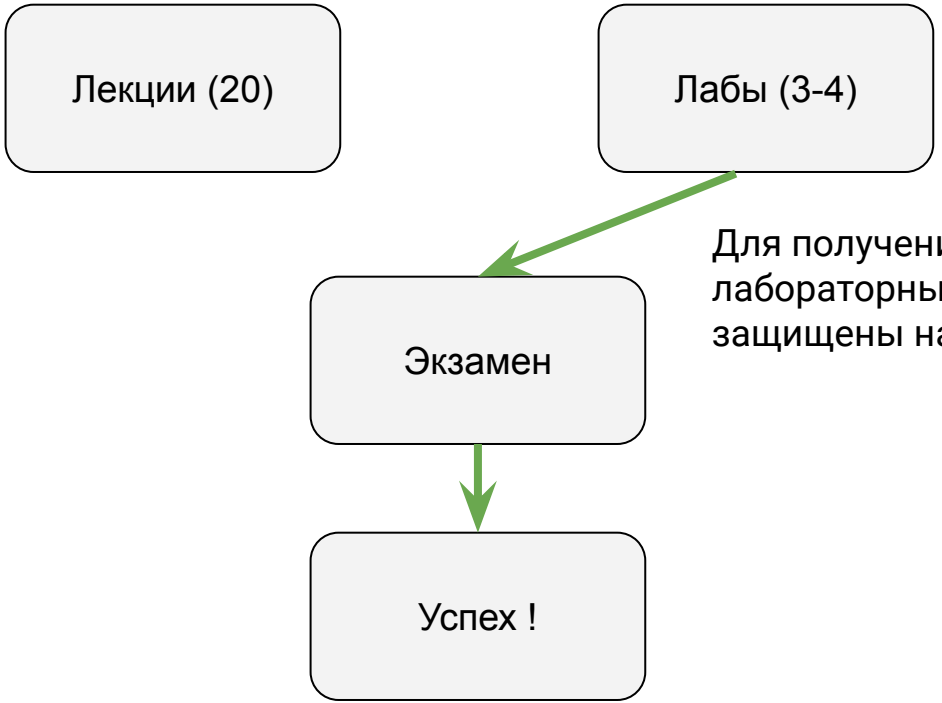
Инструменты и средства программирования

Бутома Виталий Сергеевич

vit.butoma@gmail.com
v.butoma@bsuir.by
t.me/PandaFromHell

Лекция 1. Введение.

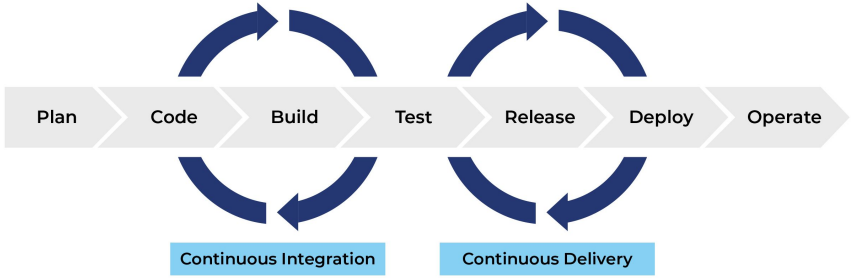
О курсе



Для получения допуска к экзамену все лабораторные должны быть выполнены и защищены на оценку 4+

Тематика курса

CI/CD



Git

Git - самая популярная распределенная система контроля версий.

Основное предназначение Git – это сохранение снимков последовательно улучшающихся состояний вашего проекта

Git не хранит и не обрабатывает данные как список изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Git. Базовые понятия

- репозиторий
- ветка (branch)
- коммит (commit)
- head
- pull
- push
- merge
- rebase
- stash
- reset

необходимый минимум !

Git. Установка и настройка

Варианты установки:

- через пакетный менеджер
- установщик, скачанный с официального сайта (https://git-scm.com/downloads)

Файлы настроек (зависит от OS):

/etc/gitconfig — Общие настройки для всех пользователей и репозиториев

~/.gitconfig или ~/.config/git/config — Настройки конкретного пользователя

.git/config — Настройки для конкретного репозитория

Git. Установка и настройка

Также настройка возможна через консольную команду *git config*

Используйте

```
git config name value
```

где name это название параметра, а value его значение, для того чтобы задать настройки.

Пример установки редактора по умолчанию nano:

```
git config --global core.editor nano
```

Git. Начало работы. Создание репозитория

В самом начале заходим на https://github.com (https://bitbucket.org/ , https://gitlab.com/) и создаем пустой репозиторий.

Git. Начало работы. Создание репозитория

```
vitaly@vitaly-machine:~/BSUIR/PythonISP$ mkdir iitp-python-2021
vitaly@vitaly-machine:~/BSUIR/PythonISP$ cd iitp-python-2021/
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git init
Инициализирован пустой репозиторий Git в /home/vitaly/BSUIR/PythonISP/iitp-python-2021/.git/
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git remote add origin git@github.com:vbutoma/iitp-python-2021.git
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git remote -v
origin  git@github.com:vbutoma/iitp-python-2021.git (fetch)
origin  git@github.com:vbutoma/iitp-python-2021.git (push)
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ echo "# iitp-python-2021" >> README.md
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git status
На ветке master

Еще нет коммитов

Неотслеживаемые файлы:
  (используйте «git add <файл>…», чтобы добавить в то, что будет включено в коммит)

        README.md

ничего не добавлено в коммит, но есть неотслеживаемые файлы (используйте «git add», чтобы отслеживать их)
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$
```

Git. Начало работы. Создание репозитория

- git init — инициализация репозитория
- git remote add origin …. — добавление ссылки на репозиторий
- git remote -v — вывод адресов для чтения и записи, привязанных к репозиторию
- echo "# iitp-python-2021" >> README.md — запись заголовка в файл README.md
- git status — проверка состояния файлов

Git. Начало работы. Первый коммит

```
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git add README.md
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git commit -m "First commit"
[master (корневой коммит) 80aed9f] First commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git status
На ветке master
нечего коммитить, нет изменений в рабочем каталоге
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git push
fatal: Текущая ветка master не имеет вышестоящей ветки.
Чтобы отправить текущую ветку и установить внешнюю ветку как вышестоящую для этой ветки, используйте

    git push --set-upstream origin master

vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git push --set-upstream origin master
Подсчет объектов: 3, готово.
Запись объектов: 100% (3/3), 231 bytes | 231.00 KiB/s, готово.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:vbutoma/iitp-python-2021.git
 * [new branch]      master -> master
Ветка «master» отслеживает внешнюю ветку «master» из «origin».
```

Git. Начало работы. Первый коммит

- git add README.md — индексируем файл
- git commit -m “First commit” — создаем коммит с сообщением

- git push — попытка запушить изменения. Не сработало, так как у текущей локальной ветки не настроена удаленная ветка
- git push --set-upstream origin master — установка удаленной ветки для текущей локальной

Git. Ветвление

Ветка в Git — это простой перемещаемый указатель на один из коммитов. Git хранит специальный указатель HEAD (указатель на текущую локальную ветку).

