:80"
```

```
    volumes:
```

```
      - "/var/www/mysite/data:/var/www"
```

```
      - "/var/www/mysite/nginx/nginx.conf:/etc/nginx/conf.d/nginx.conf"
```

```
  db:
```

```
    image: mysql:8.0
```

```
    container_name: db
```

```
    restart: unless-stopped
```

```
    environment:
```

```
      - MYSQL_DATABASE=my_db
```

```
      - MYSQL_ROOT_PASSWORD=my_root_password
```

```
      - MYSQL_USER=my_user
```

```
      - MYSQL_PASSWORD=my_password
```

```
  app:
```

```
    restart: unless-stopped
```

```
    container_name: app
```

```
    build:
```

```
      context: .
```

```
      dockerfile: ./Dockerfile_custom
```

```
    entrypoint: /entrypoint.sh
```

```
    volumes:
```

```
      - "/var/www/mysite/data:/var/www"
```

```
    networks:
```

```
      - my-network
```

```
    depends_on:
```

```
      - nginx
```

```
      - db_mysql
```

```
    ports:
```

```
      - "13306:3306"
```

```
    volumes:
```

```
      - "/var/www/mysite/database/data:/var/lib/mysql"
```

```
    networks:
```

```
      - my-network
```

```
networks:
```

```
  my-network:
```

```
    driver: bridge
```



# Пример. Networks

```
networks:  
  my-network:  
    driver: bridge
```

В данной секции задаются правила создания и функционирования docker-сетей. В Docker используется сетевой мост (bridge), обычно его имя в host ОС - docker0. Для каждого контейнера создается свой виртуальный сетевой интерфейс, он и подключается к сети обычно при помощи bridge.

В Docker по умолчанию представлены сетевые драйвера:

- **bridge** — мост, самый распространенный интерфейс.
- **macvlan** — контейнер подключается при помощи виртуального интерфейса, подключенного к физическому. При этом у каждого из них есть свой MAC-адрес.
- **host** — как видно из названия, подключение происходит к сети хоста. Это значит, что контейнер может общаться с сервисами, которые запущены на локальном интерфейсе так, как если бы он был запущен прямо на хосте. Крайне небезопасно, т.к. из контейнера можно получить доступ к основной ОС.
- **none** — означает, что сети нет.

# Пример. Services. Nginx

```
nginx:
  container_name: nginx
  image: nginx:latest
  restart: unless-stopped
  networks:
    - my-network
  ports:
    - "333:80"
  volumes:
    - "/var/www/mysite/data:/var/www"
    - "/var/www/mysite/nginx/nginx.conf:/etc/nginx/conf.d/nginx.conf"
```

Контейнер, который отвечает за запуск веб-сервера nginx. Данный контейнер собирается из образа (image) nginx:latest, однако можно задать специфическую версию.



# Пример. Services. Nginx

**restart** обозначает политику перезапуска контейнера в случаях, если он находится в состоянии, отличном от “запущен”.

**networks** - список docker-сетей, доступ к которым имеет данный контейнер, внутри которых он идентифицирован.

**ports** говорит о том, что внутри сети (в данном случае my-network) контейнер запускается на порту 80, и происходит “проброс портов”. Т.е. из-вне сети контейнер nginx будет доступен на 333 порту. Например, из host ОС, можно обратиться к nginx из браузера, введя <http://localhost:333> в строке ввода адреса.

**volumes** позволяет синхронизировать файлы host ОС и контейнера. В данном примере синхронизируются все файлы приложения PHP (необходимо для работы php-fpm) и файл конфигурации nginx.

# Пример. Services. App

```
app:
  restart: unless-stopped
  container_name: app
  build:
    context: .
    dockerfile: ./Dockerfile_custom
  entrypoint: /entrypoint.sh
  volumes:
    - "/var/www/mysite/data/:/var/www"
  networks:
    - my-network
  depends_on:
    - nginx
    - db_mysql
```

Данный контейнер также собирается из определенного образа (php:8.1-fpm), однако данный образ расширяется с помощью Dockerfile.

Приводя аналогию с ООП, производится наследование и расширение базового класса.

# Пример. Services. App

