ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Лекция 12. Docker контейнеры

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Основные темы

- Контейнеры
- Архитектура Docker
- Сетевые режимы
- Хранение данных
- Docker Swarm

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Словарь

- Образ
- Контейнер
- Сеть
- Том
- CoW
- Swarm
- Manager
- Worker
- Service
- Task
- Raft

ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Контейнеры

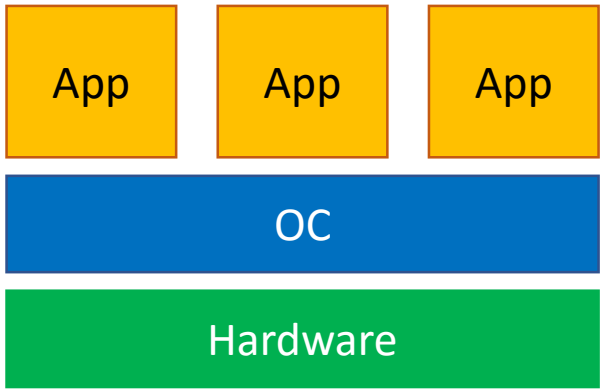
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

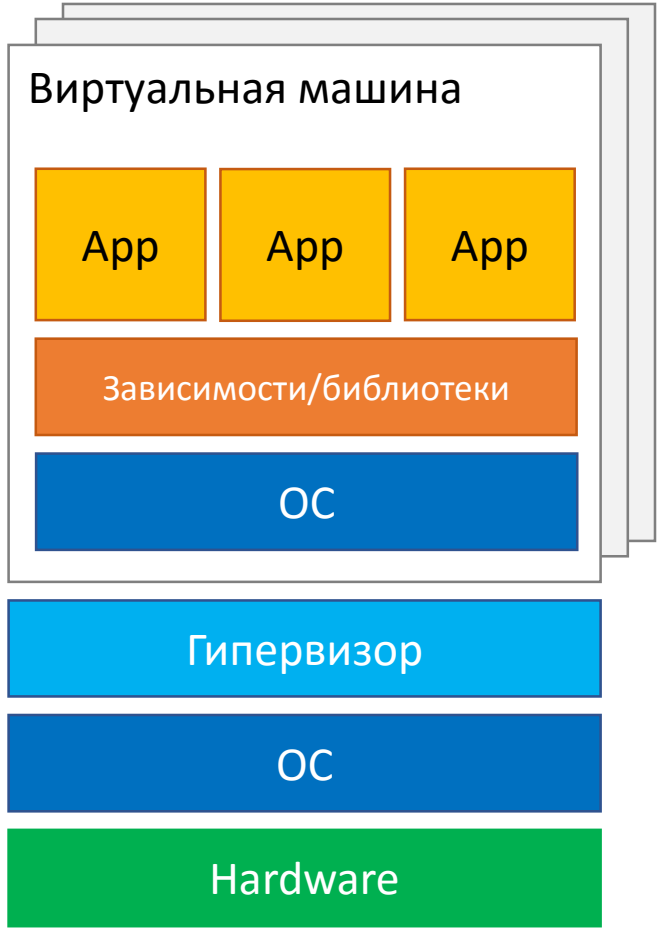
Контейнер

- Контейнер – изолированная среда для запуска сервисов/приложений, которая ограничивает взаимодействии с только ей приписанными ресурсами
- Приложения запускаются в контейнерах, которые имеют свое изолированное пространство имен файловой системы, сети, ограничены вычислительными ресурсами и др.
- На одном хосте могут быть запущены множество контейнеров со своими приложениями
- Как правило, один контейнер отвечает за один сервис/приложение, например, веб-сервер, СУБД

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

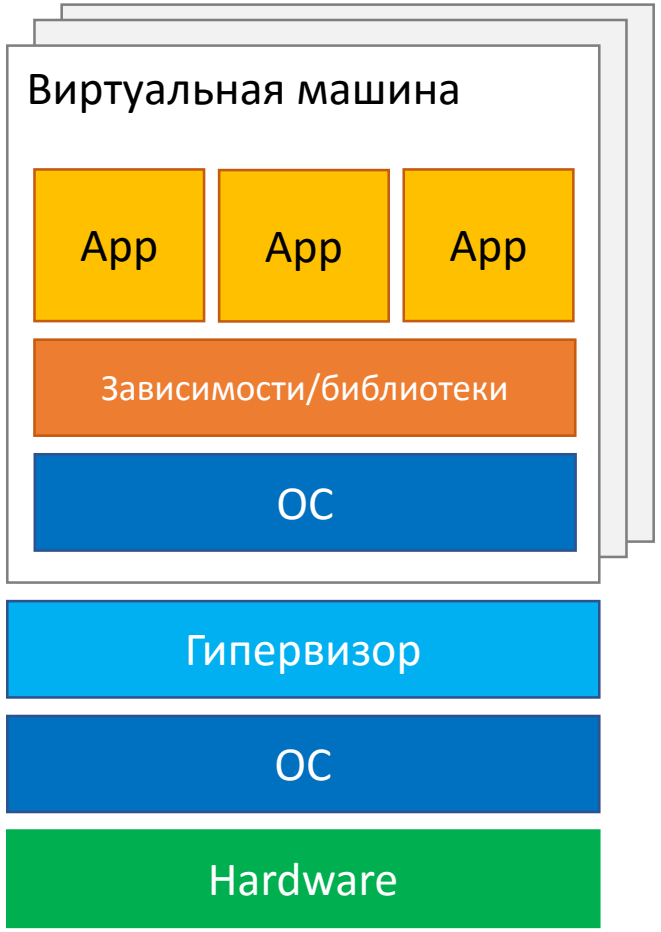
Виртуальная машина

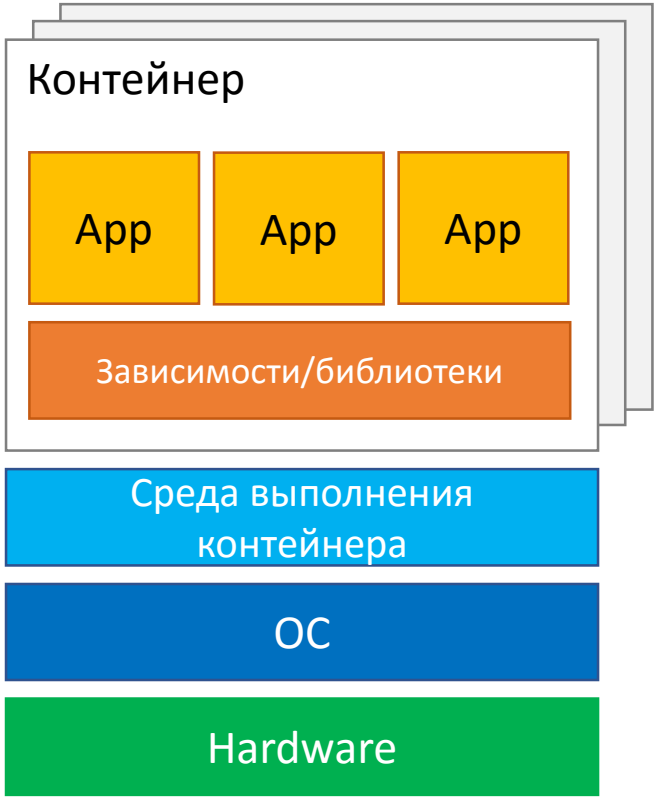




ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Контейнер





ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Why not

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Виды контейнеров

- Основанные на cgroup
  - LXC
  - systemd-nspawn
  - **Docker**
  - rkt, runC
- Другие:
  - OpenVZ
  - Jails/Zones

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Контейнеры на основе cgroup

- **cgroup** – ограничивает вычислительные ресурсы
  - memory
  - cpu, cpuset
  - blkio
  - net_cls, net_prio
  - devices
  - freezer
  - pids
- **namespaces** – ограничивает видимость
  - cgroup
  - pid
  - network
  - user
  - ipc
  - mount
  - uts

ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Docker

Docker Engine

Docker Engine – клиент-серверное приложение со следующими компонентами:

- **Docker daemon**

Сервер слушает Docker API запросы и управляет Docker **объектами*** (команда dockerd). Docker daemon также может взаимодействовать с серверами на других хостах для управления сервисами (services) в swarm режиме

- **REST API**

Интерфейс для взаимодействия с сервером

- **Command line interface** (CLI)

Клиент (команда docker). Может взаимодействовать с несколькими серверами

- **Registry**

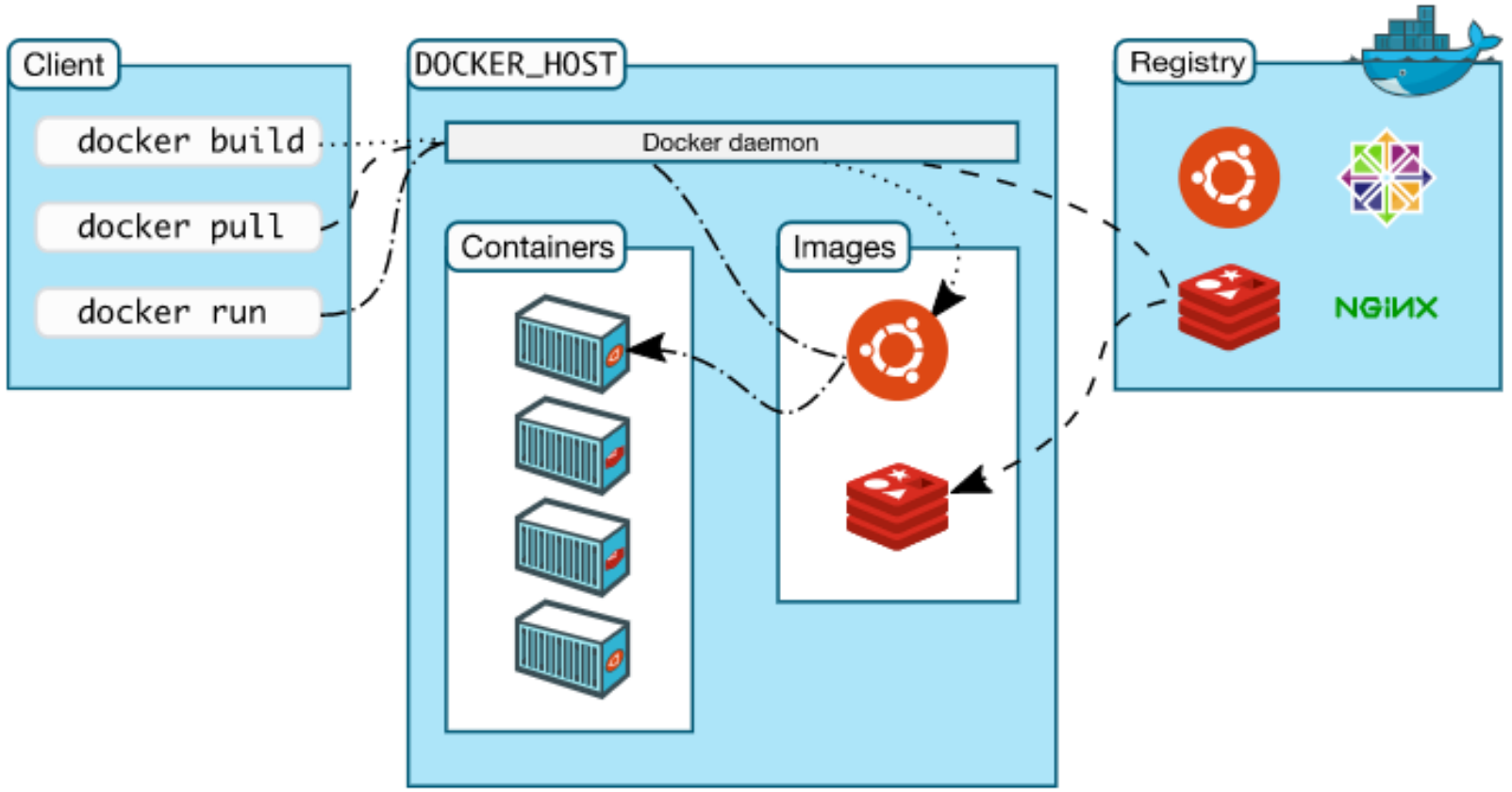
Хранилище образов

***Docker объекты**:*

- Образы (Images)
- Контейнеры (containers)
- Сети (networks)
- Тома (volumes)
- и др.