- git branch new_branch — создание новой ветки (но не переключаемся на нее)
- git checkout new_branch — переключаемся на ветку new_branch
- git checkout -b new_branch — создание новой ветки и переключение на нее

Git. Ветвление

```
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git branch
* master
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git branch develop
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git branch
  develop
* master
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git checkout develop
Переключено на ветку «develop»
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$ git branch
* develop
  master
vitaly@vitaly-machine:~/BSUIR/PythonISP/iitp-python-2021$
```

Git. Работа с удаленными ветками

Удалённые ссылки — это ссылки (указатели) в ваших удалённых репозиториях, включая ветки, теги и т.д..

- git checkout origin/new_branch — создание локальной ветки
- git fetch origin | git pull origin branch — получение всех последних изменений с удаленных веток
- git push origin branch — отправка изменений в удаленную ветку

Git. Слияние веток

- git checkout -b fix/bug_001 — создаем и переключаемся на новую ветку
- echo “print(‘fix’)” > main.py — создаем новый файл
- git add main.py && git commit -m “Added log” — добавляем в индекс и создаем коммит
- git checkout master — переключаемся в master ветку
- git merge fix/bug_001 — мержим ветки (fix/bug_001 -> master)
- git branch -d fix/bug_001 — удаляем ветку
- git checkout master — переключаемся в master ветку

При слиянии веток могут происходить конфликты. Для разрешения конфликтов можно использовать git mergetool. (Используйте десктопные версии с UI).

Git. Хорошие манеры при работе

Хорошие манеры:

- коммитьте понятные и цельные изменения
- коммитьте чаще, но не коммитьте недоделанную работу
- тестируйте ваш код перед коммитом
- пишите понятные сообщения к коммиту
- используйте git flow
- помечать релизы с помощью тегов (почитайте про semver)

Git. Полезные материалы

https://learngitbranching.js.org/ — интерактиваная обучалка ветвлению в гите

https://githowto.com/ru — отличный курс обучения гиту на русском с примерами

https://git-scm.com/book/ru/v2/ — pro git book

Docker

Где запускаются программы ?

Ядро и ОС

“Железо” — основа любого компьютера.

Железо (минимальный набор):

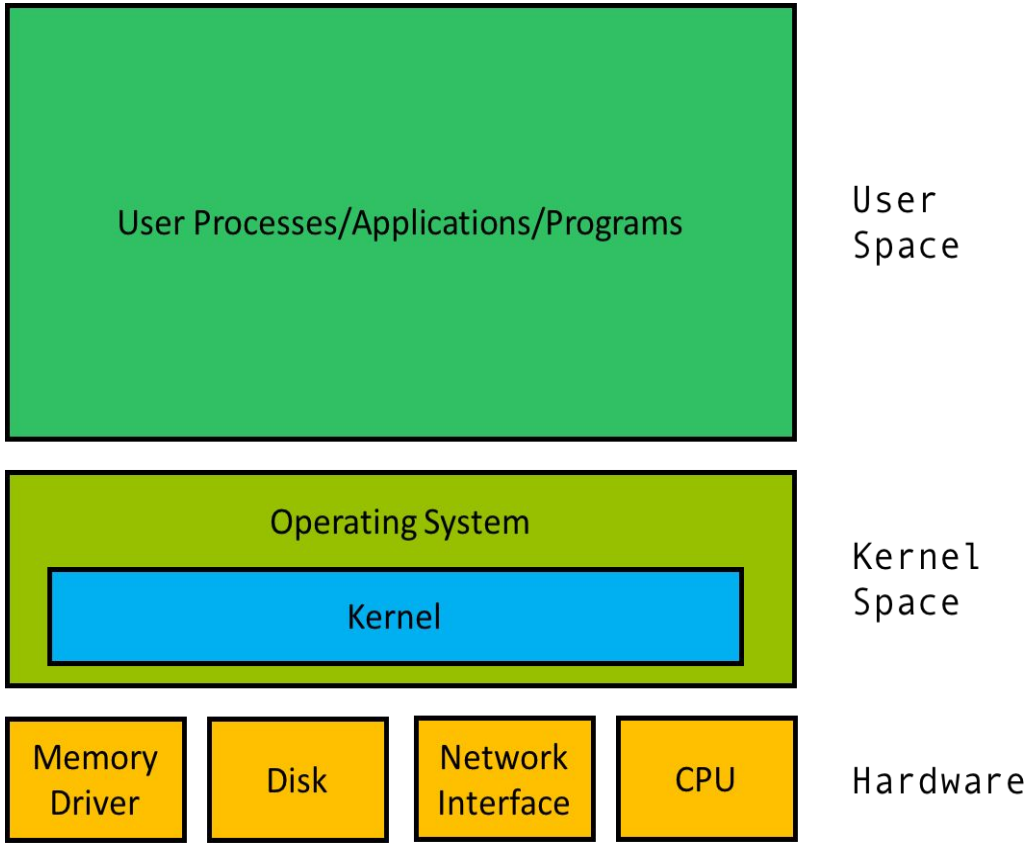
- процессор
- накопитель
- оперативка
- материнка

В ОС есть часть программного кода, которая служит мостом между софтом и железом, он называется — kernel (ядро). Ядро координирует запуск процессов (программ), управляет устройствами (чтение и запись адресов на диск и в память) и многое другое.

