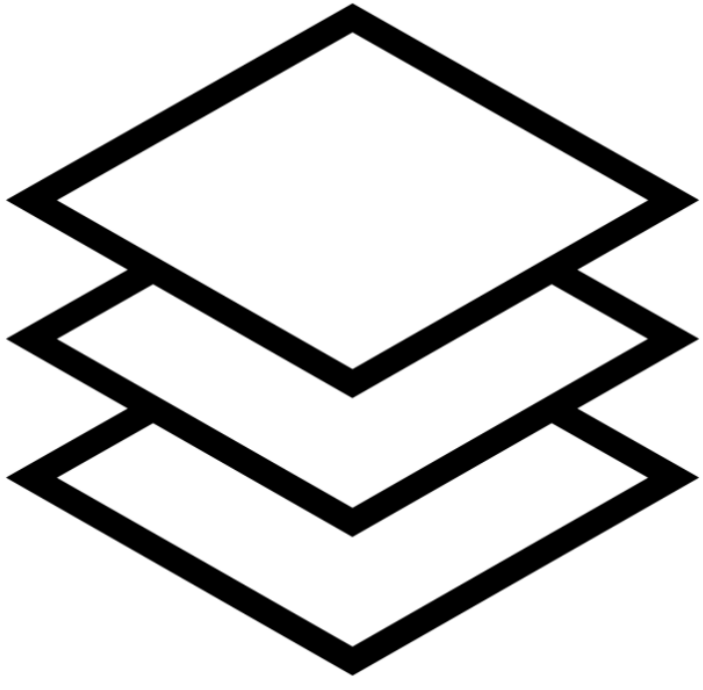
Контейнерная оркестрация

Не забудь включить запись!

План

- Оркестрация
- Контейнерная оркестрация
- Docker Swarm
- Демо

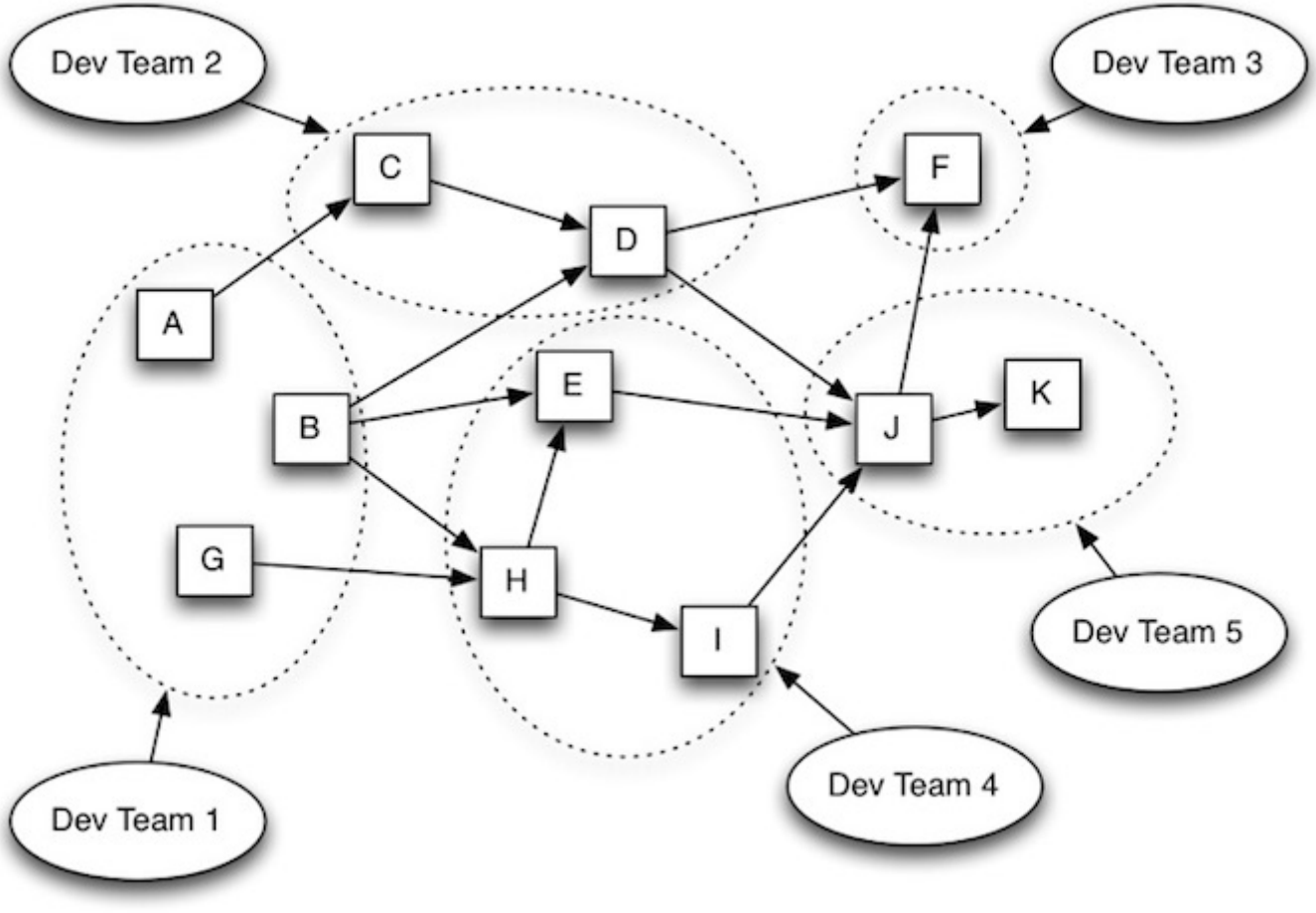
3-Tier архитектура



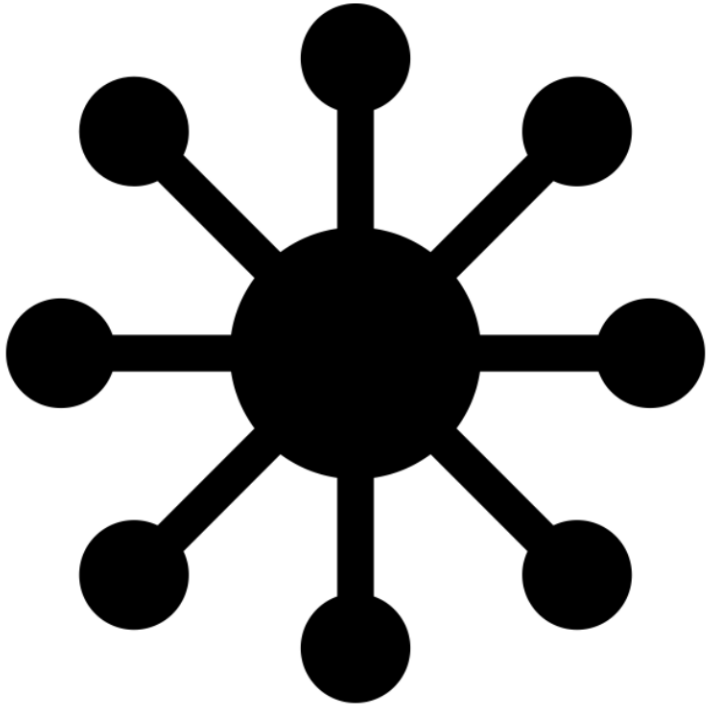
Устройство проекта:

- Несколько сервисов/приложений
- Просто деплоить и управлять конфигурацией
- Инженеры принимают решение куда деплоить
- Инженеры хранят знания о том, сколько ресурсов нужно

