

Docker



..T..Systems.....

Saint Petersburg, 2018

СЕГОДНЯ В ПРОГРАММЕ

- 1 ВИРТУАЛИЗАЦИЯ
- 2 КОНТЕЙНЕРИЗАЦИЯ
- 3 АРХИТЕКТУРА DOCKER
- 4 СЕТЬ И ФАЙЛОВАЯ СИСТЕМА
- 5 DOCKER В РАЗРАБОТКЕ ПО
- 6 СОЗДАНИЕ ОБРАЗОВ
- 7 КЛАСТЕРИЗАЦИЯ
- 8 МИКРОСЕРВИСЫ И DOCKER

...T...Systems.....

Виртуализация — предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию друг от друга вычислительных процессов, выполняемых на одном физическом ресурсе.

ЦЕЛИ ВИРТУАЛИЗАЦИИ

- **Повышение изоляции**

- Ограничение одной или группы тесно связанных служб собственной виртуальной машиной;

- **Безопасность**

- Распределение задач администрирования — возможность ограничить права каждого администратора только самыми необходимыми;
- Снижение потенциальных вредных последствий взлома какой-либо из служб.

- **Распределение ресурсов** — каждая машина получает столько ресурсов, сколько ей необходимо, но не более того.

- Выделение памяти по требованию;
- Гибкое распределение сетевого трафика между машинами;
- Распределение дисковых ресурсов;

- **Постоянная доступность**

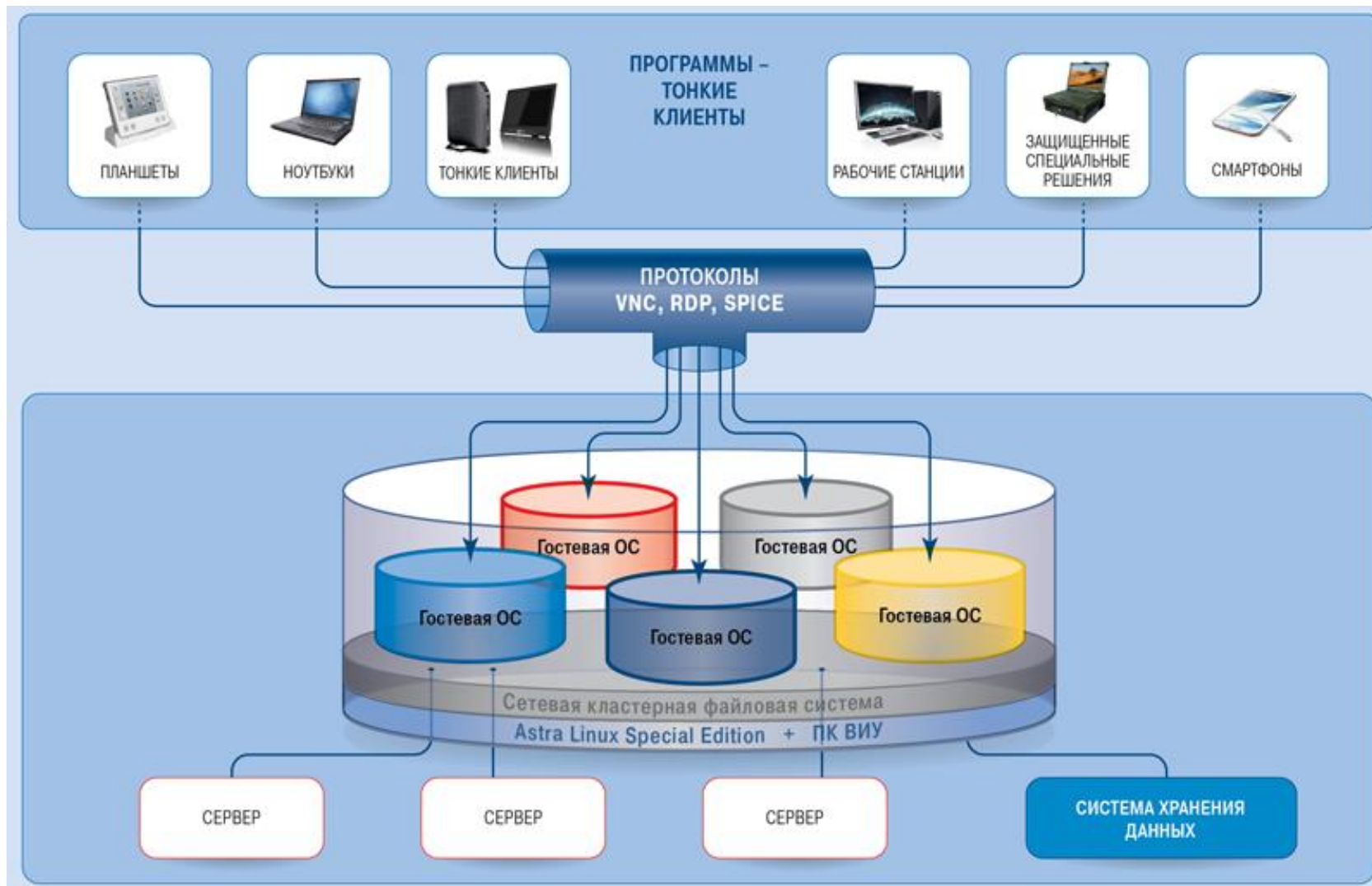
- Есть возможность live-миграции машин;
- Плавный апгрейд критических серверов.

- **Повышение качества администрирования**

- Возможность выполнения регрессионных тестов;
- Возможность экспериментирования и исследования.

.. **T** .. **Systems** ..

ВИРТУАЛИЗАЦИЯ



.. T .. Systems ..

- Аппаратная виртуализация
- Виртуализация на уровне операционной системы
- Виртуальные машины
- Виртуализация ресурсов
- Виртуализация приложений

ВИРТУАЛИЗАЦИЯ

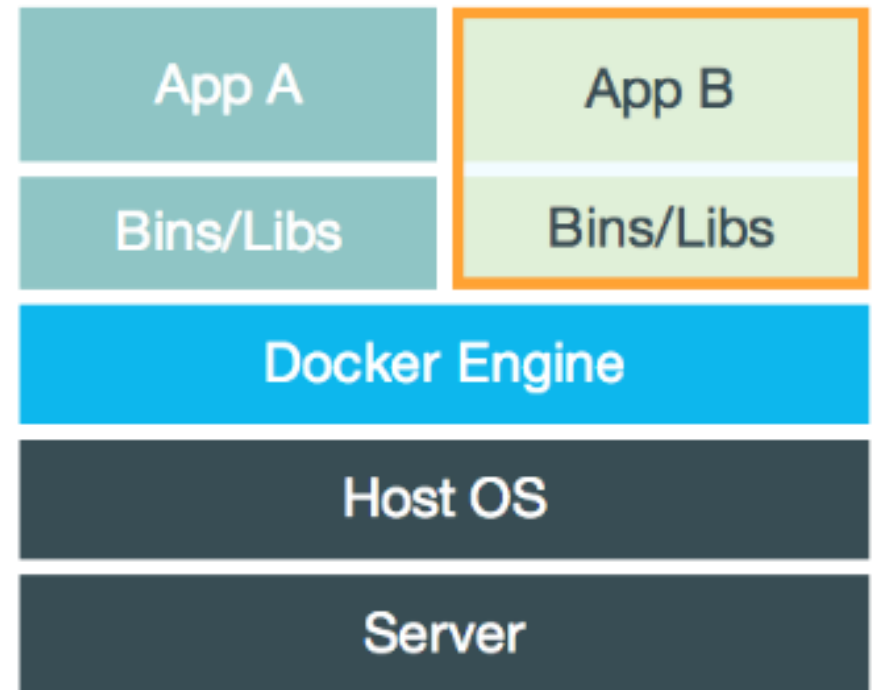
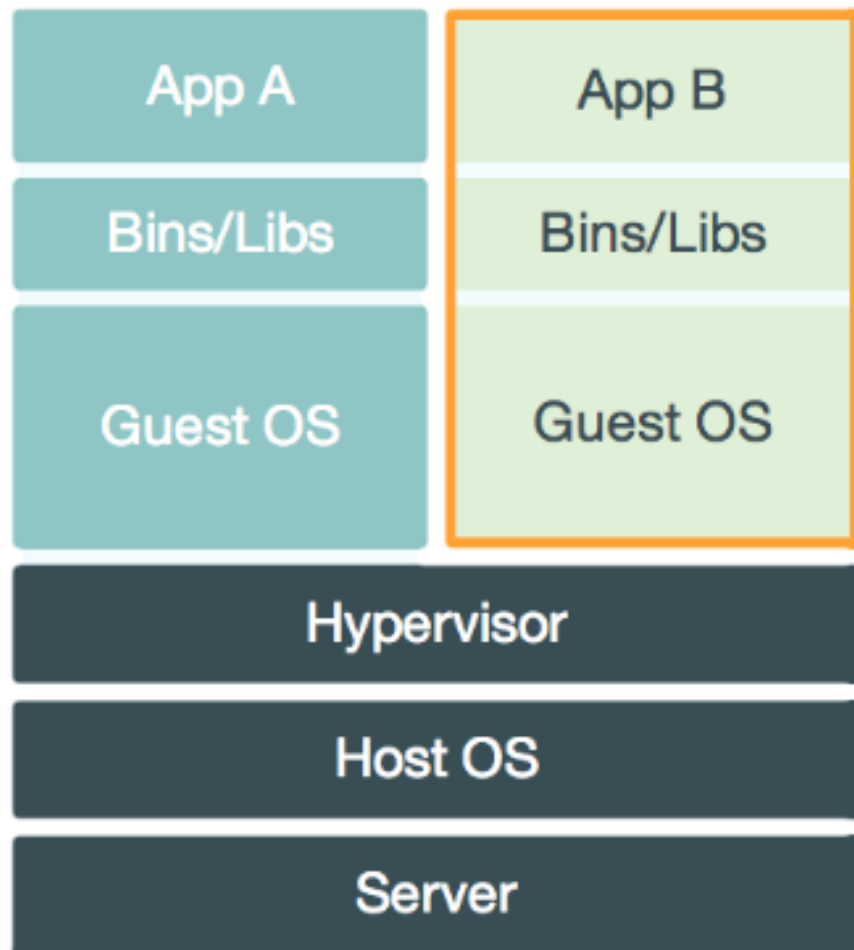


.. T .. Systems ..

Виртуализация на уровне операционной системы — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя, вместо одного. Эти экземпляры (часто называемые *контейнерами* или *зонами*) с точки зрения пользователя полностью идентичны реальному серверу.

Контейнерезация— легковесные механизмы виртуализации в пользовательском пространстве (виртуализация на уровне операционной системы), которые применяют контрольные группы и пространства имен, чтобы управлять изолированием ресурсов.

Docker vs VMs



...T...Systems...

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.



.. T .. Systems ..

Зачем нужен Docker?

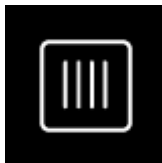
- Масштабирование
- Управление версиями и зависимостями
- Изолирование среды
- Использование слоев
- Компоновка

...T...Systems.....