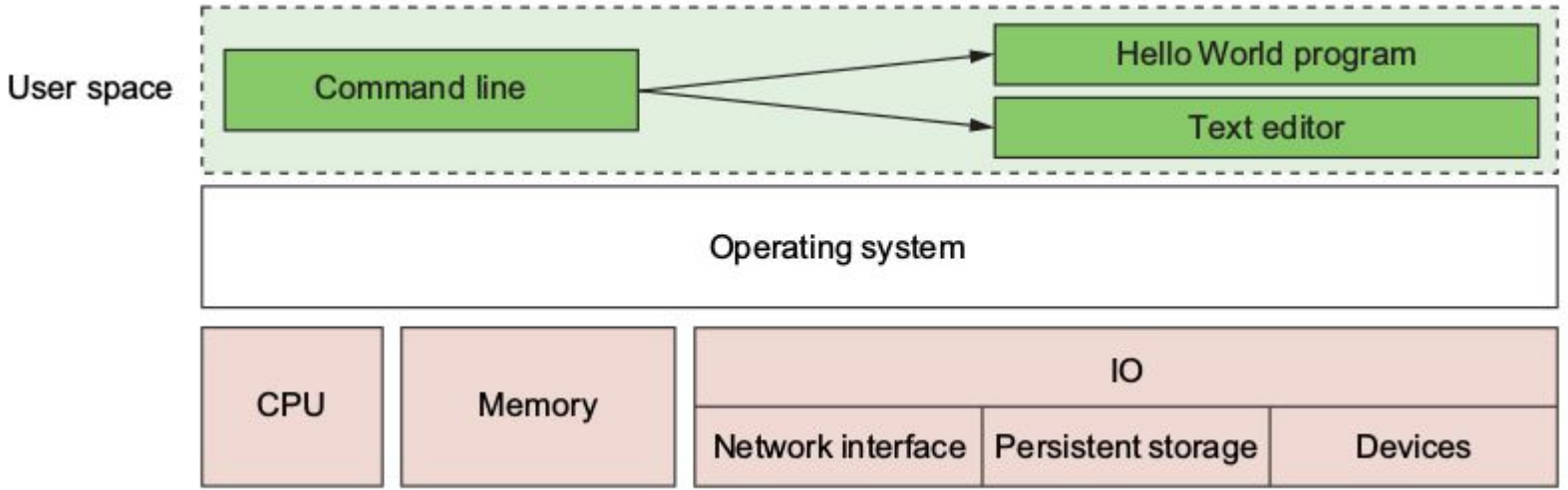
Остальная часть ОС служит для загрузки и управления пользовательским пространством, где запускаются и постоянно взаимодействуют с ядром процессы пользователя.

Ядро и ОС

Ядро является частью ОС и интерфейсом для железа. ОС «живёт» в “kernel space”, а программы пользователя в “user space”. Kernel space — управляет user space.



Как запускаются программы ?



Запуск двух программ из терминала

Docker

Docker - это платформа для создания, совместного использования и запуска приложений в контейнерах.

Docker

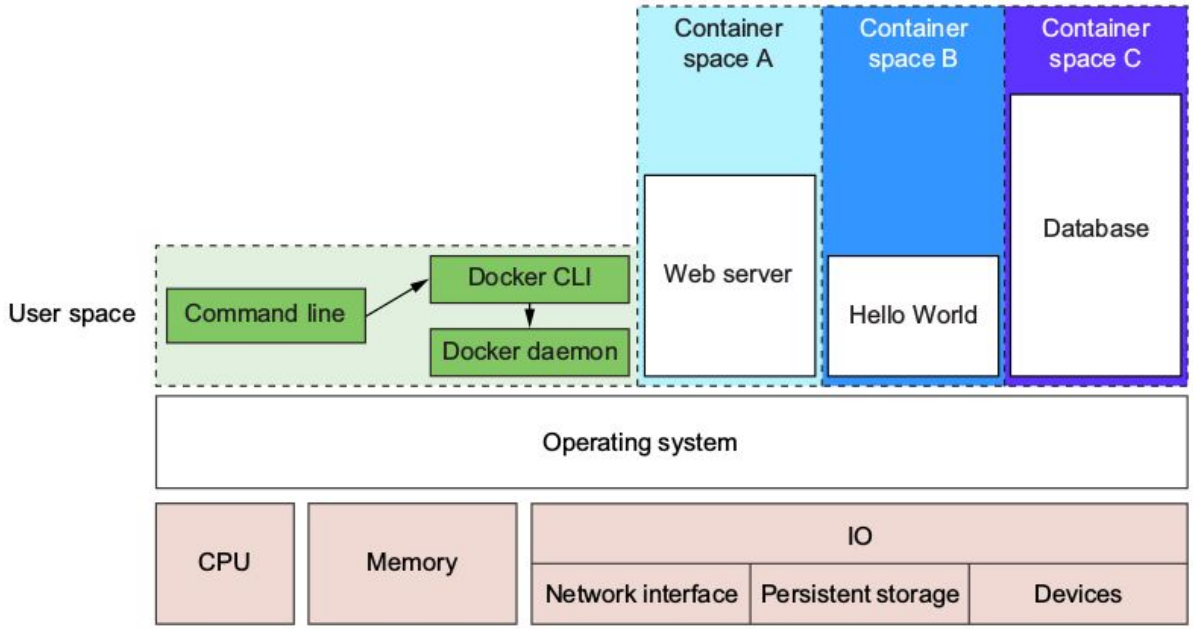
Docker состоит из командной строки, демона и набора удаленных сервисов.

Основные фичи контейнеров:

1) контейнер делит ядро хост-системы с другими контейнерами;
2) особый тип файловой системы.

Использование контейнеров для развертывания приложений называется **контейнеризацией**.

Docker



Запуск 3-х контейнеров

Как работает изоляция?

Cgroups — это аббревиатура от Linux “control groups”

Это функция ядра Linux, которая изолирует и контролирует использование ресурсов для пользовательских процессов. Её создали инженеры из Google в 2006 году.

Процессы могут быть помещены в пространства имён, то есть группы процессов, у которых общие ограниченные ресурсы. В компьютере может быть несколько пространств имен, у каждого из которых есть свойства ресурса, закрепленные ядром.