Микросервисы



Микросервисы

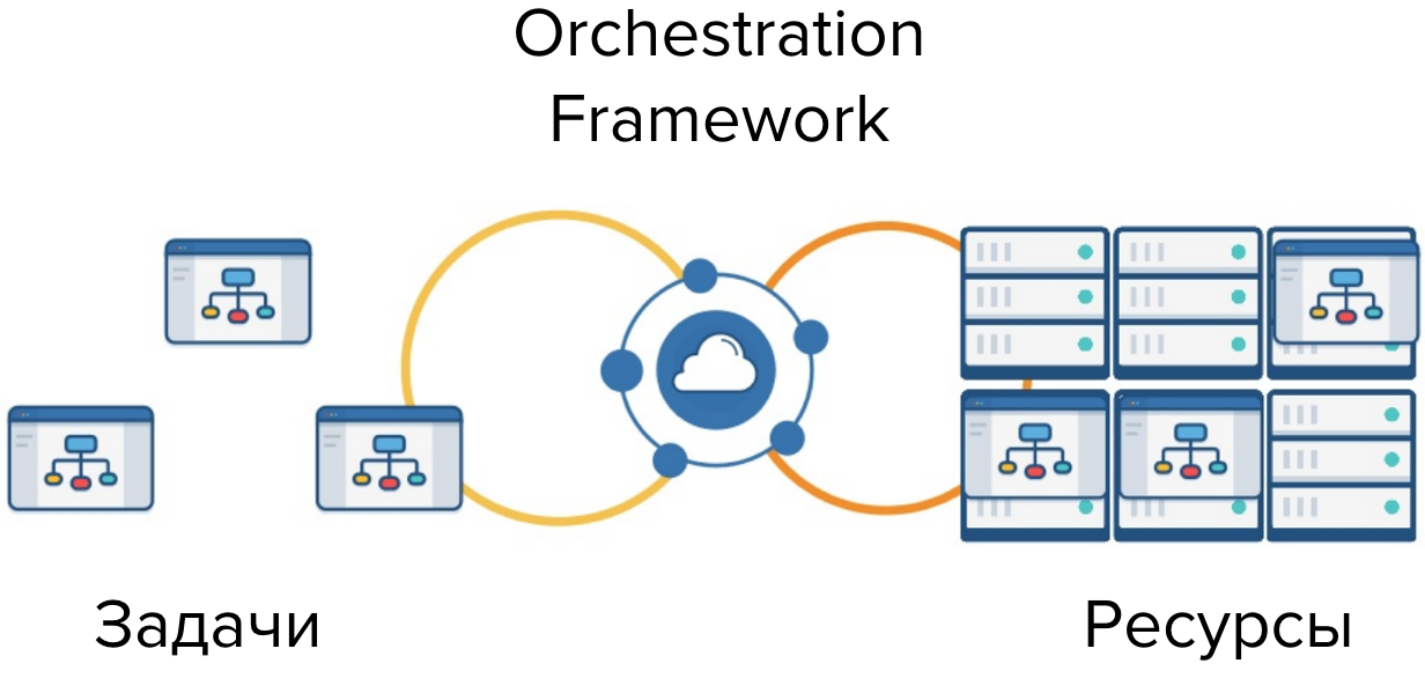


Поддержка

- Рассчитать потребление ресурсов сложно
- Ручная аллокация приложений становится проблемой
- Деплой становится рутиной

Оркестрация

- Управление кластером хостов
- Планирование и распределение задач
- Автоматизация



Управление кластером

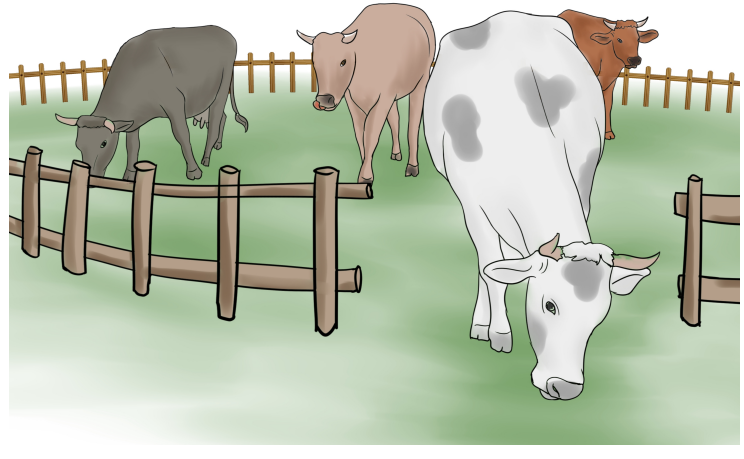
- Cattle, NOT pets
- Добавление новых хостов (provisioning)
- Управление хостами в кластере

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 и заменяем другими такими же



Планирование

Когда?

- Есть больше 1-го приложения
- Гибкое управление большим количеством ресурсов
- Нужен контроль над состоянием

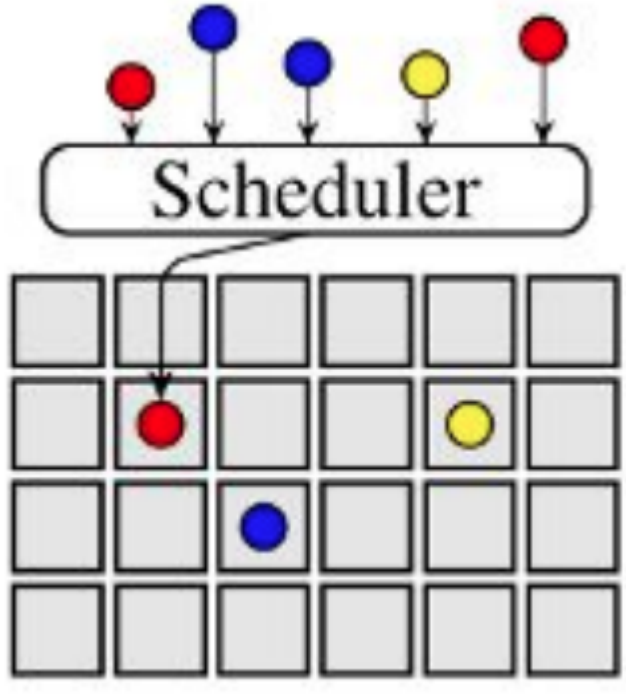
Чем занимается планировщик?

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

Планировщики: Архитектурные отличия

Монолитное планирование (`kube-scheduler`, `Borg`)

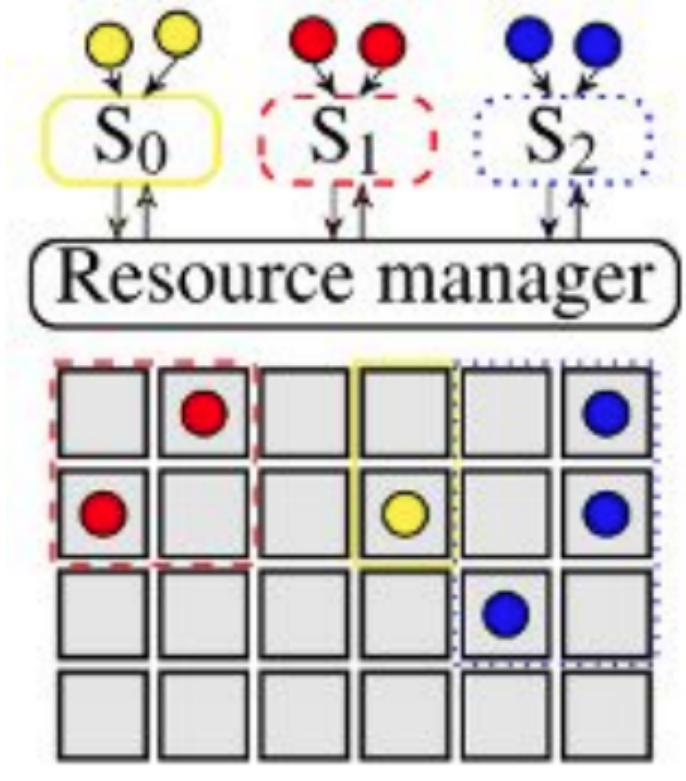
- Планировщик сам выполняет оценку доступных ресурсов
- Технически простое решение
- Могут возникать задержки планирования (*head-of-line blocking*), особенно для batch-заданий



Планировщики: Архитектурные отличия

Двухуровневое планирование (`Mesos + Marathon`)

