Контейнерная оркестрация

План

- Оркестрация
- Контейнерная оркестрация
- Docker Swarm

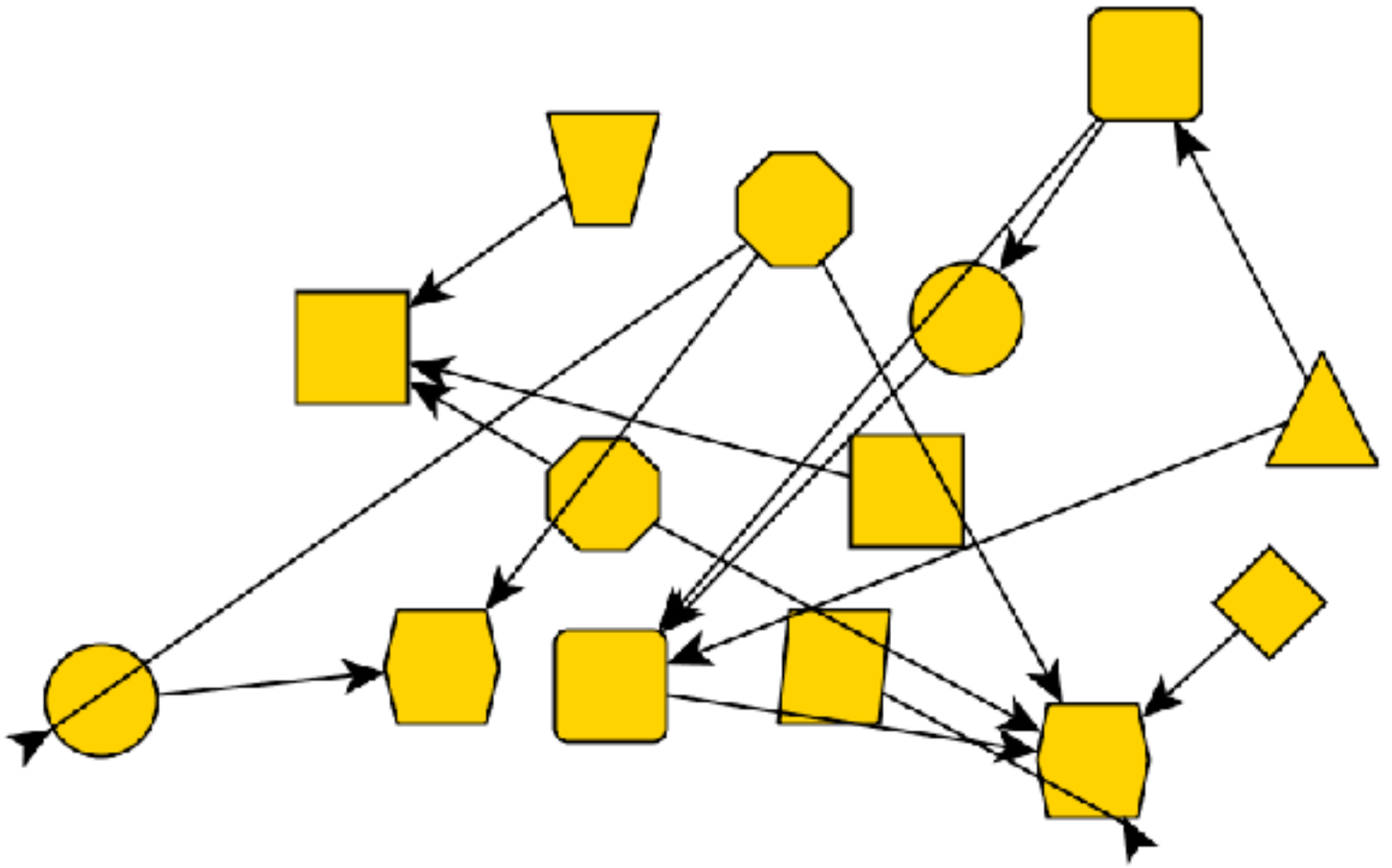
Оркестрация

front/back/db

Устройство проекта:

- 1/2/3 приложения или сервиса
- Просто деплоить
- Знаем куда деплоить
- Знаем сколько ресурсов нужно

Микросервисы



Микросервисы

Поддержка:

- Рассчитать потребление ресурсов сложно
- Ручная аллокация приложений
- Проблемы с деплоем

Оркестрация

- Управление кластером хостов
- Планирование и распределение задач
- Автоматизация

Управление кластером

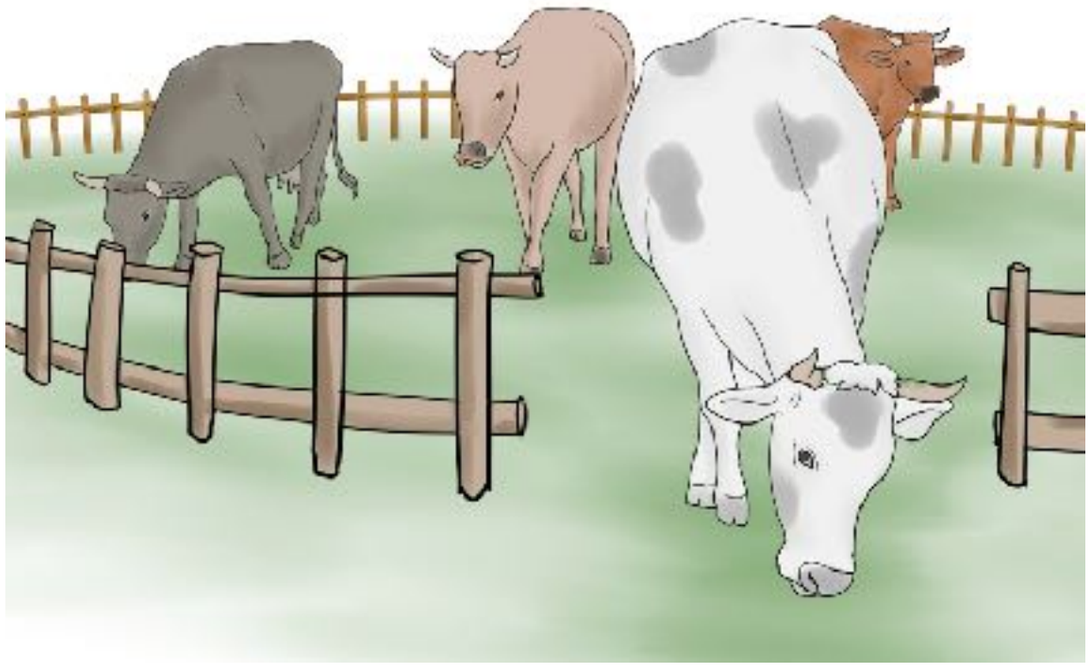
- Cattle, NOT pets
- Добавление новых хостов (provisioning)
- Управление хостами в кластере

Pets

- Уникальны и неповторимы
- Имеют особые имена
- Не могут заболеть
- Если болеют, то пытаемся лечить

Cattle

- Все подобны друг другу
- Не имеют специальных имен, обычно нумеруются
- Если болеют, то убиваем, заменяем другими таким же



Планирование

Когда?

