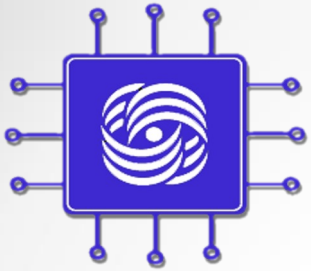
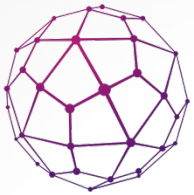




APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

# Системы моделирования компьютерных сетей #2

Доп. главы Компьютерных сетей и телекоммуникации

к. ф.-м. н. Антоненко В.А.

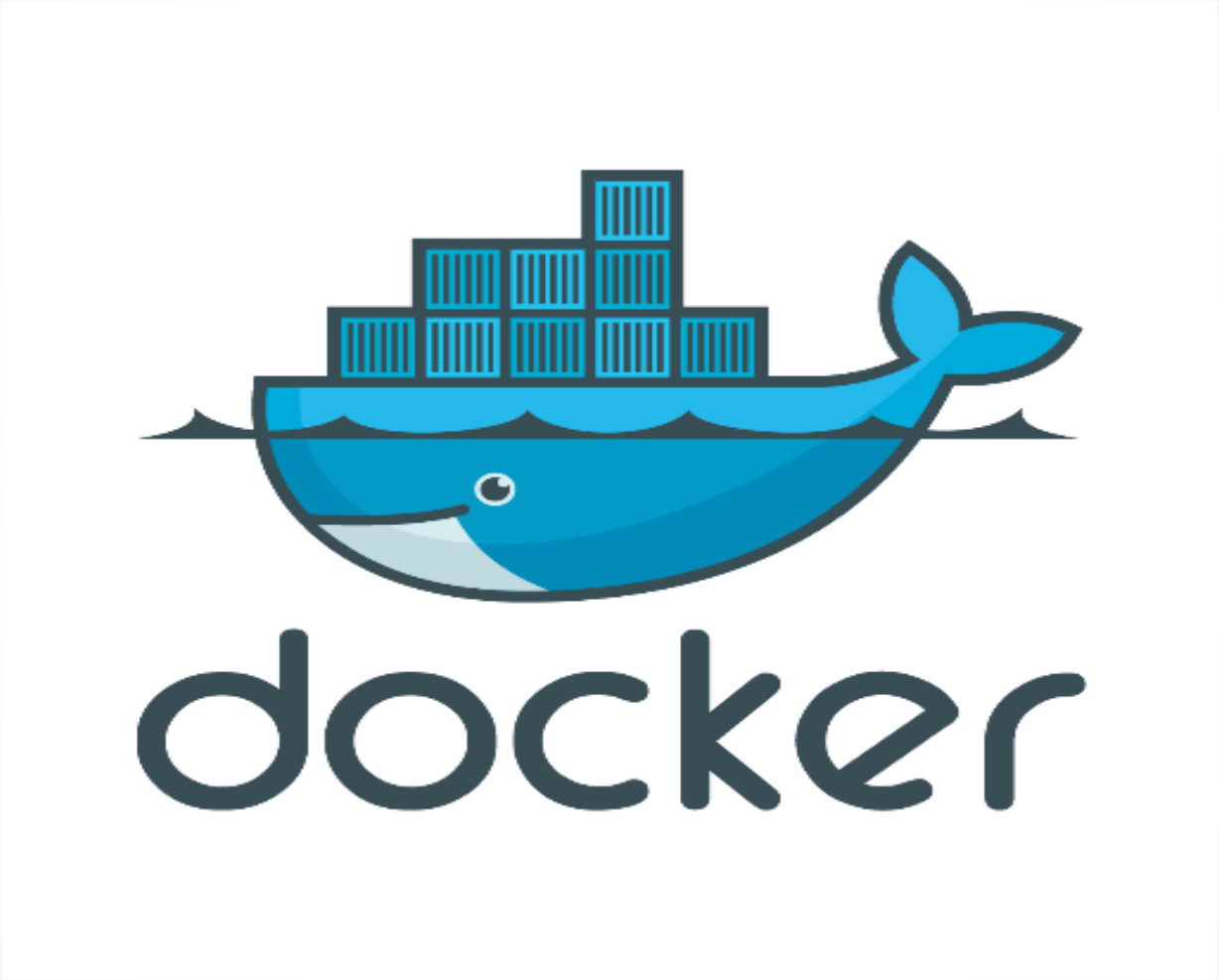
# Вопросы

1. Что такое имитационное моделирование?
2. Что такое аналитическое моделирование?
3. Что такое агентное моделирование?
4. Что такое системная динамика?
5. Что такое дискретно-событийная модель?