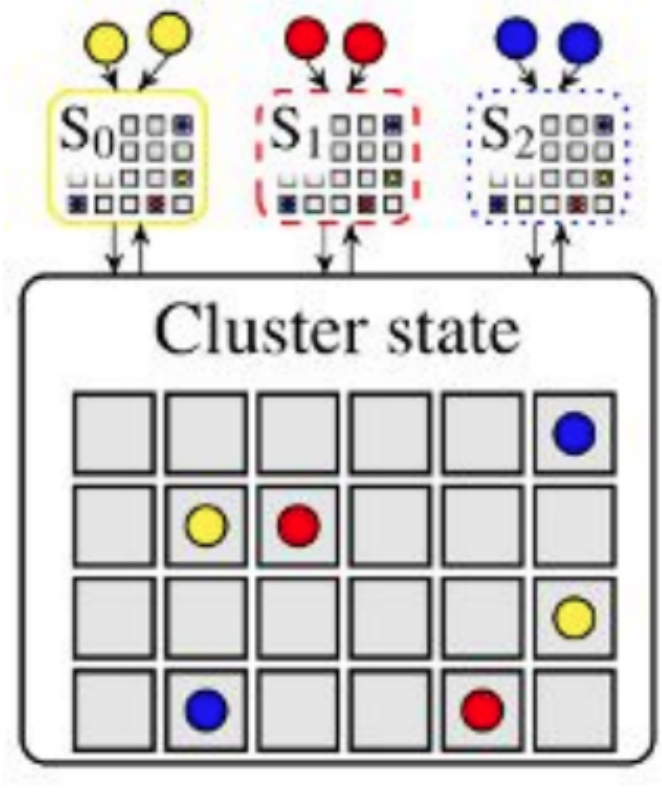
- Существуют отдельные сущности: планировщик ресурсов и планировщик задач
- Легко добавлять новые типы планировщиков
- Ресурсы кластера могут быть недоиспользованы



Планировщики: Архитектурные отличия

Shared-state архитектура (`Nomad`, `Google Omega`)

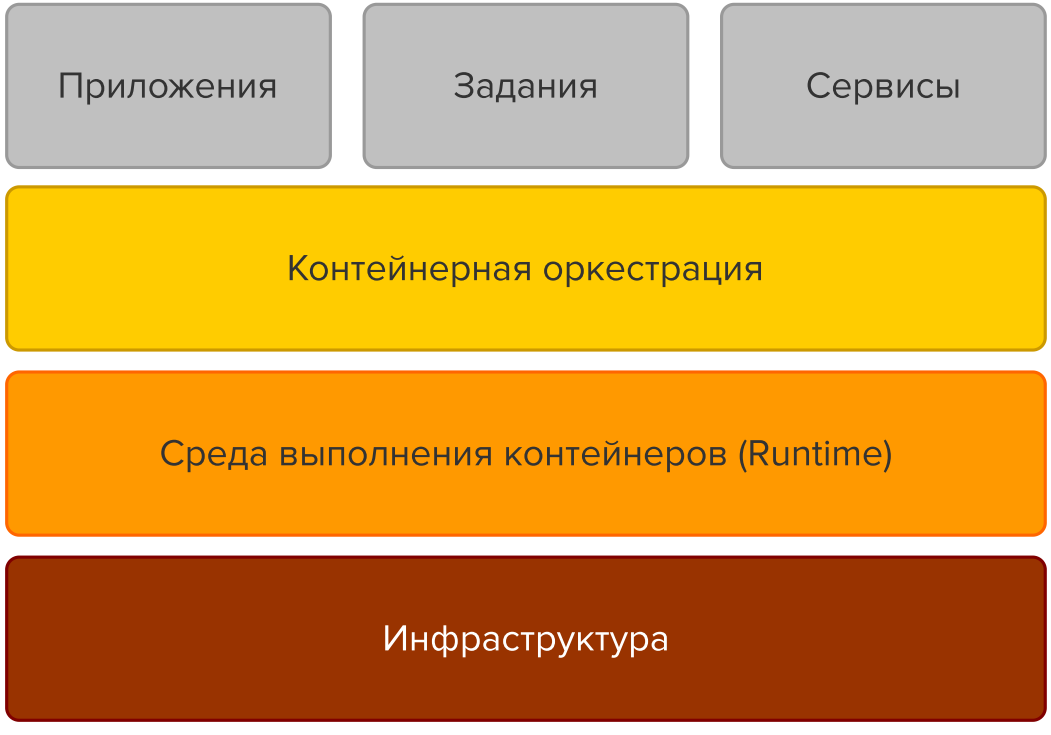
- Каждый планировщик независимо выполняет выделение ресурсов, используя общее хранилище состояния кластера
- После выделения ресурсов он коммитит транзакцию задачу в общий стейт
- Есть шанс множественного Split Brain и затраты на разрешение конфликтов



Контейнерая оркестрация

- Управление кластером
- Планирование и распределение задач
- Автоматизация
- Управление сервисами для контейнеров
  - Service Discovery
  - Load balancing, networking
  - Storages
  - ...

Контейнерная оркестрация



Пользовательская нагрузка

Распределенное управление контейнерами

Управление контейнерами на локальной машине

Инфраструктура, независимая от контейнеров (agnostic)

Контейнерная платформа

Пользовательская нагрузка

Управление системными службами

Распределенное управление контейнерами

Управление контейнерами на локальной машине

Управление инфраструктурой, специфичной для контейнеров

Инфраструктура, независимая от контейнеров (agnostic)

Управление ресурсами

- Память
- CPU
- GPU
- Хранение данных
  - локальные хранилища
  - удаленные хранилища
- Сетевые ресурсы
  - Порты
  - Управление IP адресами
  - Управление туннелями и overlay-сетями

