

Некоторые вопросы безопасности и производительности web-приложений. Docker

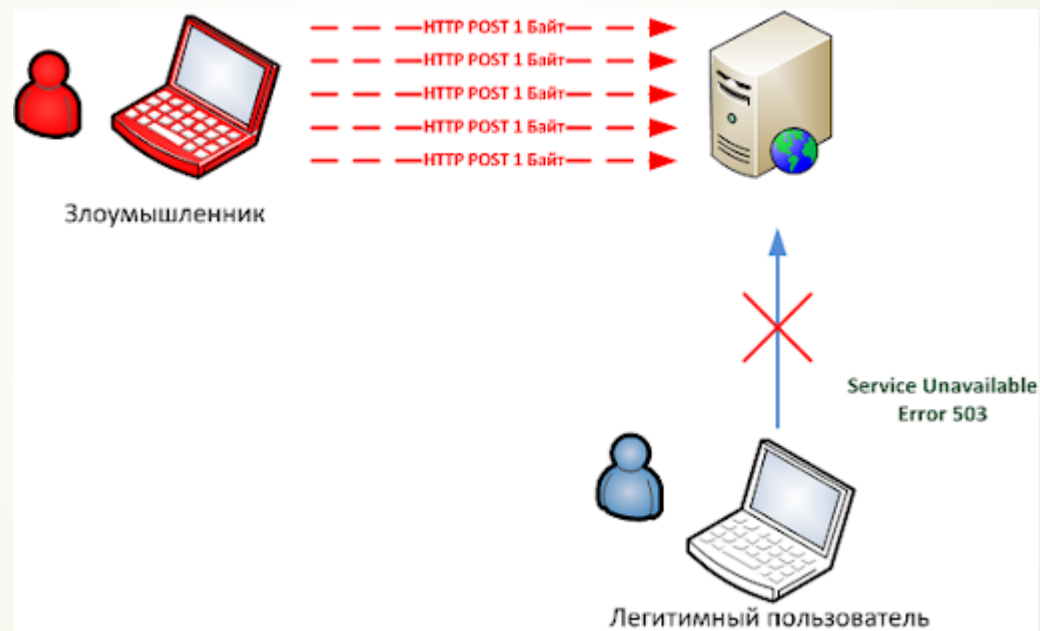


Apache

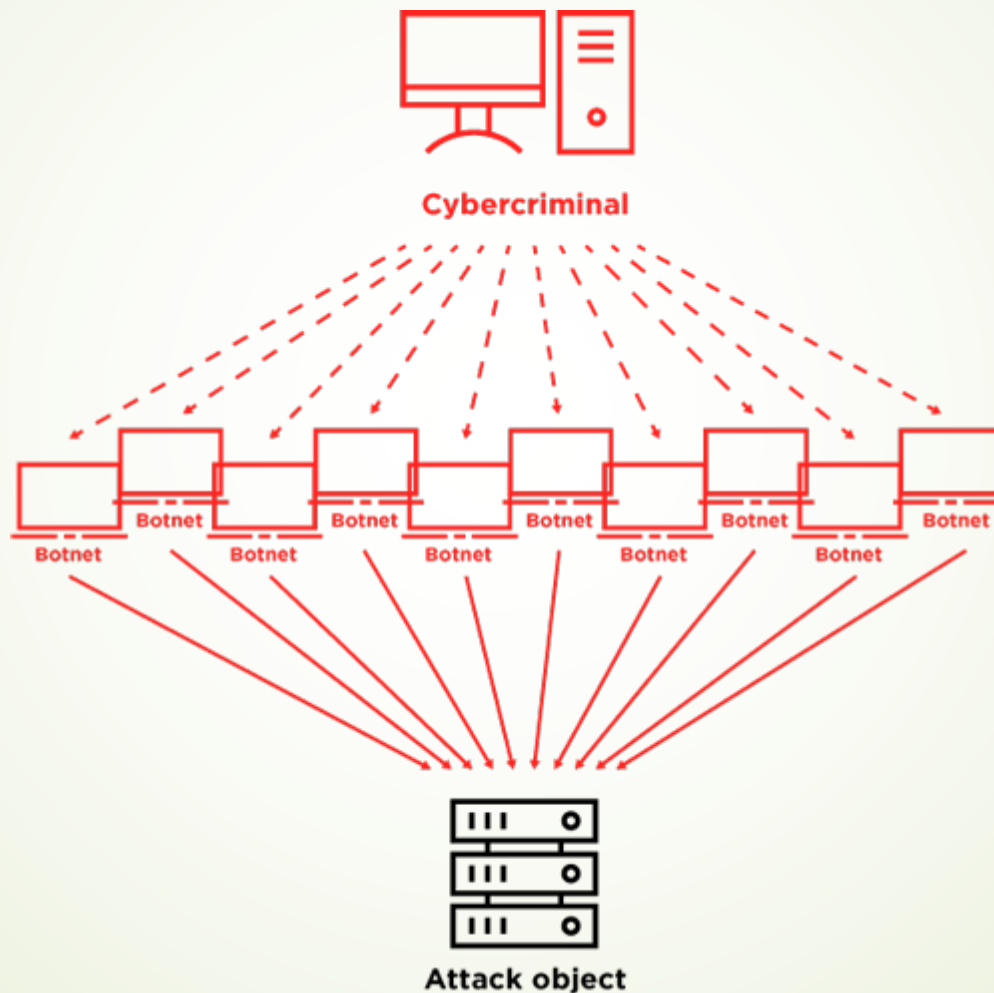


- Веб-сервер может позволять просматривать не только файлы, но и директории. Apache имеет такое поведение по умолчанию
- Запрет листинга каталогов через .htaccess:
Options -Indexes
- Данное условие действует на все вложенные/дочерние папки. Следовательно, добавление данного условия в корневой файл .htaccess запрещает листинг директорий для всего сайта. При необходимости всегда можно разрешить выдачу листинга для конкретно взятой директории, для этого достаточно разместить в ней все тот же .htaccess:
Options Indexes

DoS-атаки (Denial of Service)



DDoS-атаки (Distributed DoS)





Уязвимости БД

- Пользователи на уровне СУБД в случае нескольких БД на одном сервере
- Служебный пользователь с правами root
- Хранение пароля в открытом виде
- SQL-инъекции

SQL-инъекции

- SQL-инъекция – изменение структуры SQL-запроса путем манипулирования входными данными (в параметрах HTTP-запроса или input-элементах)

- Примеры:

`'SELECT user_pass_hash FROM users WHERE user_id ='+id`

`//id: 1 OR 1=1`

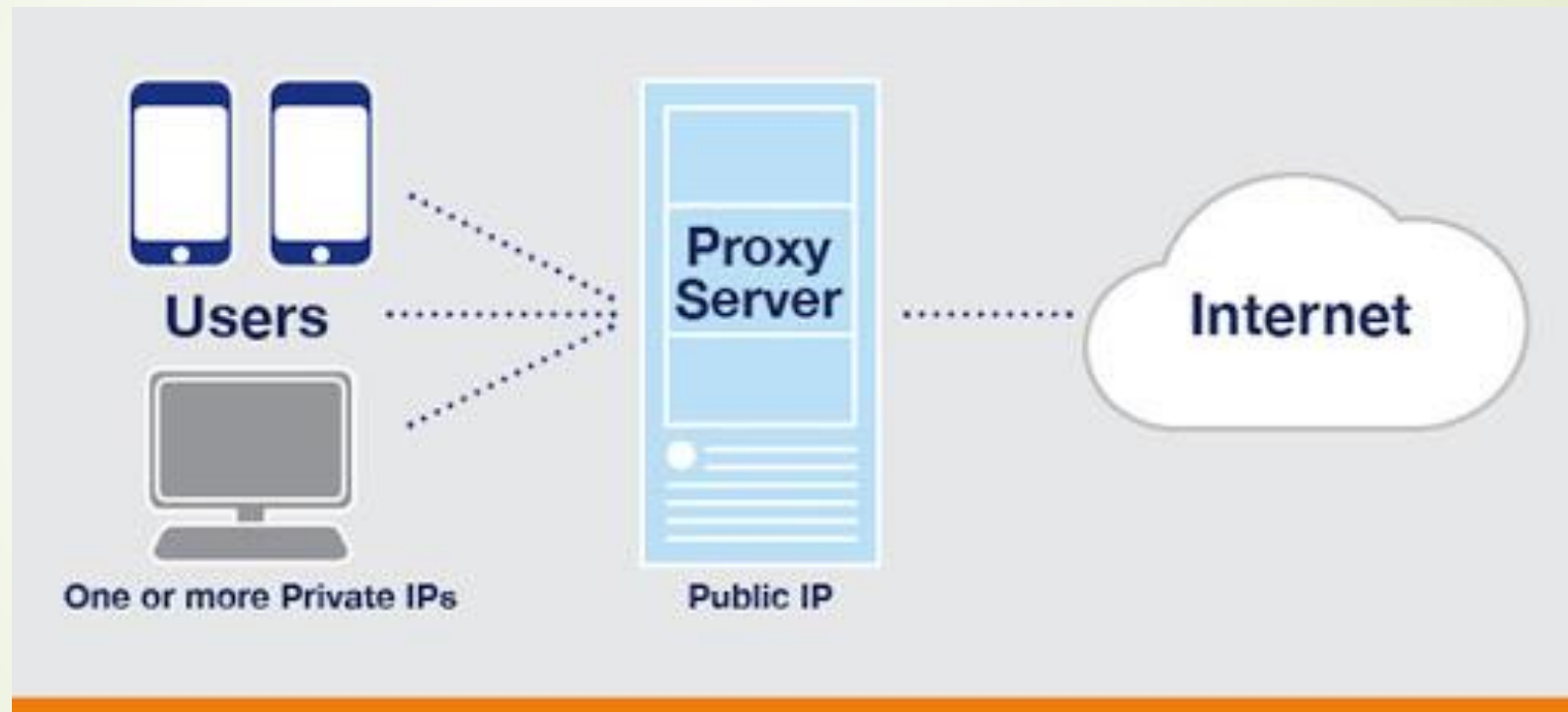
`//id: 1'; DROP TABLE users;`

- Решение:

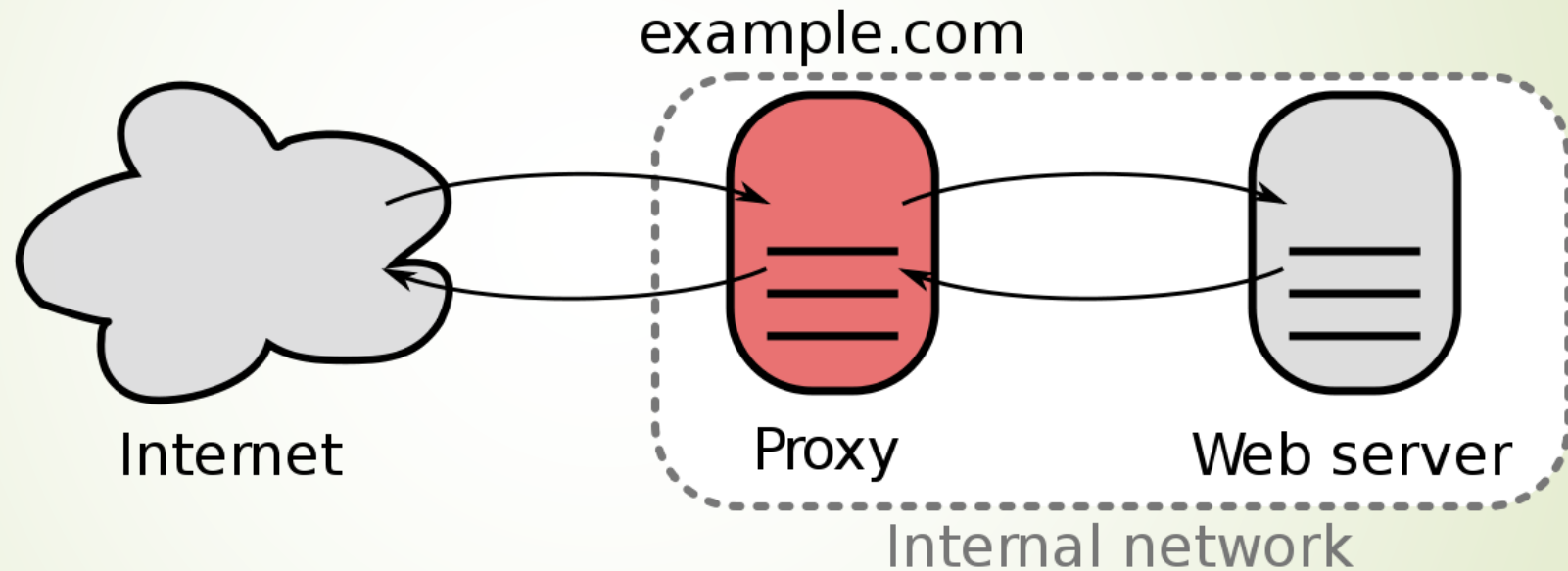
Параметризованные запросы

`'SELECT user_pass_hash FROM user WHERE user_id = %(name)s', {'id': id}`

Proxy



Reverse Proxy (Обратный прокси)