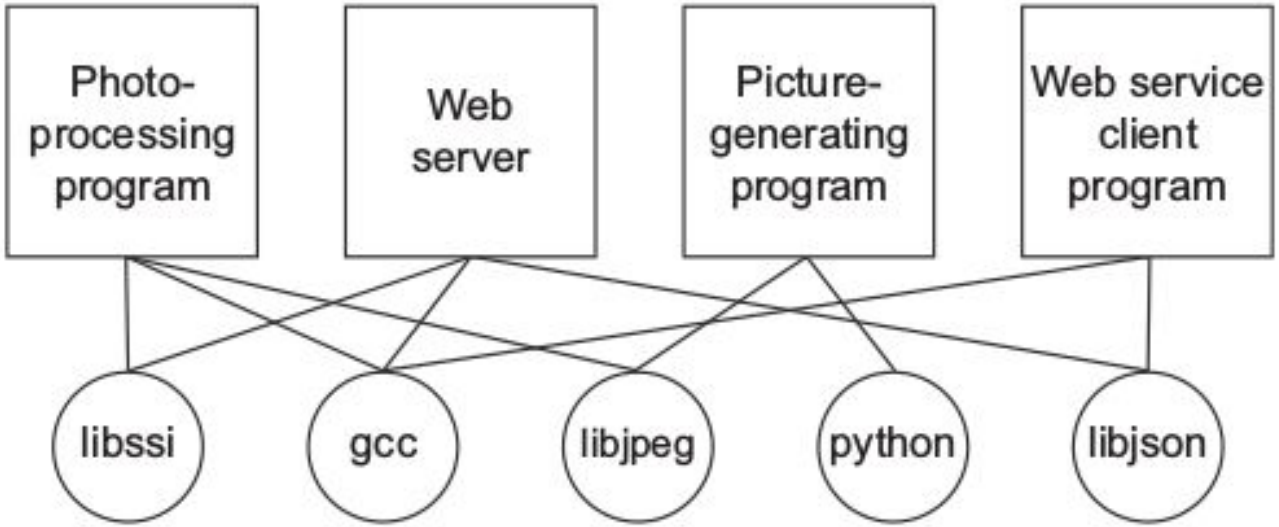
Как работает изоляция?

Докер контейнеры изолированы по следующим пунктам:

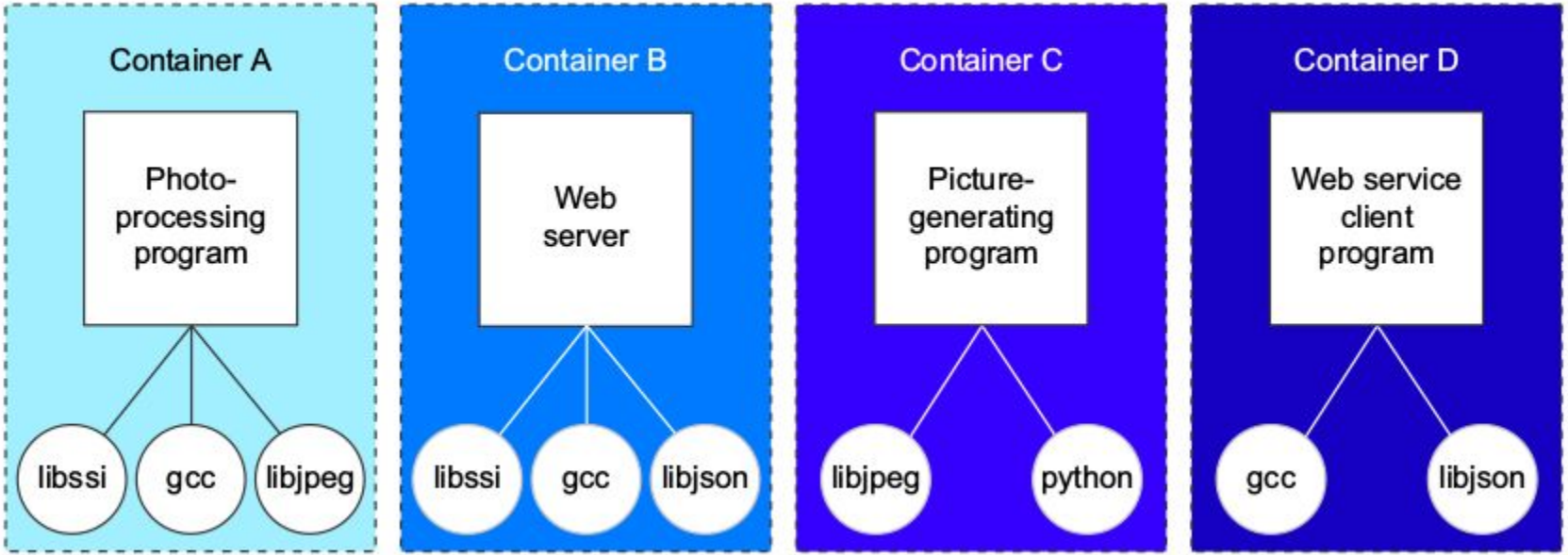
- PID namespace — идентификация процессов и из возможностей
- UTS namespace — хост и доменное имя
- MNT namespace — изоляция файловой системы
- IPC namespace — коммуникация процессов через разделяемую память
- NET namespace — изоляция контроллера сетевого интерфейса
- USR namespace — изоляция пользователей
- chroot() — контроль местоположения корня файловой системы
- cgroups — защита ресурсов

Какие проблемы решает контейнеризация ?

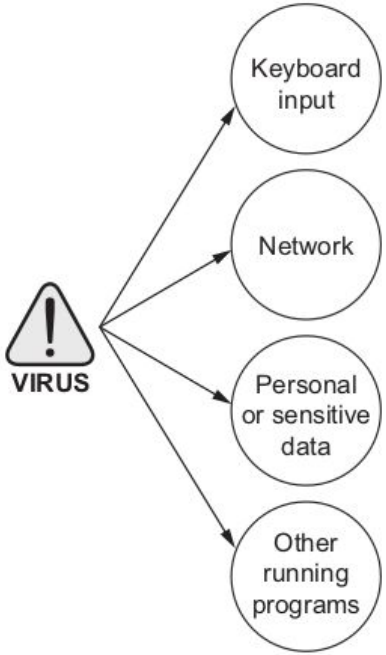
Множественные зависимости

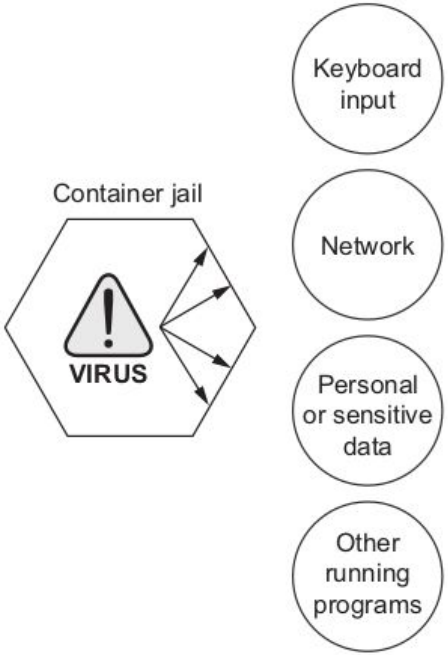


Множественные зависимости (решение)