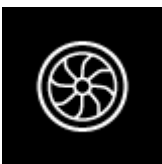
Docker Image

Упакованная версия приложения вместе с зависимостями и необходимым окружением необходимым для запуска



Docker Container

Это запущенный экземпляр образа



Docker Engine

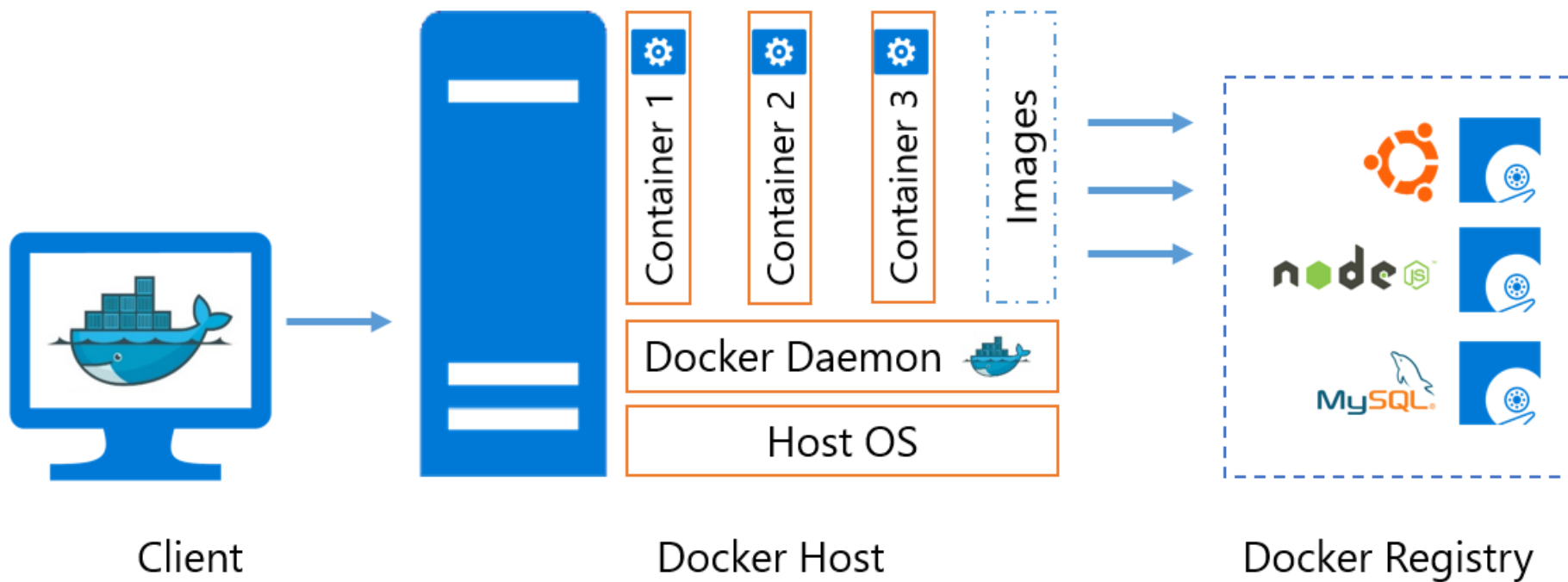
Центральная часть докера, демон работающий на хост-машине и умеющий скачивать и заливать образы, запускать из них контейнеры, следить за запущенными контейнерами, собирать логи и настраивать сеть между контейнерами



Registry Service (Docker Hub or Docker Trusted Registry)

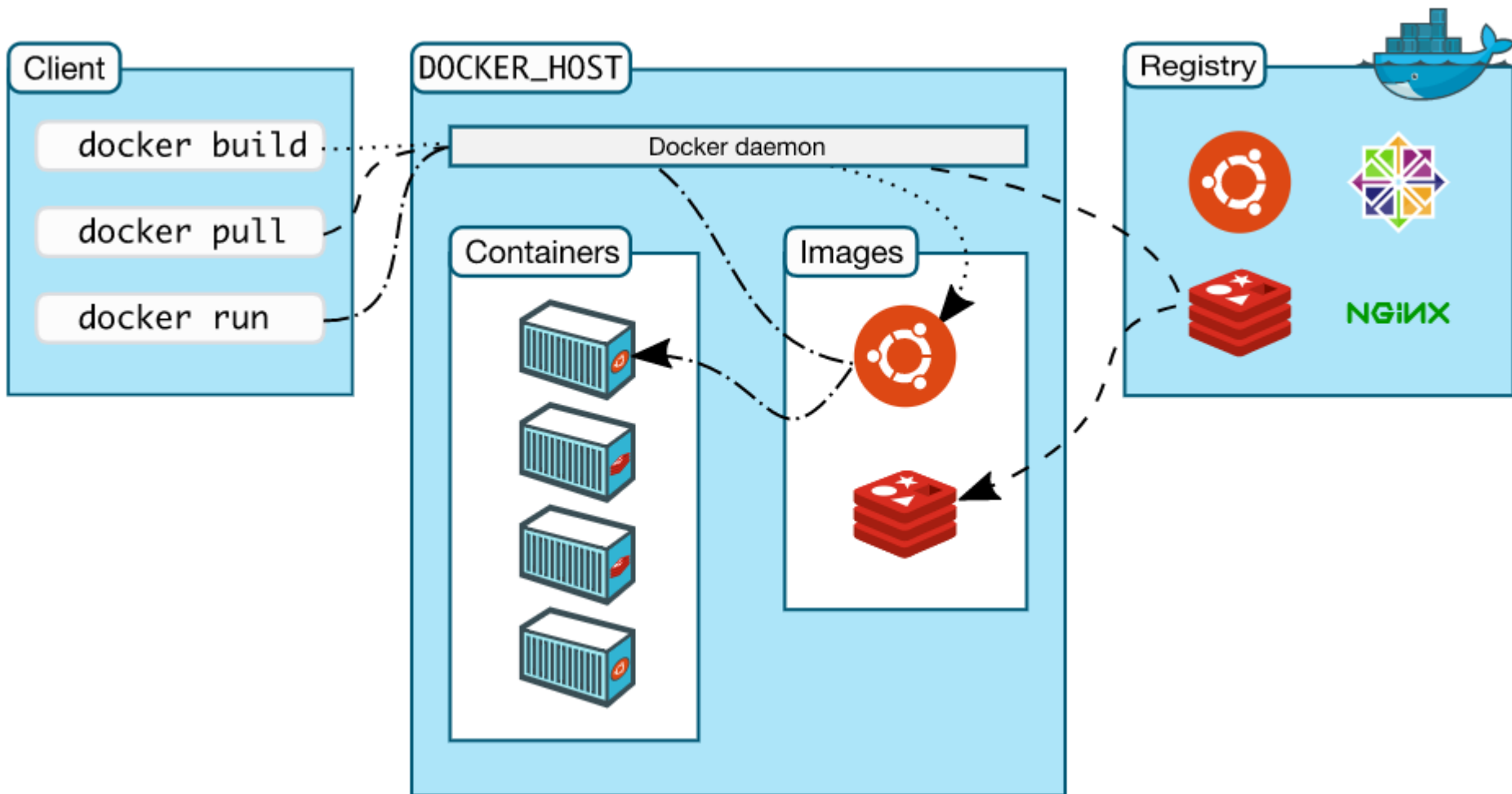
Сервер для хранения образов

Зачем нужен Docker?



...T...Systems.....

Жизненный цикл



... T ... Systems ...

Использование Docker CLI

Команда	Описание
<code>docker images</code>	Список образов на локальной машине
<code>docker ps</code>	Список запущенных контейнеров на локальной машине
<code>docker ps -a</code>	Список всех контейнеров на локальной машине
<code>docker run <image></code>	Запуск контейнера из образа
<code>docker run --name="Some Name <image></code>	Запуск контейнера с заданным именем
<code>docker run -d <image></code>	Запуск контейнера из образа в демон моде
<code>docker run -it <container></code>	Запуск контейнера в интерактивном моде
<code>docker run -p 8080:80</code>	Редирект 80 порта контейнера на 8080 порт хоста
<code>docker start <container></code>	Запуск существующего контейнера
<code>docker stop <container></code>	Завершение работы контейнера

Использование Docker CLI

Команда	Описание
<code>docker rm <containerid></code>	Удаление контейнера
<code>docker rm `docker ps -a -q`</code>	Удаление всех контейнеров
<code>docker rmi <image></code>	Удаление образа
<code>docker rmi -r <image></code>	Удаление образов если есть их запущенные контейнеры
<code>docker exec -it <name> <command></code>	Выполнение команд на запущенном образе
<code>docker pull repository/image</code>	Скачивание образа из репозитория

Code time

..T..Systems.....

Задание #1

- 1) Запустить образ `parkito/hellodocker`
- 2) Запустить контейнер, используя `run` без параметров . Сколько раз встречается слово `Java` и `Python` в выводе?
- 3) Используя интерактивный режим воспользуйтесь `ash`-интерпритатором для доступа к внутреннему каталогу `/var/lib/app`.
- 4) Какие файлы вы найдёте в `/var/lib/app`?

Преимущества Docker

Докер — универсальный способ доставки приложений на машины (локальный компьютер или удаленные сервера) и их запуск в изолированном окружении.

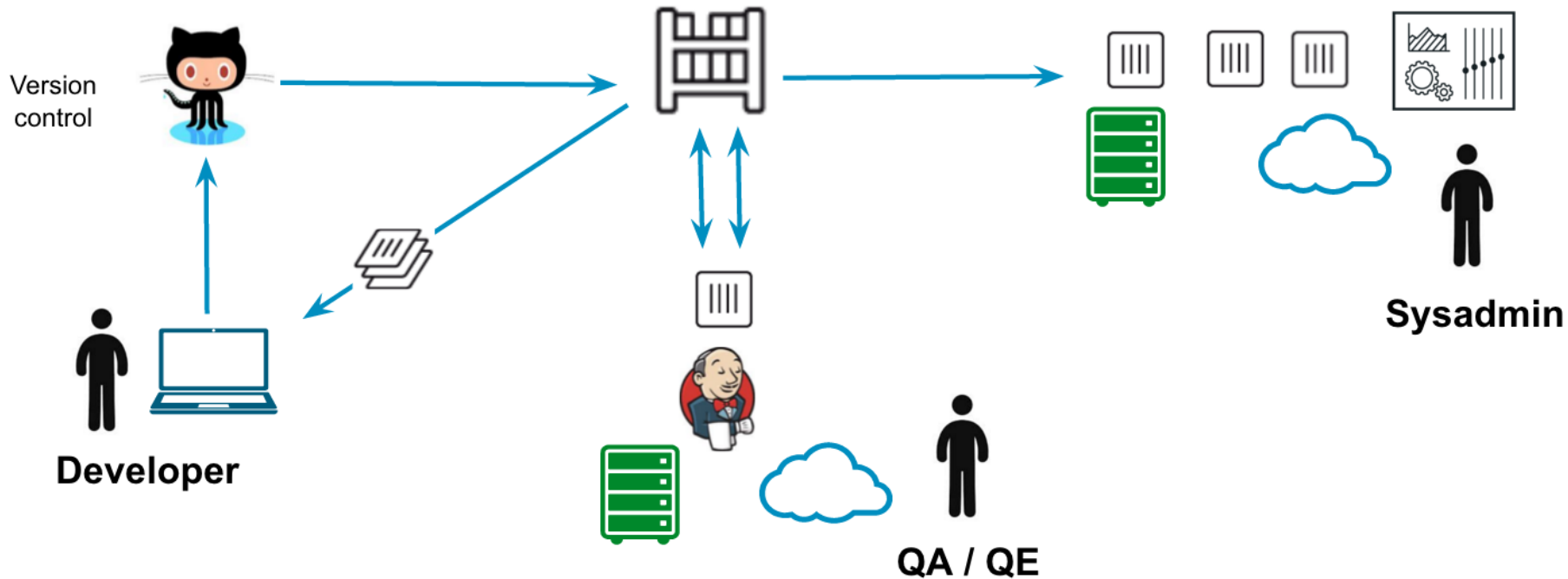
- Абстрагирование хост-системы от контейнеризованных приложений
- Простота масштабирования
- Простота управления зависимостями и версиями приложения
- Чрезвычайно легкие, изолированные среды выполнения
- Совместно используемые слои
- Возможность компоновки и предсказуемость