```
app:
  restart: unless-stopped
  container_name: app
  build:
    context: .
    dockerfile: ./Dockerfile_custom
  entrypoint: /entrypoint.sh
  volumes:
    - "/var/www/mysite/data/:/var/www"
  networks:
    - my-network
  depends_on:
    - nginx
    - db_mysql
```

**build** - секция, описывающая правила создания нового образа на основе базового.

**context** в данном примере говорит о том, внутри какой директории должен работать сборщик образа, а **dockerfile** указывает, какой файл использовать при сборке.

Важно - в случае, если у файла имя Dockerfile, указывать **dockerfile** в секции **build** необязательно.

# Пример. Services. App

```
app:
  restart: unless-stopped
  container_name: app
  build:
    context: .
    dockerfile: ./Dockerfile_custom
  entrypoint: /entrypoint.sh
  volumes:
    - "/var/www/mysite/data/:/var/www"
  networks:
    - my-network
  depends_on:
    - nginx
    - db_mysql
```

**entrypoint** - указание файла, который описывает действия, которые должны привести к запуску контейнера.

В данном примере происходит запуск php-fpm в режиме демона.

```
#!/bin/bash
set -e
php-fpm --daemonize
```

**depends\_on** - список сервисов, которые должны запуститься до запуска данного контейнера. Проще говоря - это зависимости контейнера.

# Пример. Services. DB

```
db:
  image: mysql:8.0
  container_name: db
  restart: unless-stopped
  environment:
    - MYSQL_DATABASE=my_db
    - MYSQL_ROOT_PASSWORD=my_root_password
    - MYSQL_USER=my_user
    - MYSQL_PASSWORD=my_password
  ports:
    - "13306:3306"
  volumes:
    - "/var/www/mysite/database/data:/var/lib/mysql"
  networks:
    - my-network
```

Контейнер, запускающий сервис СУБД. В данном примере это - MySQL.

Помимо вышеуказанных параметров стоит отметить **environment**.

**environment** - это список аргументов, которые могут быть использованы внутри контейнера для определенных процедур.



# Пример. Services. DB

```
db:
  image: mysql:8.0
  container_name: db
  restart: unless-stopped
  environment:
    - MYSQL_DATABASE=my_db
    - MYSQL_ROOT_PASSWORD=my_root_password
    - MYSQL_USER=my_user
    - MYSQL_PASSWORD=my_password
  ports:
    - "13306:3306"
  volumes:
    - "/var/www/mysite/database/data:/var/lib/mysql"
  networks:
    - my-network
```

Например, в данном образе предусмотрено создание базы данных (если она еще не создана), создание пользователя с паролем, а также установка пароля для root.

# Пример. Services. DB

```
db:
  image: mysql:8.0
  container_name: db
  restart: unless-stopped
  environment:
    - MYSQL_DATABASE=my_db
    - MYSQL_ROOT_PASSWORD=my_root_password
    - MYSQL_USER=my_user
    - MYSQL_PASSWORD=my_password
  ports:
    - "13306:3306"
  volumes:
    - "/var/www/mysite/database/data:/var/lib/mysql"
  networks:
    - my-network
```

Также стоит отметить, что в случае, если не указан **volume**, то при перезапуске контейнера, все данные из БД будут удалены, т.к. хранилище docker-контейнера не персистентно, и будет произведена установка пустой БД.

Использование **volume** в данном примере позволяет хранить файлы СУБД персистентно в host ОС, а при запуске контейнера маунтить этот **volume** к контейнеру.

# Пример. Services. DB

```
db:
  image: mysql:8.0
  container_name: db
  restart: unless-stopped
  environment:
    - MYSQL_DATABASE=my_db
    - MYSQL_ROOT_PASSWORD=my_root_password
    - MYSQL_USER=my_user
    - MYSQL_PASSWORD=my_password
  ports:
    - "13306:3306"
  volumes:
    - "/var/www/mysite/database/data:/var/lib/mysql"
  networks:
    - my-network
```

Таким образом СУБД будет понимать, что БД уже существует и инициализирует лишь подключение.