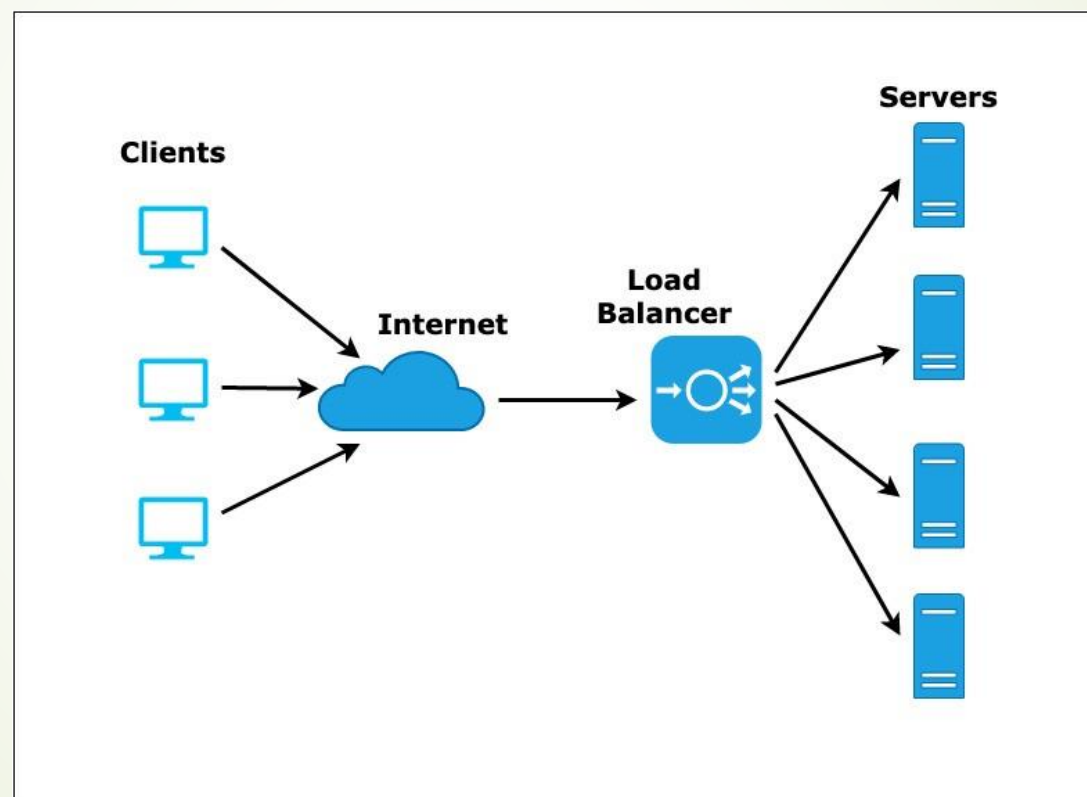


NGINX в роли обратного прокси

```
location /static/ {  
    proxy_pass http://127.0.0.1:3000;  
    proxy_http_version 1.1;  
    proxy_set_header Upgrade $http_upgrade;  
    proxy_set_header Connection 'upgrade';  
    proxy_set_header Host $host;  
    proxy_cache_bypass $http_upgrade;  
}
```

```
location /sockjs-node/ {  
    proxy_pass http://127.0.0.1:3000;  
    proxy_http_version 1.1;  
    proxy_set_header Upgrade $http_upgrade;  
    proxy_set_header Connection 'upgrade';  
    proxy_set_header Host $host;  
    proxy_cache_bypass $http_upgrade;  
}
```

Load Balancers (балансировщики нагрузки)