Docker workflow

1. Development

2. Test

3. Stage / Production

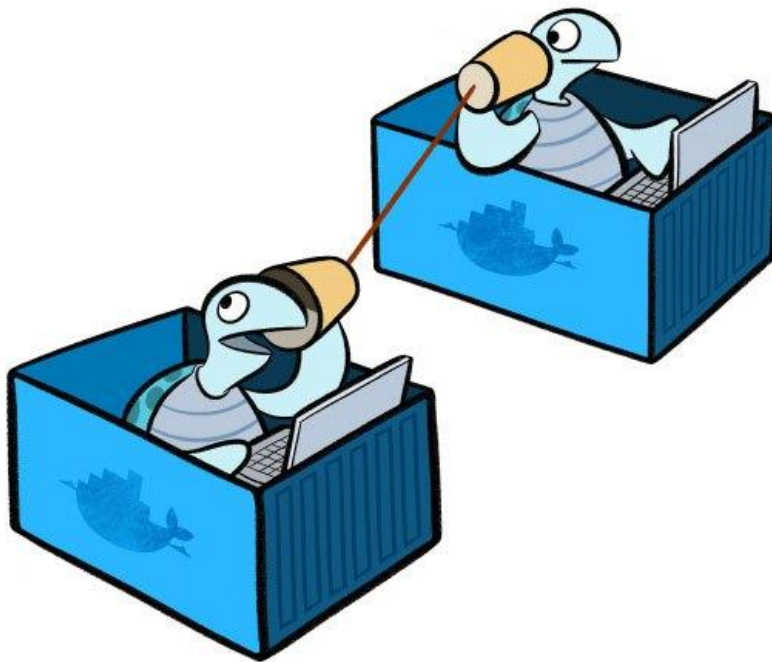


... T ... Systems ...

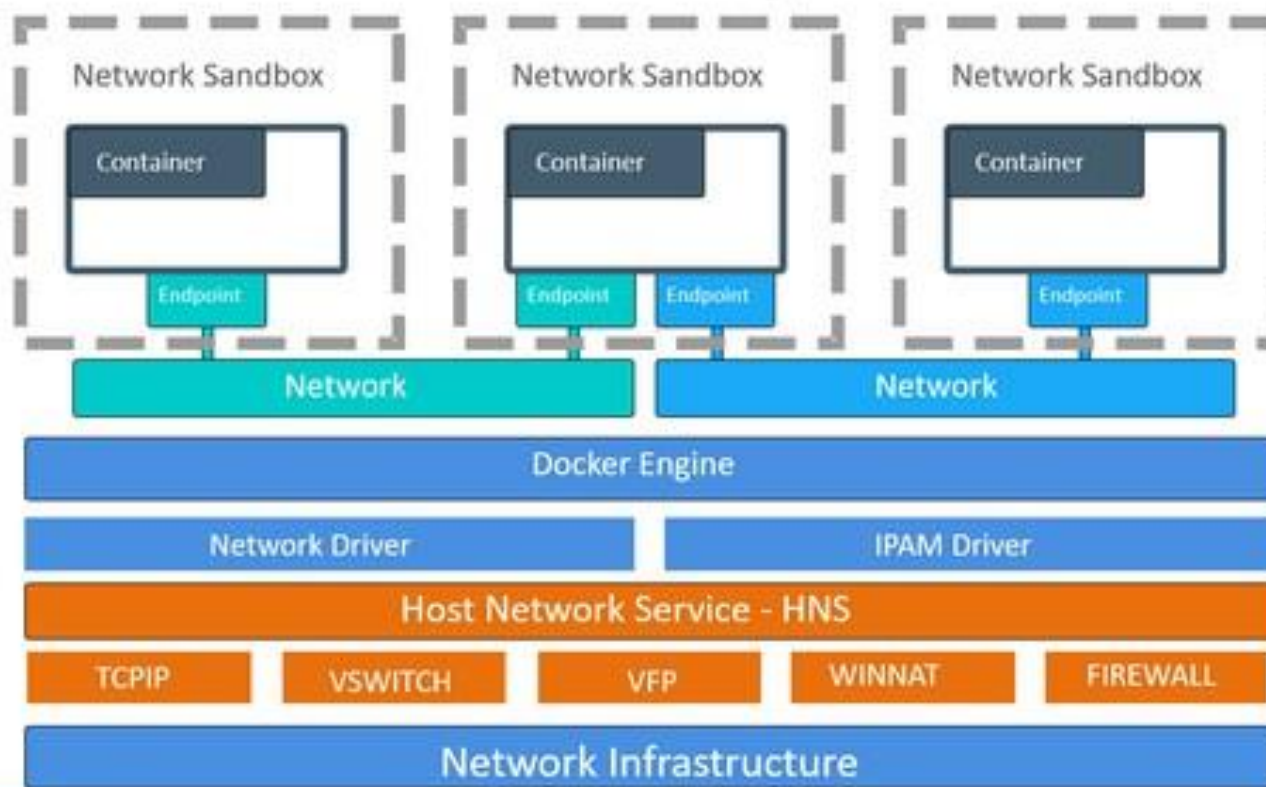
Docker в деталях

...T...Systems.....

Сеть Docker построена на Container Network Model (CNM), которая позволяет кому угодно создать свой собственный сетевой драйвер. Таким образом, у контейнеров есть доступ к разным типам сетей и они могут подключаться к нескольким сетям одновременно.



Container Networking Model



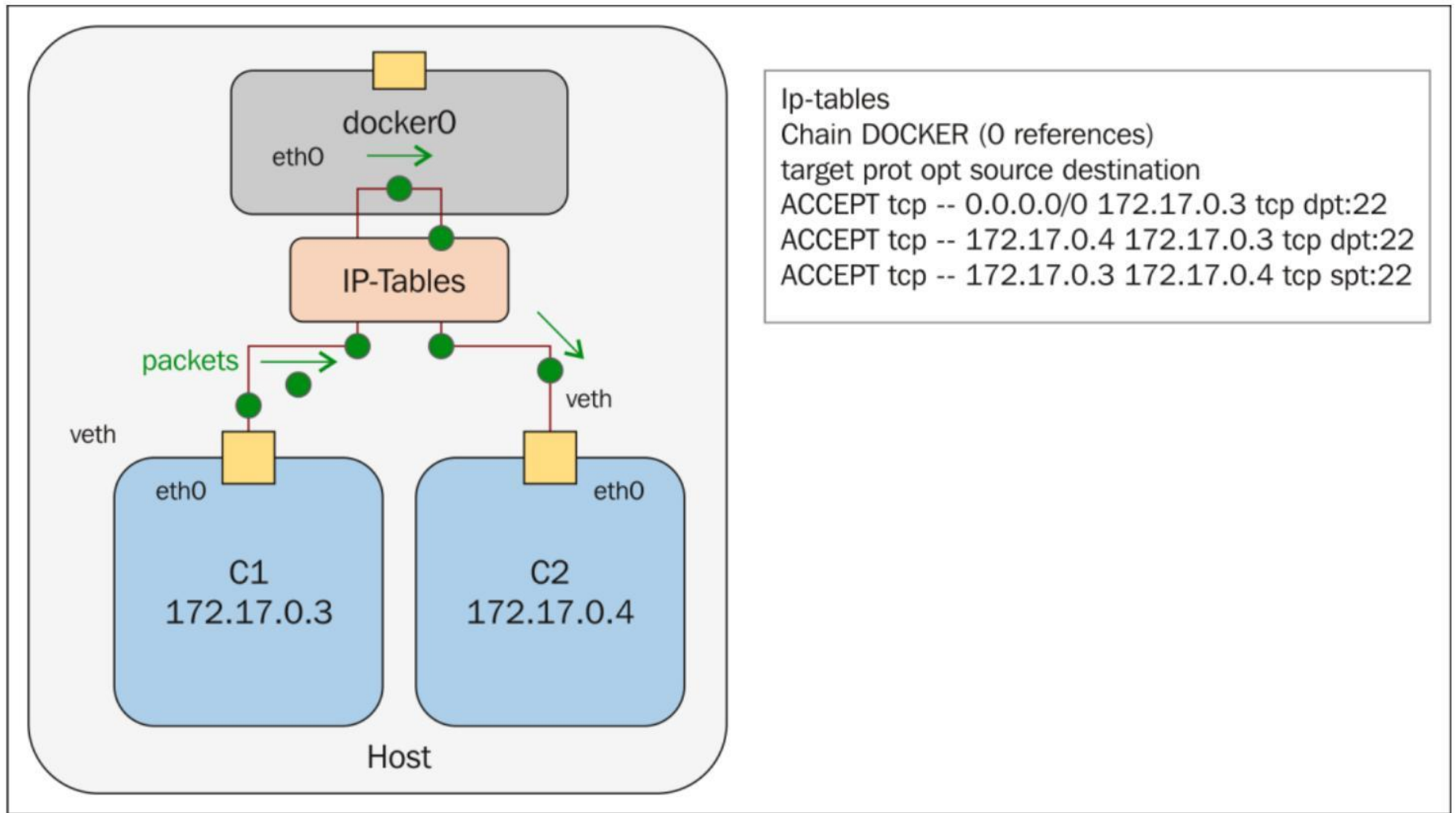
...T...Systems.....

Сетевое окружение

- **Bridge:** в этой сети контейнеры запускаются по умолчанию. Связь устанавливается через bridge-интерфейс на хосте. У контейнеров, которые используют одинаковую сеть, есть своя собственная подсеть, и они могут передавать данные друг другу по умолчанию.
- **Host:** этот драйвер дает контейнеру доступ к собственному пространству хоста (контейнер будет видеть и использовать тот же интерфейс, что и хост).
- **Macvlan:** этот драйвер дает контейнерам прямой доступ к интерфейсу и суб-интерфейсу (vlan) хоста. Также он разрешает транкинг.
- **Overlay:** этот драйвер позволяет строить сети на нескольких хостах с Docker (обычно на Docker Swarm кластере). У контейнеров также есть свои адреса сети и подсети, и они могут напрямую обмениваться данными, даже если они располагаются физически на разных хостах

..T..Systems.....