Защита системы





Какие проблемы решает Docker:

- Улучшение портируемости приложений - возможность запуска практически везде
- Защита системы
- Организованность зависимостей - мы всегда будем знать что у нас стоит внутри контейнера

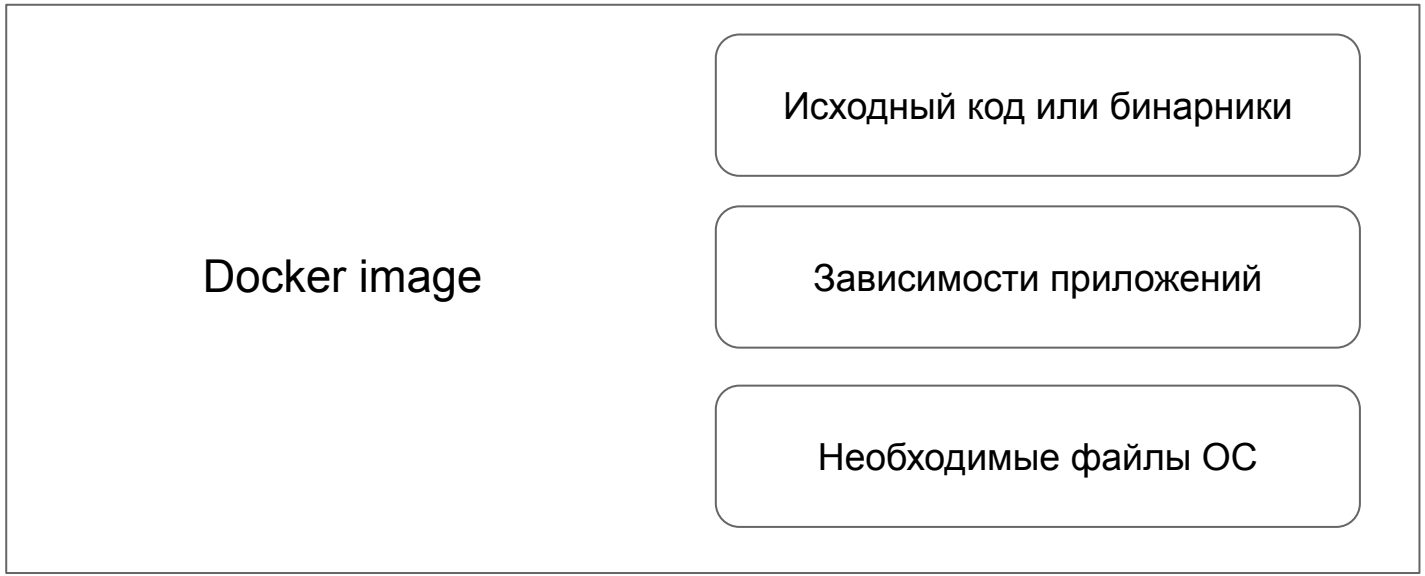
Плюсы контейнеризации

- Безопасность
- Легковесность
- Портируемость
- Масштабируемость
- Слабая связность

Docker. Создание образов (images)

Docker images

Для запуска контейнеров вначале необходимо собрать docker image (образ)



Docker images

Название docker образа состоит из разделенных слешем частей, где префиксом иногда может служить хост-имя репозитория с образами.

Хост-имя должно соответствовать стандартам DNS, однако не может содержать нижних подчеркиваний.

У каждого образа есть тэг.

Dockerfile - описание докер образа. Содержит в себе информацию о том, на базе какой ОС будет контейнер и описание необходимых зависимостей и библиотек.

Docker images


```
grafana/grafana                                 7.0.5                       89ec51554c46        7 months ago        159MB
istio/examples-bookinfo-reviews-v3              1.16.2                      83e6a8464b84        7 months ago        694MB
istio/examples-bookinfo-reviews-v2              1.16.2                      39cff5d782e1        7 months ago        694MB
istio/examples-bookinfo-reviews-v1              1.16.2                      181be23dc1af        7 months ago        694MB
istio/examples-bookinfo-ratings-v1              1.16.2                      99ce598b98cf        7 months ago        161MB
istio/examples-bookinfo-details-v1              1.16.2                      edf6b9bea3db        7 months ago        149MB
istio/examples-bookinfo-productpage-v1          1.16.2                      7f1e097aad6d        7 months ago        207MB
k8s.gcr.io/etcd                                 3.4.9-1                     d4ca8726196c        7 months ago        253MB
jaegertracing/all-in-one                        1.18                        0e2e4c273c1d        7 months ago        48.3MB
k8s.gcr.io/coredns                              1.7.0                       bfe3a36ebd25        7 months ago        45.2MB
prom/prometheus                                 v2.19.0                     39d1866a438a        8 months ago        142MB
k8s.gcr.io/metrics-server/metrics-server        v0.3.7                      07c9e703ca2c        9 months ago        55.4MB
datamolahub/nginxplus_lb                        r21-waf                     b3f37f3c009e        10 months ago       262MB
quay.io/coreos/flannel                          v0.12.0-amd64               4e9f801d2217        11 months ago       52.8MB
k8s.gcr.io/pause                                3.2                         80d28bedfe5d        12 months ago       683kB
store/ibmcorp/db2wh_ce                          v3.10.0-db2wh_client-linux  cf3e5951b1de        17 months ago       3.11GB
jimmidyson/configmap-reload                     v0.3.0                      7ec24a279487        18 months ago       9.7MB
```

Docker images. Dockerfile

Пример Dockerfile:

```
FROM node:10

WORKDIR /usr/src/app

COPY package.json .

# Run the command inside your image filesystem.
RUN npm install

# Inform Docker that the container is listening on the specified port at runtime.
EXPOSE 3000

# Run the specified command within the container.
CMD node app.js

# Copy the rest of your app's source code from your host to your image filesystem.
COPY . .
```

Docker images. Dockerfile

Подробно про структуру Dockerfile можно почитать в официальной документации:

https://docs.docker.com/engine/reference/builder/

Docker images. Build.