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Docker контейнер



Docker объекты

Образ (image) – последовательный набор инструкций для создания контейнера. Представляется в виде неизменяемого шаблона. Образы можно строить на базе других образом за счет добавления новых инструкций. Например, чтобы получить новый образ, к готовому ubuntu образу можно добавить инструкции по установке и запуску Apache веб-сервера с собственным приложением.

Контейнер (container) – выполняемый экземпляр образа с конфигурируемой изолированной средой. Контейнер можно создать, запустить, остановить, переместить, удалить посредством Docker API или CLI. При удалении контейнера изменения в файловой системе не сохраняются. К контейнеру можно подключить сеть, постоянное хранилище данных и др.

Сеть (network) предназначена для организации коммуникаций между контейнером и хостом, между контейнерами. Сетевой режим определяется драйвером (встроенные, *host, bridge, overlay, none, mavclan*)

Том (volume) применяется для постоянного хранения данных. Данные сохраняются после удаления контейнера и могут быть повторно использованы

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Образы, слои, стратегия Copy-on-Write

- Docker образ состоит из набора слоев
- Каждый слой представляет инструкцию в файле образа Dockerfile
- Каждый слой – разница между предыдущим и текущим состоянием
- Когда создается новый контейнер, добавляется новый слой с возможностью записи поверх остальных нижележащих – «container layer»
- Все изменения, такие как запись нового файла, изменение и удаление файлов, записываются в этот слой
- Docker использует **storage driver** для управления содержанием слоев образа и слоя контейнера
- Существует несколько реализаций драйвера (*aufs, overlay, overlay2, btrfs, zfs*), но каждая использует стек слоев и стратегию copy-on-write

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

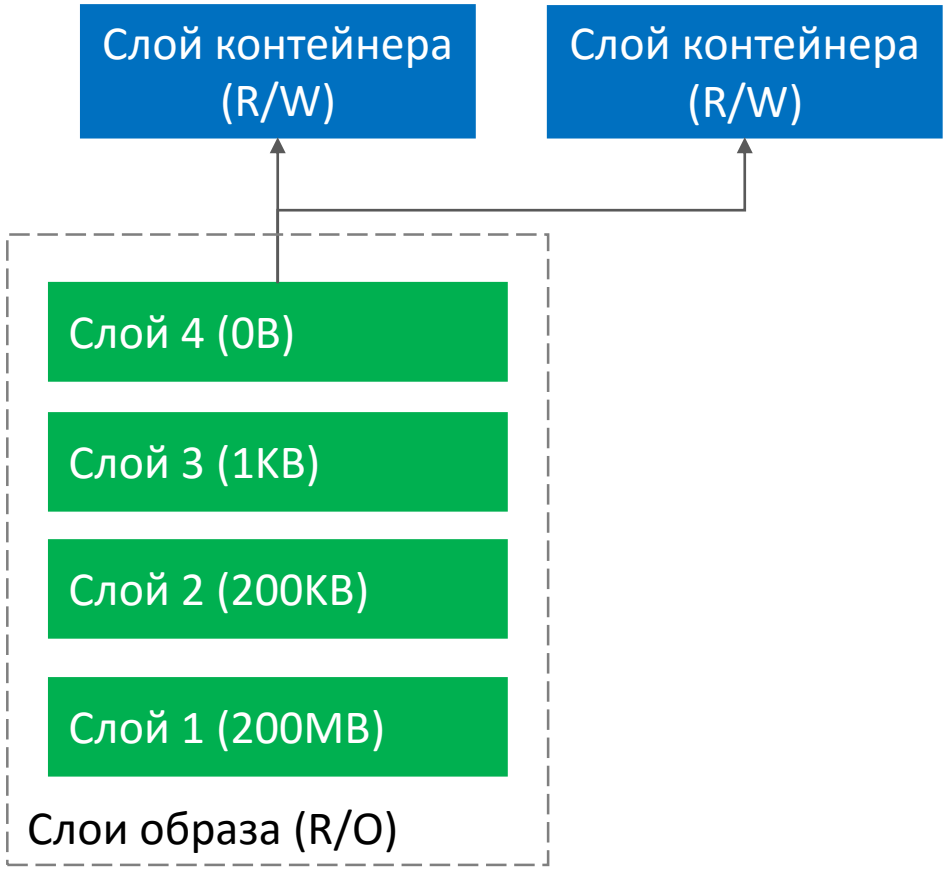
Copy-on-Write

- Все изменения хранятся в слое для записи (слой контейнера)
- Когда контейнер удаляется, слой контейнера также удаляется, а образ остается неизменным
- Так как каждый контейнер имеет свой собственный слой для записи, то множество контейнеров могут использовать один и тот же образ
- Если файл или директория существует в нижележащих слоях образа и другой слой, в том числе слой контейнера, хочет получить доступ к ним, то он просто использует существующий файл
- Если необходимо изменить существующий файл (при построении образа или в работающем контейнере), то он копируется в слой, который требует изменений

Пример слоя контейнера для записи

Dockerfile

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



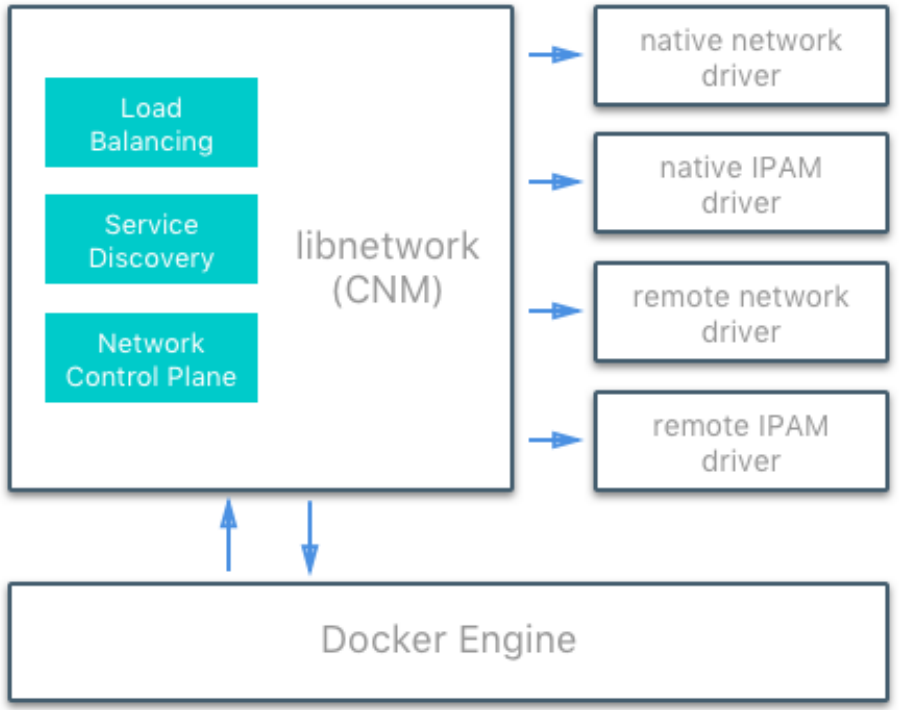
ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Сетевые режимы

Модель управления сетью

- **Load Balancing** – балансировка нагрузки, например, при обращении к сервису (service) с множеством одинаковых задач (tasks), распределенным по нескольким хостам
- **Service Discovery** – аналог DNS сервера для разрешения имен контейнеров, сервисов и пр. (поиск IP по имени)
- **Network Control Plane** управляет состоянием сетей в swarm режиме
- Драйвер сети (**Network Driver**) – реализация Docker сети контейнеров

  Стандартные (native): host, bridge, overlay, macvlan и none
- Драйвер управления IP адресами (**IPAM driver**) – назначает подсети и IP адреса для сетей и интерфейсов контейнеров, если они не определены.



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

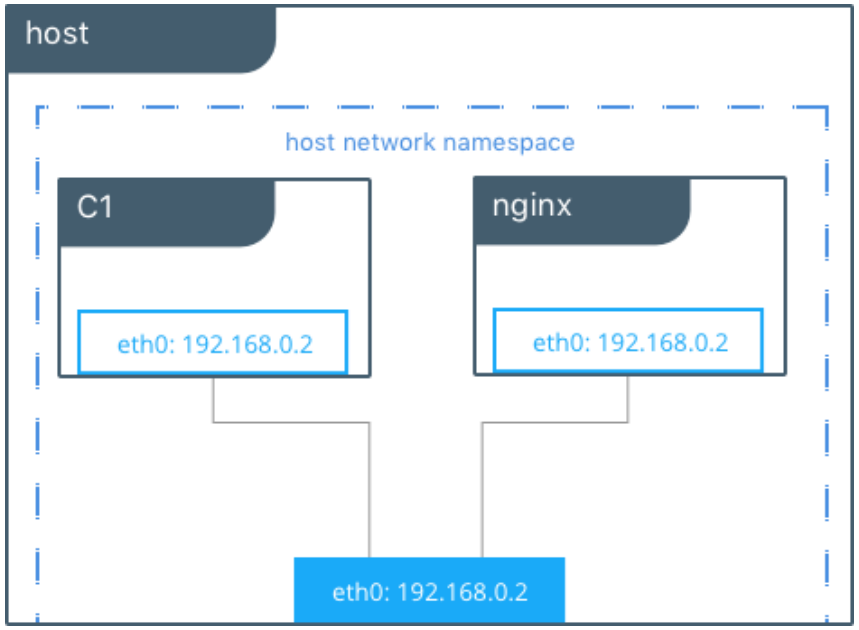
Стандартные сетевые драйверы

- **Host.** Контейнер использует сетевой стек хоста. В этом случае нет разделения пространства имен и все интерфейса хоста могут быть использованы напрямую контейнером
- **Bridge.** Создает Linux bridge на хосте, который управляется Docker’ом. По умолчанию контейнеры в bridge могут общаться между собой. Также может быть настроен внешний доступ к контейнерам
- **Overlay.** Создает сеть, которая поддерживает взаимодействие контейнеров между несколькими хостами. Используется комбинация локальных Linux мостов (bridges) и VXLAN для реализации коммуникации поверх физической сети
- **Macvlan.** Используется MACVLAN bridge режим для установки соединения между интерфейсами контейнеров и интерфейсом родительского хоста. Используется для назначения контейнерам IP адресов физической сети.
- **None.** Создает для контейнера сетевой стек и сетевое пространство имен, но не настраивает интерфейс контейнера. Без дополнительного конфигурирования контейнер полностью изолирован от сетевого стека хоста