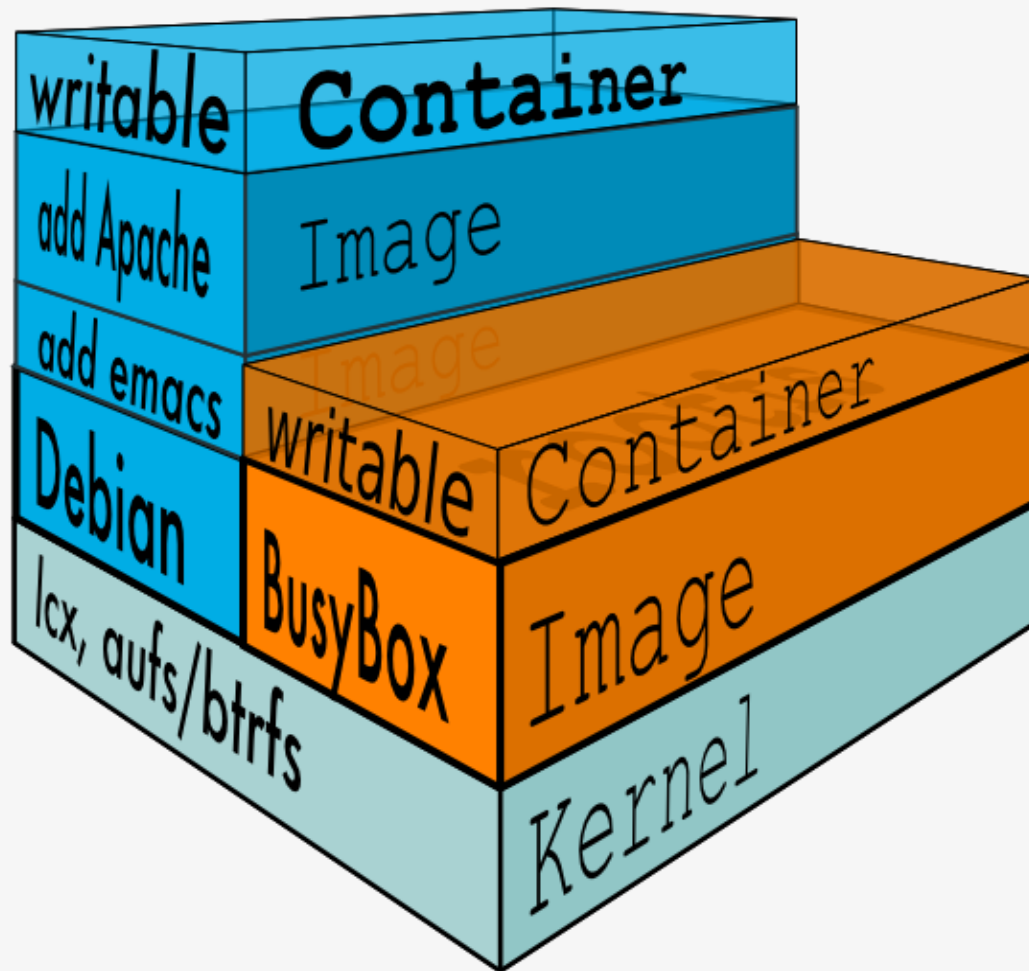
Сетевое окружение



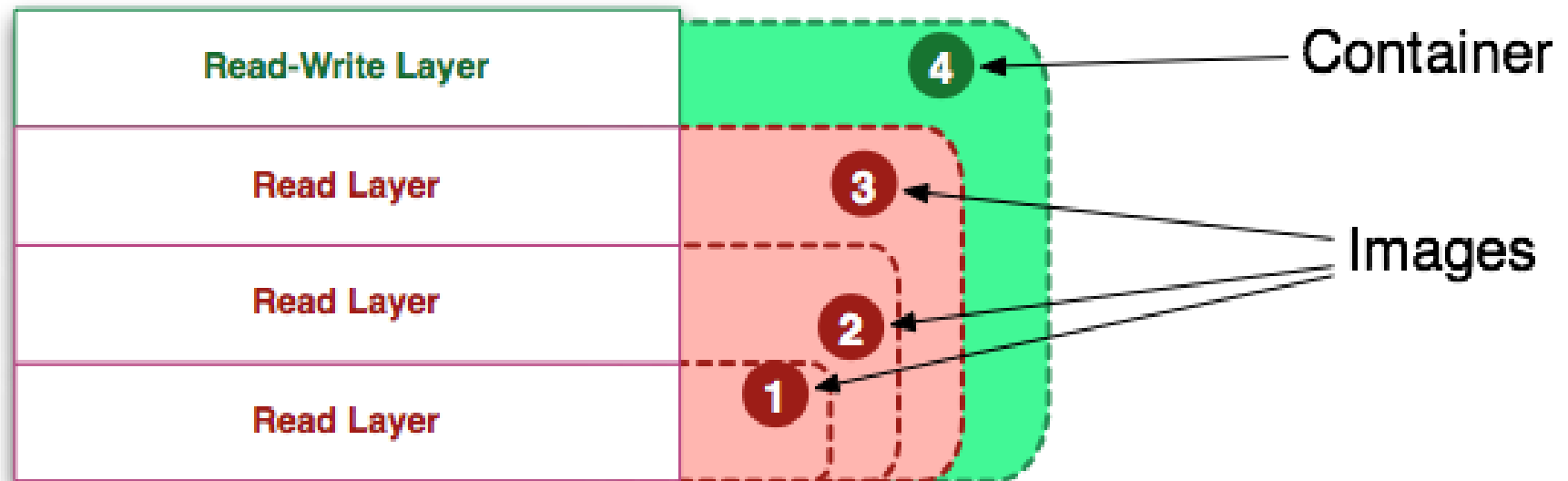
...T...Systems...

Union File System или **UnionFS** — вспомогательная файловая система для производящая каскадно-объединённое монтирование других файловых систем.

Это позволяет файлам и каталогам изолированных файловых систем, известных как ветви, прозрачно перекрываться, формируя единую связанную файловую систему. Docker использует UnionFS для создания блоков, из которых строится контейнер.

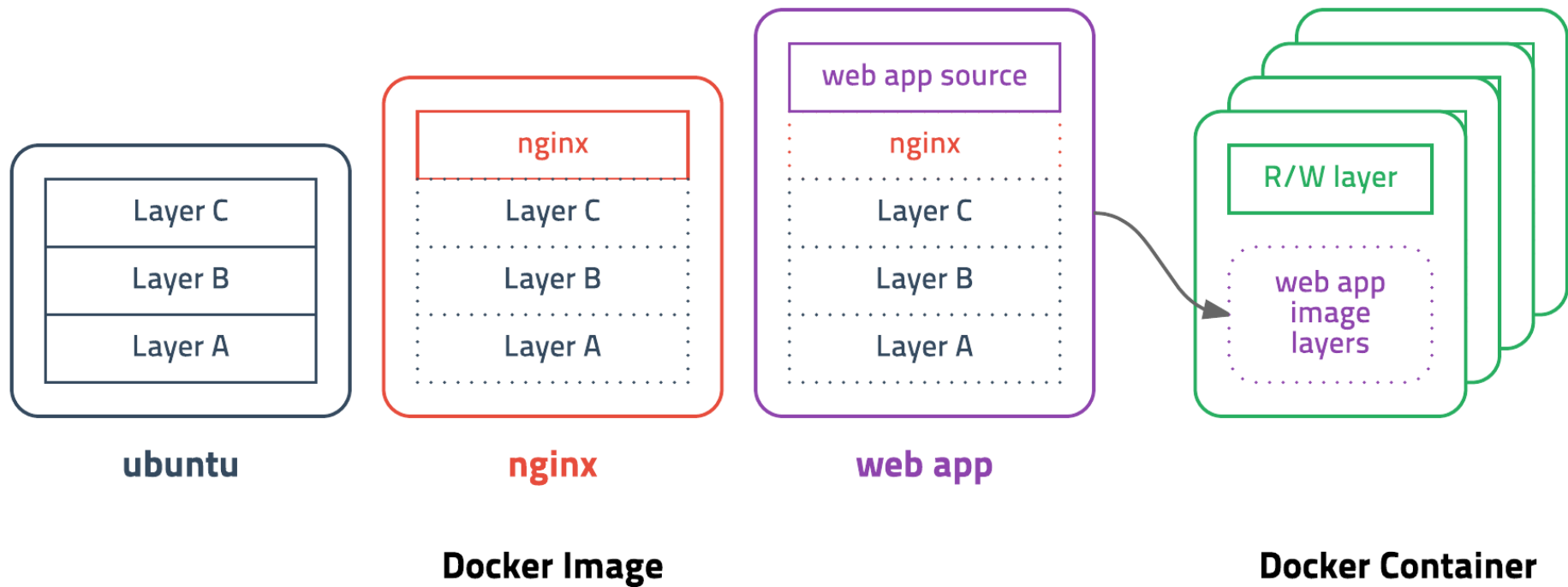


• • T • • Systems • • • • •



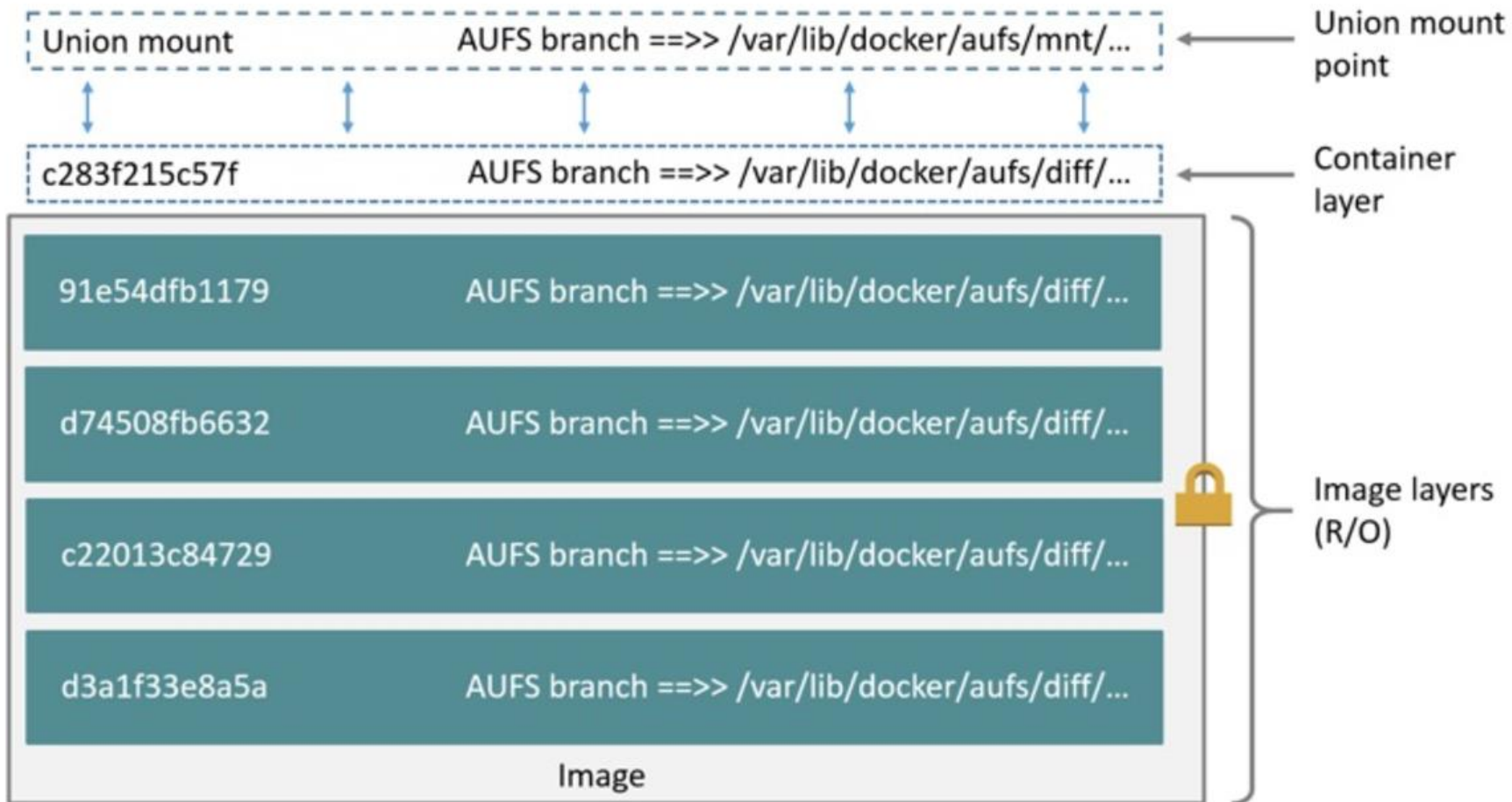
... T ... Systems ...