Виды балансировщиков

- Сетевой - этот метод предполагает использование набора физических серверов. Это достаточно дорогой, но очень эффективный метод, который надежно защищает вебмастера от превышения нагрузки на сервер. Суть его сводится к тому, что несколько разных машин отвечают за работу одного IP-адреса.
- Транспортный - этот тип методов отличается простотой и эффективностью. Транспортное снижение нагрузки на сервер предполагает использование балансировщика, который распределяет запросы по пулу в соответствии с заданными алгоритмами. Балансировщик передает выбранному серверу запросы, а затем получает на них ответ и перенаправляет его обратно пользователю. Транспортные методы работают как прокси — они сами обмениваются данными с сервером, а не связывают клиента и сервер напрямую.
- Прикладная балансировка похожа на транспортную — здесь тоже используется прокси, и запросы пользователей передаются на серверы. Но, в отличие от транспортной балансировки, прикладная распределяет нагрузку с учетом запрашиваемых страниц, контента или действий. Например, запись на сайт (создание учетных записей и подобные действия) будут проводиться через один сервер, а чтение — через другой. Можно распределить нагрузку по типу контента (аудио, видео, изображения, текст).



Алгоритмы распределения нагрузки

- Round Robin – поочередная «закольцованная» отправка запросов на сервера
- Weighted Round Robin – аналогичен предыдущему, но учитывает производительность серверов за счет весовых коэффициентов
- Least Connections - каждый последующий запрос направляется на сервер с наименьшим количеством поддерживаемых подключений
- Sticky Sessions - алгоритме запросы распределяются в зависимости от IP-адреса пользователя. Sticky Sessions предполагает, что обращения от одного клиента будут направляться на один и тот же сервер, а не «скакать» в пуле.



NGINX в роли балансировщика нагрузки

```
http {  
    upstream myapp1 {  
        server srv1.example.com;  
        server srv2.example.com;  
        server srv3.example.com;  
    }  
  
    server {  
        listen 80;  
  
        location / {  
            proxy_pass http://myapp1;  
        }  
    }  
}
```



Sticky Sessions :

```
upstream myapp1 {  
    ip_hash;  
    server srv1.example.com;  
    server srv2.example.com;  
}
```

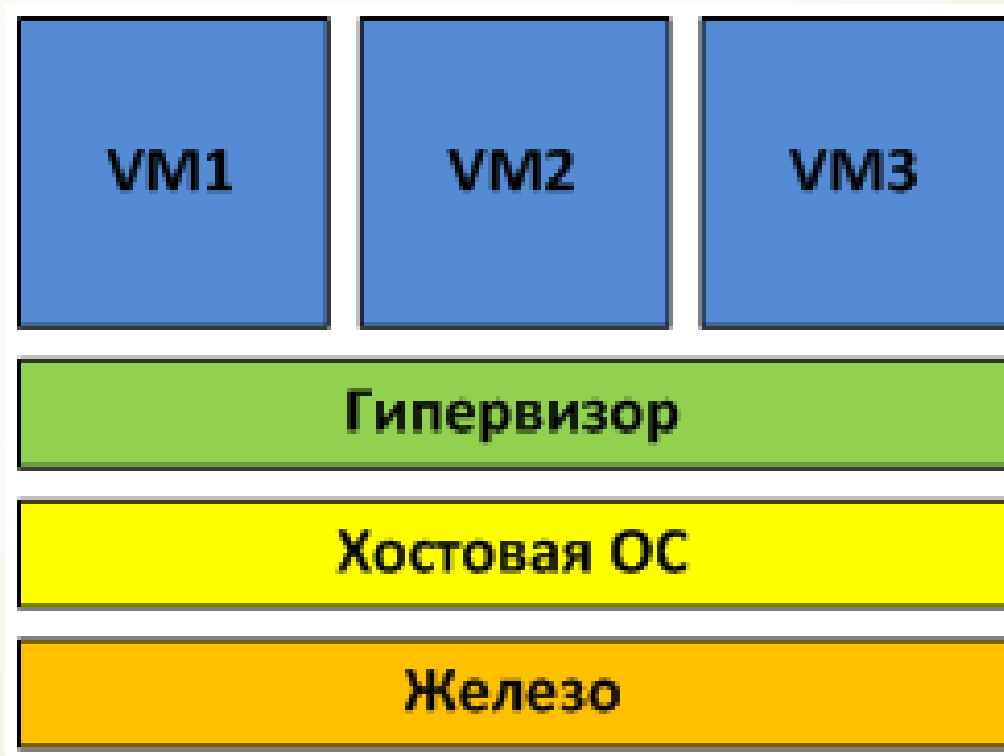
Weighted Round Robin :

```
upstream myapp1 {  
    server srv1.example.com weight=3;  
    server srv2.example.com;  
}
```