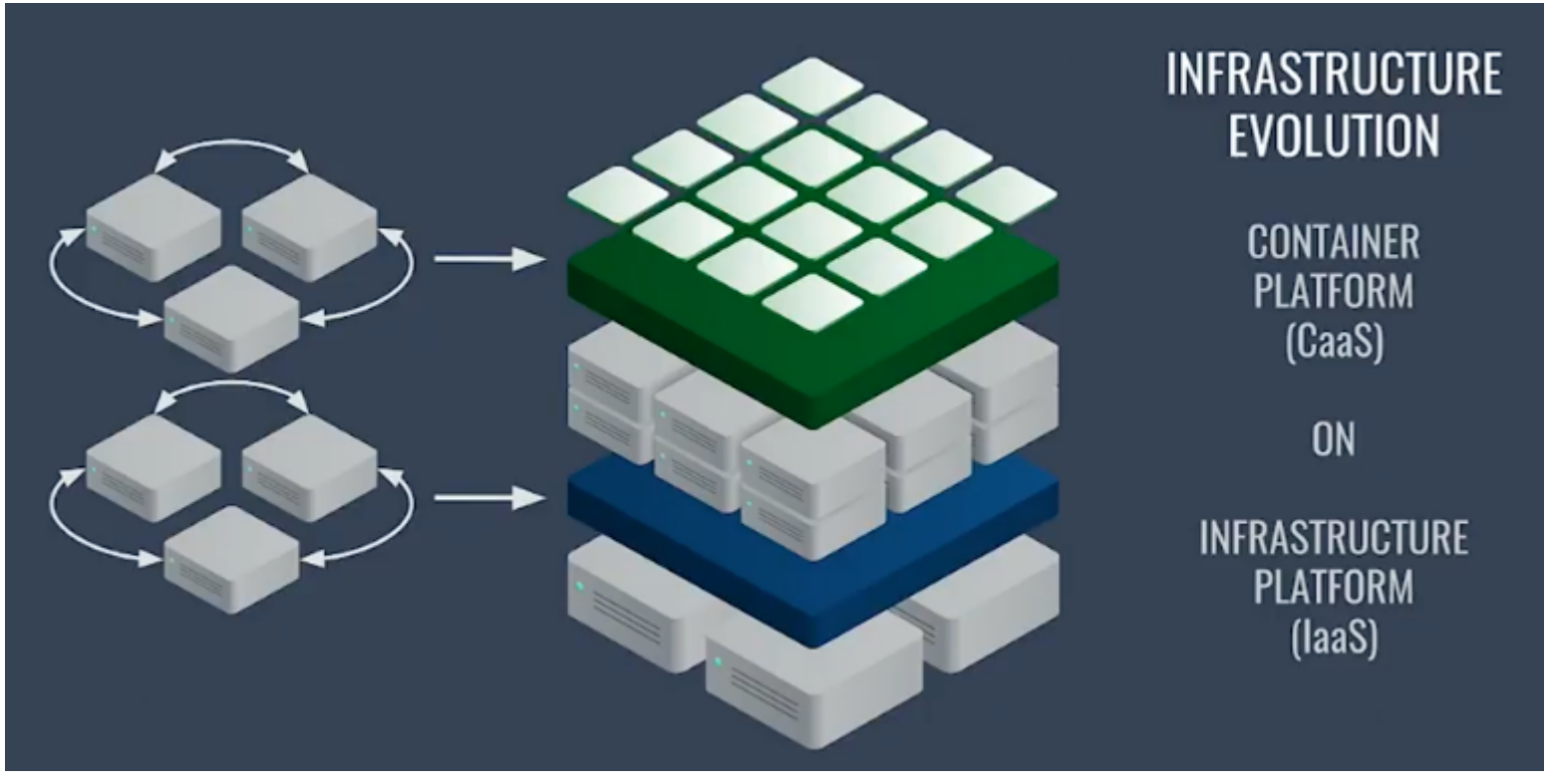
Планирование

- Расположение (bin packing, affinity rules)
- Реплицирование/масштабирование
- Проверки готовности/жизнеспособности
- Воскрешение
- Перепланирование (rescheduling)
- Cron/batch jobs
- Запуск демонов

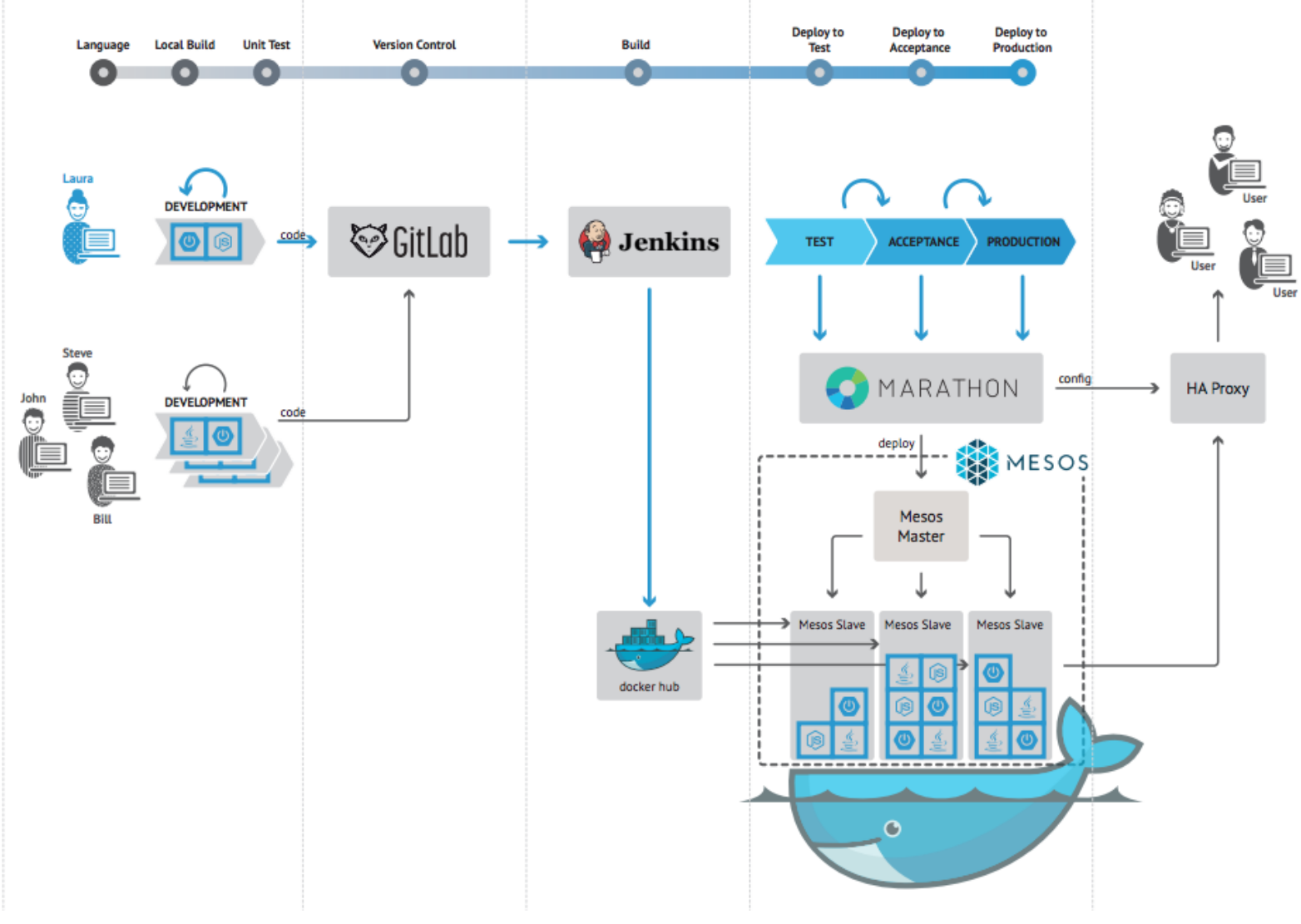
Управление сервисами

- Метки
- Группы
- Зависимости
- Балансировка нагрузки
- Virtual IP
- DNS и регистрация в Service Discovery
- Управление секретами
- Хранение и шаблонизация конфигурационных файлов

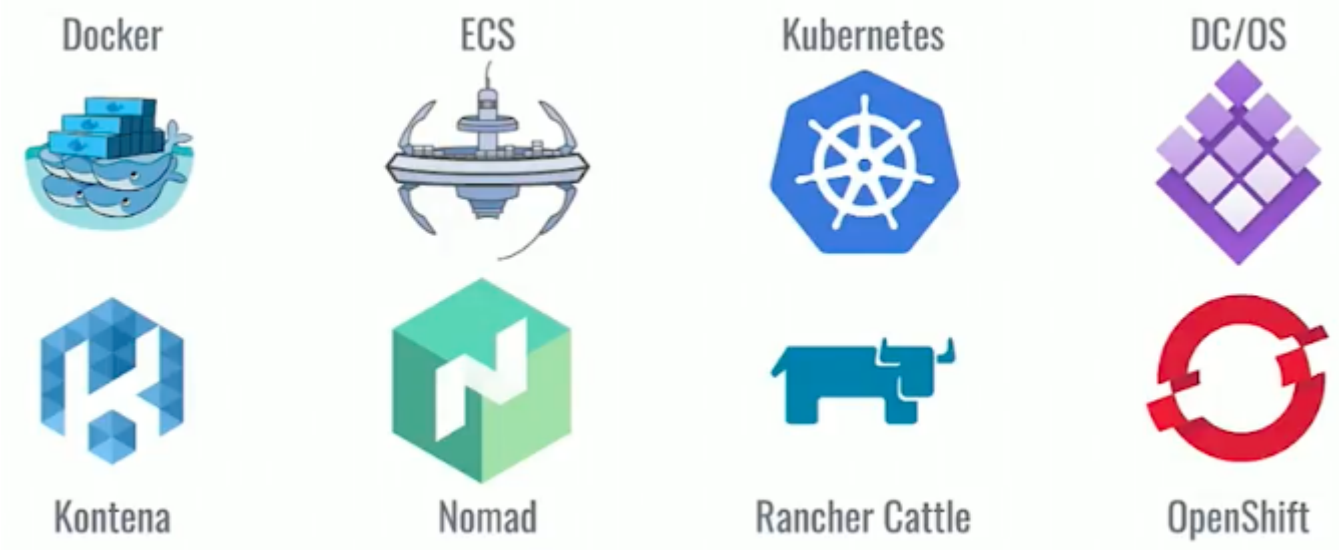
Контейнерная оркестрация



Место в конвейере



Примеры



У большинства под капотом - Kubernetes. И почти все копируют архитектуру Google Borg/Omega.