Файловая система



...T...Systems...

Файловая система

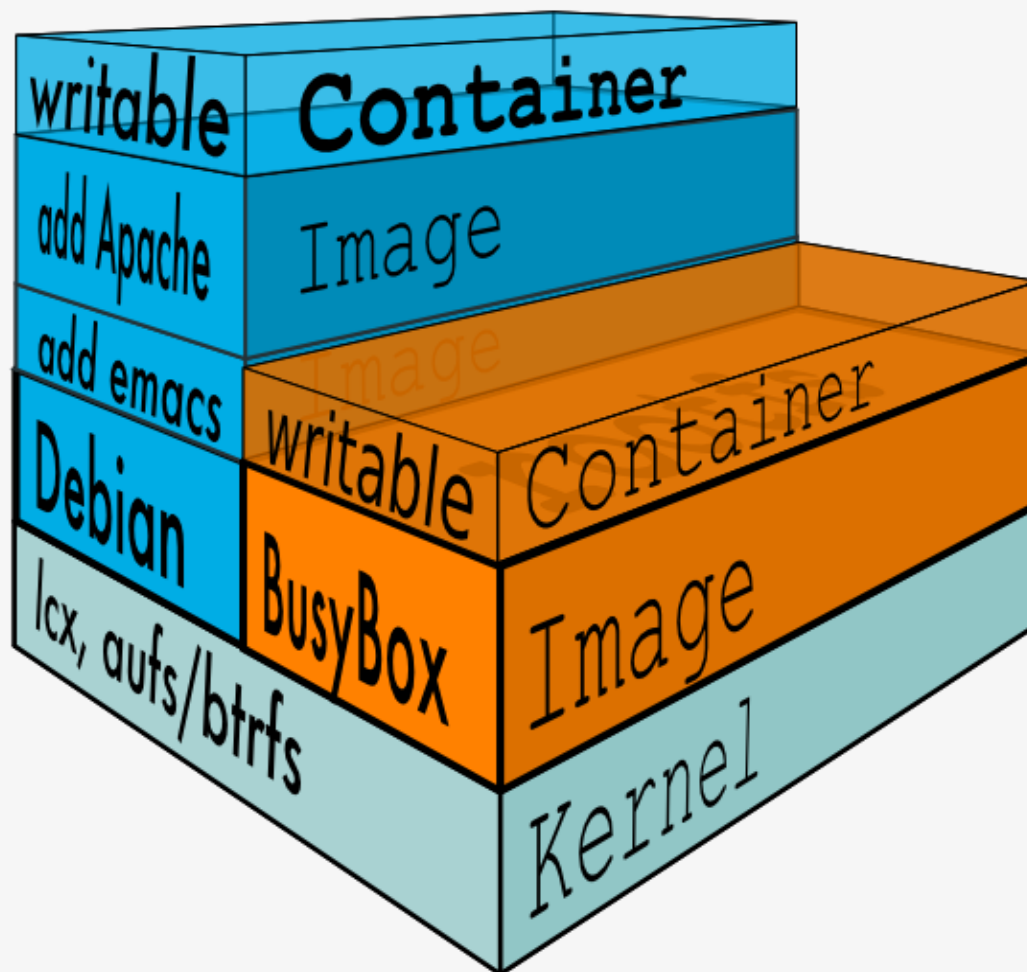


... T ... Systems ...

Взаимодействие с персистентными данными

Докер задуман быть stateless. Контейнеры не имеют хранилища на диске, всё что происходит — эфемерно и уходит, когда контейнер останавливается. Не подразумевается, что контейнеры будут хранить данные. На самом деле, они спроектированы чтобы НЕ ХРАНИТЬ данные. Любая попытка действовать против этой философии приводит к несчастьям.

Взаимодействие с персистентными данными



... **T** ... **Systems** ...

Докер НЕ ДОЛЖЕН ЗАПУСКАТЬ базы данных в продакшене, by design.

...T...Systems.....

Вы ОБЯЗАНЫ НЕ ЗАПУСКАТЬ на Докере базы данных в продакшене, НИКОГДА.

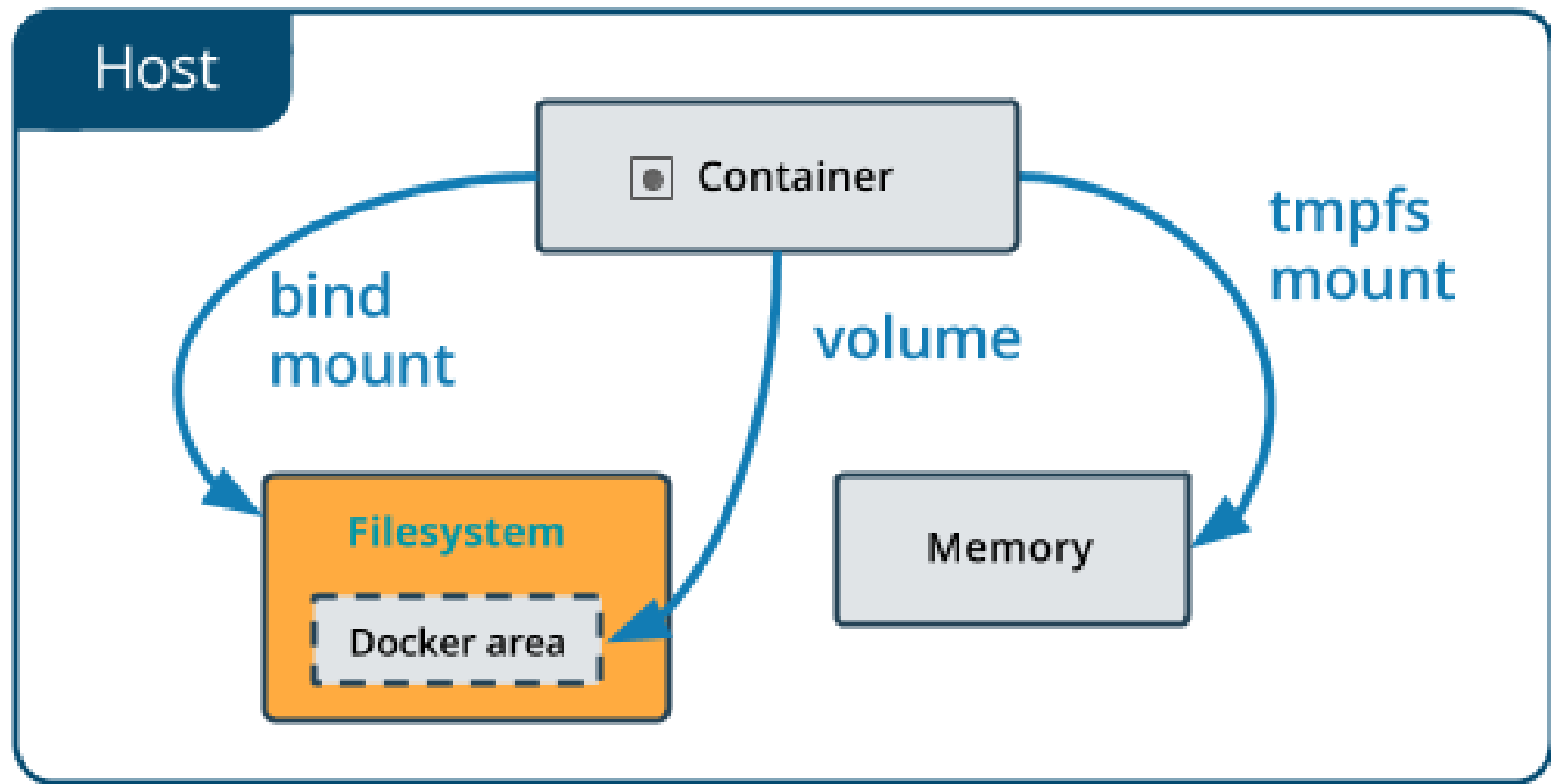
...T...Systems.....

Взаимодействие с персистентными данными

Для того, чтобы иметь возможность сохранять данные, а также использовать эти данные несколькими контейнерами, Docker предоставляем вам возможность тома (Docker volumes).

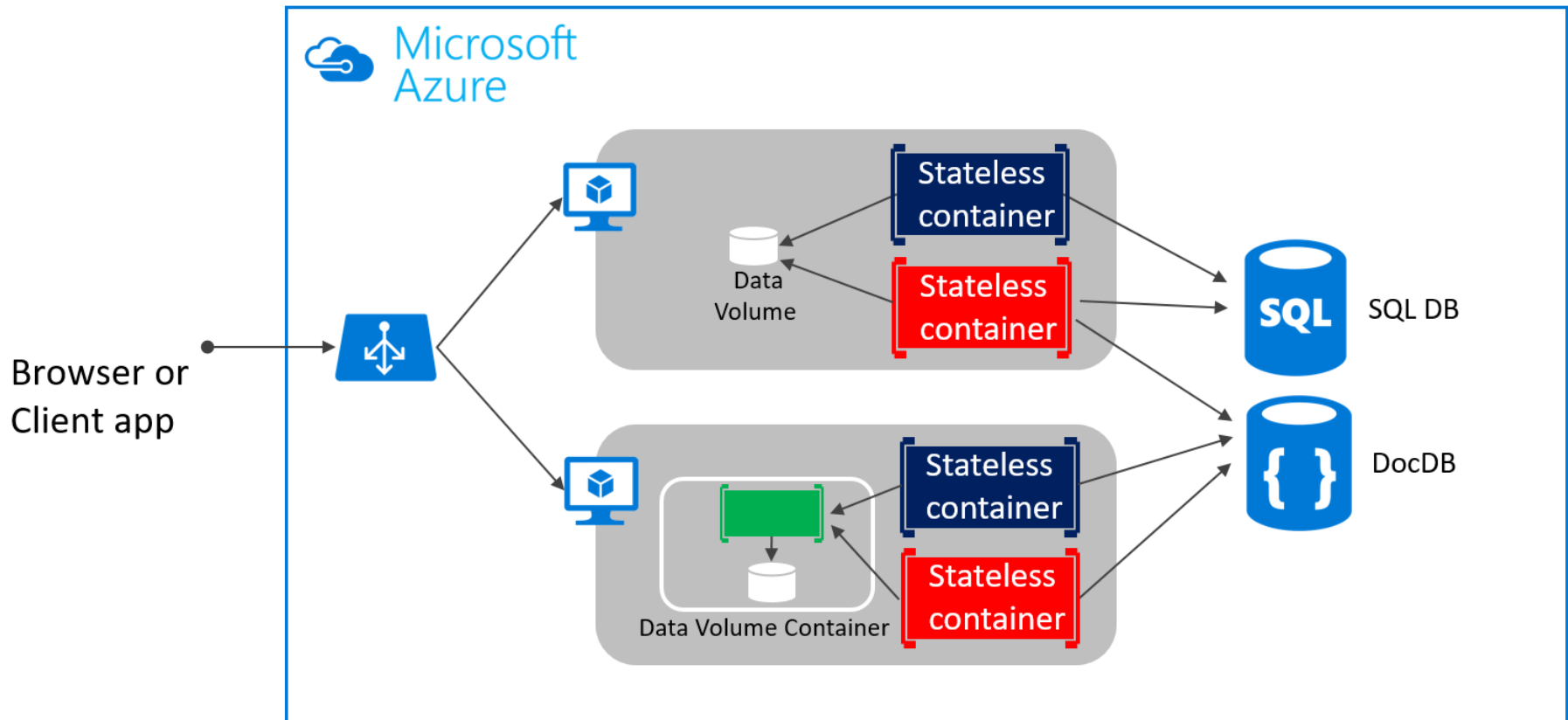
..T..Systems.....