- Есть больше 1-го приложения
- Гибкое управление большим количеством ресурсов

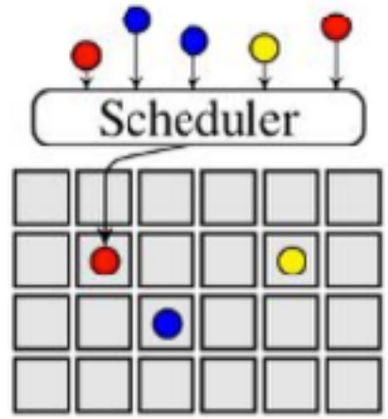
Чем занимается планировщик?

- Выделение (Аллокация) ресурсов для запуска задачи
- Дать все необходимое для запуска задачи
- Контроль ресурсов
- Контроль состояния задач
- Реакции на изменения состояний задач

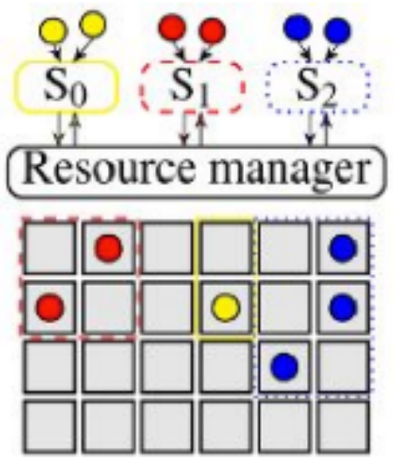
Планировщики

Архитектуры

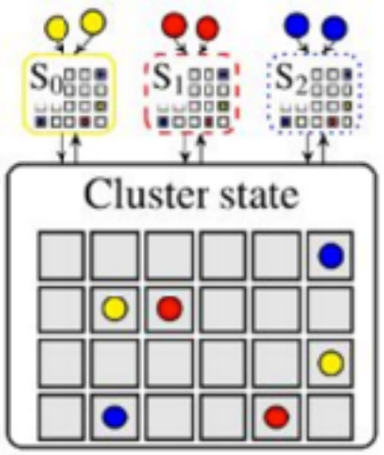
1) **Монолитное** планирование- (kube-scheduler)
   - Планировщик сам выполняет оценку ресурсов
   - Простое решение
   - Могут возникать очереди



2) **Двухуровневое** планирование (Mesos + Marathon)
   - Планировщик ресурсов + планировщик задач
   - Легко добавлять другие типы планировщиков для других задач
   - Ресурсы могут быть недозагружены



3) **Shared-state** архитектура (Nomad)
   - Каждый планировщик поддерживает собственное представление состояния кластера
   - когда нужно выполнить новую задачу, то он коммитит транзакцию задачу в общий стейт
   - Есть шанс множественного Split Brain



Задачи

Ресурсы

Оркестрация

Orchestration
Framework

Задачи

Ресурсы

Оркестрация для контейнеров

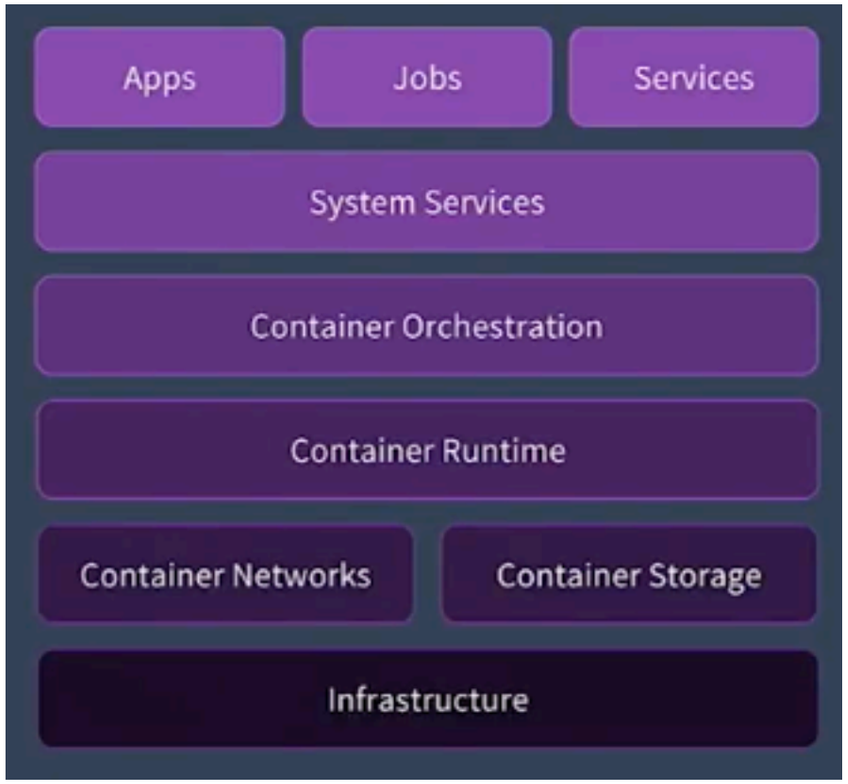
Контейнерая оркестрация

- Управление кластером
- Планирование и распределение задач
- Автоматизация
- Управление сервисами для контейнеров
  - Service Discovery
  - LB
  - Storages
  - ...

Контейнерная оркестрация

- Нагрузка пользователей
- Распределенное управление контейнерами
- Управление контейнерами на локальной машине
- Контейнеро-независимая инфраструктура

Контейнерная платформа



- Нагрузка пользователей
- Управление сервисами
- Распределенное управление контейнерами
- Управление контейнерами на локальной машине
- Инфраструктура для контейнеров
- Контейнеро-независимая инфраструктура

Управление ресурсами

- Память
- CPU
- GPU
- Storages
  - локальные хранилища
  - удаленные хранилища
- Ports
- IP address management