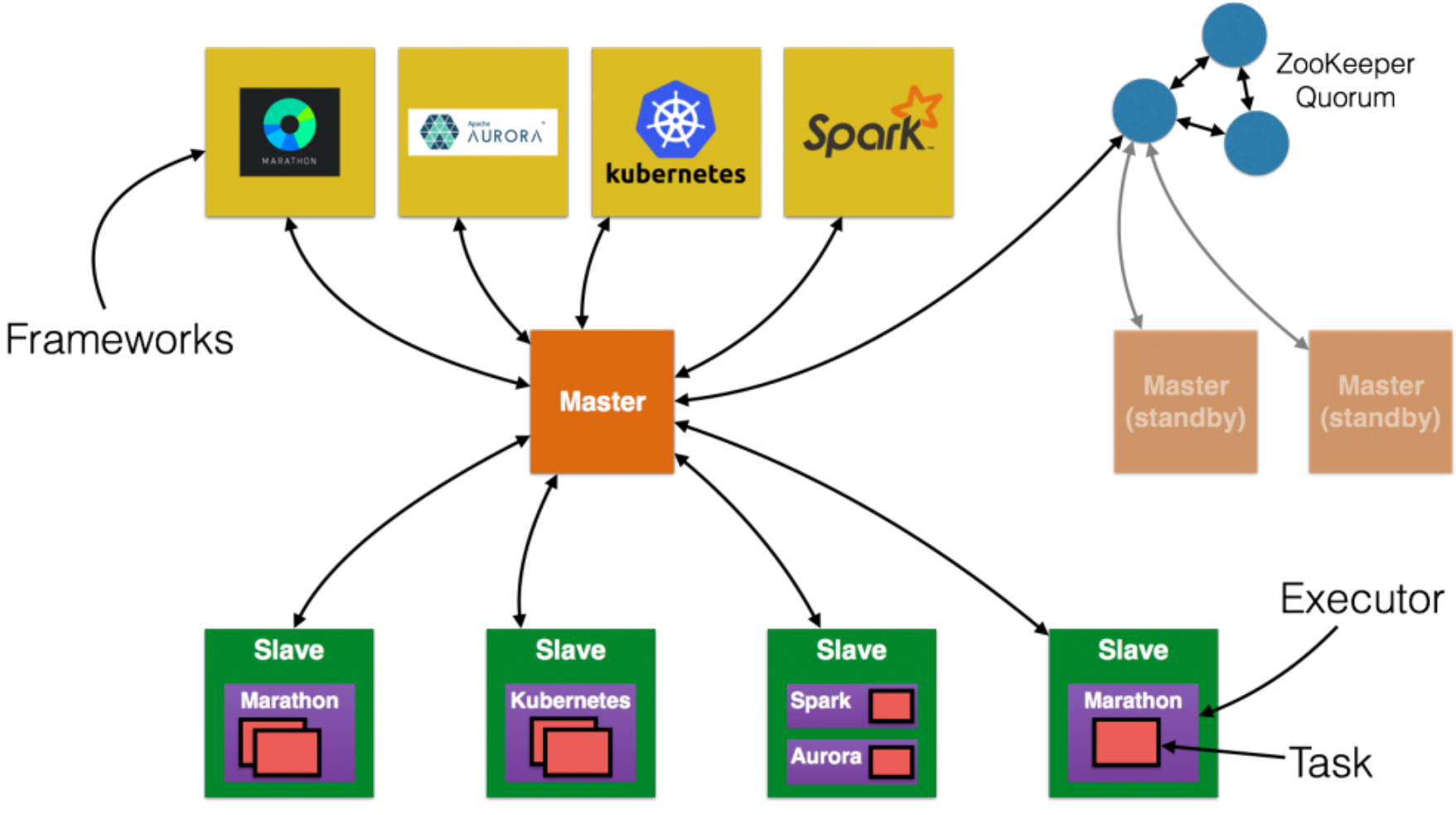
DC/OS (Mesos + Marathon, Aurora)

- 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 (планировщик ресурсов)
- **Marathon** - планировщик для Docker-контейнеров
- **Aurora** - WebUI для Mesos
- Поддерживает **Kubernetes (beta)** в виде "приложения"

DC/OS (Mesos + Marathon, Aurora)



HashiCorp Nomad

HashiCorp
Nomad

- Поставляется одним бинарником
- Запуск задач с помощью различных драйверов (Docker, rkt, Isolated Fork/Exec, Java, Qemu)
- Большой упор сделан на поддержку мультирегиональности
- Три типа планировщиков: Service, System и Batch (включая периодические и параметризованные задачи)
- Конфигурация в формате HCL

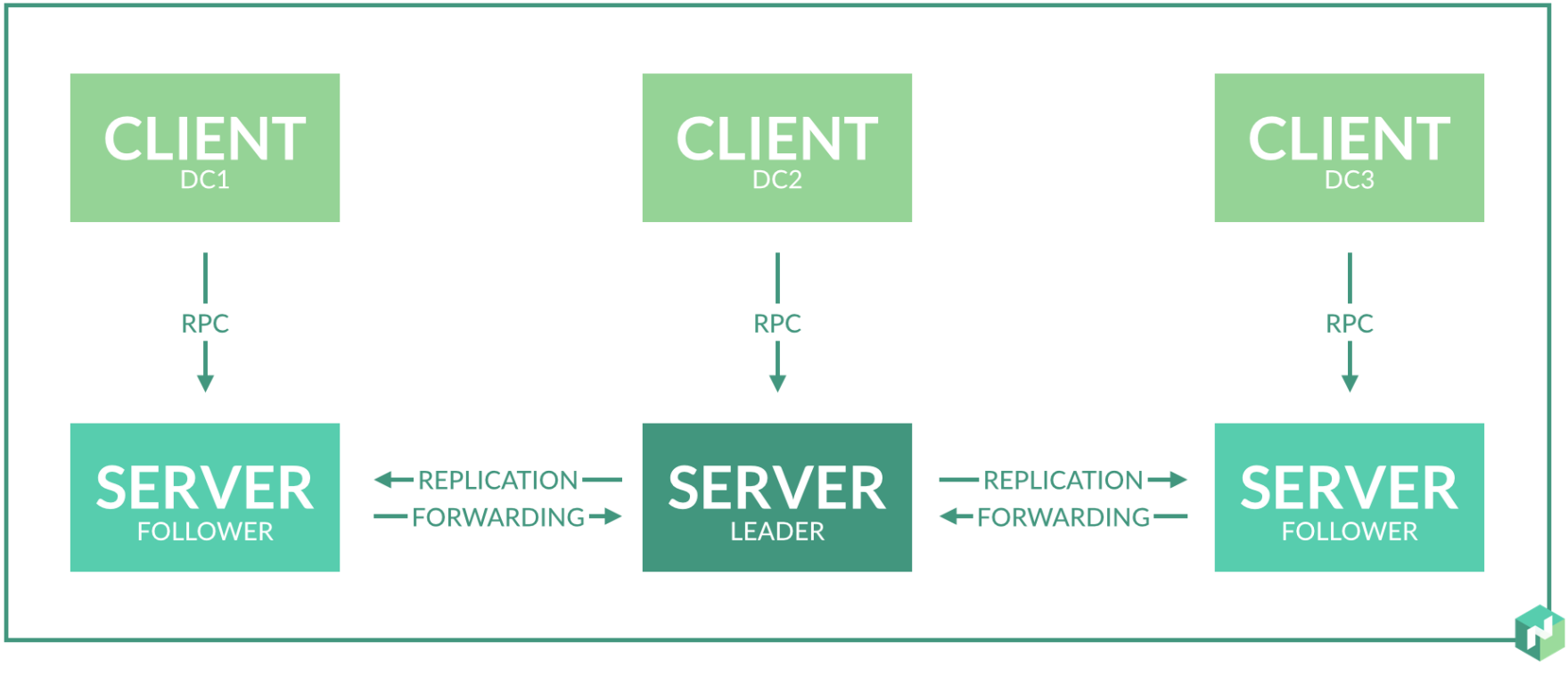
HashiCorp Nomad

HashiCorp
Nomad

- Service discovery в HashiCorp Consul
- Хранение секретов в HashiCorp Vault
- Нет встроенного Load-Balancing
  - зато это есть в Consul Connect, Traefik и Fabio
- Сам не управляет сетями и хранением данных (использует функционал Docker и т.п.)