Взаимодействие с персистентными данными



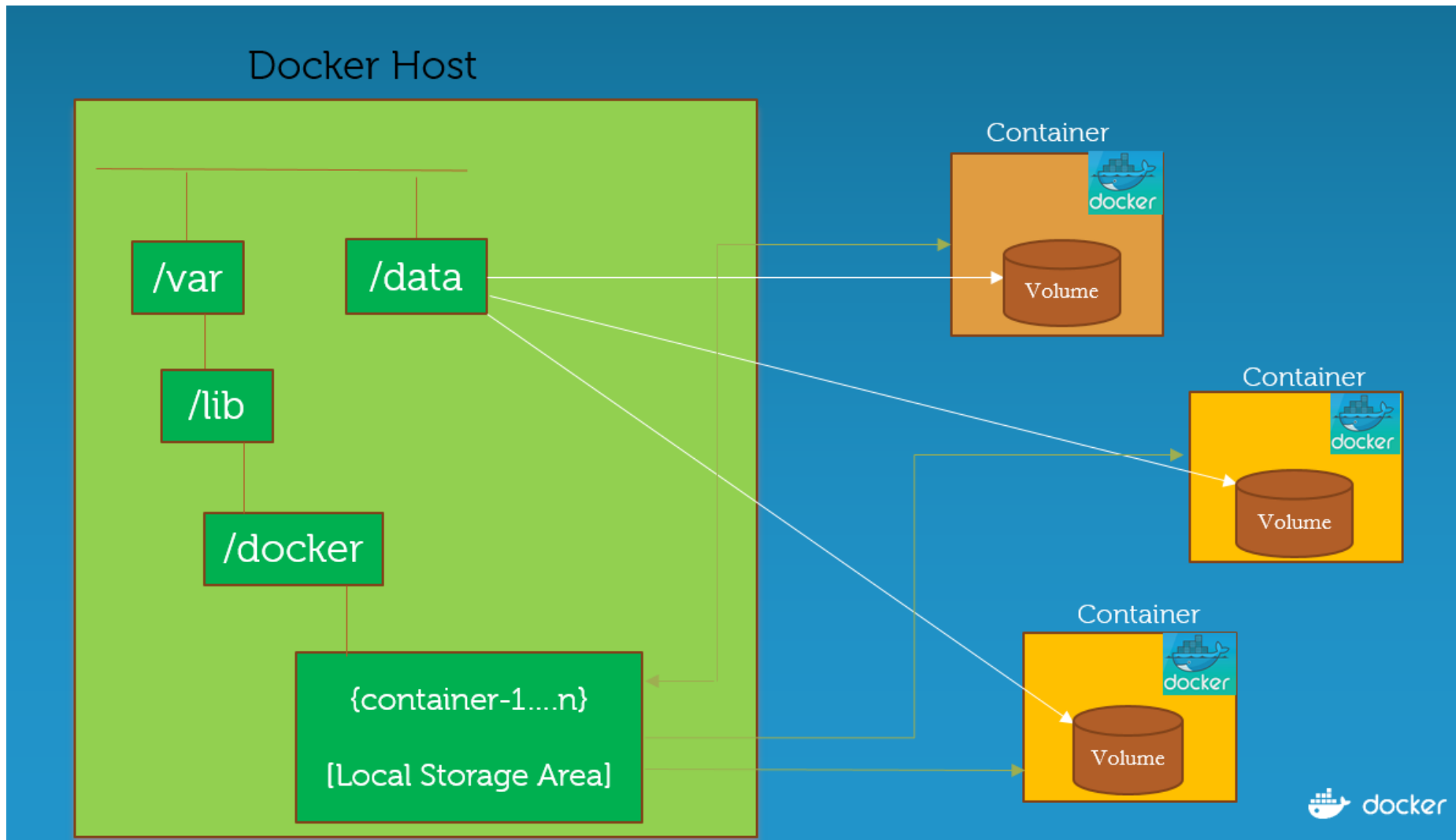
... T ... Systems ...

Data Volume and Data Volume Container



...T...Systems...

Взаимодействие с персистентными данными



...T...Systems.....

Code time

..T..Systems.....

Задание #2

	Внутриконтейнерный порт	Адрес эндпоинта
layer-server	:2141	/messages
layer-client	:2142	/swagger-ui.html

1) Запустить образ `parkito/layer-server` и `parkito/layer-client`

2) Послать сообщение из `layer-client` на `layer-server` через http-протокол (Адрес: `http://имя_контейнера:2141/`)

3) Подмонтировать к контейнеру `layer-client` локальную дерикторию и создать в ней файл через web-интерфейс контейнера

Используйте следующие форматы

Path: /my_folder/

FileName: Test.txt

.. **T** .. **Systems** ..

Задание #2 повышенной сложности

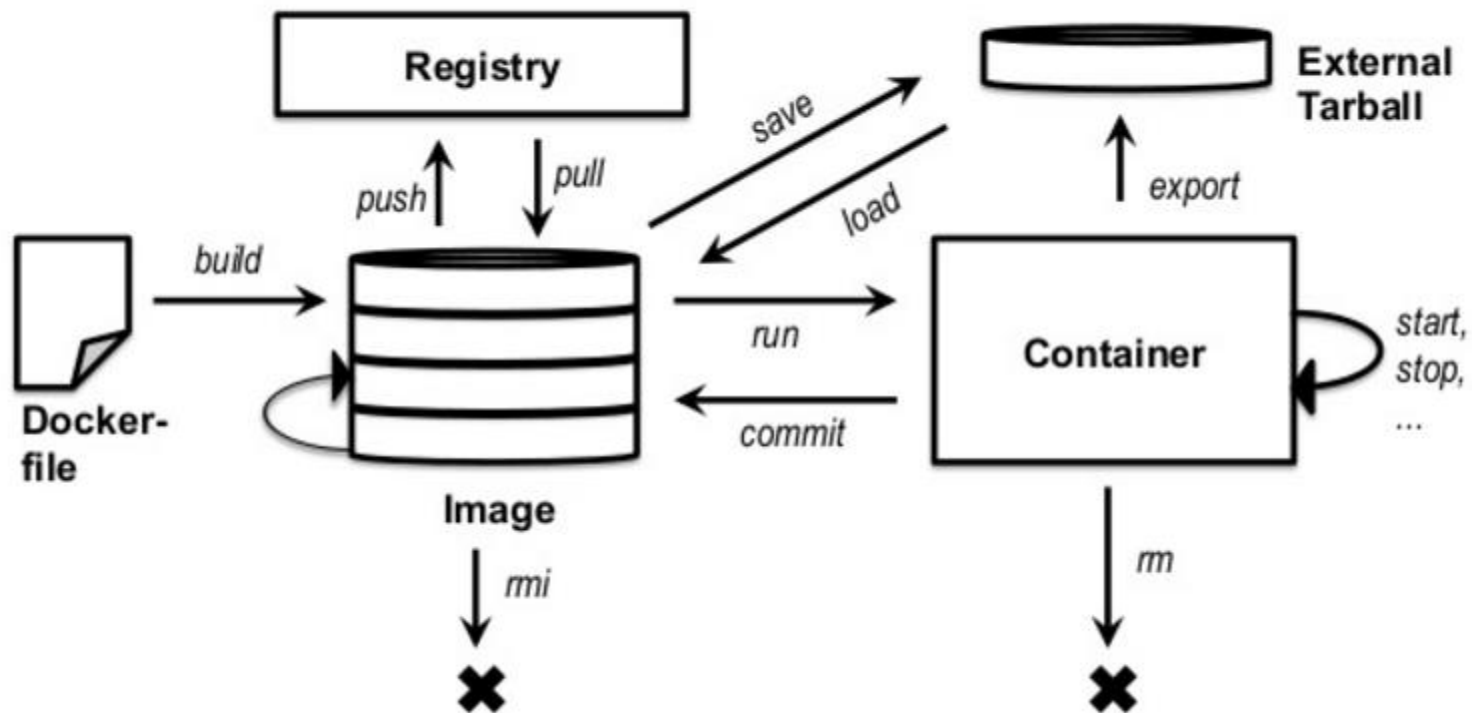
1) Запустить образы `parkito/layer-server` и `parkito/layer-client` с настроенной сетевой маршрутизацией между ними и примонтированной хостовой файловой системой

..T..Systems.....

Создание образов

...T...Systems.....

Создание образов



... T ... Systems ...

Изменение существующих образов



Docker Image

Упакованная версия приложения вместе с зависимостями и необходимым окружением необходимым для запуска

- Изменение существующего образа
- Использование Dockerfile

Изменение существующих образов

docker commit <CONTAINER_ID>



... **T** ... Systems ...

Dockerfile



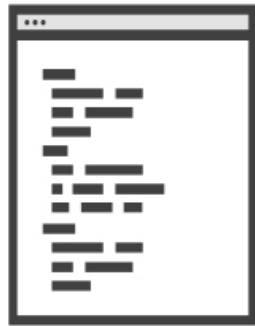
Dockerfile



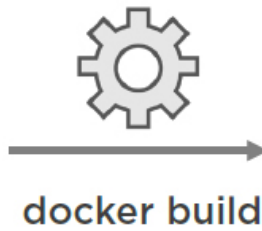
Docker Image

...T...Systems.....

Dockerfile



Dockerfile



Docker Image



...T...Systems.....

Команды Dockerfile

Синтаксис Dockerfile состоит из двух основных блоков:

- комментарии и команды
- аргументы

```
# Line blocks used for commenting
command argument argument ..
# Print "Hello docker!"
RUN echo "Hello docker!"
```

Все команды в файле Dockerfile нужно упорядочить по мере их выполнения. Однако некоторые команды (например, MAINTAINER) могут находиться в любом месте файла (но всегда после команды FROM).

Команды Dockerfile

Команда ADD

Команда ADD имеет два аргумента: источник и назначение. Команда копирует исходный файл в целевой каталог файловой системы контейнера. Если в источнике указан URL-адрес, команда загрузит его содержимое.

```
# Usage: ADD [source directory or URL] [destination directory]
ADD /my_app_folder /my_app_folder
```

Команда CMD

CMD, аналогично команде RUN, можно использовать для запуска других команд. В отличие от RUN, эту команду нельзя использовать при сборке, она выполняет команду при запуске контейнера

К примеру, команда CMD может запустить приложение во время создания контейнера, установленного с помощью RUN. Команда CMD будет командой по умолчанию и заменит любую другую команду, запущенную во время создания.

```
# Usage 1: CMD application "argument", "argument", ..
CMD "echo" "Hello docker!"
```

Команды Dockerfile

Аргумент ENTRYPOINT

ENTRYPOINT задаёт приложение по умолчанию, которое используется во время создания контейнера. К примеру, если образ предназначен только для запуска определённого приложения, это приложение можно обозначить в ENTRYPOINT.