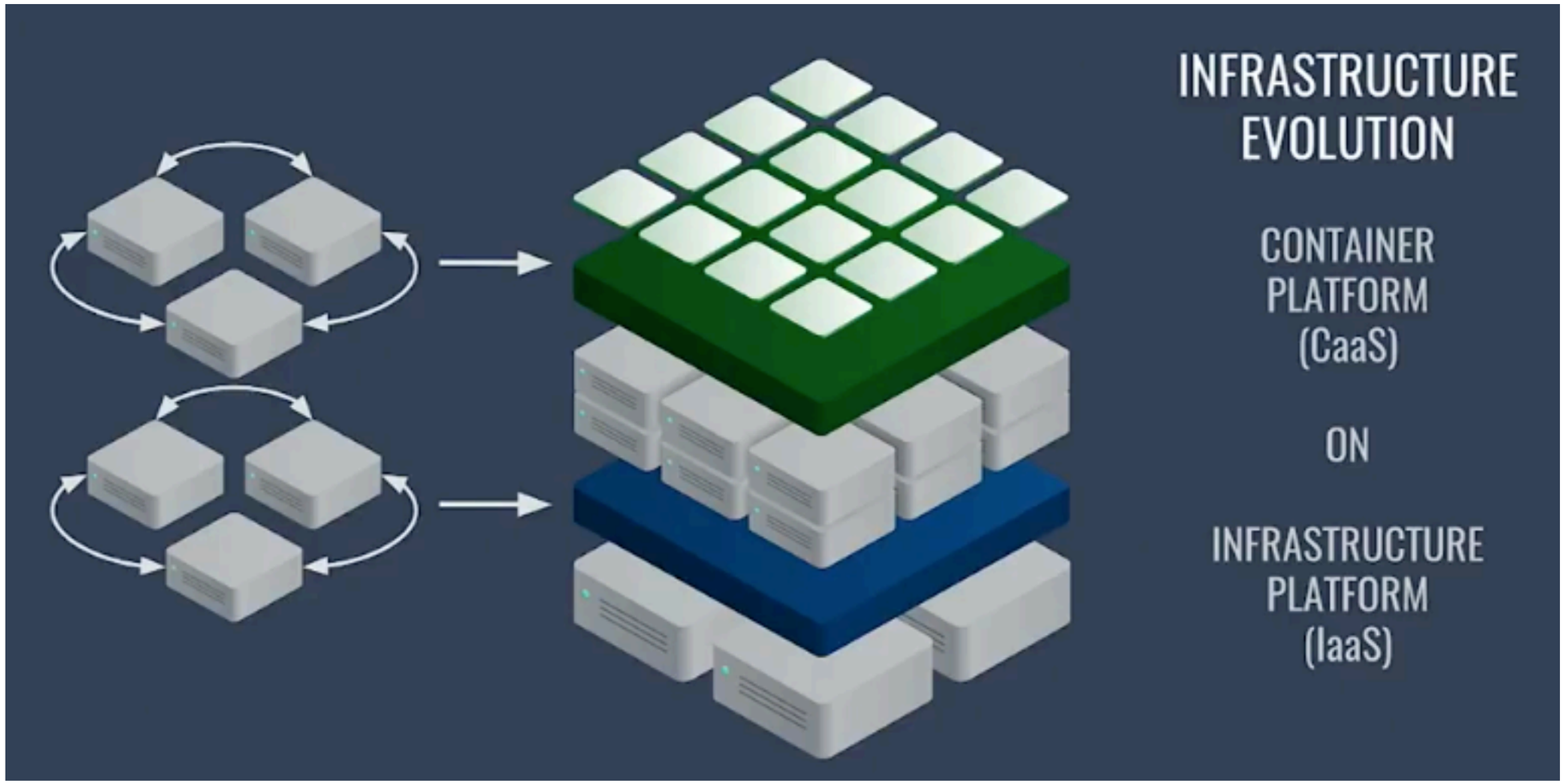
Планирование

- Расположение
- Реплицирование/масштабирование
- Проверки готовности/жизнеспособности
- Воскрешение
- Перепланирование (rescheduling)
- Cron/batch jobs
- Запуск демонов

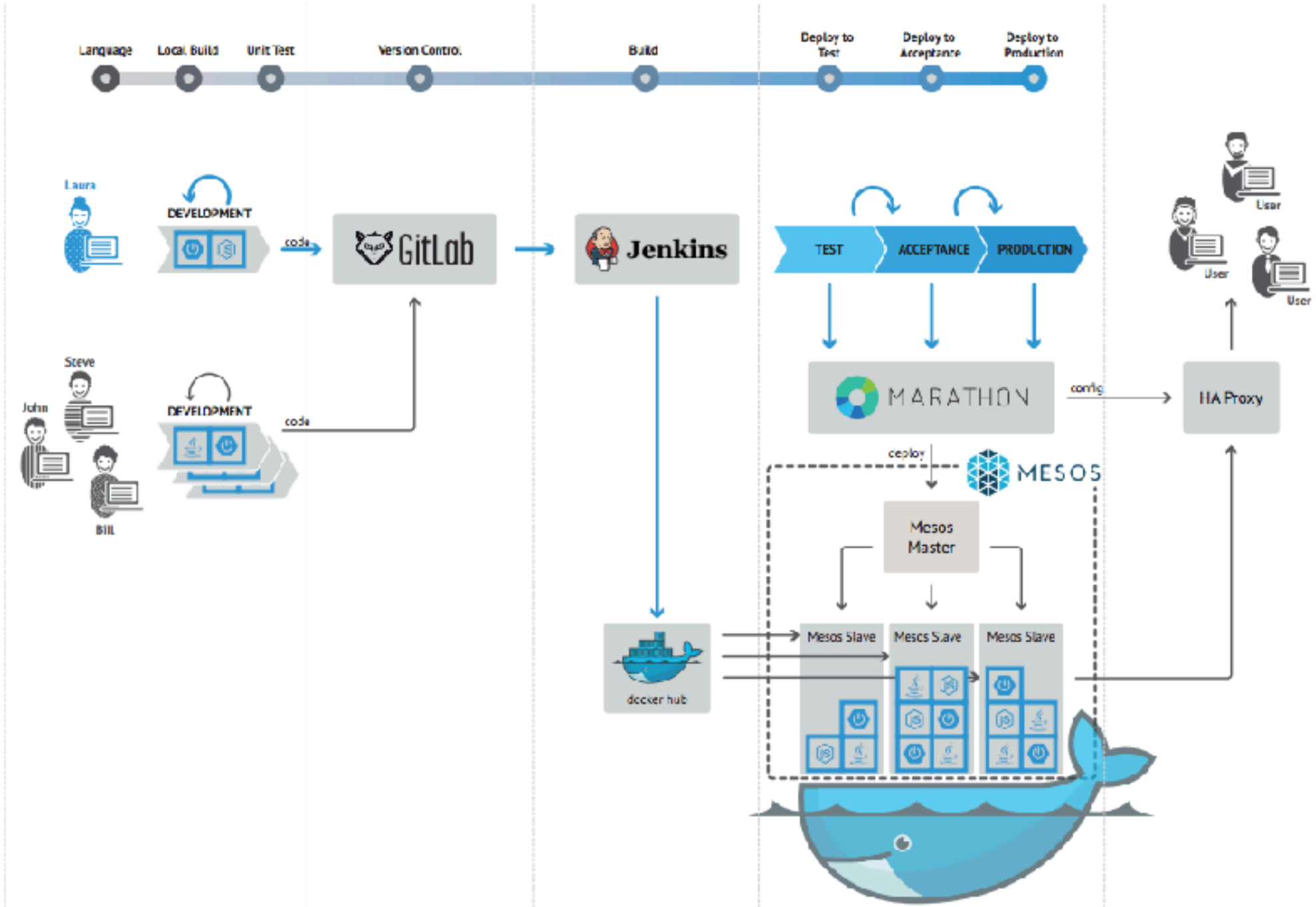
Управление сервисами

- Метки
- Группы
- Зависимости
- Балансировка нагрузки
- Virtual IP
- DNS
- Управление секретами
- Хранение конфигурационных файлов