Общий вид команды сборки:

docker build [options] path | url

options - все опции доступны по https://docs.docker.com/engine/reference/commandline/build/
path - путь к контексту сборки
url - url по которому находится контекст сборки

Docker images. Build.

Соберем наш первый образ!

```
docker build -t web-app:1.0.0
```

После успешной сборки затегируем образ и зальем в удаленный репозиторий

```
docker tag web-app:1.0.0 your-registry/web-app:1.0.0
```

```
docker push your-registry/web-app:1.0.0
```

Docker images. Build.

После пуша в dockerhub можно спокойно удалять локальный образ, если он вам не нужен

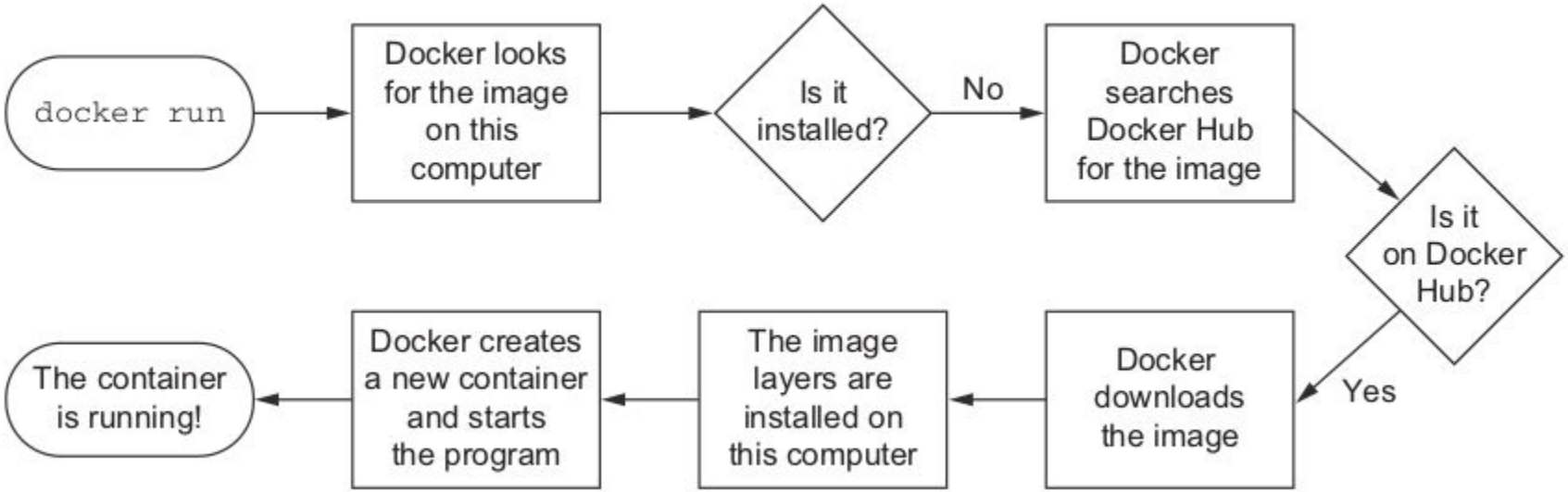
```
docker rm you-registry/web-app:1.0.0
```

Если же он понадобится вам снова, то можно его скачать следующей командой:

```
docker pull your-registry/web-app:1.0.0
```

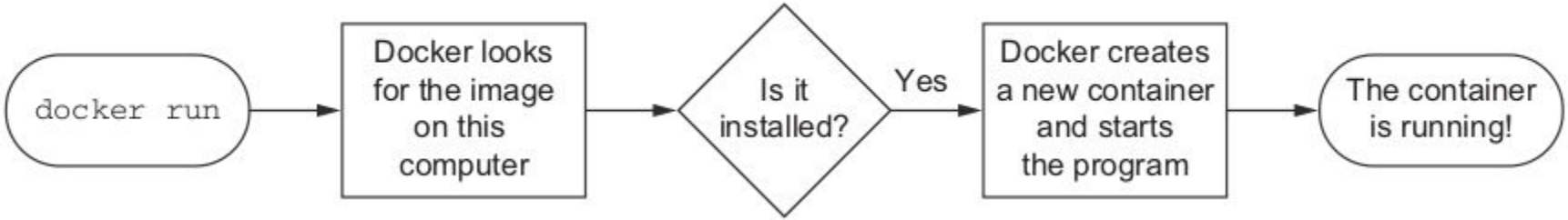
Docker. Запуск контейнеров

Docker. Запуск контейнеров



Запуск контейнера, образ которого локально отсутствует

Docker. Запуск контейнеров



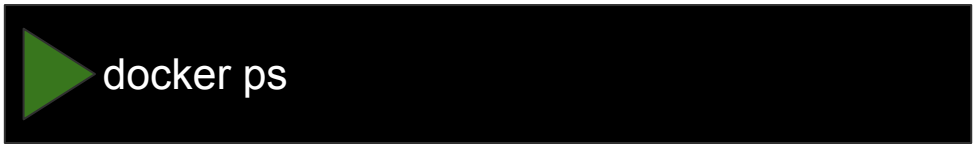
Запуск контейнера, образ которого локально присутствует

Docker. Запуск контейнеров


```
vitaly@vitaly-machine:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
vitaly@vitaly-machine:~$ docker run -d -p 3000:3000 --name web web-app:1.0.0
58ff2ea38c4583223e986546624144e82befc45b2165f27c2fcfca58be8b032f
vitaly@vitaly-machine:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
58ff2ea38c45        web-app:1.0.0       "docker-entrypoint.s…"   15 seconds ago      Up 14 seconds       0.0.0.0:3000->3000/tcp   web
vitaly@vitaly-machine:~$
```

Docker. Запуск контейнеров

Для вывода статуса запущенных контейнеров используется команда:



1) ID контейнера
2) Используемый образ
3) Команда запуска контейнера
4) Время от создания контейнера
5) Время работы контейнера
6) Открытые порты
7) Имя

Docker. Запуск интерактивных контейнеров

```
docker run -ti --name web-interactive web-app:1.0.0 bash
```

В команде выше есть два флага: --interactive (или -i) и --tty (или -t).