Аргумент ENTRYPOINT можно использовать с командой CMD.

```
# Usage: ENTRYPOINT application "argument", "argument", ..
# Remember: arguments are optional. They can be provided by CMD
#           or during the creation of a container.
ENTRYPOINT echo

# Usage example with CMD:
# Arguments set with CMD can be overridden during *run*
CMD "Hello docker!"
ENTRYPOINT echo
```

Команды Dockerfile

Команда ENV

Команда ENV задаёт переменные среды в формате «ключ = значение», которые в дальнейшем можно использовать в сценариях и приложениях внутри контейнера. ENV обеспечивает гибкость запуска команд.

```
# Usage: ENV key value  
ENV SERVER_WORKS 4
```

Команда EXPOSE

Команда EXPOSE задаёт порт, с помощью которого приложение в контейнере может взаимодействовать с внешним миром.

```
# Usage: EXPOSE [port]  
EXPOSE 8080
```

Команды Dockerfile

Команда FROM

FROM – пожалуй, одна из самых важных команд Dockerfile. Она определяет базовый образ, на основе которого будет собран новый образ. В качестве базового можно использовать любой доступный образ, включая созданные ранее. Если указанный образ не найден, Docker попытается найти и загрузить его из индекса образов. С этой команды должен начинаться Dockerfile.

```
# Usage: FROM [image name]
FROM ubuntu
```

Команда MAINTAINER

Одна из команд, которые можно поместить в любую точку сценария (хотя рекомендуется всё же указывать её в начале). Эта команда не выполняется, она позволяет задать имя автора. Она всегда должна идти после FROM.

```
# Usage: MAINTAINER [name]
MAINTAINER authors_name
```

Команды Dockerfile

Команда RUN

RUN – основная команда Dockerfile для запуска других команд. Она запускает указанную команду внутри контейнера с учётом всех аргументов. В отличие от CMD, её можно использовать для сборки образа (формирования нового уровня).

```
# Usage: RUN [command]
RUN aptitude install -y riak
```

Директива USER

Эта директива задаёт имя пользователя (или UID), с помощью которого нужно запустить контейнер.

```
# Usage: USER [UID]
USER 751
```

Команды Dockerfile

Команда VOLUME

VOLUME разрешает контейнеру доступ к заданному каталогу на локальной машине.

```
# Usage: VOLUME ["/dir_1", "/dir_2" ..]  
VOLUME ["/my_files"]
```

Директива WORKDIR

WORKDIR устанавливает рабочий каталог, в котором будет выполнена команда, указанная в CMD.

```
# Usage: WORKDIR /path  
WORKDIR ~/
```

Code time

..T..Systems.....

Задание #3

1) `$git clone github.com/parkito/LearnDocker`

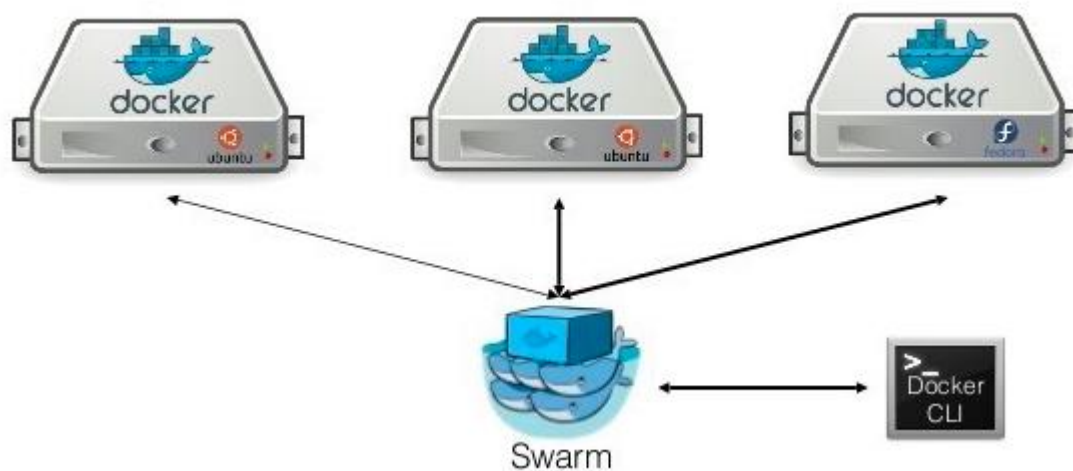
2) Докеризовать Dockerfile/WEB_APP приложение (использовать 2144-ый порт)

3) Запустить образ докеризированного приложения и обратиться к его эндпоинтам (`localhost:2144/swagger-ui.html`)

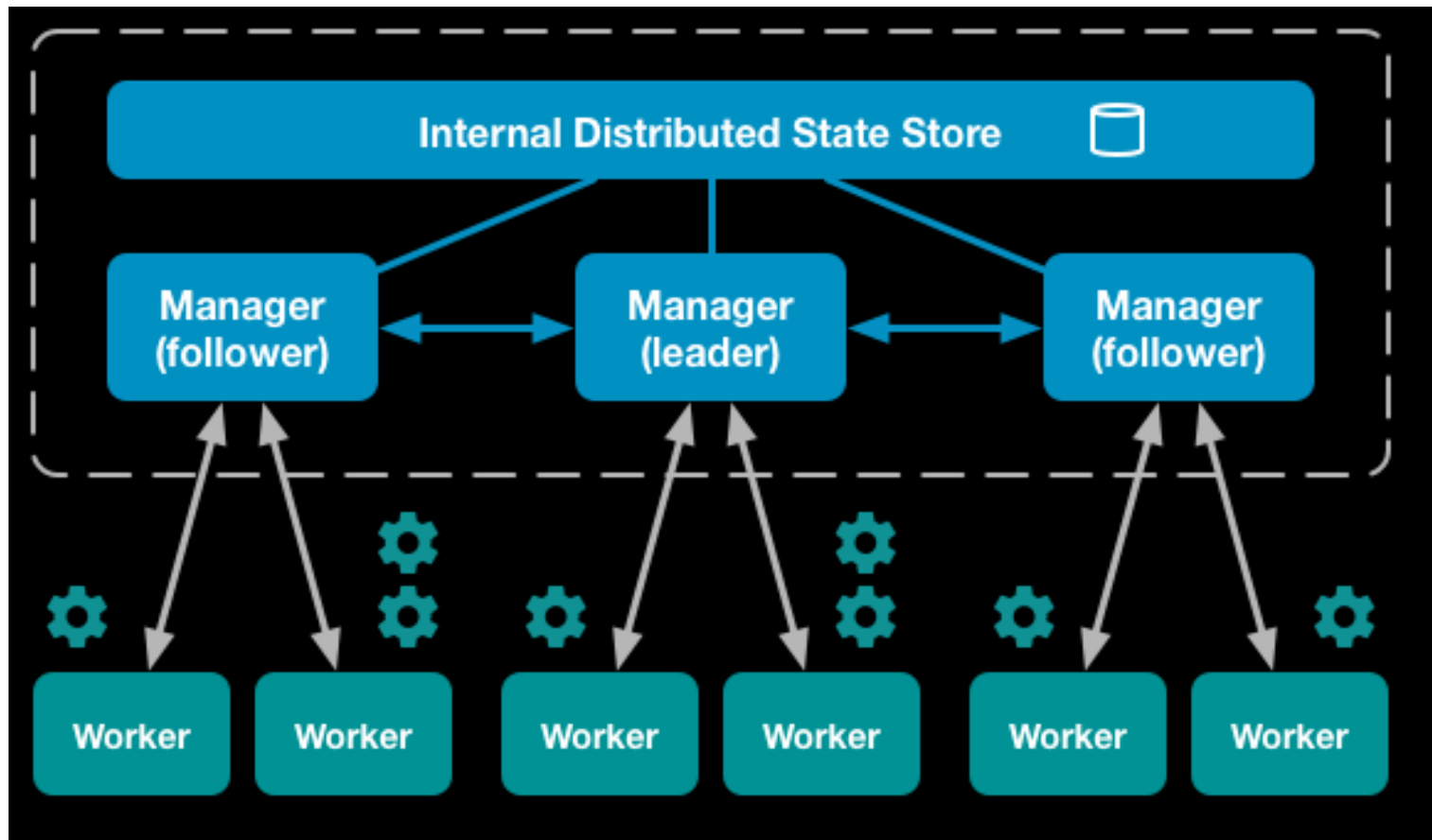
Кластеризация

.. T .. Systems

Docker Swarm — это кластер Докеров. Множество хостов, на каждом из которых запущен отдельный Docker Engine, объединены под общим управлением, и выглядят как один большой Docker. Хосты могут добавляться в кластер, и вычислительная ёмкость вашего большого Докера будет тоже увеличена.

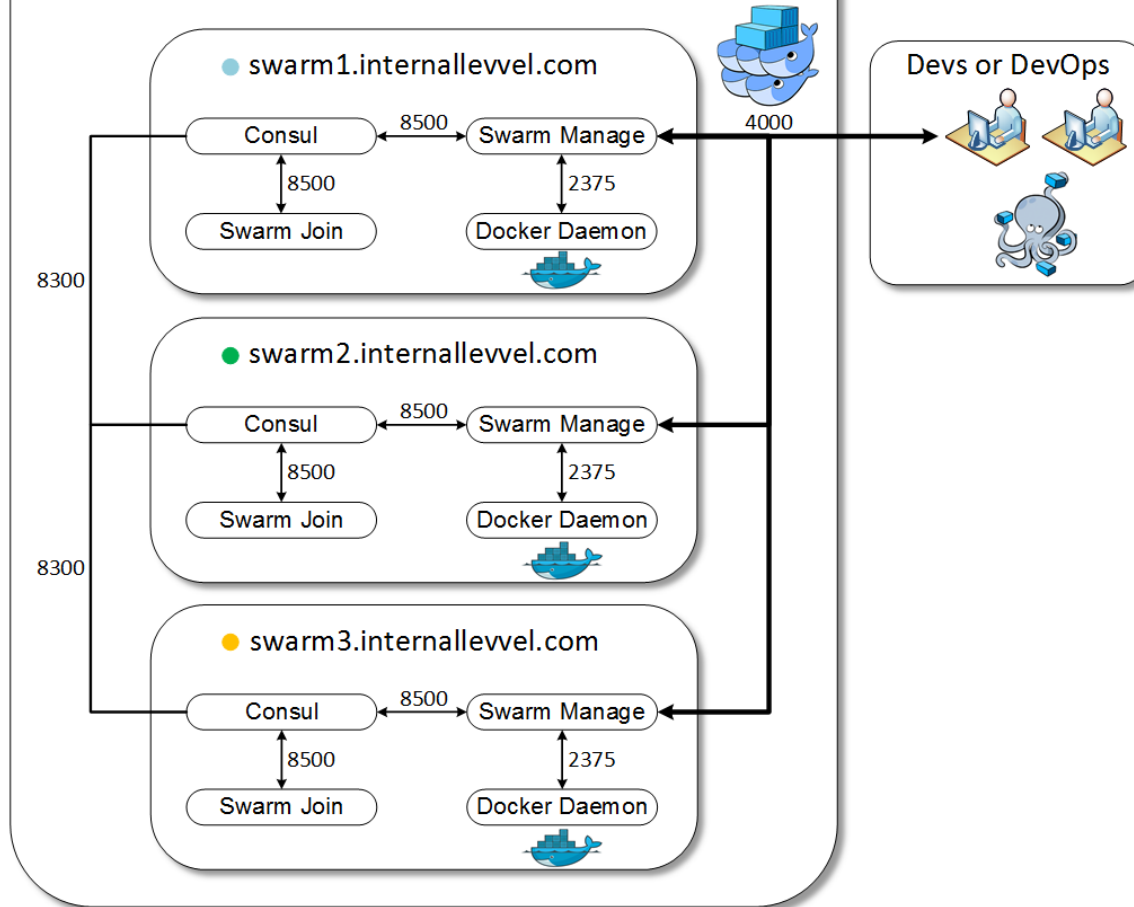


.. T .. Systems ..