Контейнерная оркестрация



Место в конвейере



Примеры


Docker
ECS
Kubernetes
DC/OS
Kontena
Nomad
Rancher Cattle
OpenShift

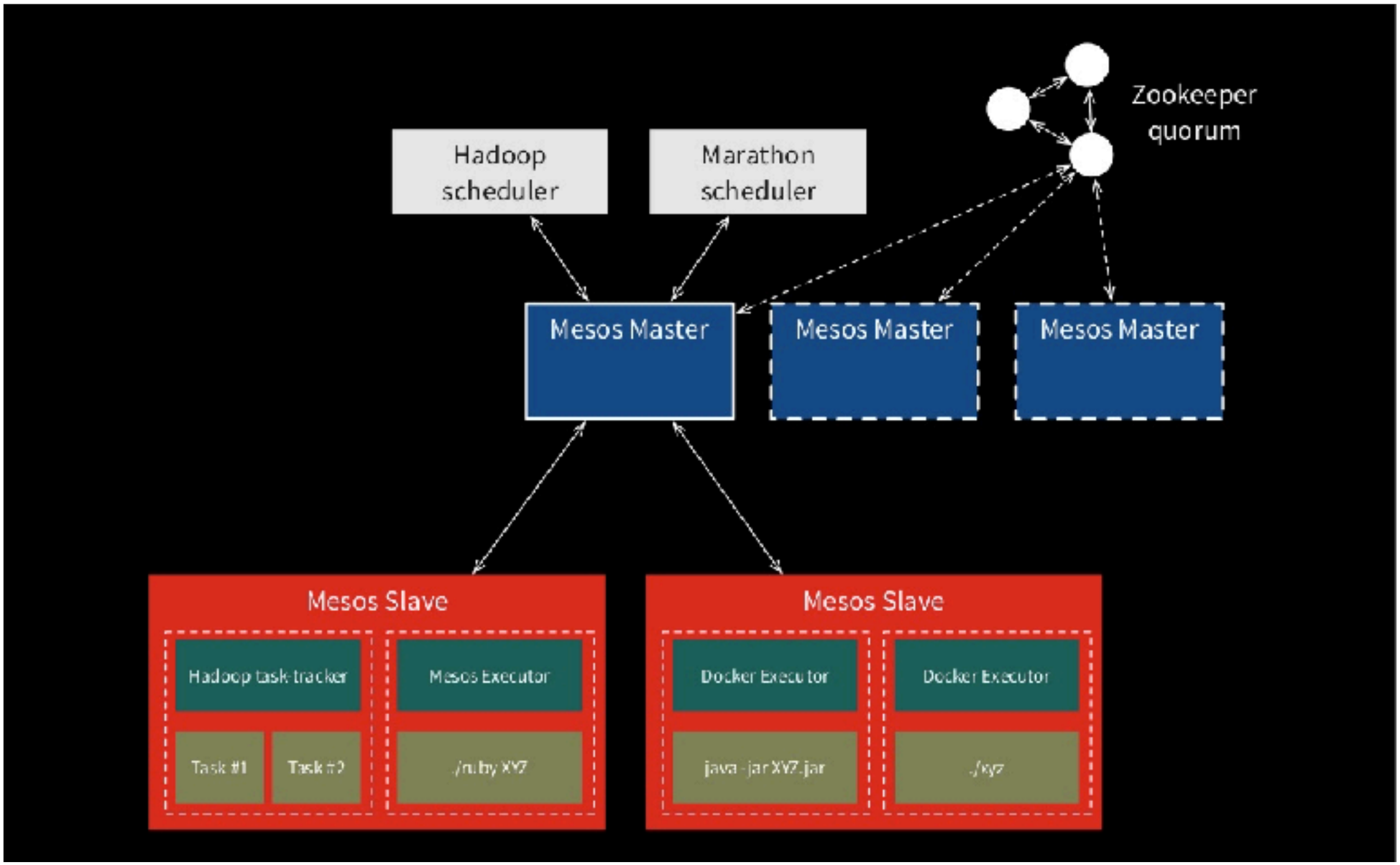
DC/OS (Mesos + marathon, aurora)

- Появился еще в 2009 году
- Позиционируется как Data Center Operating System
- Поддерживает 2 типа контейнеров (Docker, Mesos)
- Поддерживает различные типы выполняемых задач с помощью Mesos фреймворков (kafka, spark, Elastic, Cassandra, etc)

DC/OS (Mesos + marathon, aurora)

- zookeeper для хранения состояния кластера
- **Mesos** - кластерный менеджер
- **Marathon** - планировщик для Docker-контейнеров
- Поддерживает **Kubernetes** (beta) в виде “приложения”

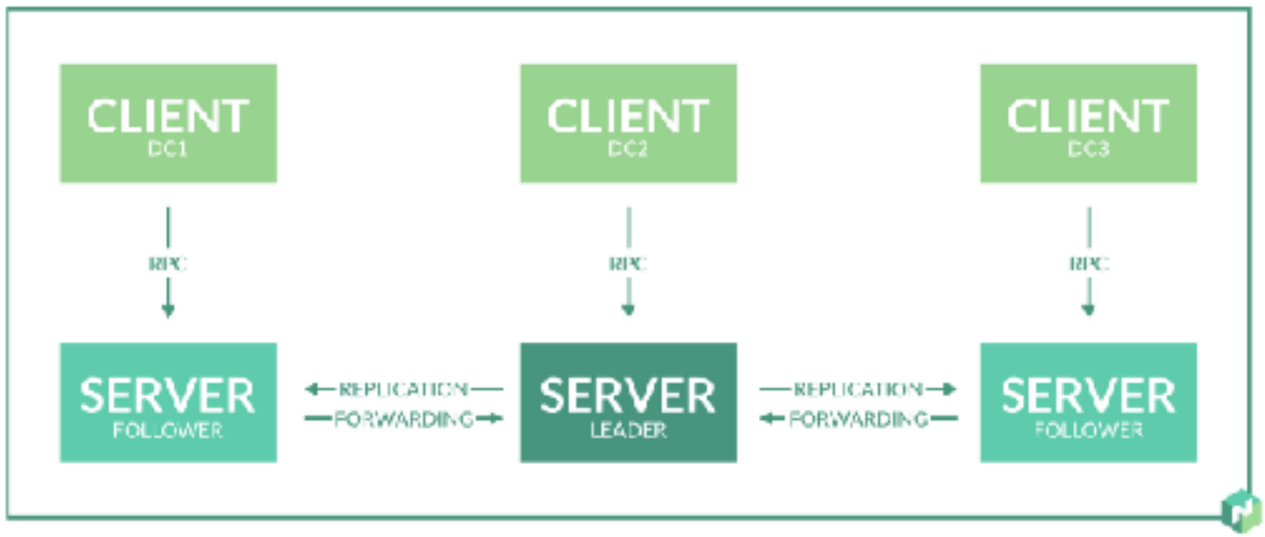
DC/OS (Mesos + marathon)