# План лекции

1. Основы контейнерной виртуализации – Docker
2. Tcpdump
3. NPS

# Introduction to Docker



# История DOCKER ……

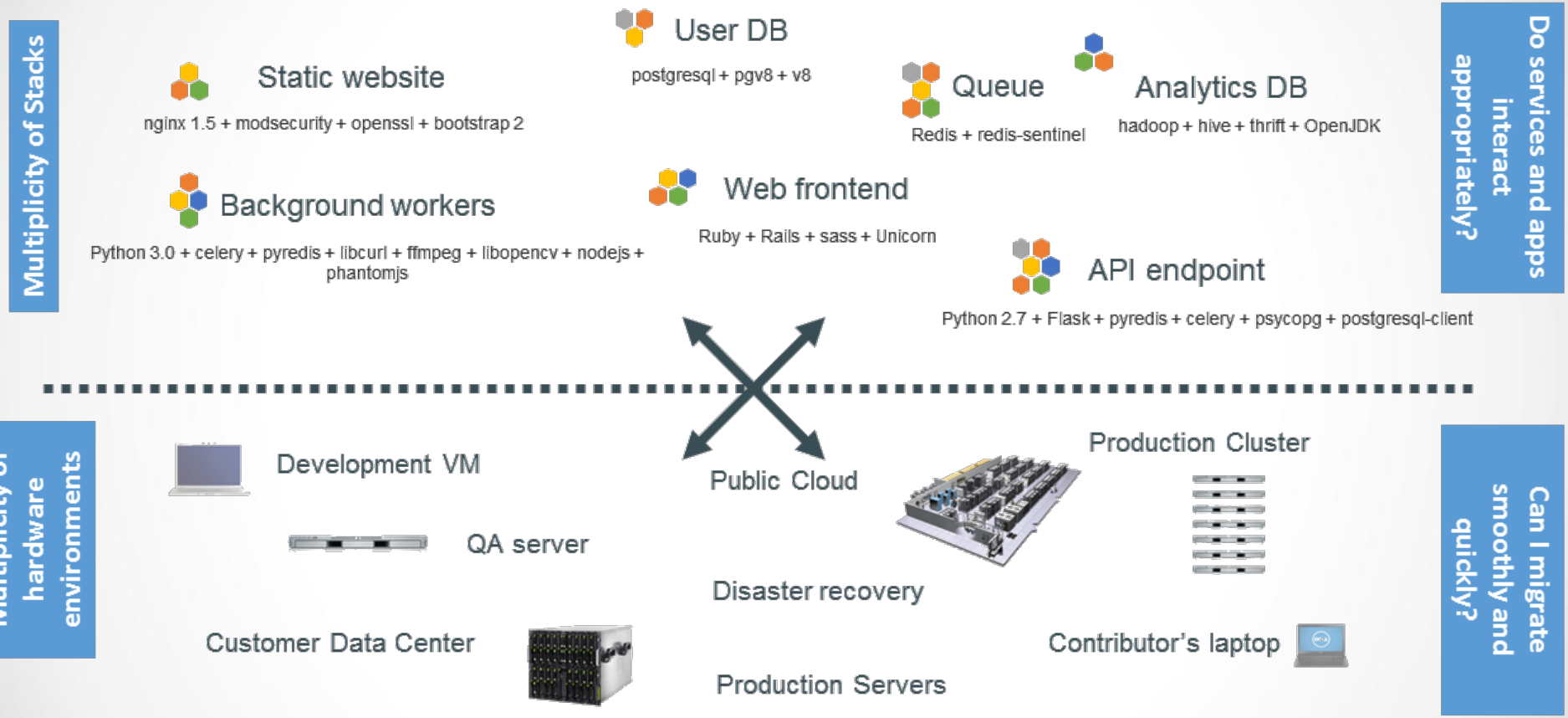
- A dotCloud (PAAS provider) project
- Первый коммит January 18, 2013
- Docker 0.1.0 March 25, 2013
- 18,600+ github stars, 3800+ forks,740 Contributors…. and continues

# Что такое Docker?

- Докер — это открытая платформа для разработки, доставки и эксплуатации приложений.
- отделить ваше приложение от вашей инфраструктуры
- позволяет запускать практически любое приложение, безопасно изолированное в контейнере

# Когда Docker полезен?

- упаковывание приложения (и так же используемых компонент) в docker контейнеры;
- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
- размещение контейнеров на серверах, как в дата центры так и в облака.


Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Do services and apps interact appropriately?
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Disaster recovery
Production Servers
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly?