... T ... Systems ...

Distributed Docker Swarm Environment



Kubernetes является проектом с открытым исходным кодом, предназначенным для управления кластером контейнеров Linux как единой системой. Kubernetes управляет и запускает контейнеры Docker на большом количестве хостов, а так же обеспечивает совместное размещение и репликацию большого количества контейнеров.



.. **T** .. **Systems** ..

Apache Mesos — это централизованная отказоустойчивая система управления кластером. Она разработана для распределенных компьютерных сред с целью обеспечения изоляции ресурсов и удобного управления кластерами подчиненных узлов (**mesos** slaves)



MESOS

...T...Systems.....

Project

Hello Openshift

1

Add to project

2

?

admin

▼

3

Overview

4

Applications

5

Builds

6

Resources

7

Storage

8

Monitoring

▼ NODEJS

<http://nodejs-hello-openshift.xip.io>

Build **nodejs, #1** ✓ Complete 9 minutes ago [View Log](#)

nodejs

Deployment **nodejs** - 7 minutes ago

CONTAINER: NODEJS

Image: hello-openshift/nodejs

Ports: 8080/TCP

1 pod

postgresql

Deployment **postgresql** - 8 days ago

CONTAINER: POSTGRESQL

Image: centos/postgresql-95-centos7

Ports: 5432/TCP

3 pods

▼ RAILS POSTGRESQL EXAMPLE

<http://rails-postgresql-example-hello-openshift.xip.io>

Build **rails-postgresql-example, #1** ✓ Complete 8 days ago [View Log](#)

rails-postgresql-example

Deployment **rails-postgresql-example** - 8 days ago

CONTAINER: RAILS-POSTGRESQL-EXAMPLE

Image: hello-openshift/rails-postgresql-example

Ports: 8080/TCP

1 pod

postgresql

Deployment **postgresql** - 8 days ago

CONTAINER: POSTGRESQL

Image: centos/postgresql-95-centos7

Ports: 5432/TCP

3 pods

Code time

..T..Systems.....

Задание #4

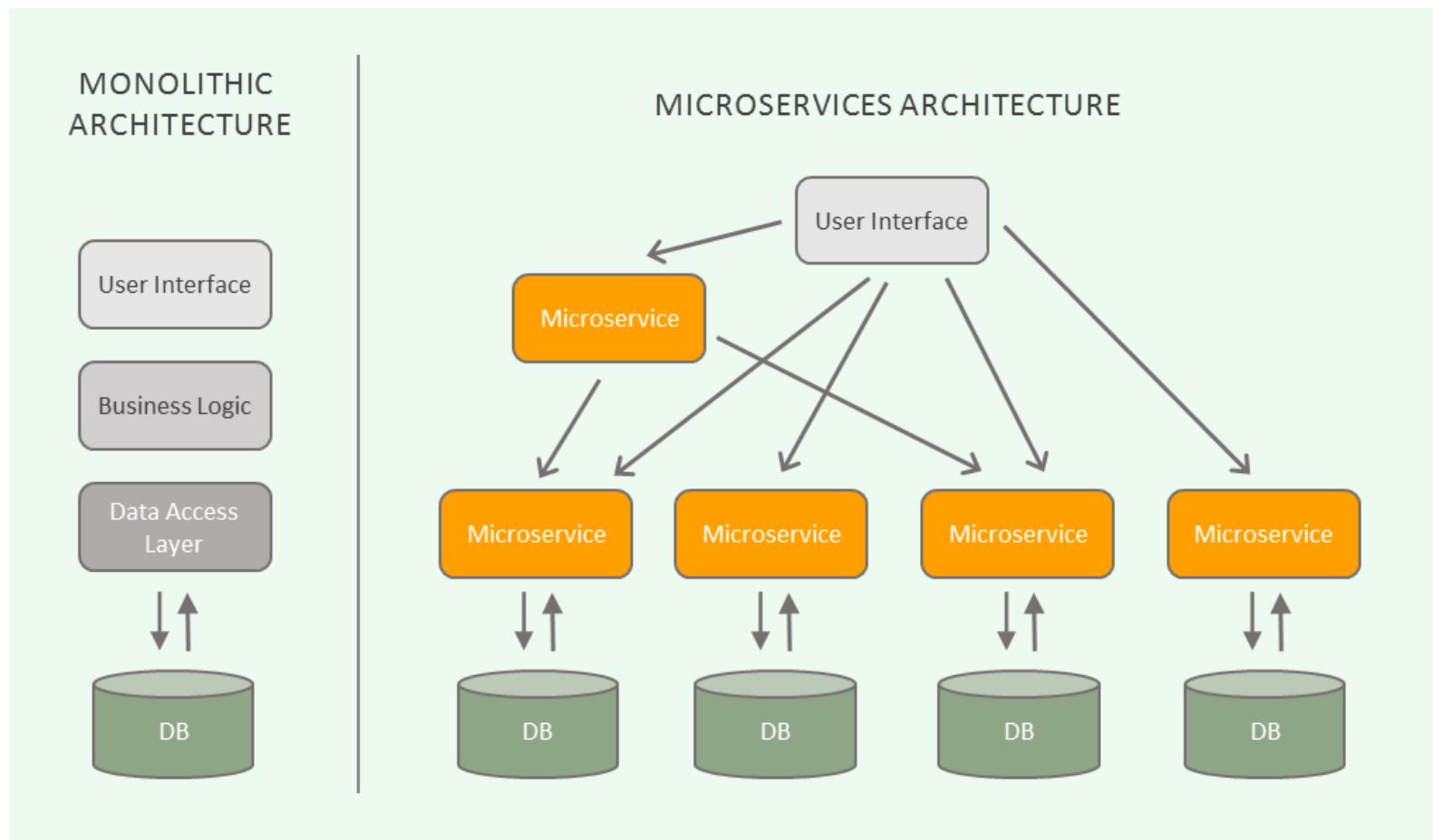
1) На основе docker-compose.yml из Clustering напишите скрипт запуска запуска приложений из задания №2. (Конкретнее из задания повышенной сложности – с настроенным сетевым взаимодействием и примонтированной хостовой файловой системой)

Микросервисная архитектура

...T...Systems.....

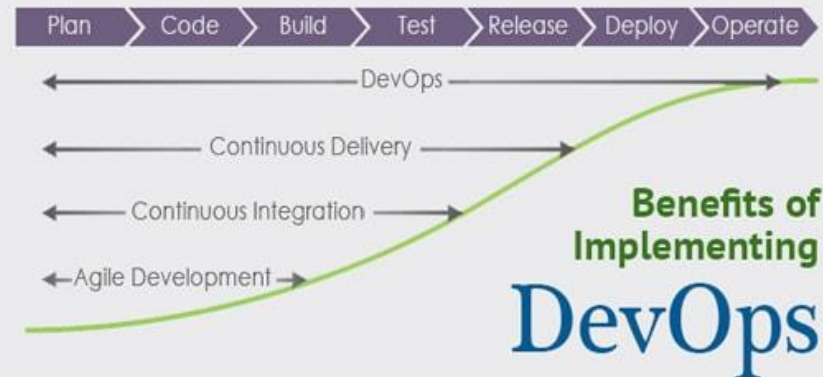
Микросервисы

Микросервисы – это архитектурный подход, в котором большие комплексные системы состоят из одного или более компонентов.



...T...Systems...

DevOps Life cycle



21% Reduction
in time spent fixing & maintaining applications

19% Improvement
in application quality and performance

18% Increase
in revenue

33% Increase
in time for infrastructure improvements

15% Increase
in time for self-improvement

60% Decrease
in application release time

... T ... Systems ...

Достоинства и недостатки



Абстрагирование приложение от хоста

Тонкая настройка

Масштабирование

Обратная совместимость

Управление версиями и зависимостями

Удаление

Изолирование среды

Производительность

Использование слоев

Архитектура

Компоновка

Поддержка

Заклучение



45%

of Docker users have been able to increase the frequency of software releases

93%

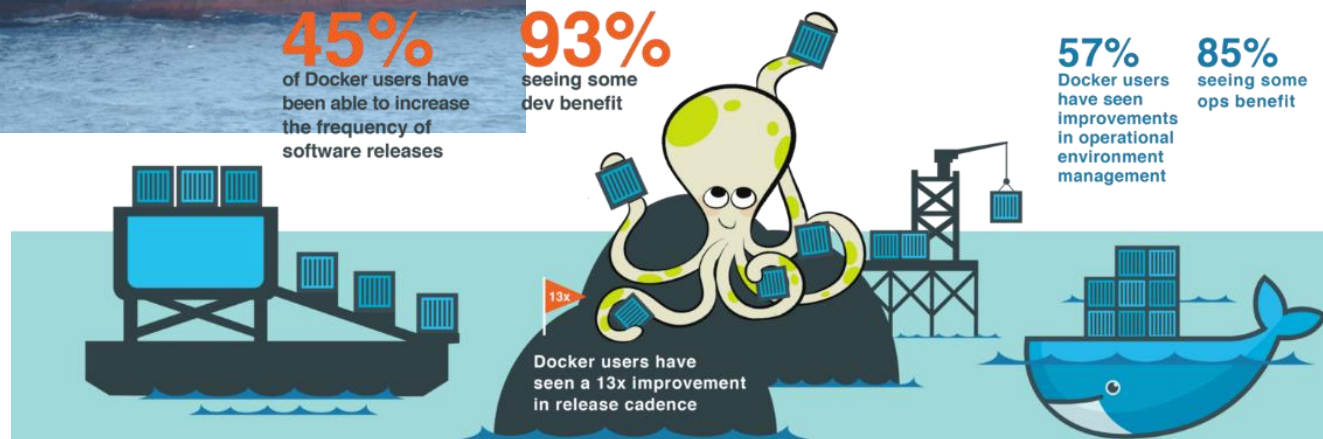
seeing some dev benefit

57%

Docker users have seen improvements in operational environment management

85%

seeing some ops benefit



70%

of Docker users say 'Docker has dramatically transformed...' etc



62%

have seen improved MTTR on software issues.



...T...Systems...

Ресурсы

- <https://www.youtube.com/watch?v=Q5POuMHxW-0> (Introduction to Docker By Solomon Hykes) – Введение от создателя
- Using Docker (Adrian Mouat) - хорошая кига по основам
- http://openit.readthedocs.io/03-cloud/600_docker_intro/ (Докер основы) – Краткая выжимка всего докера
- <https://habr.com/post/332450> (Docker в продакшене. История провала 1) - Статья на подумать
- <https://habr.com/post/346430> (Docker в продакшене. История провала 2) - Статья на подумать
- <https://xakep.ru/2015/06/01/docker-advanced-usecases/#toc01> (Идеальные задачи для докера) – хороший обзор инфраструктуры

.. **T** .. Systems ..



THANK YOU!