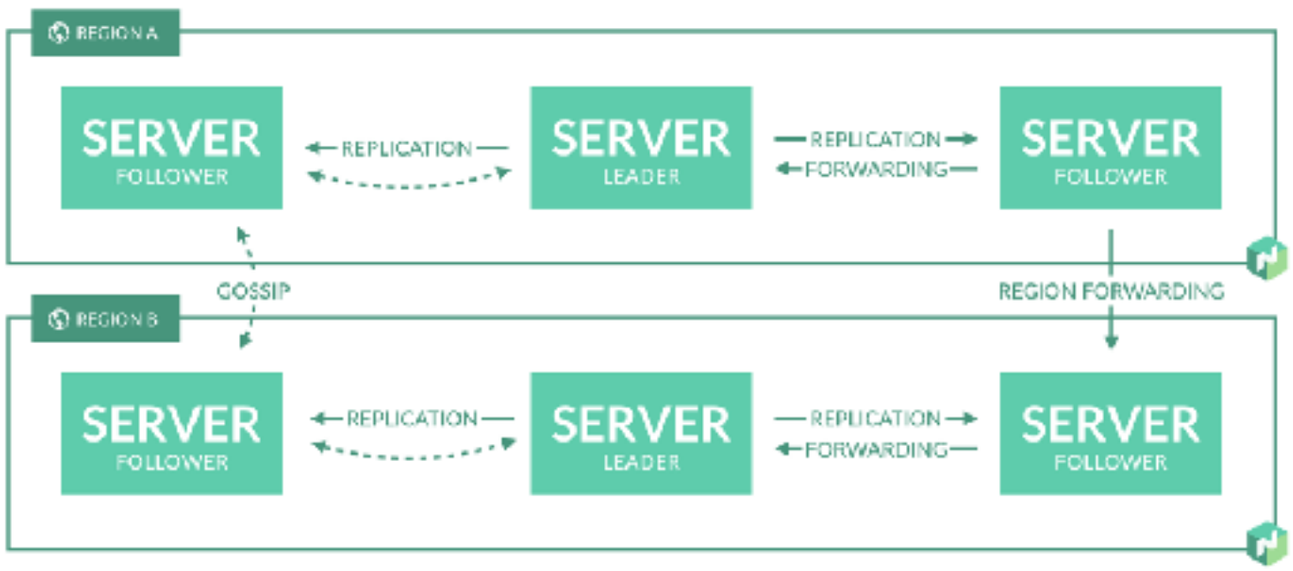
Nomad

- Поставляется одним бинарником
- Запуск задач с помощью различных драйверов (Docker, rkt, Isolated Fork/Exec, Java, Qemu, ...)
- Большой упор сделан на поддержку мультирегиональности
- Выполняет как долгоживущие задачи, так и периодические короткие
- Service discovery через Consul
- Хранение секретов через Vault
- Нет Load-Balancing

Nomad



Client-Server



Multi-Region

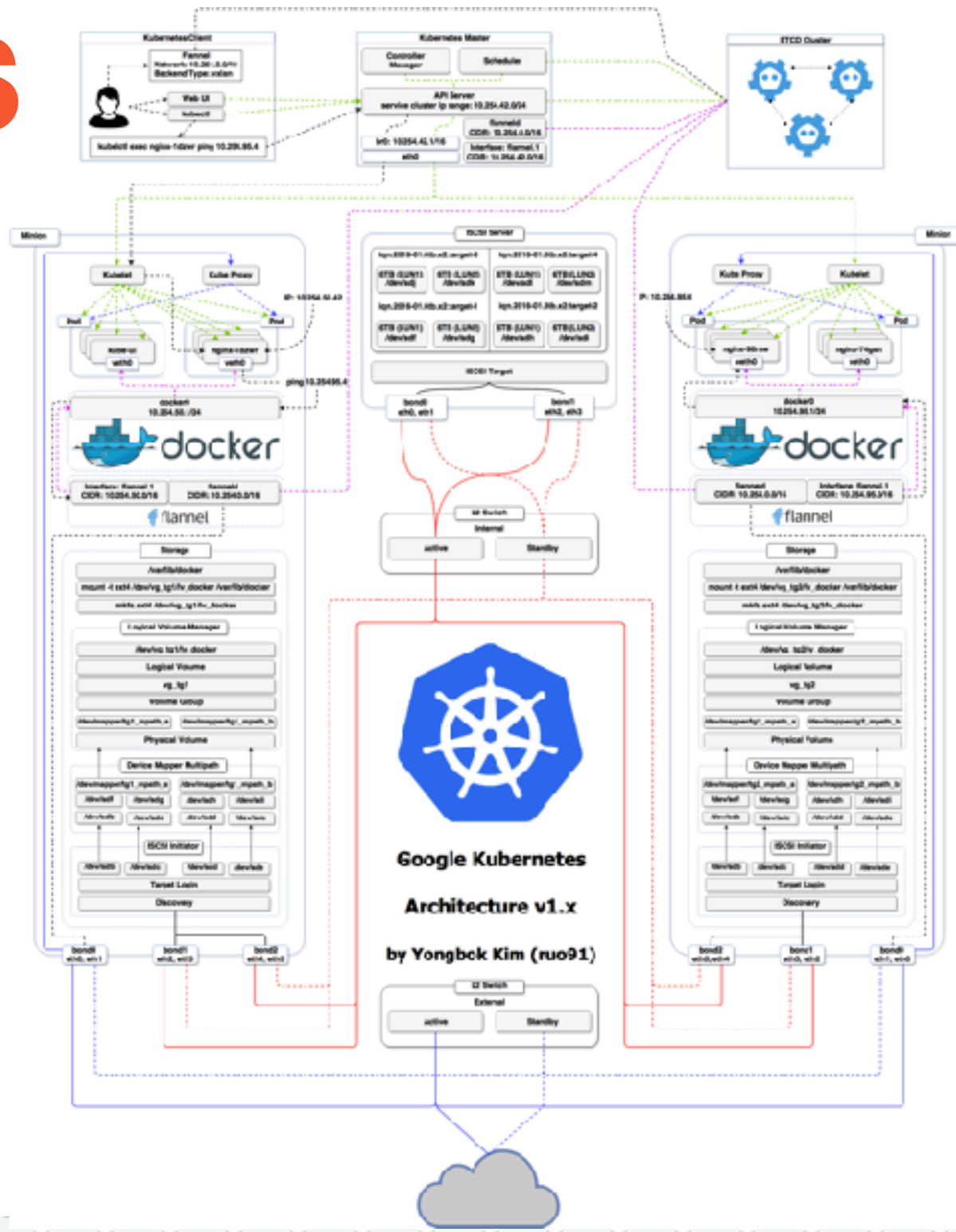
Kubernetes

- Самое большое сообщество, поддержка Google, Red Hat, CoreOS
- Поддержка нескольких Runtime-систем: (CRI-O, Docker, rkt)
- Описание всех сущностей в собственном YAML-формате
- Хранилище состояния в **etcd**
- Большое кол-во поддерживаемых плагинов сетей и хранилищ
- Мощный API для добавления новых сущностей

Концепция **POD**-ов - управление группами контейнеров (group container management)

Kubernetes

Что видит человек впервые?