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Режим Host

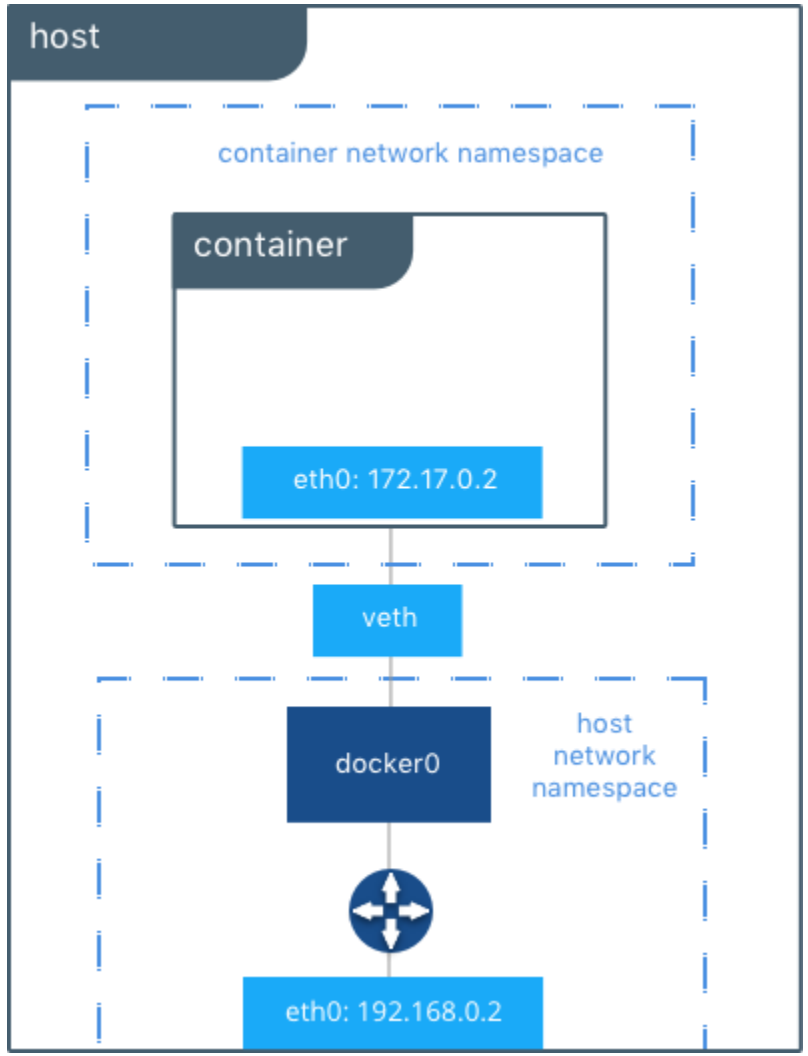
- С host драйвером контейнеры принадлежат сетевому пространству имен хоста и используют сетевые интерфейсы и IP стек хоста.
- Все контейнеры в этом случае могут взаимодействовать друг с другом по интерфейсам хоста, как обычные процессы.
- Соответственно, два контейнера не могут иметь один TCP порт
- Работает только с Linux хостами



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Режим Bridge

- По умолчанию при создании контейнера на Linux хосте bridge драйвер создает Linux bridge называемый docker0
- Таблица маршрутизации обеспечивает соединение между docker0 и eth0 внешней сети, тем самым связывая внутренний контейнер с внешней сетью
- По умолчанию bridge назначает одну подсеть из диапазона 172.[17-31].0.0/16 или 192.168.[0-256].0/200, отличную от интерфейса хоста

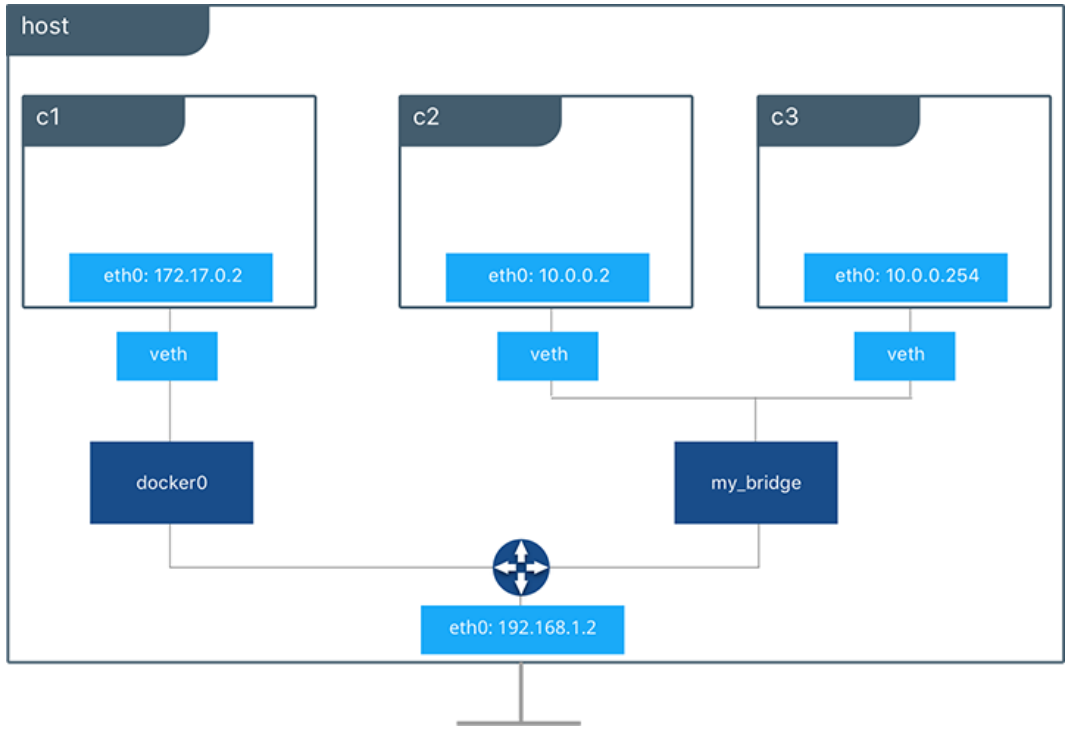


ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Bridge

- Можно создать собственную сеть с bridge драйвером и указать подсеть и IP контейнера

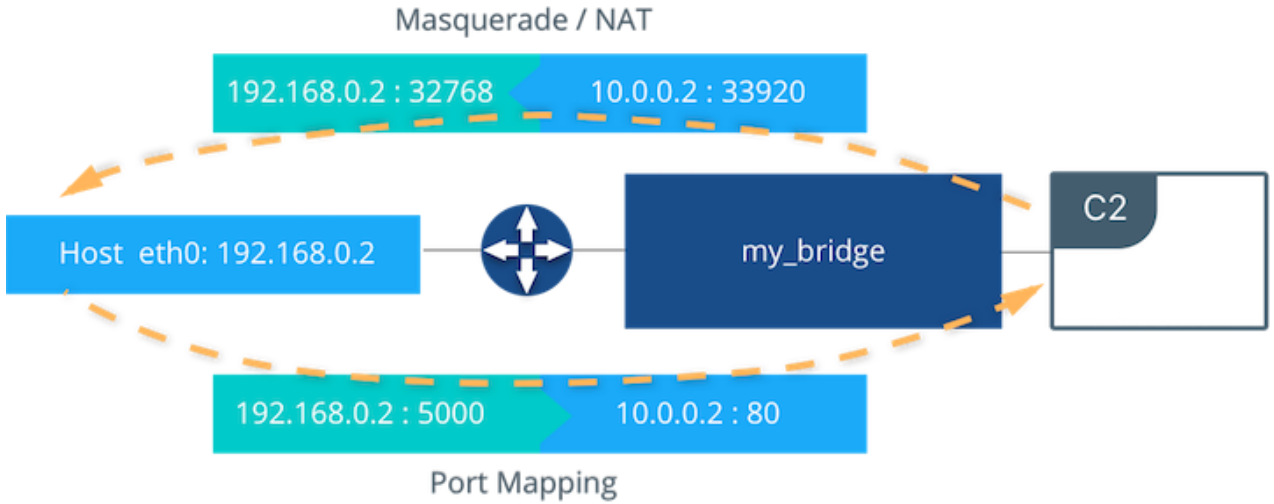
- По умолчанию все контейнеры в одной Docker сети соединены друг с другом и могут использовать для связи все порты



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Bridge. Внешний трафик

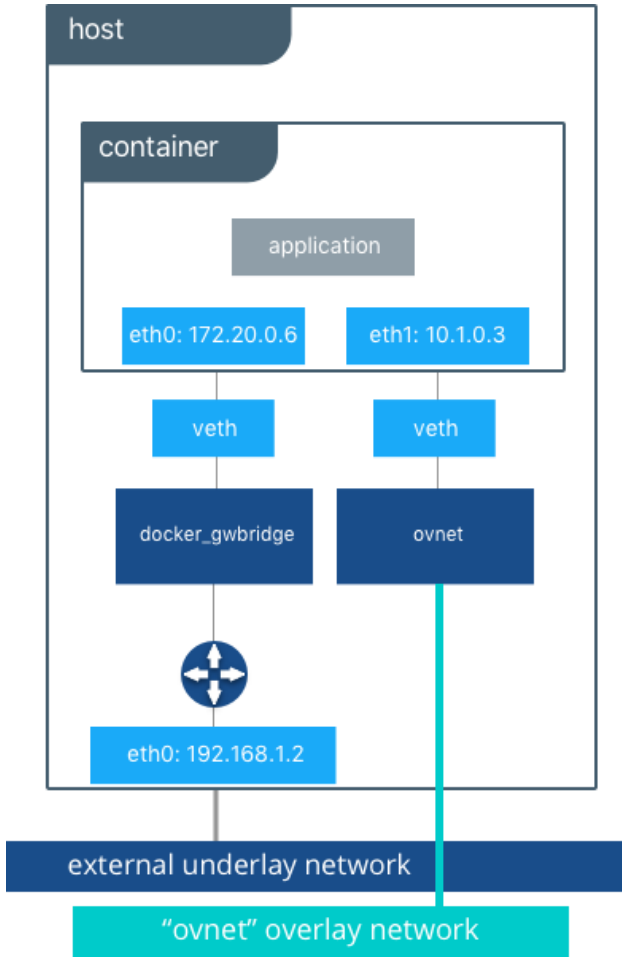
- Коммуникация между различными Docker сетями и входящий трафик контейнера внешний по отношению к Docker ограничены брандмауэром
- Доступ входящего трафика обеспечивается за счет явной публикации портов, связывая интерфейсы хоста с интерфейсами контейнера
- Для исходящего трафика контейнера используется NAT с портом из диапазона 32768-60999. Соответственно, ответы на исходящие запросы не блокируются