Multiplicity of Goods















Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing















Can I transport quickly and smoothly (e.g. from boat to train to truck)

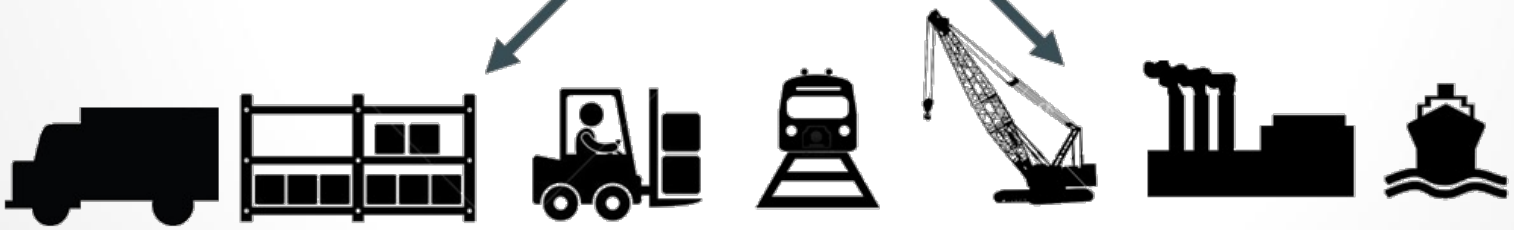
Multiplicity of Goods

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

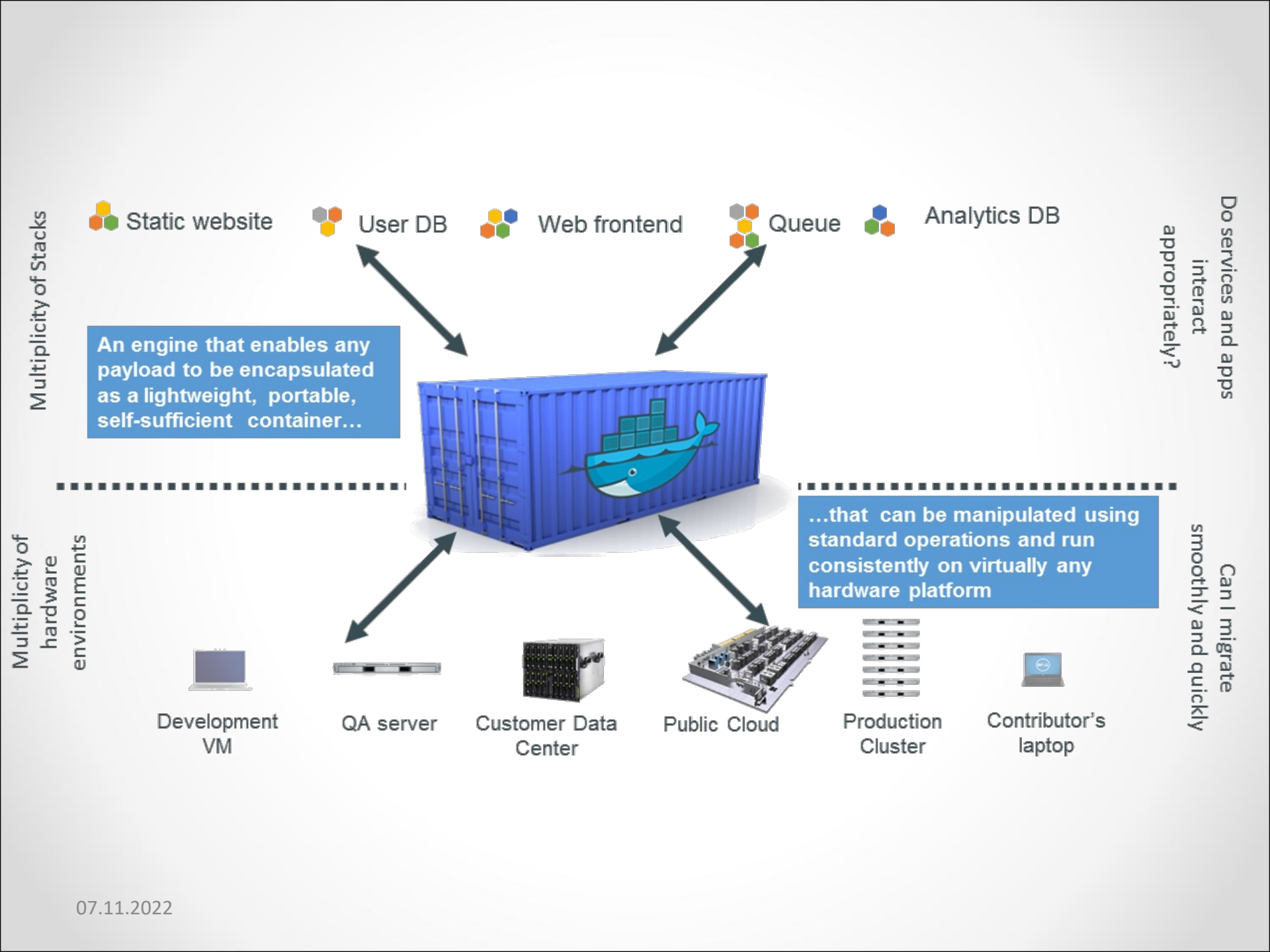
Do I worry about how goods interact (e.g. coffee beans next to spices)