Docker compose

- Только планирование
- Декларативное описание в YAML-формате
- Управление группами сервисов
- Работает только для одного Docker хоста

Docker Swarm

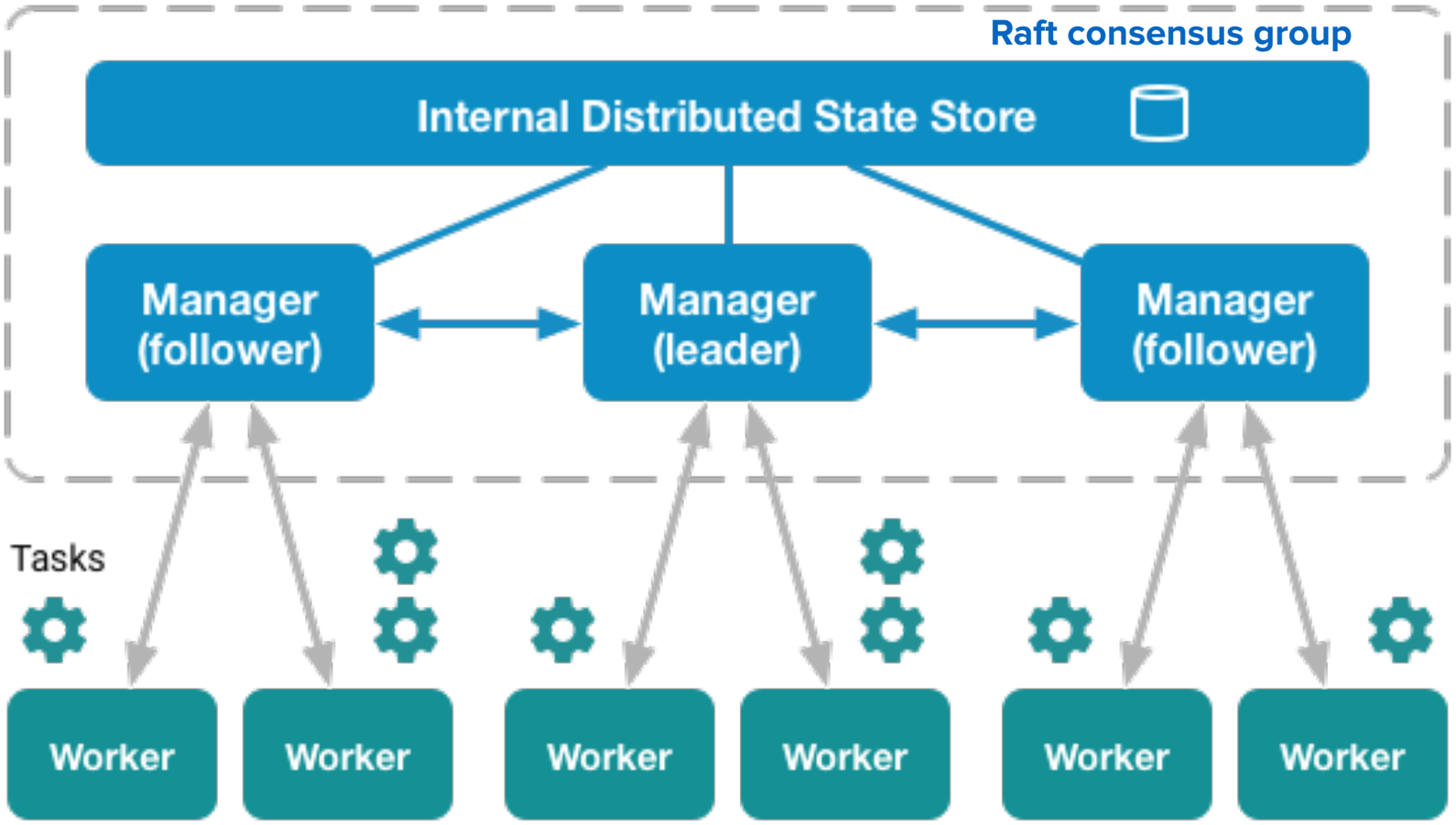
- Поставляется из коробки с Docker (начиная с версии 1.12)
- Разворачивается 1-й командой (начиная с версии 1.12)
- Не использует внешние распределенные хранилища
- Позволяет описывать конфигурацию в формате Docker Compose
- Есть встроенный Service Discovery
- Есть встроенный Load Balancer

Docker swarm

Основные концепции

- **Кластер** (Cluster) - организованный набор Docker-Engine'ов, сконфигурированный для запуска сервисов. Состоит из *нод*
- **Нода** (Node) - активный член кластера. Может выполнять задачи и/или управлять кластером
- **Manager** - управляющая нода. Участвует в выборах лидера (**Leader**) (создание *кворума*).
  - Управляет кластером (контроль состояний хостов, удаление, генерация секретов)
  - Принимает запросы на управление сервисами по API.
  - Выполняет планирование (аллокация ресурсов, контроль состояний сервисов, ...)
- **Кворум** - состояние, в которому несколько Manager-ов выбрали Лидера по протоколу Raft. В этом состоянии все запросы принимает *Лидер*, а остальные менеджеры перебрасывают на него.
- **Worker** - *нода*, выполняющая задачи, объявляющая об их статусах