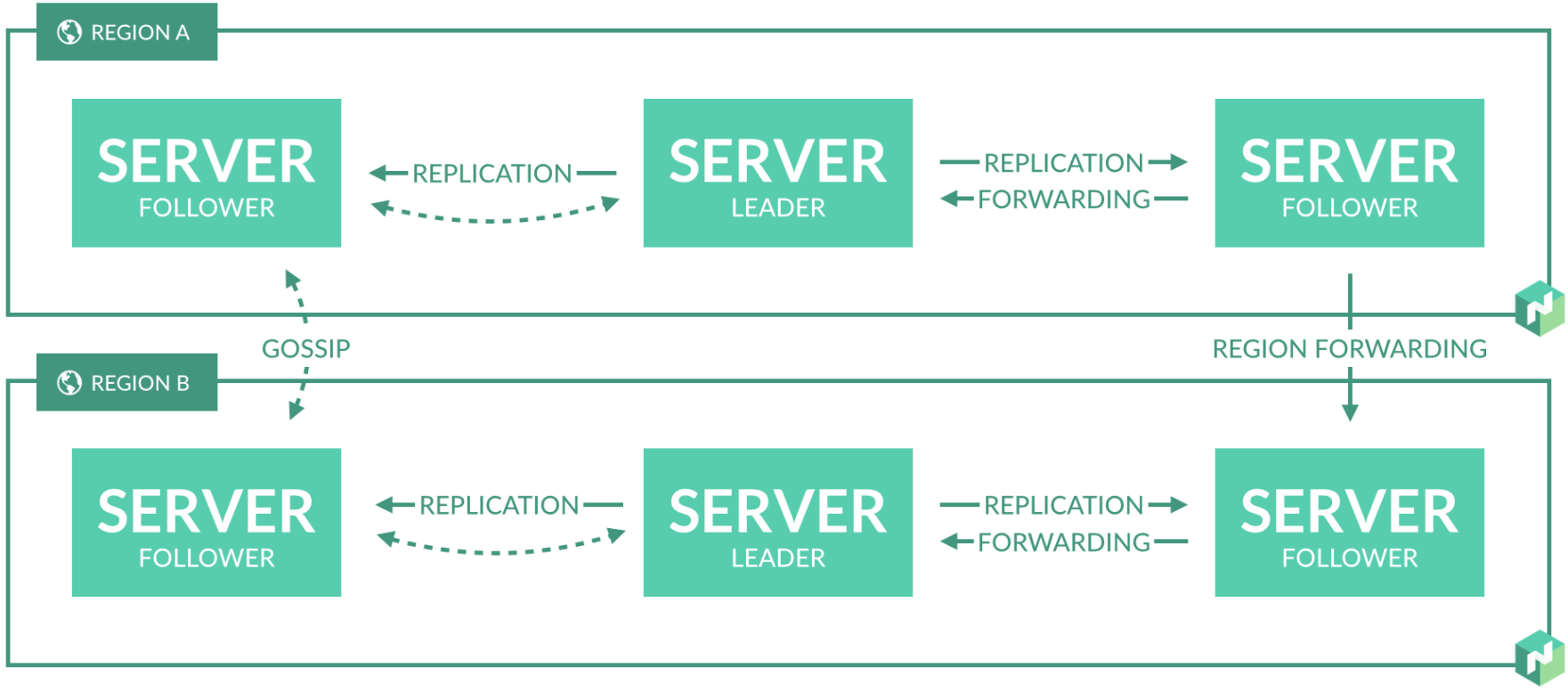
HashiCorp Nomad

Client-Server (внутри региона)



HashiCorp Nomad

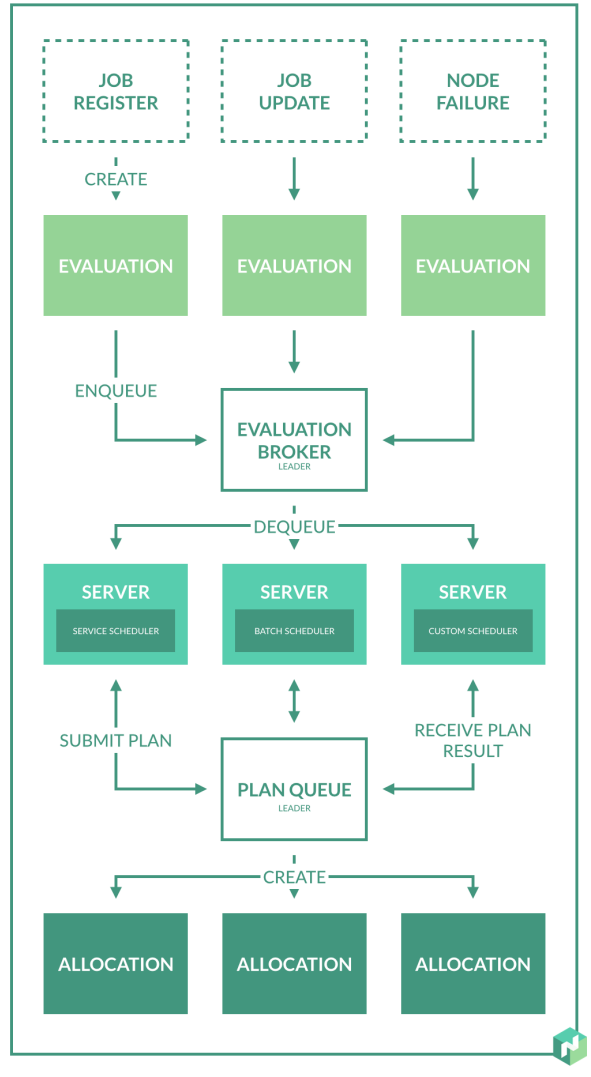
Multi-Region



HashiCorp Nomad

Сущности Nomad:

- **Evaluation** - задача по выделению ресурсов
- **Job** - набор связанных групп задач
- **Group** - задачи, которые всегда размещаются на одной ноде (аналог K8s Pod)
- **Task** - конкретная задача (например, контейнер Docker)
- **Allocation** - набор ресурсов для экземпляра группы задач