--interactive опция говорит Docker движку оставить стандартный поток ввода (stdin) открытым в контейнере даже если терминал не присоединен;

--tty опция говорит Docker движку выделить виртуальный терминал для контейнера, который позволит отправлять сигналы внутрь контейнера.

Docker. Запуск интерактивных контейнеров

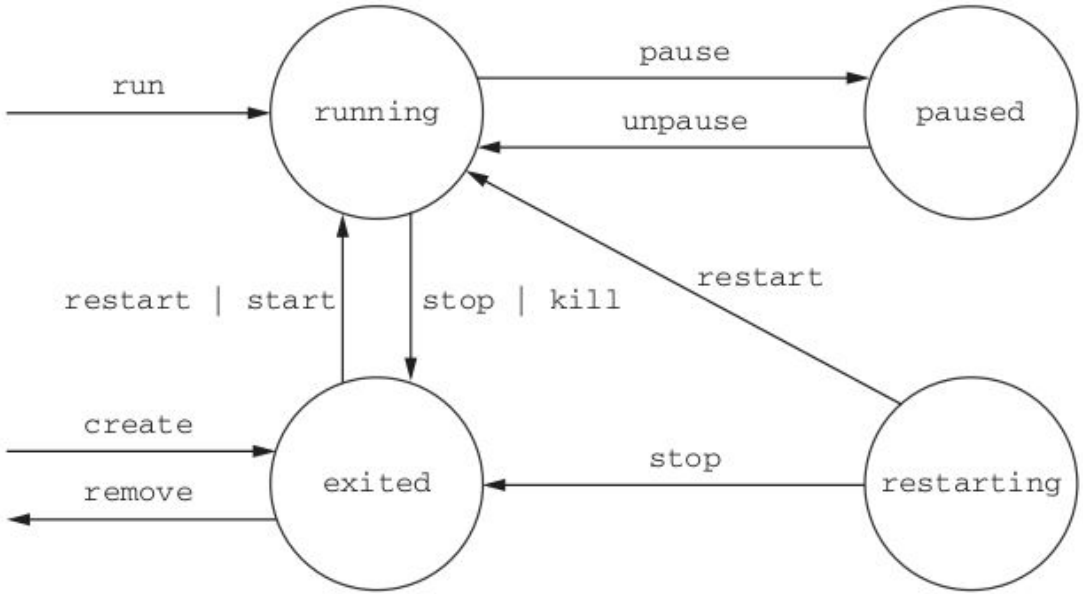

```
vitaly@vitaly-machine:~$ docker run -ti --name web-interactive web-app:1.0.0 bash
root@c6a147f27757:/usr/src/app# node -v
v10.20.1
root@c6a147f27757:/usr/src/app# npm -v
6.14.4
root@c6a147f27757:/usr/src/app# uname -a
Linux c6a147f27757 4.15.0-112-generic #113-Ubuntu SMP Thu Jul 9 23:41:39 UTC 2020 x86_64 GNU/Linux
root@c6a147f27757:/usr/src/app#
```

Docker. Рестарт контейнера

docker restart name

```
vitaly@vitaly-machine:~$ docker restart web
web
vitaly@vitaly-machine:~$
```

Docker. Жизненный цикл контейнера



Pause vs STOP

docker pause посылает SIGSTOP сигнал всем процессам в контейнере

docker stop же посылает SIGTERM сигнал главному процессу в контейнере (PID=1) и спустя какое-то время SIGKILL.

SIGTERM — сигнал завершения. По умолчанию используется чтобы завершить процесс, но они иногда могут быть проигнорированы. Его необходимо обрабатывать, если важно провести очистку используемых ресурсов.

SIGKILL — сигнал принудительного завершения. Используется для незамедлительного завершения процесса. Отсюда следует, что о никакой очистке ресурсов и речи быть не может.

SIGSTOP — сигнал паузы. Сигнал не может быть отловлен и проигнорирован приложением. Используется для контроля над приложениями.

Docker. Просмотр логов контейнера

Для просмотра логов необходимо знать или имя, или идентификатор контейнера

```
docker logs name | id
```

```
vitaly@vitaly-machine:~$ docker logs web
Server running on http://0.0.0.0:3000
Server running on http://0.0.0.0:3000
vitaly@vitaly-machine:~$
```

Docker. Просмотр логов контейнера

Также есть несколько удобных опций. Например:

1) --follow, -f - логи в реальном времени;
2) --timestamps, -t -показывать время (timestamp) перед каждой строчкой лога;
3) --tail - количество строк для вывода (по умолчанию - все).


```
vitaly@vitaly-machine:~$ docker logs web -ft
2020-08-18T15:06:21.673833608Z Server running on http://0.0.0.0:3000
2020-08-18T15:24:31.8944666687Z Server running on http://0.0.0.0:3000
```