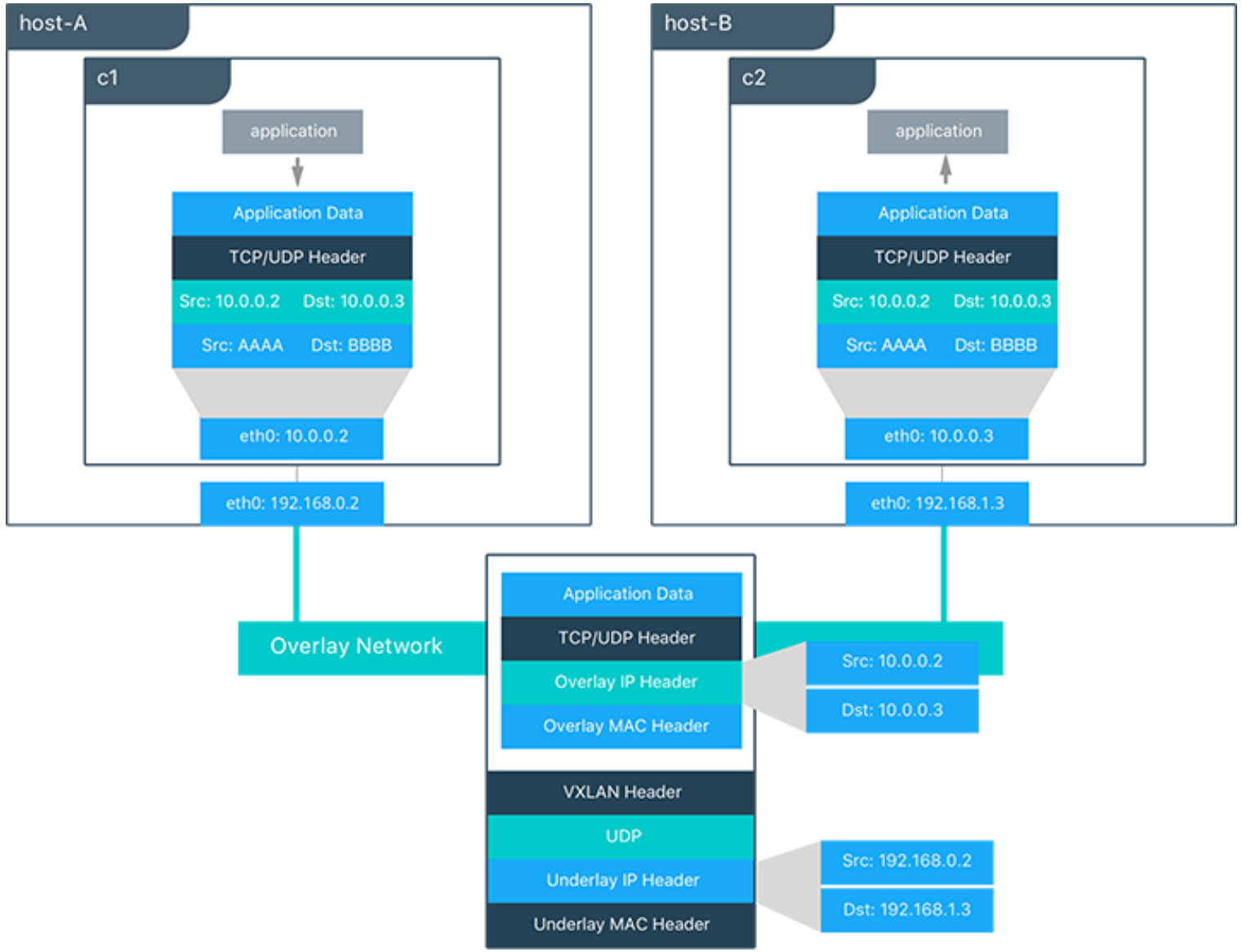
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Режим Overlay

- Overlay драйвер позволяет создать сеть, которая охватывает несколько хостов
- Каждый контейнер имеет по крайней мере два интерфейса, которые соединяют его с overlay и docker_gwbridge
- Overlay драйвер использует VXLAN туннель для передачи данных, что позволяет отделить сеть контейнеров от физической сети
- Overlay инкапсулирует трафик контейнеров в VXLAN заголовок, который далее передается по физической L2/L3 сети
- VXLAN определяется как MAC-в-UDP инкапсуляция, которая помещает L2 фрейм внутрь IP/UDP заголовка
- В overlay сеть могут быть включены как Linux, так и Windows хосты



Пример overlay

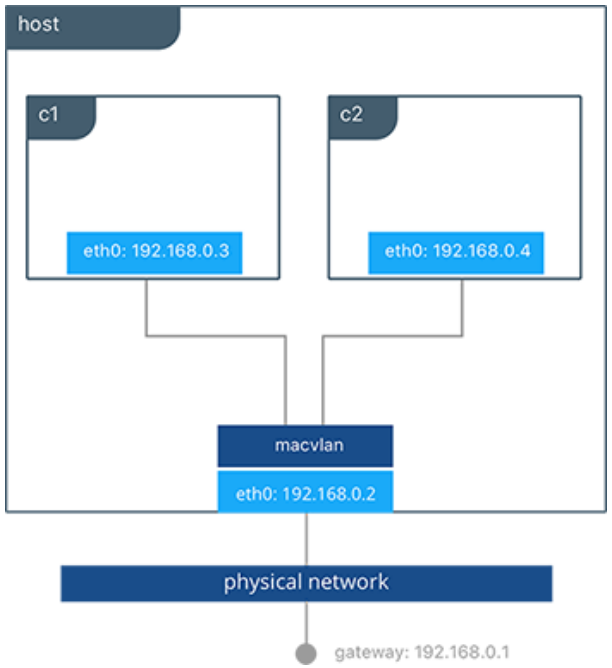


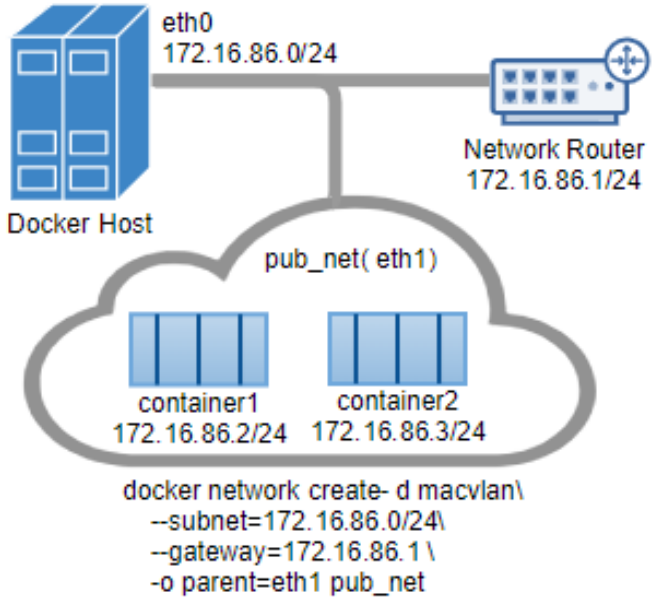
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Режим Macvlan

- Macvlan драйвер обеспечивает возможность прямого доступа между контейнерами и физической сетью

- Это позволяет контейнерам получать IP адреса подсети физической сети и использовать её шлюз

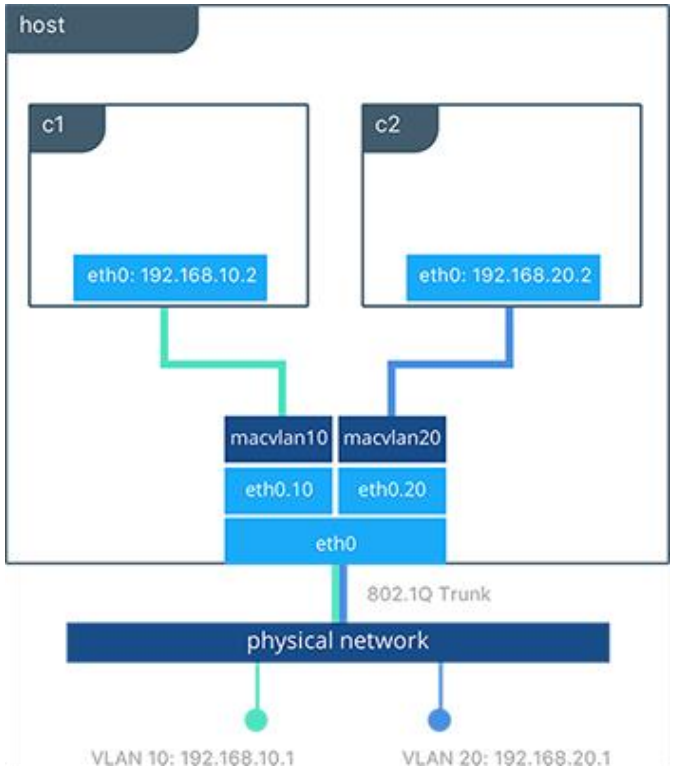


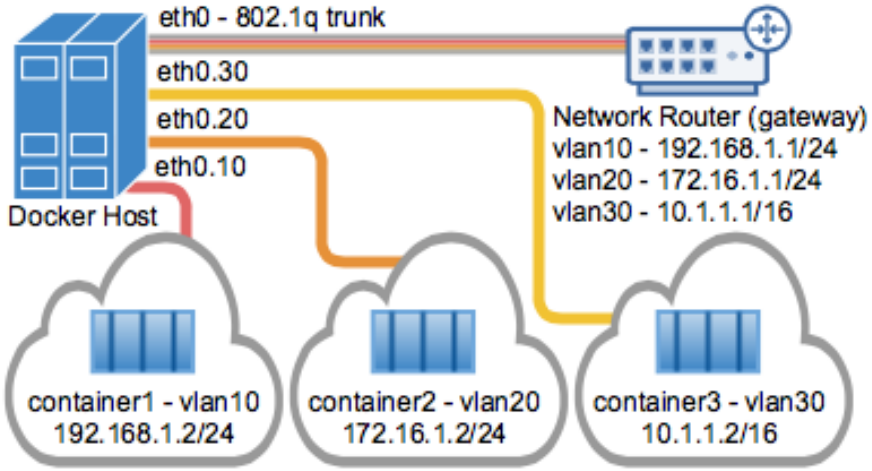


ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Режим Macvlan с VLAN Trunking

- Macvlan можно использовать с VLAN Trunking для сегментации трафика контейнеров на L2
- В этом случае macvlan создает sub-interfaces и соединяет их с интерфейсами контейнеров





ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Хранение данных

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Способы хранения данных

- Storage драйвер позволяет создавать данные в слое контейнера для записи. Эти файлы не будут сохранены после удаления контейнера. Скорость чтения и записи ниже чем у нативной файловой системы

- Docker предоставляет две опции для сохранения данных:

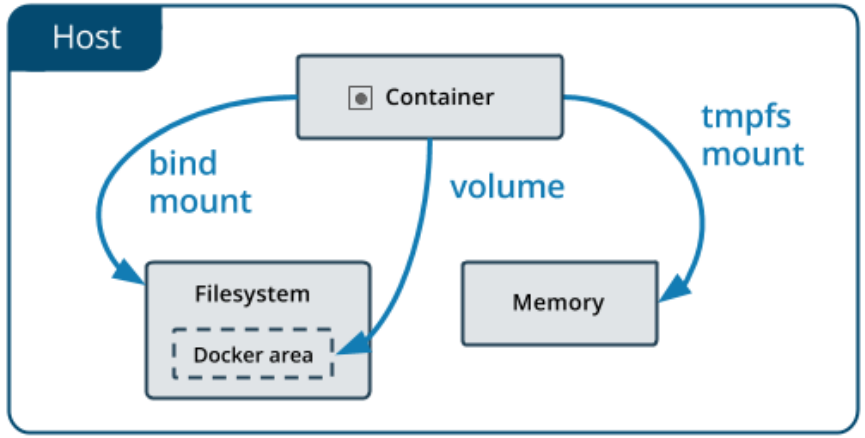
  - Volume
  - Bind mound

- Для временного хранения также можно использовать tmpfs mount (Linux)

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Постоянное хранение данных

- **Volume.** Сохраняет данные как часть файловой системы хоста, которая управляется Docker (/var/lib/docker/volumes/ в Linux). Не-Docker процессы не могут изменять эту часть файловой системы. Это рекомендуемый способ для сохранения данных
- **Bind mount.** Сохраняет данные в произвольной части файловой системы хоста, в том числе можно монтировать важные системные файлы и директории хоста. Не-Docker процессы могут изменить их в любой момент
- **Tmpfs.** Хранит данные в оперативной памяти хоста и никогда не записываются в его файловую систему
- Существуют volume дайверы для хранения данных в AWS S3, NFS и пр.



ИНСТИТУТ
ЦИФРОВОГО
ОБРАЗОВАНИЯ
МГПУ

Docker Swarm

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Docker Swarm

- **Swarm** режим предназначен для развертывания и управления задачами/контейнерами на множестве хостов

- **Swarm** состоит из набора Docker хостов, которые работают в swarm режиме и выполняют роль **manager**´ов и **worker’**ов

- **Swarm** обеспечивает:

  - Отказоустойчивость развернутых приложений
  - Масштабируемость (для Docker Swarm 1.12: ~2,300 узлов при 95,000 задачах)
  - Встроенную безопасную коммуникацию с использованием TLS

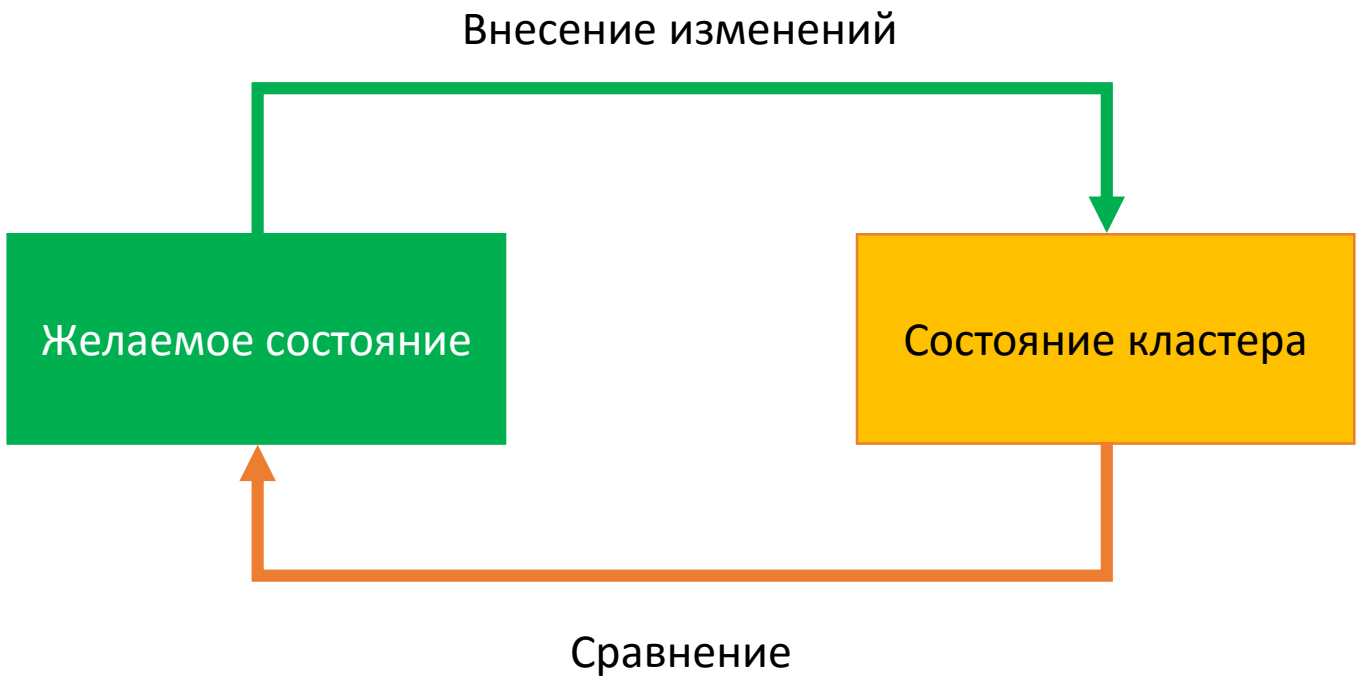
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Компоненты Docker Swarm

- **Node** – экземпляр Docker Engine в swarm. Несколько экземпляров могут быть на одном хосте (физическом или виртуальном). Наилучший вариант, когда один экземпляр соответствует одному хосту
- Для развертывания приложения в swarm кластере, на **manager** узел подается определение сервиса (**service**)
- **Manager** распределяет единицы выполнения, **tasks**, рабочим узлам (**worker**´ам)
- **Worker** получает и выполняет **задачи** от manager'а
- **Manager** поддерживает требуемое состояние сервисов и в случае несоответствия текущего состояния и требуемого предпринимает действия для устранения расхождений. Например, при выходе из строя узла, на котором была запущена задача, manager восстанавливает необходимое количество реплик за счет перезапуска данный задачи на другом доступном узле

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Поддержание требуемого состояния сервиса



Service

- Service – определение задач (tasks) для выполнения на manager или worker узлах.

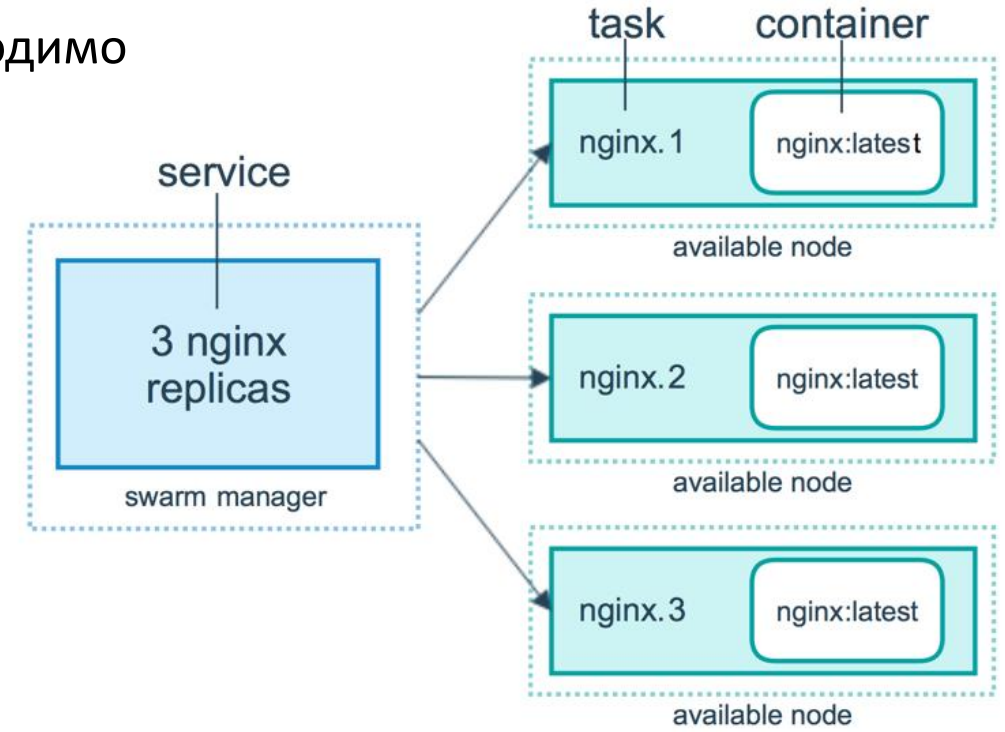
- Определение сервиса включает:
  - Образ контейнера
  - Команды на выполнения внутри контейнера
  - Внешний порт
  - Сеть (overlay)
  - CPU и память (hard и soft лимиты)
  - Предпочтения на размещение
  - Политику обновления
  - Количество реплик
  - и др.

- Режим развертывания
  - Replicated – определяет общее количество задач (реплик), которые необходимо запустить на доступных узлах
  - Global – одна задача на один узел