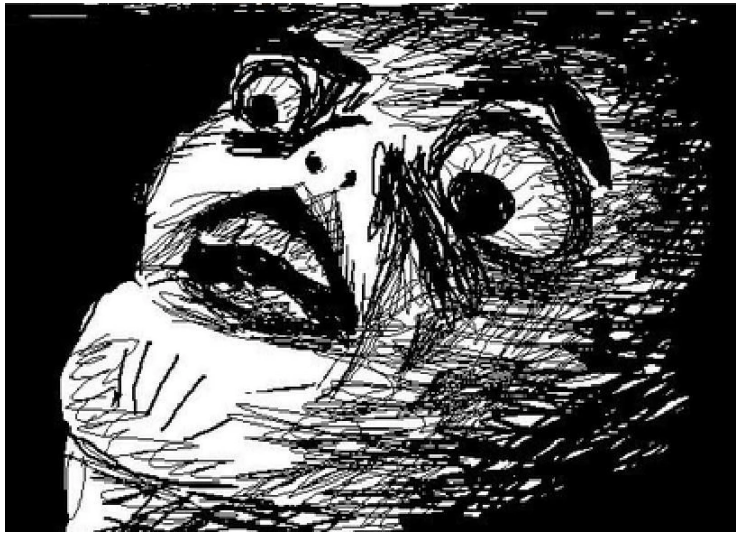


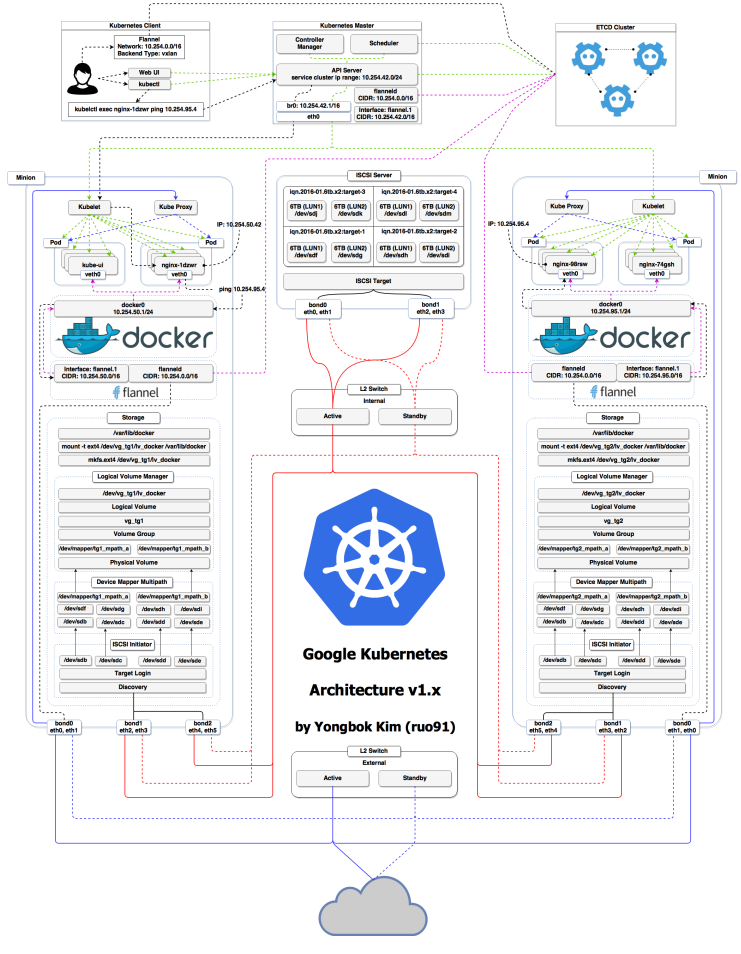
Kubernetes

- 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 число 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 Swarm

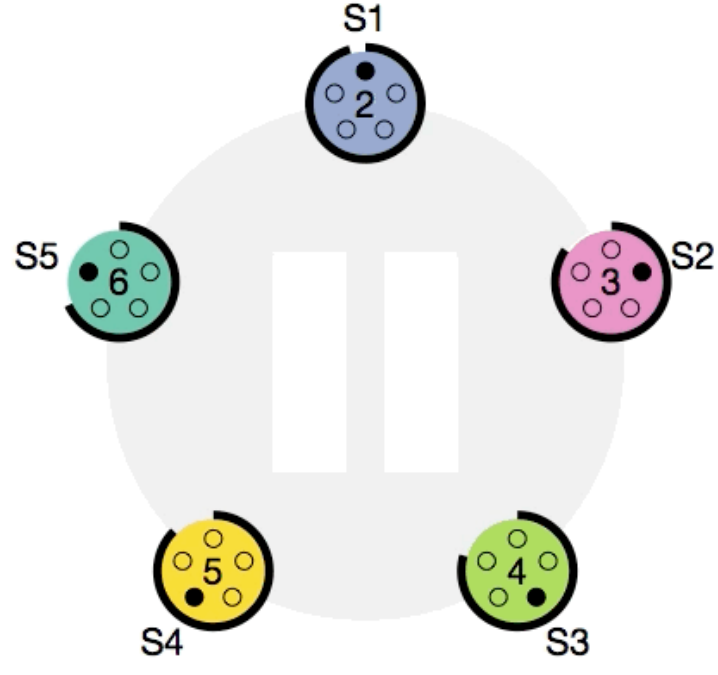
- Поставляется из коробки с Docker (начиная с версии 1.12)
- Разворачивается "одной командой" (начиная с версии 1.12)
- Может не использовать внешние распределенные хранилища
- Позволяет описывать конфигурацию в формате Docker Compose
- Есть *примитивный* встроенный Service Discovery
- Есть *примитивный* встроенный Load Balancer

Оркестраторы

- Используют общее распределенное консистентное хранилище, обычно на основе алгоритмов Raft/Paxos
- Спецификация задания описывает целевое состояние (декларативна)
- Основной процесс кластера - *reconciliation loop*, цикл обработки событий в кластере (состояние рабочих нод, изменение спецификации заданий) и запуска планировщика ресурсов
- Планировщик ресурсов является отдельной сущностью, используется для генерации плана изменений

RAFT-протокол

- Алгоритм сходимости и консистентной репликации лога изменений
- Можно потерять (N-1)/2 участников ("меньшую половину")
- Нечетное число участников лучше
- Документ об алгоритме согласования доступен по ссылке



https://raft.github.io/

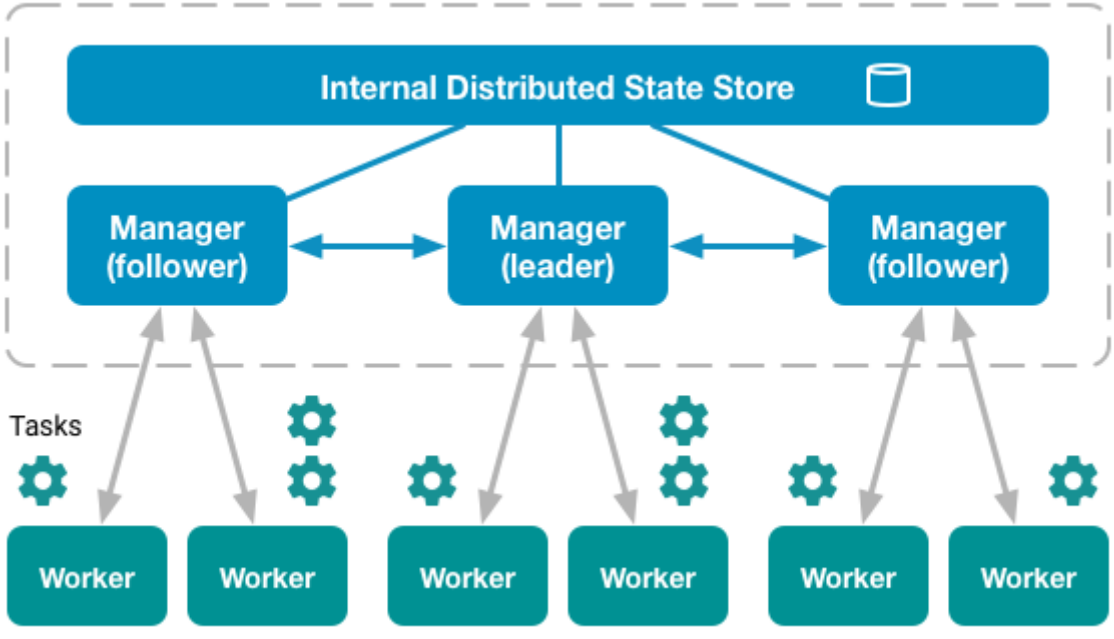
Docker Swarm: Основные концепции

- **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 состояния сервисов)