Least connected:


```
upstream myapp1 {  
    least_conn;  
    server srv1.example.com;  
    server srv2.example.com;  
}
```


Гипервизор



Настройка сервера

- Установка ОС
- Настройка сети (через виртуальный коммутатор или NAT в случае VM)
- Настройка SSH
 - `ssh-keygen -t rsa`
 - `ssh-copy-id -i ~/.ssh/id_rsa.pub user@host`
 - Файл конфигурации `/etc/ssh/sshd_config`
- Установка СУБД (установку можно проводить через менеджеры `dnf`, `apt` и др. или `wget`):
 - (возможно потребуется обновление репозитория: `sudo dnf update y`)
 - `sudo dnf install mysql-server libmysqlclient-dev`
- Запуск и добавление в автозагрузку:
 - `systemctl start mysqld`
 - `systemctl enable mysqld`
- Настройка СУБД:
 - `sudo mysql_secure_installation`
 - Создание пользователя, БД, настройка прав

- 
- Добавление порта СУБД в firewall (необязательно, нужно для удаленного подключения к БД)

```
sudo firewall-cmd --zone=public --add-port=3306/tcp --permanent
```

- Установка nginx:

```
sudo dnf install nginx
```

- Запуск и добавление в автозагрузку:

```
systemctl start nginx
```

```
systemctl enable nginx
```

- Добавление порта веб-сервера в firewall:

```
sudo firewall-cmd --zone=public --add-port=80/tcp --permanent
```

- Установка python3:

```
sudo dnf install python3
```

- Копирование на сервер микросервиса по ssh (альтернатива - git)

```
scp -P <port> <microservice> <user>@<host>:<directory>
```

- Установка необходимых python библиотек (pip)

- Создание нескольких инстансов, работающих по разным портам (файлы run_1.py, run_2.py и т.д.)

- 
- Описание сервисов в системе (/etc/systemd/system/app1.service и др)

[Unit]

Description=App

After=network.target

After=mysql.service

After=nginx.service

[Service]

User=root


WorkingDirectory=/root/app

ExecStart=/bin/python3 /root/app/run.py

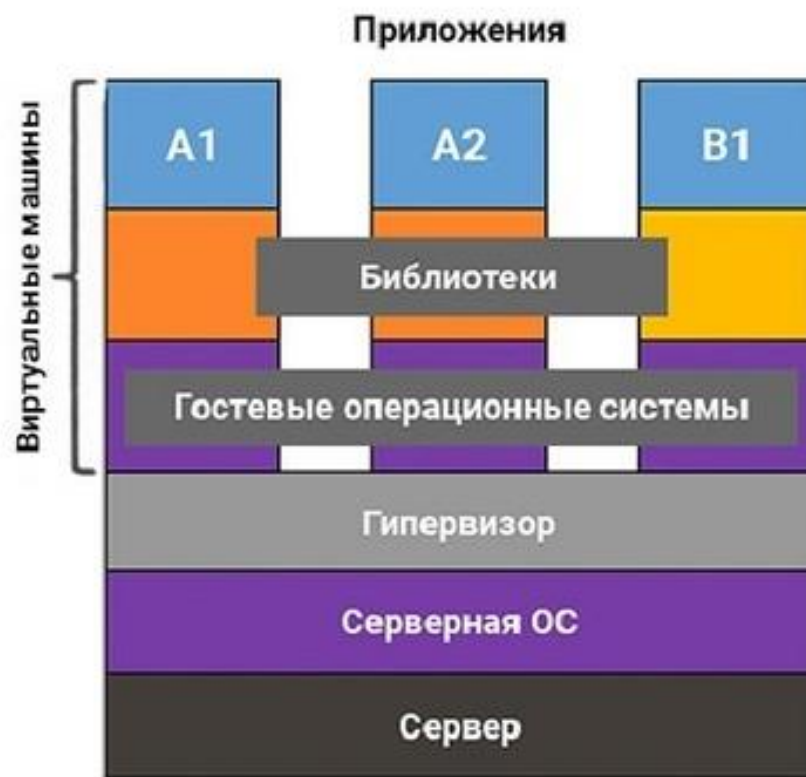
Restart=always

[Install]