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

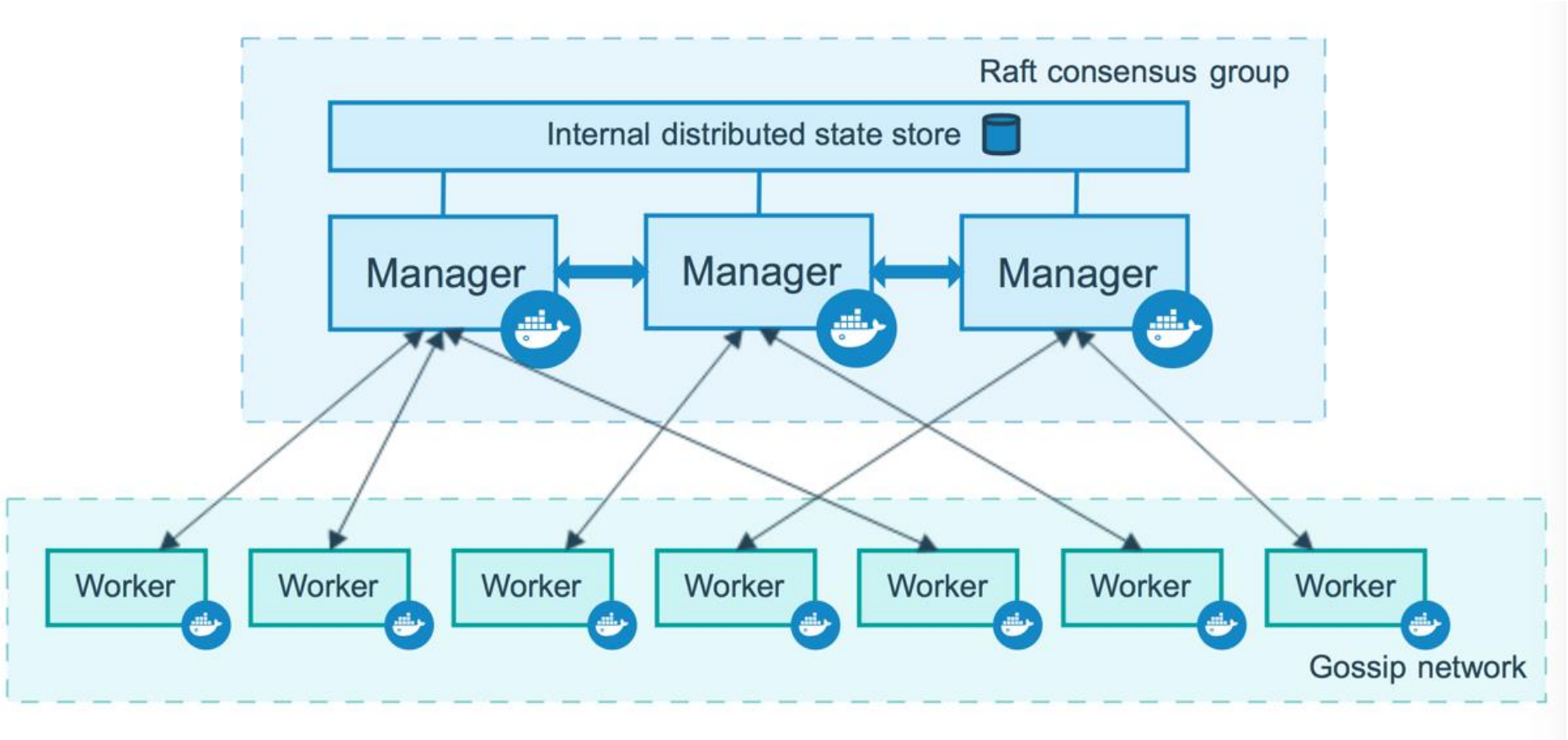
Task

- Сервис определяет количество задач, которые необходимо запустить на кластере
- Контейнер – изолированный процесс
- Одна задача (task) – один контейнер
- Task своего рода слот, куда планировщик помещает контейнер
- Пока контейнер запущен задача находится в рабочем состоянии. Если контейнер вышел из строя или остановлен, задача останавливается.



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

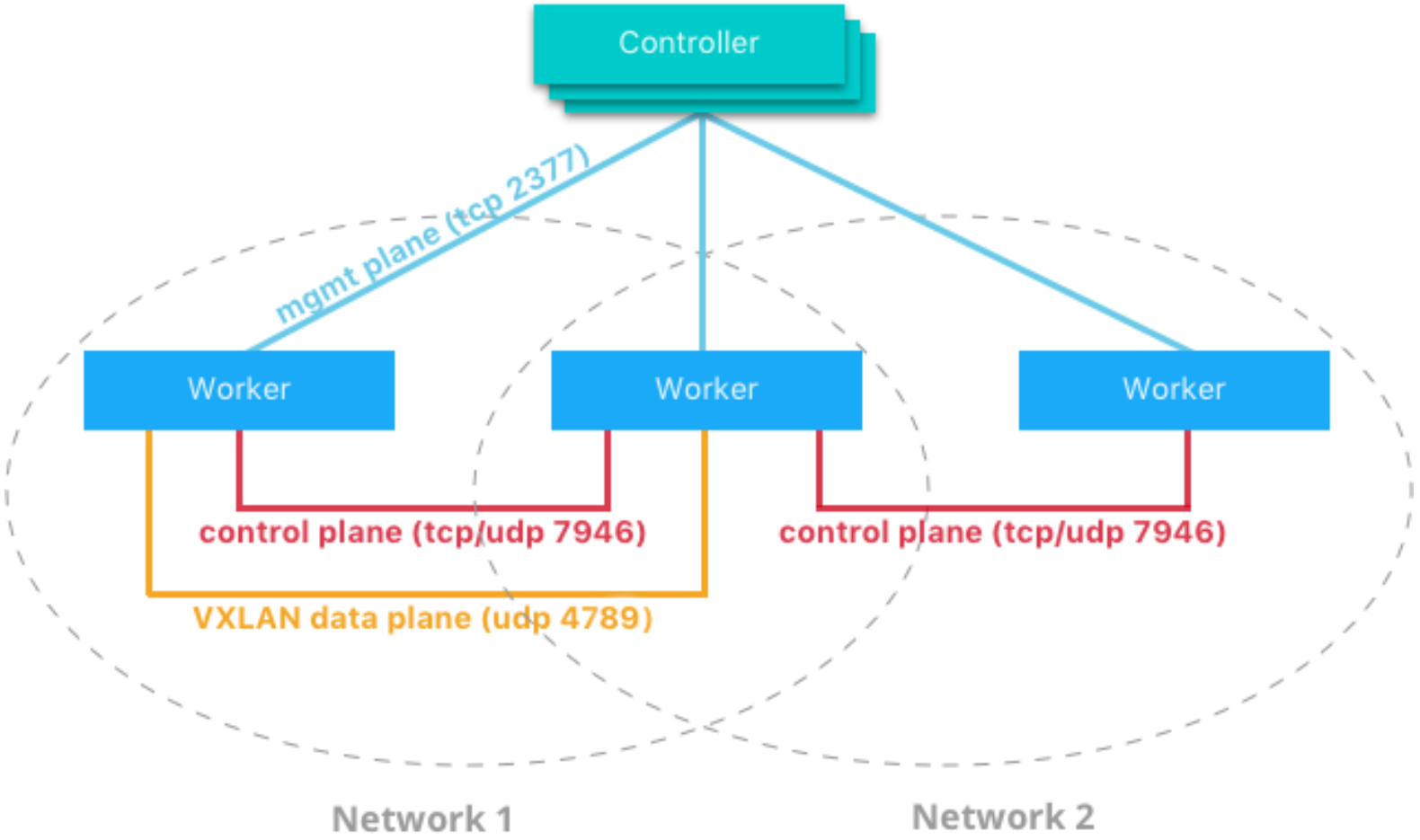
Архитектура



https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Управление сетью



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Коммуникация между Manager’ам

Для поддержания согласованного состояния кластера и информации о всех его запущенных сервисах между всеми узлами используется протокол, реализующий **RAFT** алгоритм.

RAFT отвечает за:

- Выбор лидера
- Согласование данных при чтении/записи между всеми manager узлами с использованием кворума

Серверы	Кворум	Допускает отказы
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Manager-Worker и Worker-Worker

- Коммуникация между **worker’**ами и **manager’**ами осуществляется посредством протокола **gRPC**
- **gPRC** работает поверх HTTP/2
- **Manager** посылает рабочим узлам набор задач для запуска
- **Worker’**ы отправляют **manager’**ам:
  - Статусы назначенных им задач
  - Hearbeats
- Коммуникация между рабочими узлами осуществляется по протоколу **Gossip**

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Manager

Лидер

- Отслеживает состояние кластера (nodes)
- Обрабатывает запросы пользователя на изменение состояний сервисов
- Планирует размещение сервисов (services)
- Отслеживает состояния сервисов (services, tasks)
- Дает команды worker’ам на запуск задач (tasks)

Остальные

- Необходимы в качестве быстрой замены при выходе из строя лидера
- Все API запросы перенаправляются на узел лидера

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Компоненты Manager

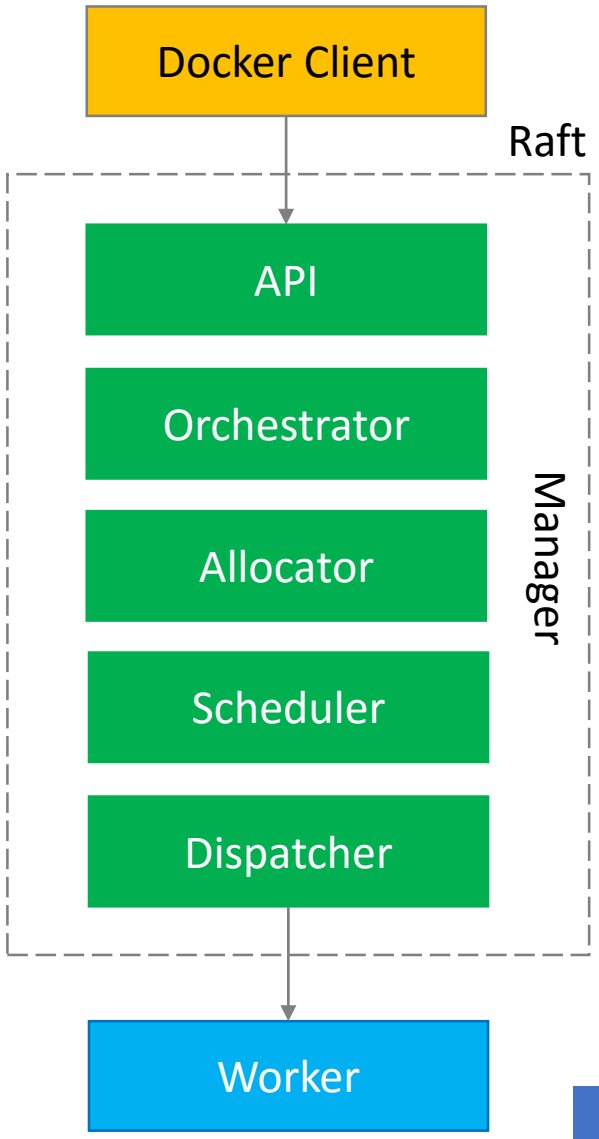
API
Принимает команды и создает новые сервис объекты

Orchestrator
Отслеживает состояния сервисов и в случае несоответствия действительного состояния и желаемого предпринимает действия для достижения желаемого состояния

Allocator
Назначает IP адреса сервисам и задачам

Scheduler
Назначает задачи рабочим узлам

Dispatcher
Дает команды рабочим узлам
Отслеживает состояния рабочих узлов и выполняемых задач