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

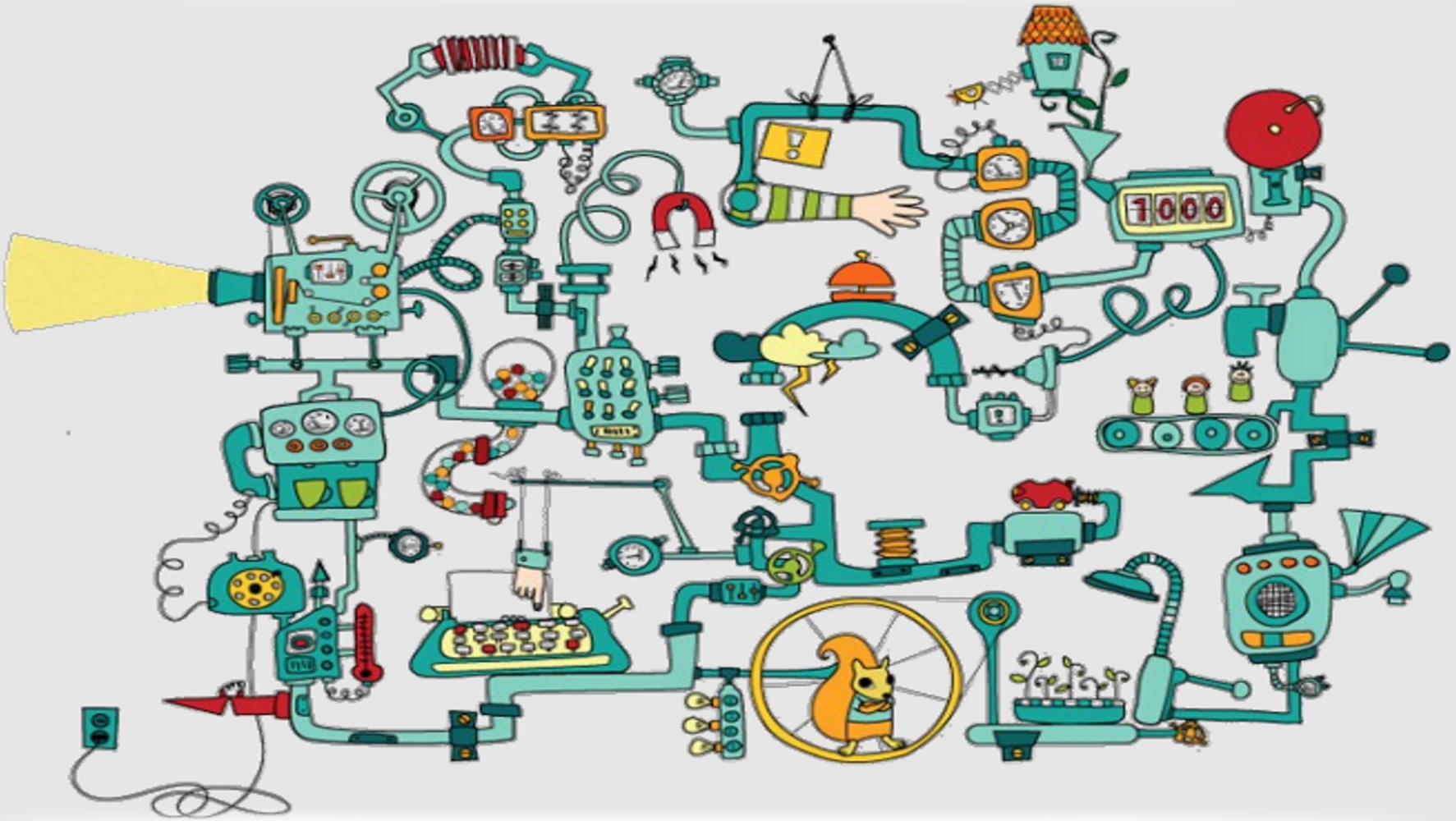
Multiplicity of methods for transporting/storing