WantedBy=multi-user.target

- 
- Запуск и добавление сервисов в автозагрузку:
`systemctl enable app1.service`
`systemctl start app1.service`
 - Отключение контроля доступа SELinux
`setsebool httpd_can_network_connect on -P`
 - Конфигурирование nginx для работы в роли обратного прокси и балансировщика (/etc/nginx/nginx.conf)

Контейнеризация





Основные понятия

- Образ - представляет собой специальную файловую систему. Помимо предоставления программ, библиотек, ресурсов и файлов конфигурации, необходимых для среды выполнения контейнера, он также содержит некоторые параметры конфигурации, подготовленные для среды выполнения (переменные среды, пользователи и т. д.)
- Контейнер - сущность образа времени выполнения. Отношения между изображением и контейнером аналогичны классам и экземплярам в объектно-ориентированном программировании. Изображение является определением, а контейнер - это объект, к которому относится изображение. Контейнеры могут быть созданы, запущены, остановлены, удалены, приостановлены и т. д.
- Репозиторий - место для централизованного хранения файлов изображений. После создания образа его можно легко запустить на текущем хосте, однако, если вам нужно использовать этот образ на других серверах, нам необходимо централизованное хранение и распространение образа.



Работа с образами

- `docker image pull IMAGE` – загрузка образа из репозитория
- `docker image push IMAGE` – загрузка образа в репозиторий
- `docker image history IMAGE` – история образа
- `docker image import TAR` – импорт tar-файла как содержимое файловой системы
- `docker image inspect IMAGE` – детальная информация об образе
- `docker image load TAR` – загрузка образа из tar-файла
- `docker image ls` – список образов в системе
- `docker image prune` – удаление неиспользуемых образов
- `docker image push IMAGE` – загрузка образа в репозиторий
- `docker image rm IMAGE` – удаление образа из системы
- `docker image save IMAGE` – сохранение образа в tar-файл
- `docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]` – создает тегированный образ
- `docker image build` - сборка образа по Dockerfile



Работа с контейнерами

- `docker ps` – просмотр списка контейнеров
- `docker logs CONTAINER` – просмотр логов контейнера
- `docker run IMAGE` – запуск контейнера
- `docker stop CONTAINER` – безопасно останавливает контейнер
- `docker kill CONTAINER` – небезопасная остановка контейнера
- `docker rm` – удаление контейнера из системы
- `docker exec CONTAINER COMAND` – выполнение команды в контейнере



Пример

- `docker pull mysql/mysql-server:latest`
- `docker run --name mysql-test -d \`
 `-p 3306:3306 \`
 `-e MYSQL_ROOT_PASSWORD=qwerty \`
 `-e MYSQL_DATABASE=test mysql/mysql-server:5.7`
- `docker exec -ti mysql-test mysql -uroot -pqwerty`

Работа с томами

- `docker volume create VOLUME` – создание тома
- `docker volume ls` – отображение списка томов в системе
- `docker volume inspect VOLUME` – просмотр информации о томе
- `docker volume rm VOLUME` – удаление тома из системы
- Запуск контейнера с томом:

```
docker run --name mysql-test -d \  
    -v db_volume:/var/lib/mysql \  
    -e MYSQL_ROOT_PASSWORD=qwerty \  
    -e MYSQL_DATABASE=test \  
mysql/mysql-server:5.7
```

Создание своего образа

- Dockerfile:

```
FROM ubuntu:22.04
```

```
RUN apt-get update -y
```

```
RUN apt-get install -y python3-pip libmysqlclient-dev
```

```
COPY . /app
```