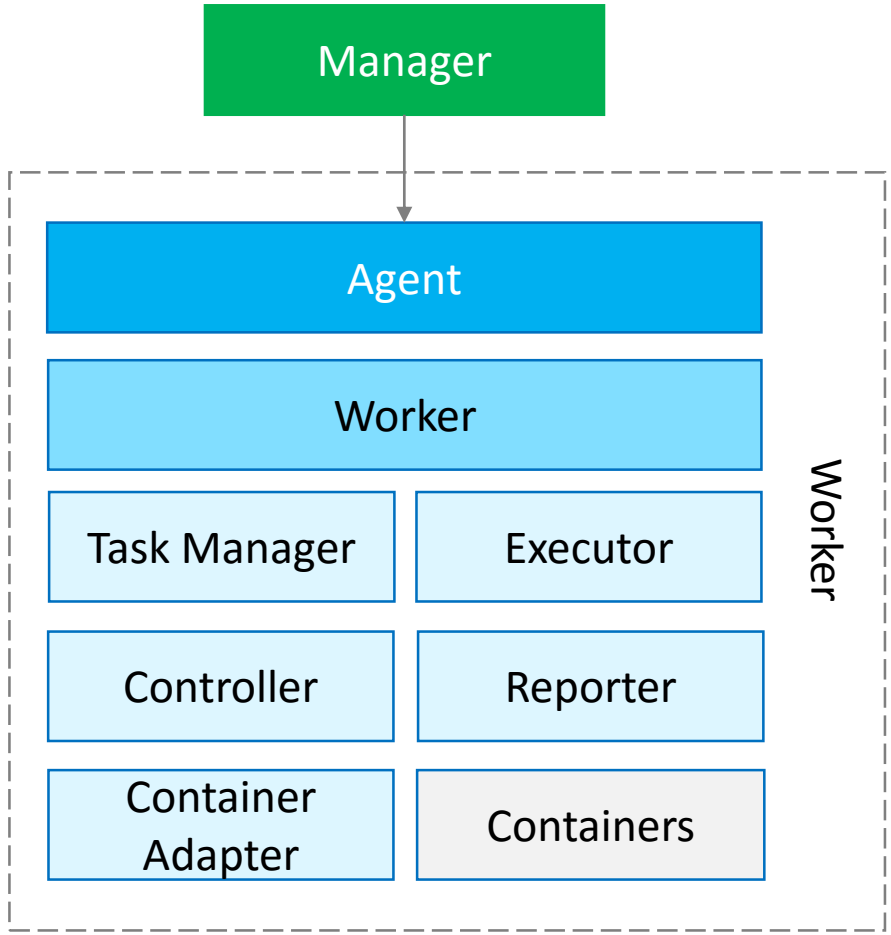
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Worker

- **Рабочий узел** (worker) получает и выполняет задачи от **manager’**а
- **Агент** запускается на каждом **рабочем узле** и информирует о выполняемых задачах на данном узле
- **Рабочий узел** уведомляет **manager** о текущих состояниях выполняемых задач. Такими образом, **manager** может поддерживать требуемое состояние
- По умолчанию **manager** запускает задачи сервиса так же как и **worker.** Можно сконфигурировать так, чтобы **manager** выполнял только свои функции, определив доступность **manager’**а как **DRAIN**

Компоненты Worker

- **Agent**
  Выполняет основные функции узла как часть swarm кластера
  Запускает и передает статусы задач
- **Worker**
  Выполняет основную логику управления задачами и данными (configs/secrets); координирует набор заданий с **executor**´ом
- **Executor**
  Предоставляет котроллеры (controllers) для задач (tasks)
- **TaskManager**
  Управляет всеми задачами (tasks)
- **Controller**
  Контролирует выполнение задачи
- **Reporter**
  Информирует об изменении статуса задачи
- **Container Adapter**
  Выполняет операции над контейнерами (использует client API Docker Engine)



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Примеры

Рассматриваемые сценарии

- Запуск сервиса с одной задачей
- Увеличение количества реплик до 3
- Обновление конфигурации
- Выход из строя рабочего узла

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Создание сервиса

1. Пользователь отправляет определение сервиса с одной задачей в manager API

2. Manager API принимает, проверяет и сохраняет сервис

3. Orchestrator согласует требуемое состоянием (определенное пользователем) с текущим (с тем, что запущено и выполняется в swarm). В данном случае он возьмет сервис и создаст на основе его определения задачу (task)

4. Allocator назначит IP адрес для сервиса и задачи

5. Scheduler отвечает за распределение задач по рабочим узлам. Определит, что задача не запущена на рабочем узле; будет пытаться найти наилучшее соответствие между параметрами доступных узлов и параметрами задачи (ограничения, лимиты ресурсов и др.); назначит задачу одному из рабочих узлов

6. Рабочие узлы подключены к dispatcher и ожидают команд от manager’а. Таким образом, назначенная задача будет передана и запущена конкретным worker’ом

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Увеличение количества реплик

1. Пользователь обновляет определение сервиса, чтобы увеличить количество реплик до 3

2. Manager API принимает, проверяет и сохраняет обновление

3. Orchestrator сравнивает требуемое состоянием с текущим. Он заметит, что должно быть три реплики, но только 1 запущена. Orchestrator создаст ещё две задачи

Далее аналогично шагам 4-6 предыдущего примера

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Обновление конфигурации

Для обновления конфигурации сервиса scheduler по умолчанию выполняет следующие действия:

1. Останавливает первую задачу
2. Планирует обновление остановленной задачи
3. Запускает контейнер с обновленной задачей
4. Если обновление для задачи возвращает RUNNING, ждет некоторое время и запускает следующую задачу
5. Если в любой момент во время обновления, задача возвратит FAILED, обновление останавливается

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Выход из строя рабочего узла

1. Dispatcher не получает положенные heartbeat’ы от некоторого узла, на котором была запущена одна задача, и признает его вышедшим из строя и помечает флагом DOWN

2. Orchestrator сравнивает требуемое состоянием с текущим и замечает, что должно быть три реплики, но одна вышла из строя и только 2 запущены. Orchestrator создаст ещё одну задачу на одном из доступных узлов

Далее аналогично шагам 4-6 первого примера

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Configs

- Конфигурационные файл без конфиденциальной информации могут быт сохранены в **swarm service configs**
- В этом случае нет необходимости монтировать (bind-mount) конфигурационные файлы или использовать переменные окружения
- Файлы будут напрямую добавлены в файловую систему контейнера
- Configs могут быть добавлены или удалены из сервиса в любое время
- Сервисы могут использовать одну и тут же конфигурацию (config)
- Значение config должно содержать строки или бинарный контент до 500КБ
- Configs доступны только для сервисов в swarm и недоступны отдельно запускаемым контейнерам

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Доступ к config

- Docker отправляет config swarm manager’у. Config сохраняется в Raft log в зашифрованном виде. Log копируется на все manager’ы в соответствии с Raft для обеспечения отказоустойчивости manager’а
- Когда предоставляется доступ к config новому или уже запущенному контейнеру, config монтируется как файл в контейнере (по умолчанию /config_name для Linux)
- В любой момент сервису можно добавить или удалить config
- Узел/хост имеет доступ к config, если он manager или на нем напущена задача сервиса
- Когда контейнер задачи останавливается, configs отсоединяются и удаляются из памяти узла
- Если узел теряет связь в swarm режиме, контейнеры все равно имеют доступ к их configs, но при этом не получают обновления

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Secrets

- Secrets используются для управления конфиденциальными данными, которые необходимы контейнерам во время работы. В этом случае нет необходимости хранить их в образе или монтировать через bind-mount.

- Примеры конфиденциальных данных:

  - Имена пользователей и пароли
  - TLS сертификаты и ключи
  - SSH ключи

- Значение secret должно содержать строки или бинарный контент до 500КБ

- Secrets передаются только тем контейнерам, которые в них нуждаются. Secrets передаются и хранятся в зашифрованном виде

- Доступны только тем сервисам, которым был явно предоставлен доступ к ним и только пока задачи сервиса запущены

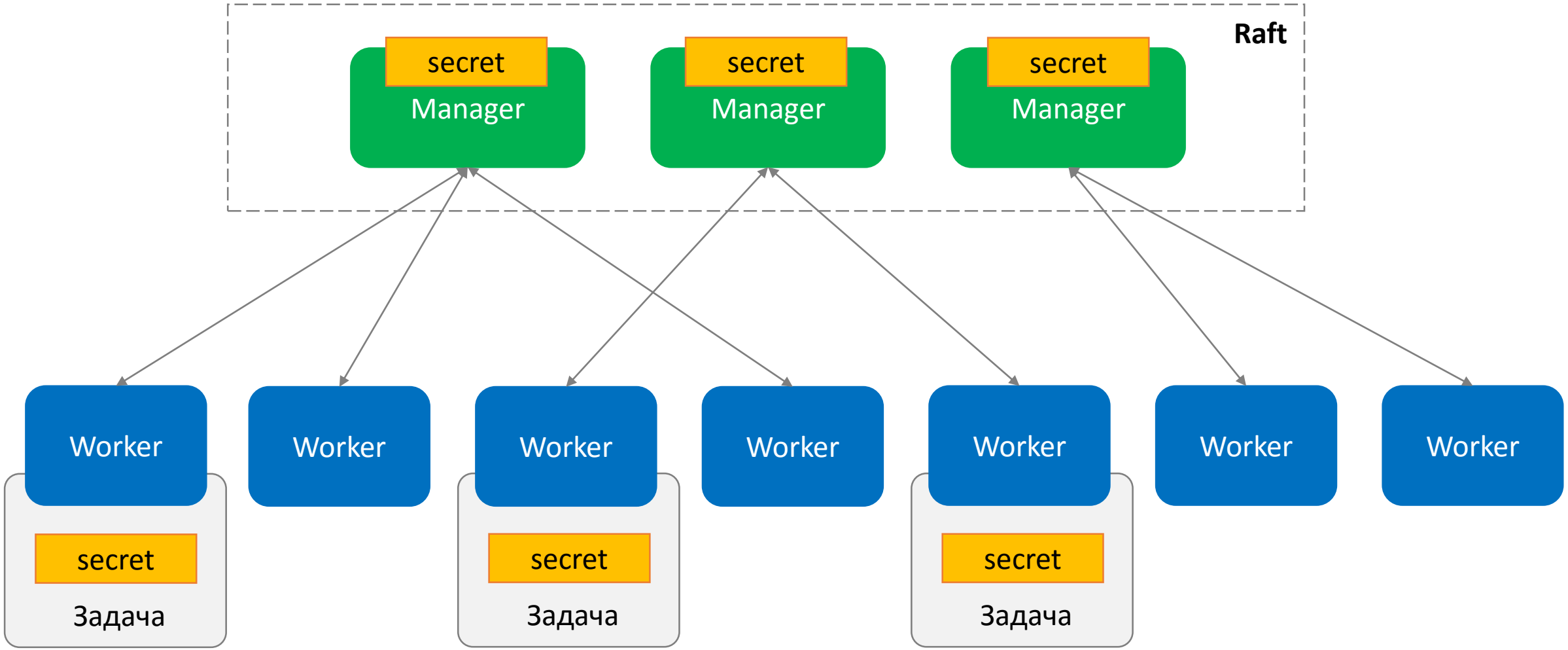
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Доступ к secrets

- Docker отправляет secrets swarm manager’у. Secret сохраняется в Raft log в зашифрованном виде. Log копируется на все manager’ы в соответствии с Raft для обеспечения отказоустойчивости manager’а
- Когда предоставляется доступ к secret новому или уже запущенному контейнеру, расшифрованный secret монтируется в контейнер как tmpfs (по умолчанию /run/secrets/secret_name в Linux)
- В любой момент можно добавить secret. Нельзя удалить secret, если его использует запущенный сервис
- Узел/хост имеет доступ к secret, если он manager или на нем напущена задача сервиса
- Когда контейнер задачи останавливается, secrets отсоединяются и удаляются из памяти узла
- Если узел теряет связь в swarm режиме, контейнеры все равно имеют доступ к их secrets, но при этом не получают обновления

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Архитектура Docker контейнера