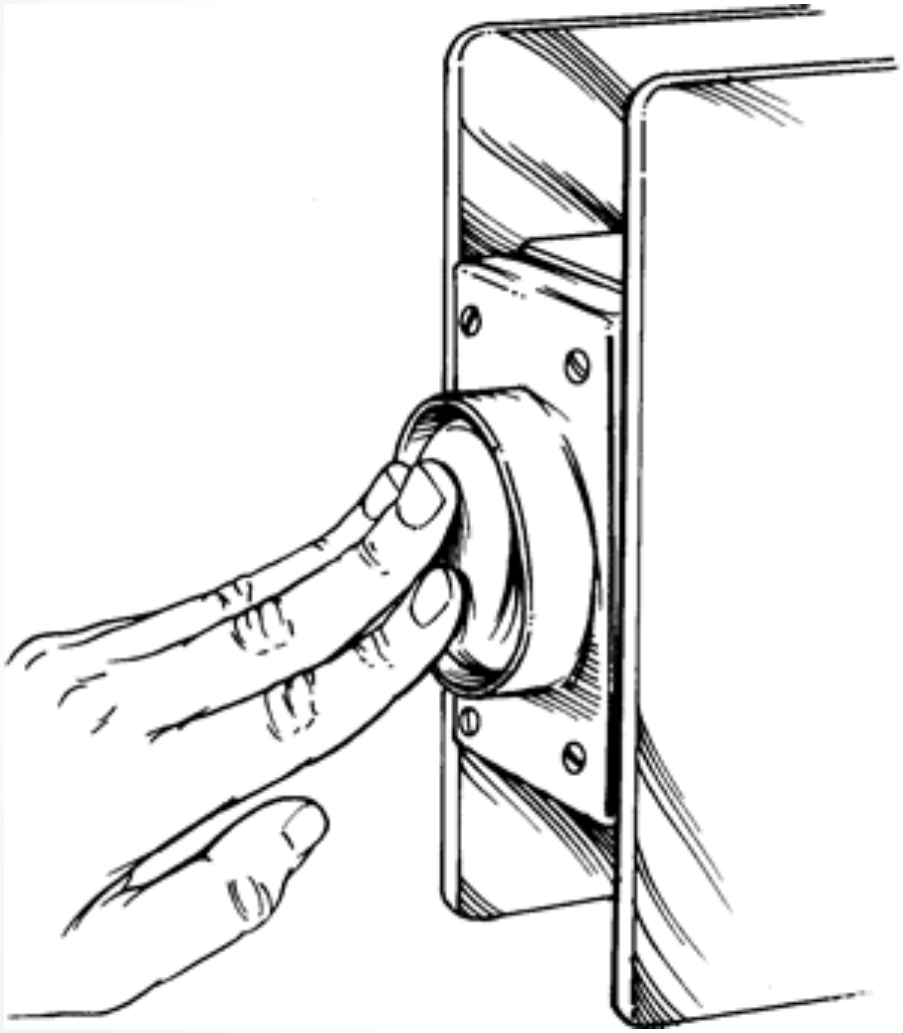
Can I transport quickly and smoothly (e.g. from boat to train to truck)


Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Can I migrate smoothly and quickly
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop

# Контейнеры до Docker



# Контейнеры после Docker ….

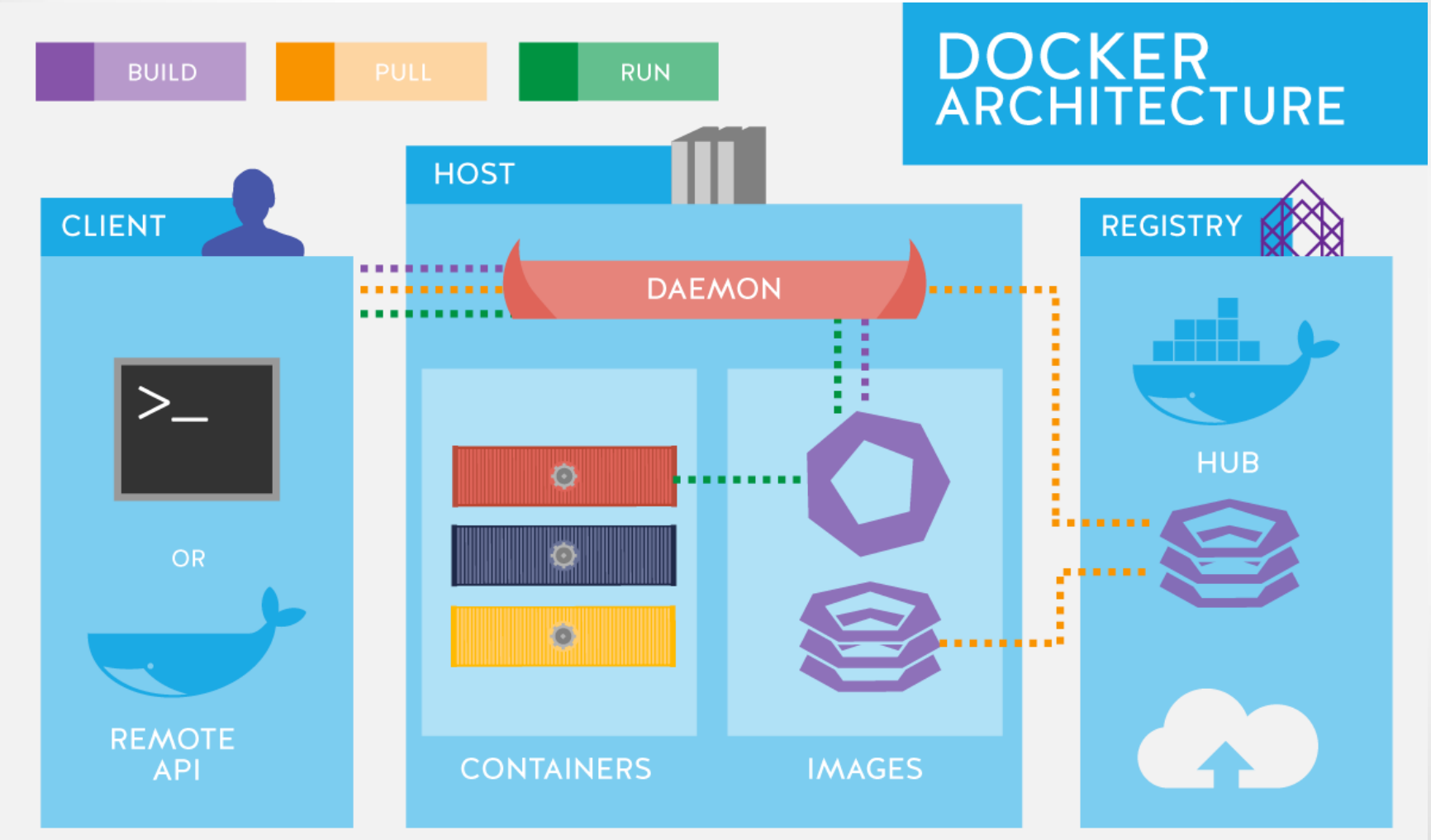


# Из чего состоит Docker?

- Docker: платформа виртуализации с открытым кодом;

- Docker Hub: платформа-как-сервис для распространения и управления docker контейнерами.

# Архитектура Docker



# Внутри docker-a

- образы (images)
- реестр (registries)
- контейнеры

# Как работает Docker?

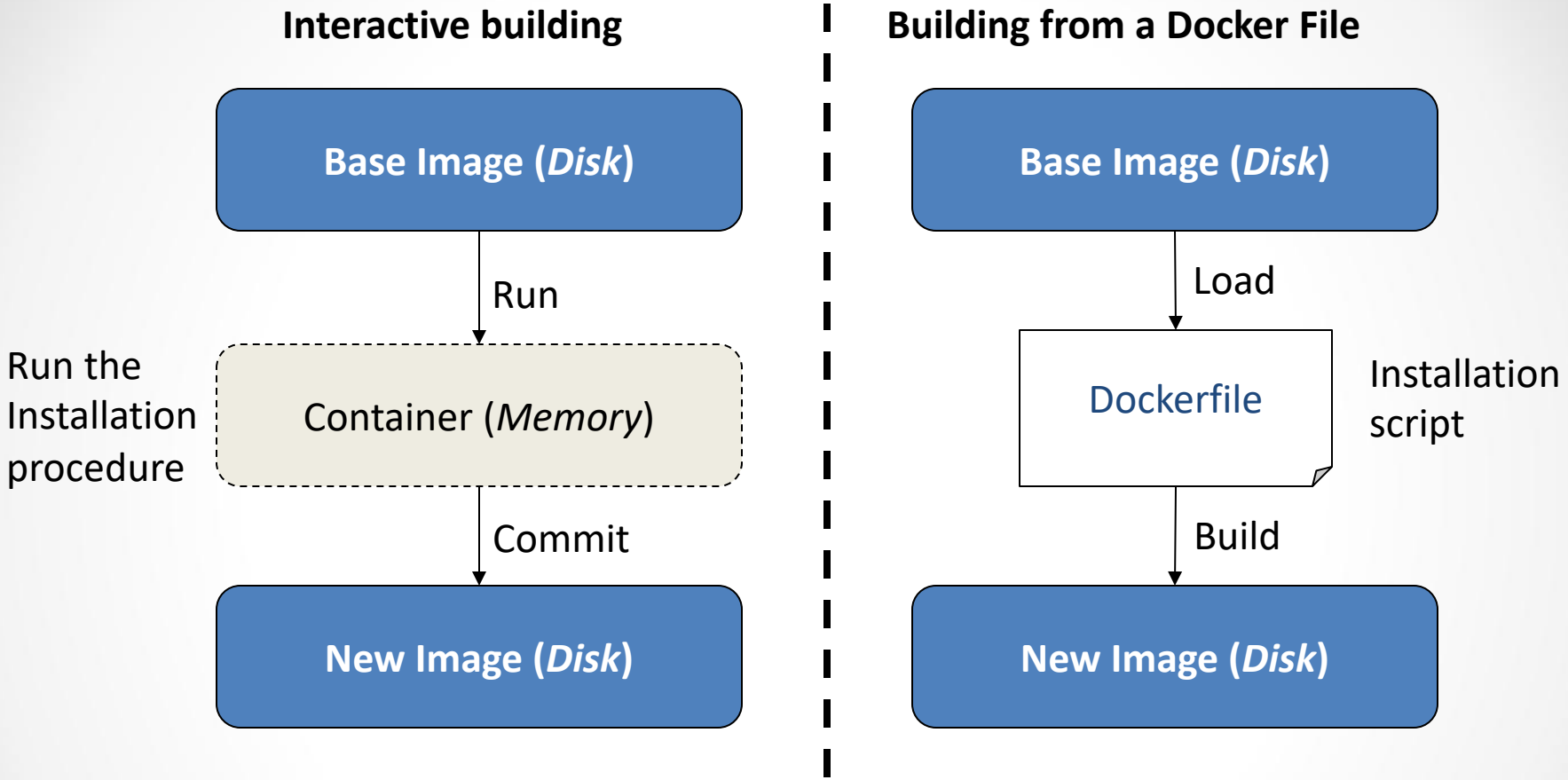
- Каждый образ состоит из набора уровней.
- Docker использует union file system для сочетания этих уровней в один образ.
- В основе каждого образа находится базовый образ.
- использовать образы как базу для создания новых образов.