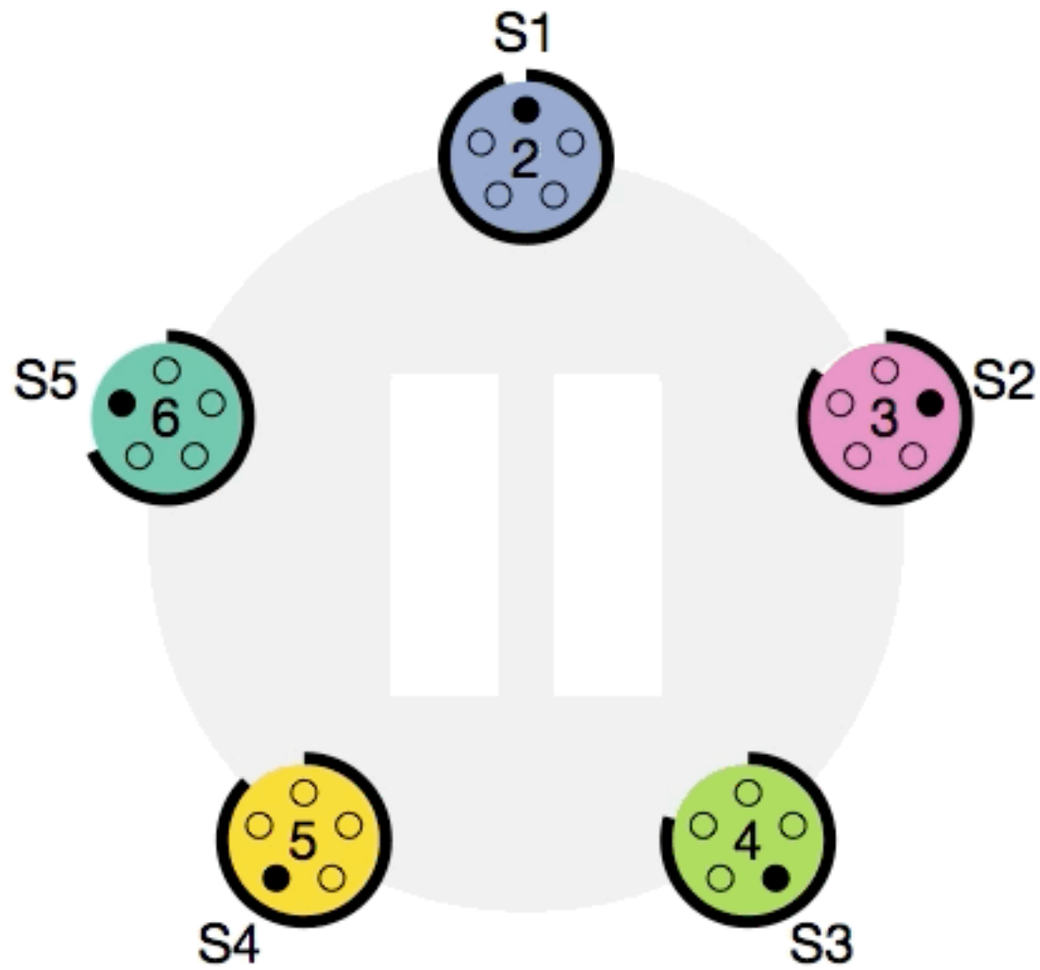
Docker Swarm



Raft в Swarm

- Алгоритм сходимости
- Можно потерять (N-1)/2 участников
- Нечетное число участников лучше
- Логи RAFT хранятся в зашифрованном виде в

/var/lib/docker/swarm/raft/



https://raft.github.io/

Docker Swarm cluster size

Swarm Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	**1**
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

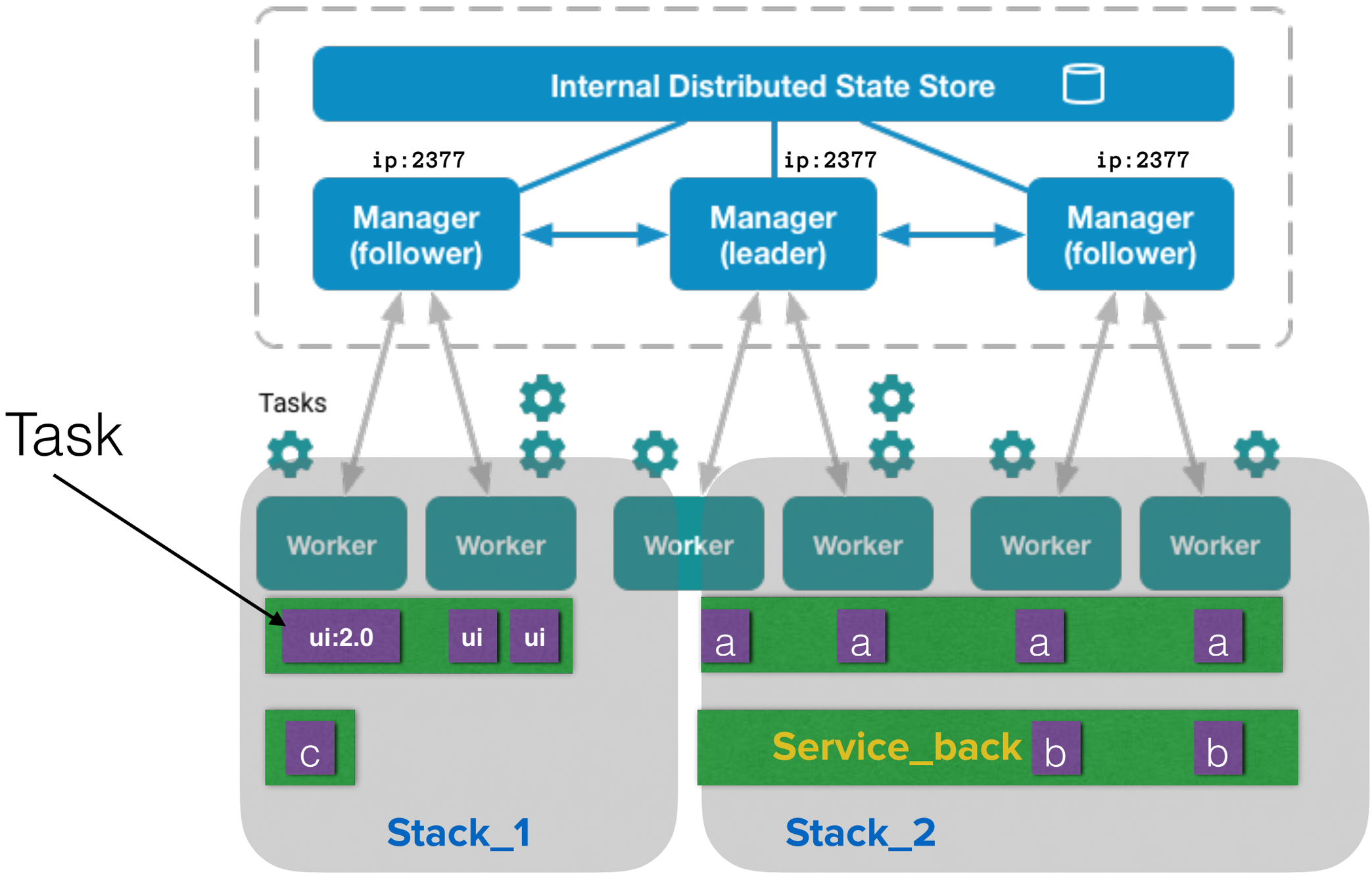
Основные концепции

- **Service** (сервис) - обозначает то, что должно быть запущено и как (количество, необходимые ресурсы и т.д.)
  При запуске Service, *лидеры* назначают *Worker*-ам необходимые задачи (*tasks*)