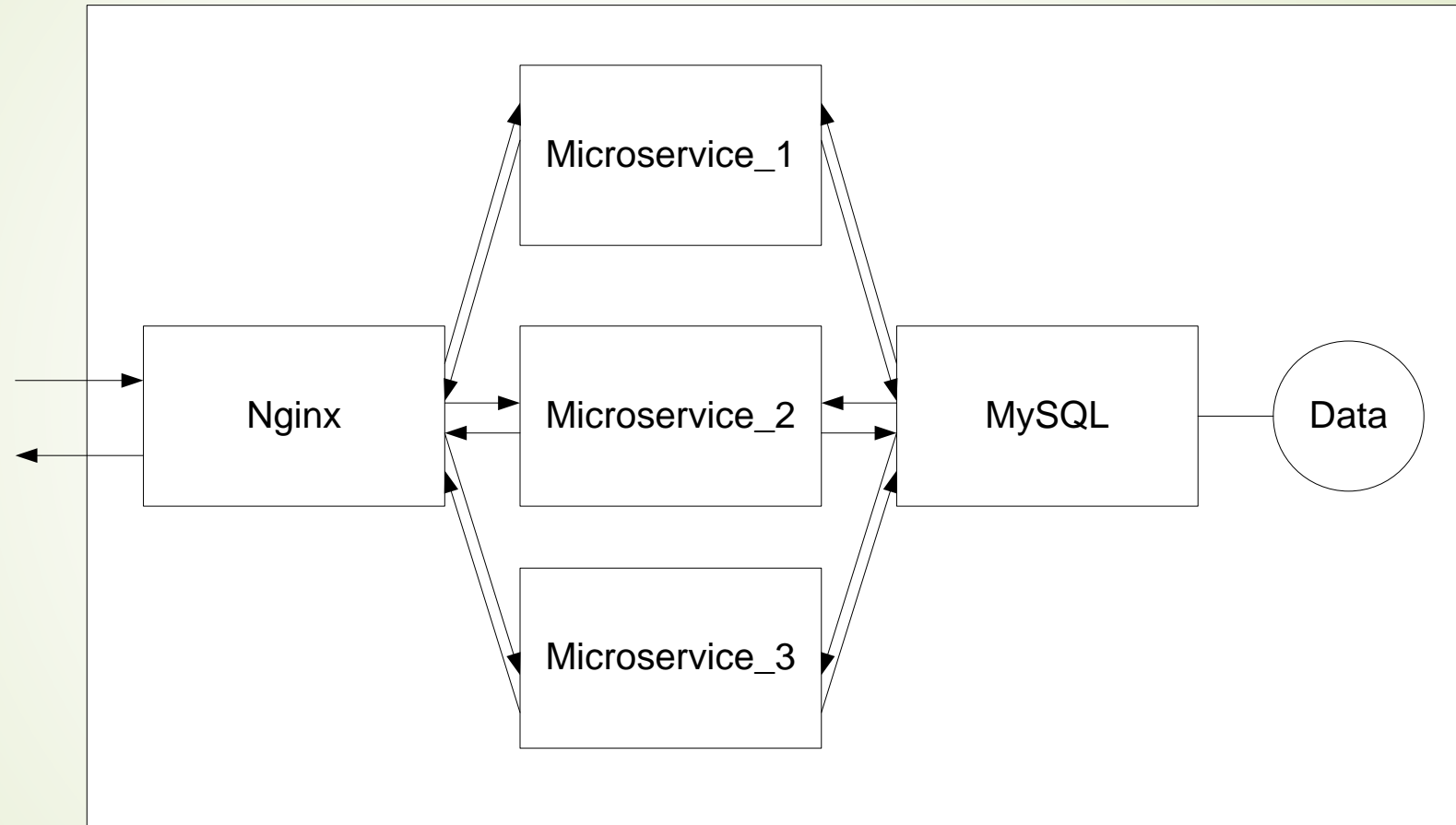
```
RUN pip3 install -r requirements.txt
```

```
ENTRYPOINT ['python3']
```

```
CMD ['run.py']
```

- Run build

```
docker build ./ -t microservice:1.0
```





Docker-compose

version: "3.3"

volumes:

db_test:

services:

db:

image: mysql/mysql-server:5.7

restart: always

environment:

- MYSQL_ROOT_PASSWORD=qwerty123
- MYSQL_DATABASE=test

volumes:

- db_test:/var/lib/mysql



microservice:

image: microservice:1.16

restart: always

depends_on:

- db

environment:

- LISTEN_PORT=8071

- DB_DRIVER=mysql

- DB_HOST=db

- DB_PORT=3306

- DB_USER=root

- DB_PASSWORD=qwerty123

- DB_NAME=test

- DB_CHARSET=utf8

loadbalancer:

image: test-nginx:1.18

restart: always

depends_on:

- microservice

ports:

- 80:80



Загрузка образов

`docker login`

`docker tag microservice:1.0 username/microservice:1.0`

`docker push username/microservice:1.0`

`docker pull username/microservice:1.0`

(или автоматически с помощью `docker-compose`)