# Как создать Docker образ?

- запуск команды
- добавление файла или директории
- создание переменной окружения
- указания что запускать, когда запускается контейнер этого образа

- инструкции хранятся в файле Dockerfile. Docker считывает это Dockerfile выполняет эти инструкции, и возвращает конечный образ.

# Как создать Docker образ? (2)



# Как работает контейнер?

- состоит из операционной системы, пользовательских файлов и метаданных.
- Docker образ доступен только для чтения.
- Когда docker запускает контейнер, он создает уровень для чтения/записи сверху образа

# Как запустить простой контейнер?

- $ sudo docker run -i -t ubuntu /bin/bash

# Docker Workflow

- скачивает образ
- создает контейнер
- инициализирует файловую систему и монтирует read-only уровень
- инициализирует сеть/мост
- Установка IP адреса
- Запускает указанный процесс
- Обрабатывает и выдает вывод вашего приложения

# Как он это делает?

- Пространство имен (namespaces)
- Control groups (контрольные группы)
- Union File System

# Пространство имен(namespaces)

- **pid:** для изоляции процесса;
- **net:** для управления сетевыми интерфейсами;
- **ipc:** для управления IPC ресурсами. (ICP: InterProccess Communication);
- **mnt:** для управления точками монтирования;
- **utc:** для изолирования ядра и контроля генерации версий(UTC: Unix timesharing system).

# Выводы

- Стейтлес
- Чистый (pure)
- Ленивый
- Декларативный
- Функциональный
- Строгий

# Docker + Mininet

# Conainernet

- Containernet — это ответвление эмулятора сети Mininet, позволяющее использовать контейнеры Docker в качестве хостов в эмулируемых сетевых топологиях.
- Предоставляет интересные функции для создания сетевых/облачных эмуляторов и испытательных стендов. Одним из примеров этого является эмулятор инфраструктуры multi-PoP NFV, который был создан в рамках проекта SONATA-NFV и теперь является частью проекта OpenSource MANO (OSM).
- Containernet активно используется исследовательским сообществом, уделяя особое внимание экспериментам в области облачных вычислений, туманных вычислений, виртуализации сетевых функций (NFV) и мобильных граничных вычислений (MEC).

# Пример создания топологии

```
          - (c)-
         |      |
(d1) - (s1) - (s2) - (d2)
"""
from mininet.net import Containernet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import info, setLogLevel
setLogLevel('info')

net = Containernet(controller=Controller)
info('*** Adding controller\n')
net.addController('c0')
info('*** Adding docker containers using ubuntu:trusty images\n')
d1 = net.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = net.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
info('*** Adding switches\n')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
info('*** Creating links\n')
net.addLink(d1, s1)
net.addLink(s1, s2, cls=TCLink, delay='100ms', bw=1)
net.addLink(s2, d2)
info('*** Starting network\n')
net.start()
info('*** Testing connectivity\n')
net.ping([d1, d2])
info('*** Running CLI\n')
CLI(net)
info('*** Stopping network')
net.stop()
```

# Запуск и взаимодействие

- Containernet требует прав sudo для настройки эмулируемой сети, описанной сценарием топологии:

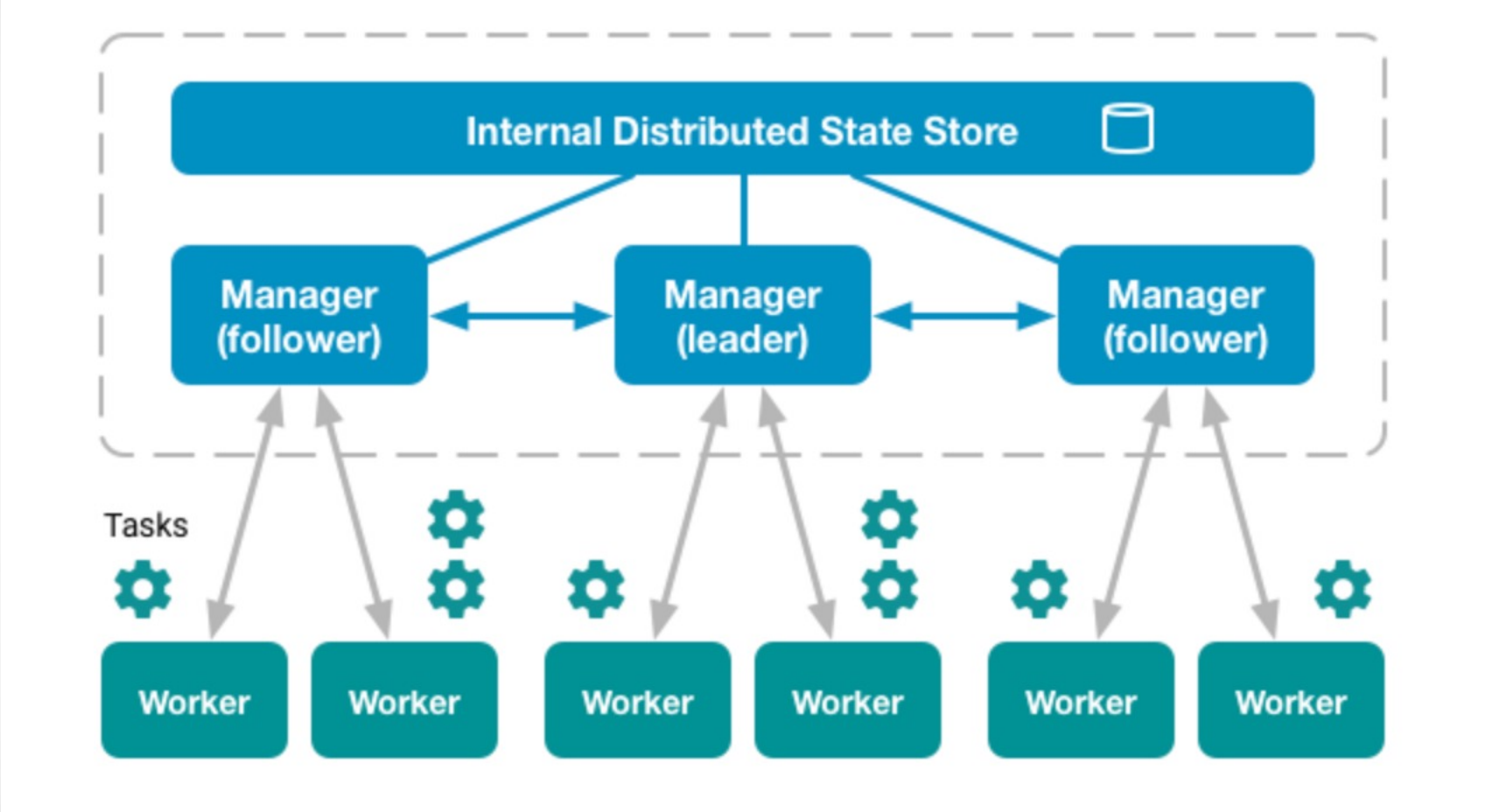
```
sudo python containernet_example.py
```

- После запуска эмулируемой сети вы можете взаимодействовать с задействованными контейнерами через интерактивный интерфейс командной строки Mininet, как показано на примере:

```
containernet> d1 ping -c3 d2
PING 10.0.0.252 (10.0.0.252) 56(84) bytes of data.
64 bytes from 10.0.0.252: icmp_seq=1 ttl=64 time=200 ms
64 bytes from 10.0.0.252: icmp_seq=2 ttl=64 time=200 ms
64 bytes from 10.0.0.252: icmp_seq=3 ttl=64 time=200 ms

--- 10.0.0.252 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 200.162/200.316/200.621/0.424 ms
containernet>
```

# Обзор Swarm mode



# Обзор Swarm mode

- **Cluster management integrated with Docker Engine:** Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. You don’t need additional orchestration software to create or manage a swarm.
- **Decentralized design:** Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine. This means you can build an entire swarm from a single disk image.
- **Declarative service model:** Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack. For example, you might describe an application comprised of a web front end service with message queueing services and a database backend.
- **Scaling:** For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

- **Desired state reconciliation:** The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager creates two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
- **Multi-host networking:** You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
- **Service discovery:** Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.

- **Load balancing:** You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.
- **Secure by default:** Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
- **Rolling updates:** At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll back to a previous version of the service.

- Open a terminal and ssh into the machine where you want to run your manager node. If you use Docker Machine, you can connect to it via SSH using the following command:

```
$ docker-machine ssh manager1
```

- Run the following command to create a new swarm:

```
$ docker swarm init --advertise-addr <MANAGER-IP>
```

```
$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a m

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wx
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager'
```

- the --advertise-addr flag configures the manager node to publish its address as 192.168.99.100. The other nodes in the swarm must be able to access the manager at the IP address.
- The output includes the commands to join new nodes to the swarm. Nodes will join as managers or workers depending on the value for the --token flag.

- Run docker info to view the current state of the swarm:

```
$ docker info

Containers: 2
Running: 0
Paused: 0
Stopped: 2
  ...snip...
Swarm: active
  NodeID: dxn1zf6l61qsb1josjja83ngz
  Is Manager: true
  Managers: 1
  Nodes: 1
  ...snip...
```

- https://habr.com/ru/company/redmadrobot/blog/318866/
- https://docs.docker.com/swarm/overview/
- https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/

# Спасибо за внимание!