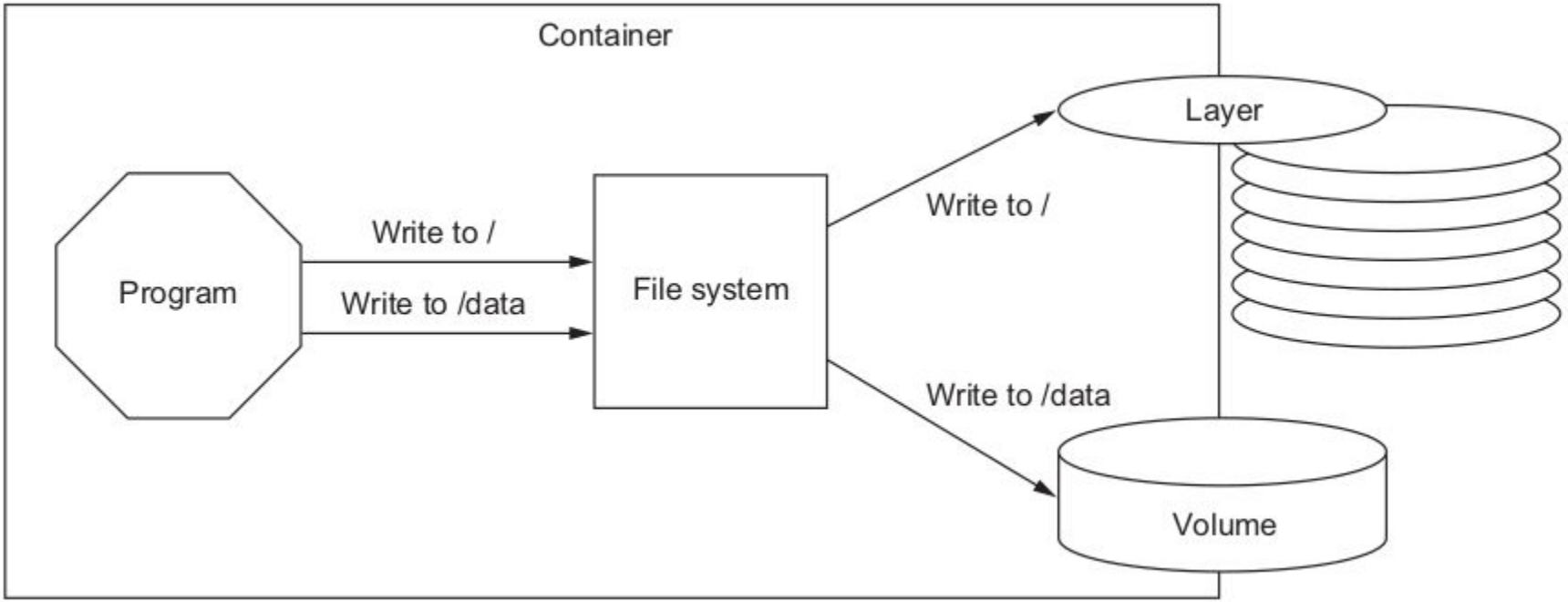
Docker. Хранилища данных

Docker. Данные в контейнерах

По умолчанию все файлы, которые создаются в контейнере, хранятся в специальном записывающем слое контейнера. Это значит:

1) Данные не буду существовать без контейнера, и данные будет очень сложно найти, если они понадобятся другому процессу;
2) Записывающий слой тесно связан с хост системой. Переместить эти данные куда-то будет непросто;
3) Для записи в этот слой необходимы специальные драйвера. Драйвер для хранилища предоставляет объединенную файловую систему, используя ядро линукс. Данный дополнительный слой абстракции замедляет производительность.

Docker. Данные в контейнерах



Docker. Типы volume

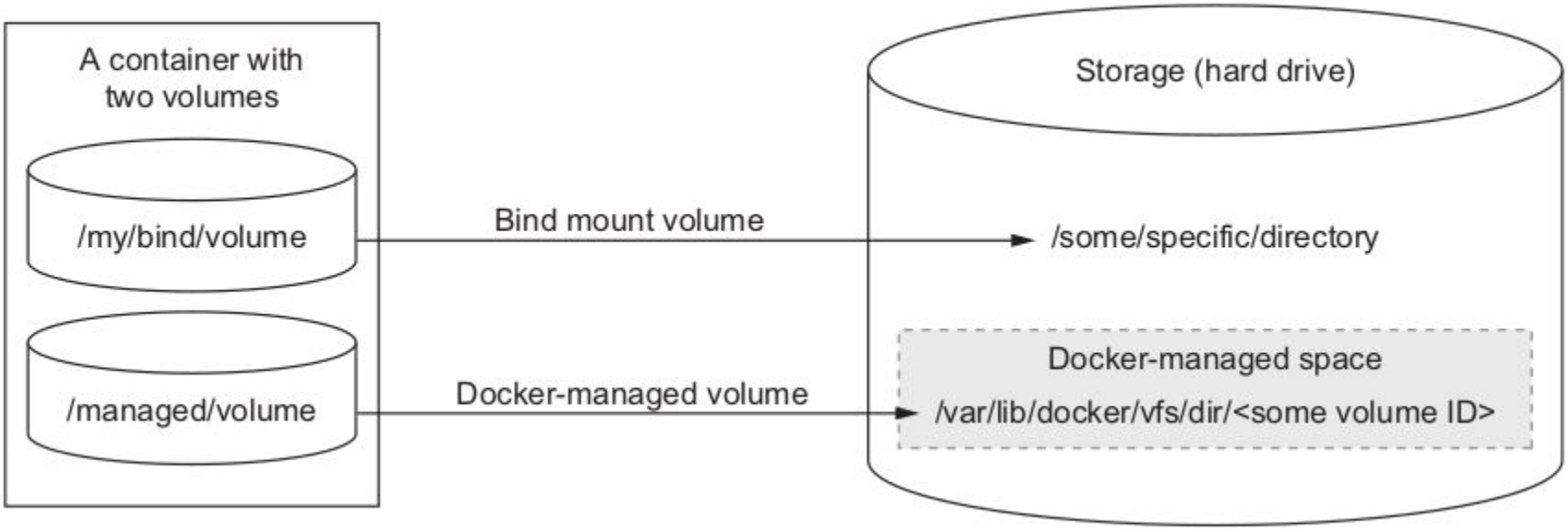
2 типа:

1) volumes;
2) mount binds;

Также, если вы работаете на Linux, то можно использовать tmpfs mount. Если на windows, то можно использовать именованные потоки.

Рекомендуется использовать 1-й тип, то есть **VOLUME**.

Docker. Типы volume



Docker. Volumes

Преимущества Volumes над mount binds:

1) У volume проще создать резервную копию или переместить ее;
2) управлять docker volumes можно через docker CLI и docker API;
3) docker volumes работают как на linux, так и на windows;
4) более безопасный шаринг volumes между контейнерами;
5) для volumes можно использовать различные драйверы, которые позволяют хранить volumes на удаленных машинах или в облаке, шифровать их или предоставляют другую функциональность;
6) Новые volumes могут презаполняться контейнером при старте (удобно, что не надо выдавать кучу лишних прав).

Docker. Volumes

Пример создания volume.

Список всех volumes:


```
vitaly@vitaly-machine:~$ docker volume create test
test
vitaly@vitaly-machine:~$ docker volume ls
DRIVER              VOLUME NAME
local               test
vitaly@vitaly-machine:~$
```

Docker. Volumes

Пример создания docker volume и запуска контейнера для хранения логов нашего приложения.

docker volume create app-logs

docker volume ls

```
vitaly@vitaly-machine:~$ docker volume create app-logs
app-logs
vitaly@vitaly-machine:~$ docker volume ls
DRIVER              VOLUME NAME
local               app-logs
vitaly@vitaly-machine:~$
```

docker run -v app-logs:/usr/src/app/log ....

Вопросы ?

Спасибо за внимание