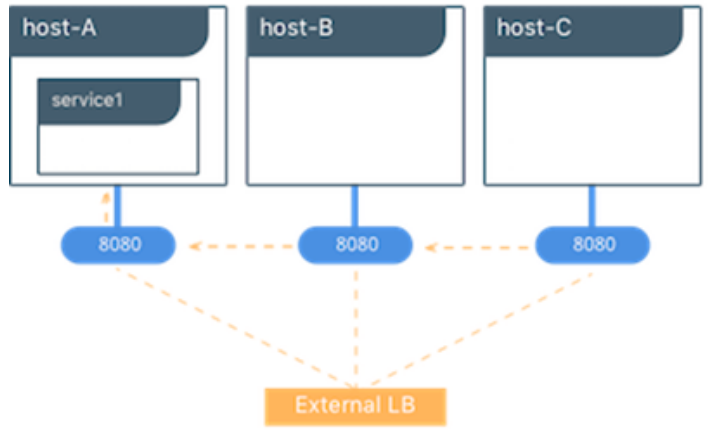


ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Внешний доступ к сервису

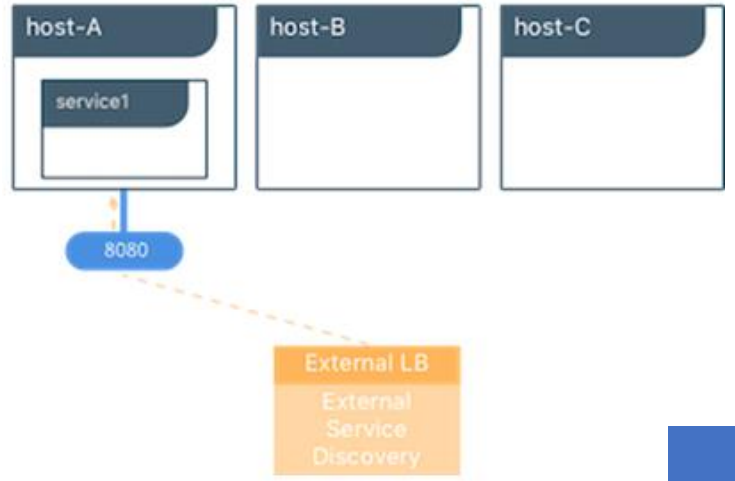
Ingress

Внешний входящий трафик на опубликованные порт балансируется посредством Routing Mesh и перенаправляется по принципу round robin одной из задач сервиса. Даже если некоторый хост не содержит запушенной задачи, порт публикуется на хосте и перенаправляет трафик хосту, на котором запущена задача



Host

Host режим при публикации порта открывает порт только на хосте, где запущены задачи сервиса



ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Балансировка нагрузки

Внутренняя

Обеспечивает балансировку между контейнерами в swarm

Внешняя

Обеспечивает балансировку входящего трафика кластера

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Внутренняя балансировка

Virtual IP (VIP)

При создании сервиса ему назначается виртуальный IP (VIP), который является частью сети сервиса. При разрешении имени сервиса возвращается этот IP. Трафик на VIP перенаправляется задачам сервиса. Docker отвечает за равномерное распределение трафика по запущенным задачам

DNS round robin (DNS RR)

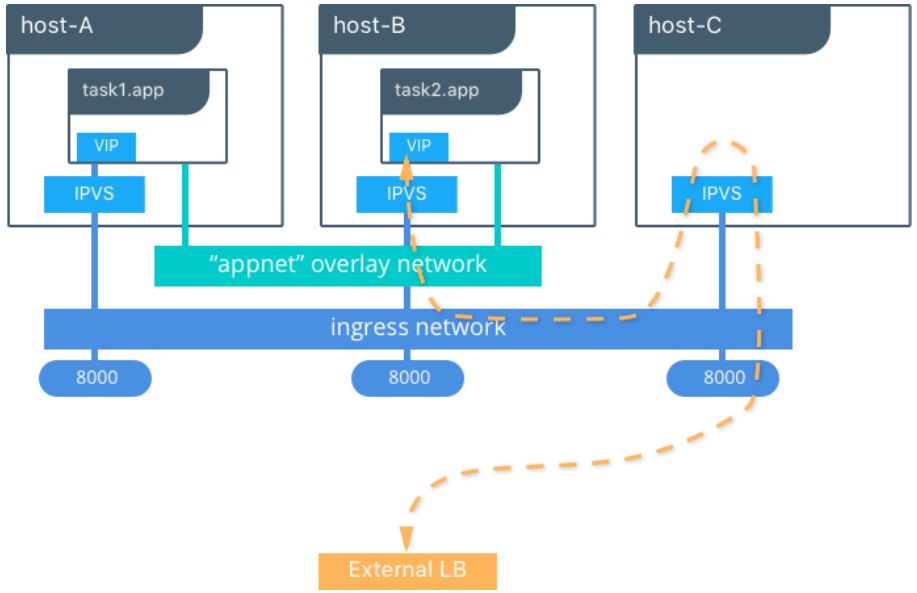
При создании сервиса с dnsrr значением параметра endpoint-mode VIP не назначается и Docker DNS просто возвращает IP контейнера запущенной задачи в round robin манере

Внешняя балансировка Routing Mesh (L4)

При публикации порта сервиса в swarm режиме каждый хост в кластере начинает слушать его и, соответственно, принимает запросы

Когда хост принимает трафик на опубликованный порт, он перенаправляет его через заранее определенную «ingress» overlay сеть

Ingress сеть работает так же как и обычная overlay сеть, но единственная её цель – это обеспечение передачи mesh routing трафика между хостами от внешних клиентов при поступлении запросов на порт сервиса. Сеть использует основанный на VIP внутренний балансировщик



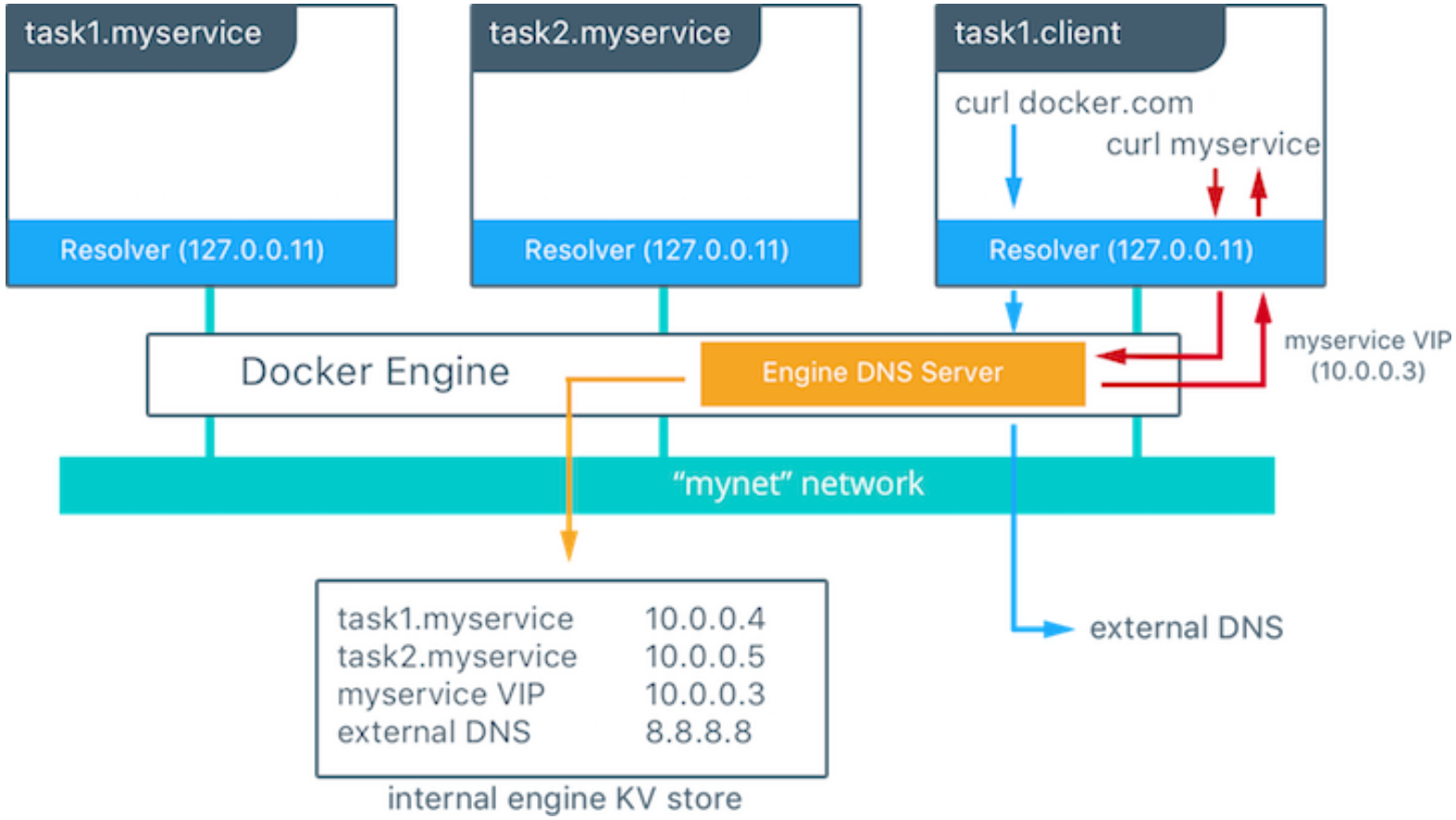
ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Обнаружение сервисов (Service Discovery)

- Docker Engine имеет внутренний DNS сервер, который обеспечивает разрешение имен всех контейнеров на хосте в bridge, overlay и macvlan сетях.
- Контейнеры находящиеся в разных сетях не могут получить адрес друг друга
- Только узлы/хосты, которые имеют контейнеры или задачи определенной сети, хранят DNS записи этой сети.
- Каждый контейнер имеет DNS resolver, который перенаправляет DNS запросы в Docker Engine. Docker Engine проверяет, принадлежит ли запрашиваемый контейнер или сервис сети отправителя
- Если принадлежит, то Docker Engine ищет IP адрес, который соответствует имени контейнера, задачи или сервиса в его kye-value хранилище и возвращает IP или VIP сервиса
- Если контейнер назначения или сервис не принадлежит той же сети, что и контейнер отправителя, то Docker Engine перенаправляет DNS запрос на DNS сервер, который установлен по умолчанию

ИНСТИТУТ ЦИФРОВОГО ОБРАЗОВАНИЯ МГПУ

Обнаружение сервисов. Пример



Источники

Docker overview (doc)

Docker Swarm Reference Architecture: Exploring Scalable, Portable Docker Container Networks (article)

Get started with Macvlan network driver (doc)

Manage data in Docker (doc)

About storage drivers (doc)

cgroups - Linux control groups (doc)

namespaces - overview of Linux namespaces (doc)

Docker source code (github)

How nodes work (doc)

How services work (doc)

SwarmKit Architecture (blog)

Docker Built-in Orchestration Ready for Production: Docker 1.12 Goes GA (blog)

Manage sensitive data with Docker secrets (doc)

Store configuration data using Docker Configs (doc)

Docker Swarm - An Analysis Of A Very-Large-Scale Container System (blog)