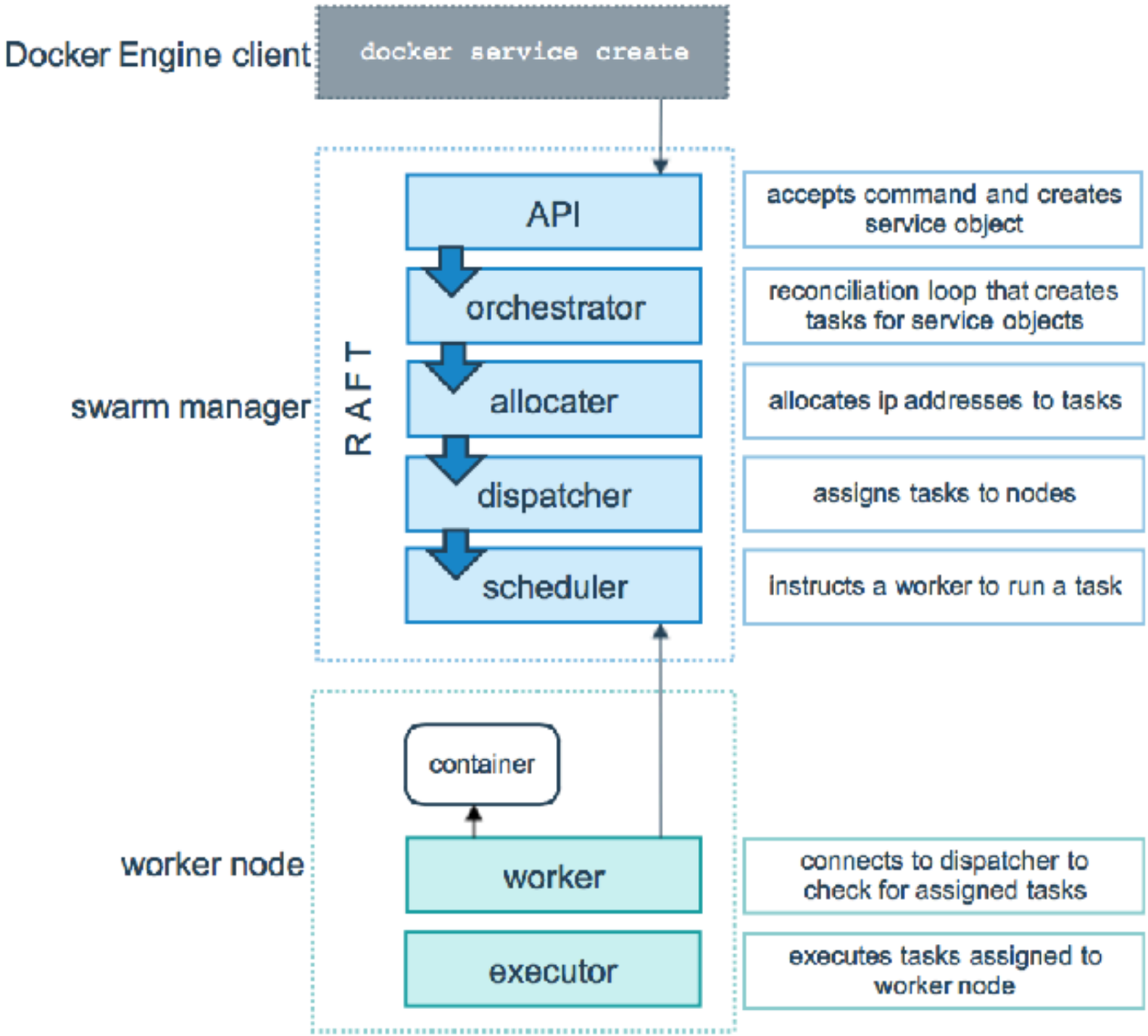
- **Task** (задача) - описывает конкретный *контейнер* на конкретной *ноде*. Может быть запущен только один раз. Если нужно перезапустить, то создается новый

- **Stack** - наборы *сервисов. Описываются с помощью compose-файла*

Основные концепции



Основные концепции



Создадим свой Swarm

```
>> docker swarm init
```

```
root@swarm-master-1:~# docker swarm init
Swarm initialized: current node (wzq2tosh1rqjg1cpthv6ofb0k) is now a manager.

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

    docker swarm join --token SWMTKN-1-20tbzjaskdjkl3pkd17nqjy2hzft8z1ujirua2uxm4c6wf475-
bmm1ovcy1pmm63p9pe4y5y463 10.132.0.2:2377
```

```
>> docker swarm join
```

```
root@swarm-worker-2:~# sudo docker swarm join --token
SWMTKN-1-20tbzjaskdjkl3pkd17nqjy2hzft8z1ujirua2uxm4c6wf475-
bmm1ovcy1pmm63p9pe4y5y463 10.132.0.2:2377
This node joined a swarm as a worker.
```

Создадим свой Swarm

```
>> docker node ls
```

Выполним на manager-ноде

```
docker-user@swarm-master-1:~$ sudo docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER
STATUS
wzq2tosh1rqjg1cpthv6ofb0k *   swarm-master-1      Ready               Active              Leader
os0hexb42nc66elc44kc7a1fx     swarm-worker-2      Ready               Active
```

Выполним на worker-ноде

```
docker-user@swarm-worker-2:~$ sudo docker node ls
Error response from daemon: This node is not a swarm manager. Worker nodes
can't be used to view or modify cluster state. Please run this command on a
manager node or promote the current node to a manager.
```

Создадим свой Swarm

Создадим сервис

```
>> docker service create --replicas 15 --ports 8080:80  nginx
```

```
root@swarm-master-1:~# docker service create --replicas 15 -p  8080:80  nginx
rkpyjqy7cc3vond5dem5e3n1a
overall progress: 15 out of 15 tasks
1/15: running   [==================================================>]
2/15: running   [==================================================>]
…
15/15: running   [==================================================>]
verify: Service converged
```

Проверим

```
docker-user@swarm-master-1:~$ curl localhost:8080
<!DOCTYPE html>
<html>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

```
docker-user@swarm-worker-2:~$ curl localhost:8080
<!DOCTYPE html>
<html>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Создадим свой Swarm

```
>> sudo docker service ps  rkpyjqy7cc3v
```

```
docker-user@swarm-master-1:~$ sudo docker service ps  rkpyjqy7cc3v
ID                  NAME                  IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
rsuzjn4yck34        reverent_lalande.1    nginx:latest        swarm-master-2      Running             Running 3 minutes ago
ku3te50h28yf        reverent_lalande.2    nginx:latest        swarm-master-1      Running             Running 3 minutes ago
zlqegiucdjug        reverent_lalande.3    nginx:latest        swarm-master-2      Running             Running 3 minutes ago
la8e4ss2ikqf        reverent_lalande.4    nginx:latest        swarm-master-2      Running             Running 3 minutes ago
```

Задачки для сервисов распределены стратегией **Spread:**

1) Смотрит, не упирается ли в лимиты по ресурсам (limits)
2) Запускает задачки на случайной ноде, на которой еще нет задач этого сервиса
3) Если задачки сервиса есть, то запускать там, где меньше всего задач этого сервиса, независимо от их состояния