Docker Swarm: Основные концепции

- **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

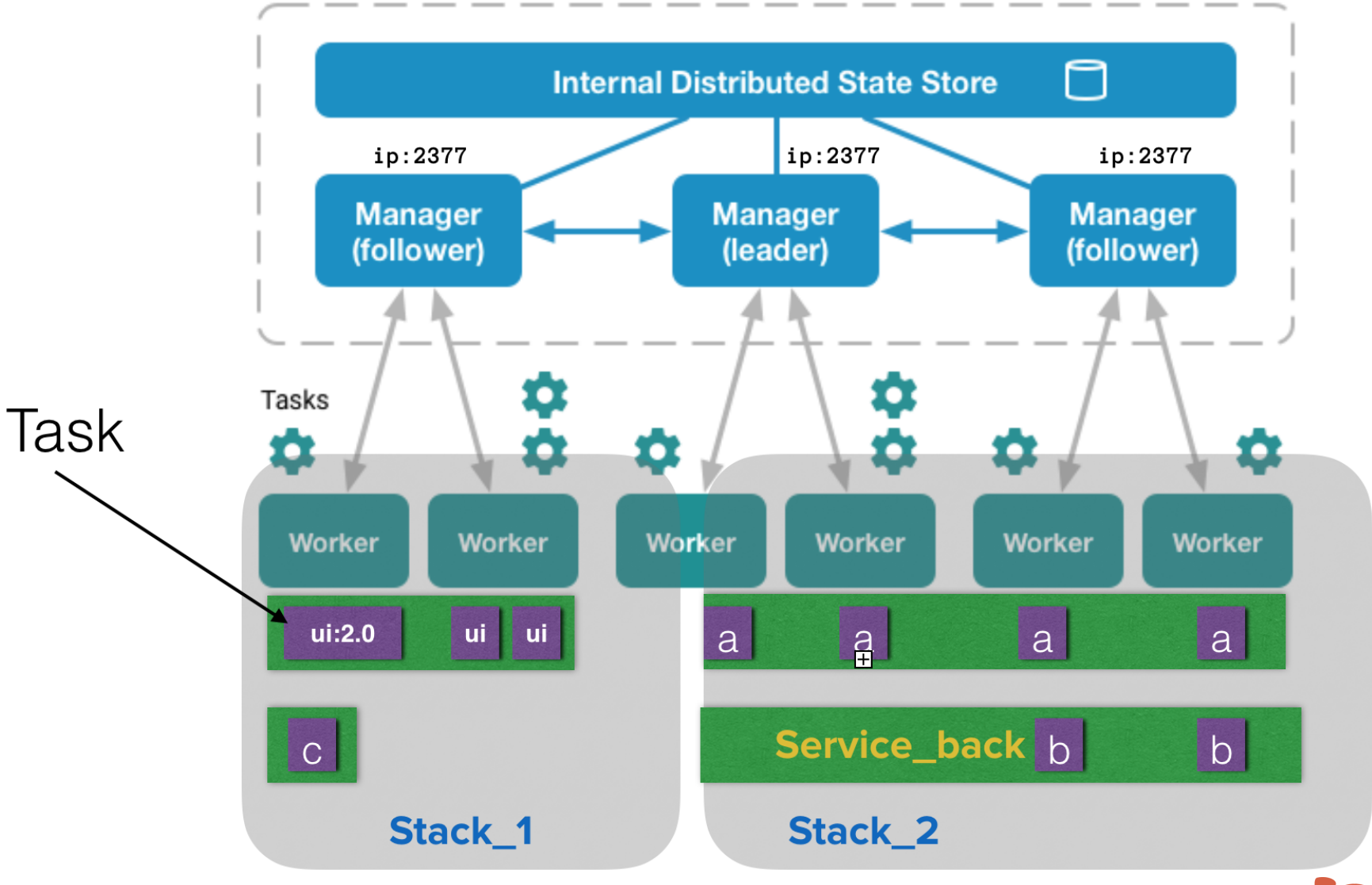
Docker Swarm



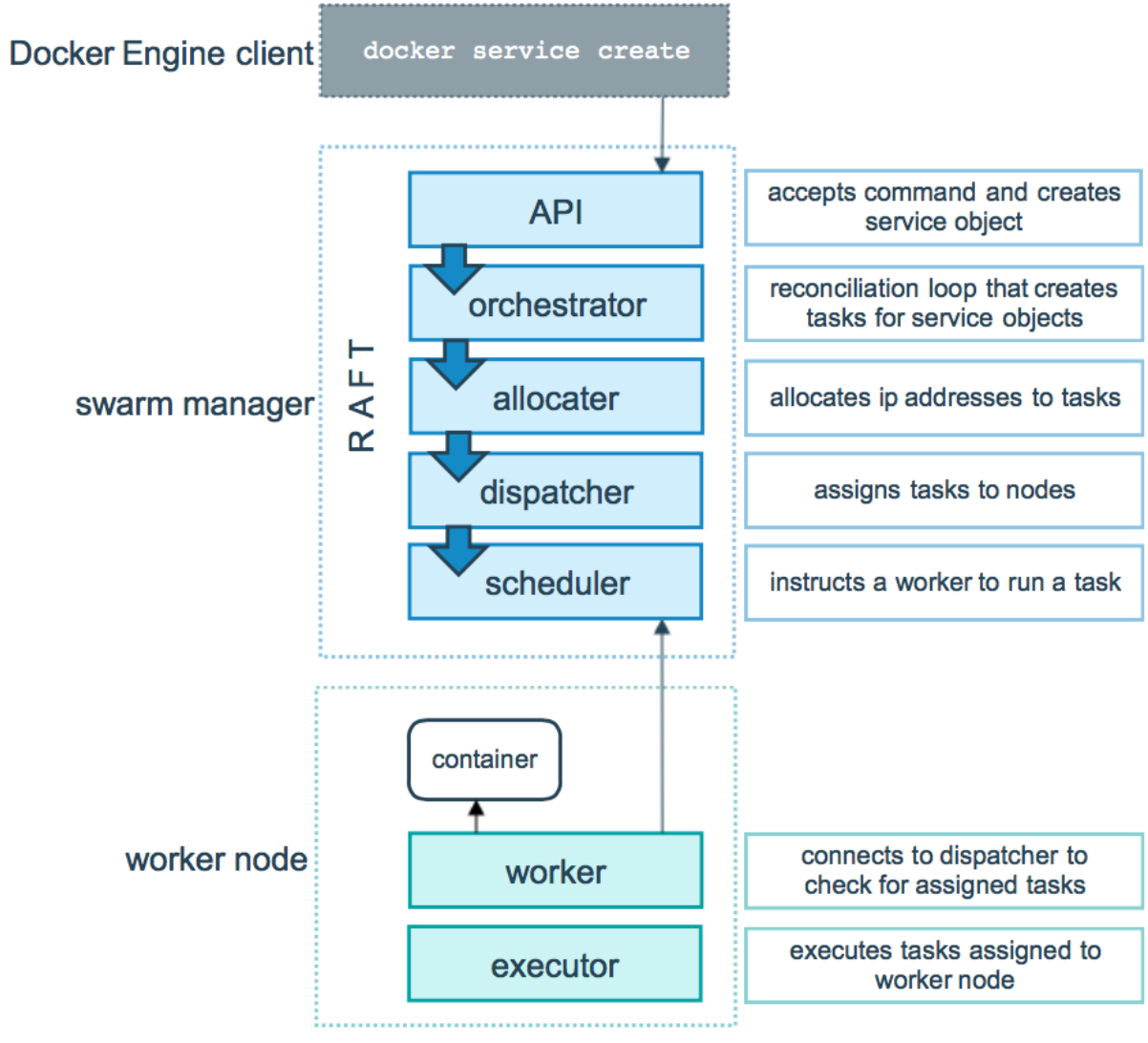
Docker Swarm: Основные концепции

- **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*

Docker Swarm



Docker Swarm: Task flow



Docker Swarm: Начало

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.
```

Docker Swarm: Начало

Выполним на manager-ноде:

```
>> docker node ls
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.
```

Docker Swarm: Работа с сервисами

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
```

Docker Swarm: Работа с сервисами

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>
```

Контейнеры и сервисы

```
$ docker-user@swarm-master-1:~$ sudo docker service ps  rkpyjqy7cc3v
ID            NAME              IMAGE         NODE     DESIRED   CURRENT STATE
                                                       STATE
rsuzjn4yck34  reverent_lala.1   nginx:latest  swarm02  Running   Running 3 minutes ago
ku3te50h28yf  reverent_lala.2   nginx:latest  swarm01  Running   Running 3 minutes ago
zlqegiucdjug  reverent_lala.3   nginx:latest  swarm02  Running   Running 3 minutes ago
la8e4ss2ikqf  reverent_lala.4   nginx:latest  swarm02  Running   Running 3 minutes ago
```

Задачи для сервисов распределены стратегией **Spread**:

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