

# Оркестрация кластером Docker контейнеров на примере Docker Swarm



Олег  
Букатчук



**Олег Букатчук**

Software Architect DevOps, [crif.com](https://crif.com)





# План занятия

1. [Введение. Возможности Docker Swarm](#)
2. [Архитектура Docker Swarm](#)
3. [Базовые команды Docker Swarm](#)
4. [Теорема CAP](#)
5. [Развёртывание стека микросервисов в Docker Swarm кластере](#)
6. [Итоги](#)
7. [Домашнее задание](#)



**Введение.**

# **Возможности Docker Swarm**

---

# Docker Swarm

**Docker Swarm** — это система кластеризации для Docker, которая превращает набор Docker хостов в полноценный кластер, называемый Docker Swarm.

Каждый хост, в составе такого кластера выступает в качестве, либо управляющей ноды (manager), либо рабочей (worker).

В кластере должен быть, как минимум, один управляющий хост (manager).

---

# Docker Swarm

Теоретически, физическое расположение машин не имеет значения, однако, **желательно иметь все Docker-ноды внутри одной локальной сети.**

В противном случае, управление операциями или поиск консенсуса между несколькими управляющими нодами может занять значительное количество времени.

Начиная с версии **Docker 1.12**, **Docker Swarm** уже интегрирован в **Docker Engine** как Swarm-режим.

В более старых версиях необходимо было запускать **swarm-контейнер** на каждом из хостов для обеспечения функционала кластеризации.

---

# Возможности Docker Swarm

- **Балансировка нагрузки**

Docker Swarm отвечает за балансировку нагрузки и назначение **уникальных DNS-имен**, чтобы приложение, развернутое в кластере, можно было использовать так же, как, если бы приложение было развернуто на одном Docker Engine хосте.

Другими словами, Docker Swarm может публиковать порты так же, как контейнер в Docker Engine, а затем **управляющая нода** **распределяет запросы между service-ами** в кластере.

# Возможности Docker Swarm

- **Динамическое управление ролями: `manager/worker`**

Docker-хосты могут быть добавлены Swarm кластеру без необходимости перезапуска кластера.

Более того, роль узла (управляющий или рабочий) также может динамически меняться на лету.

**Для того чтобы динамически добавить/убрать роль `manager` нужно выполнить команду:**

```
# Добавление роли manager
$ docker node promote <node name>
# Удаление роли manager
$ docker node demote <node name>
```



---

# Возможности Docker Swarm

- **Динамическое масштабирование сервисов**

Каждый **service**, запущенный в Swarm кластере, может динамически масштабироваться, как в сторону увеличения, так и в сторону уменьшения **количества реплик**.

Управляющая нода (manager) заботится о добавлении или удалении контейнеров на рабочих узлах кластера.

```
# Добавление реплик сервиса
$ docker service update --replicas=3 my-service
# Откат изменений (отмена последнего изменения конфигурации)
$ docker service rollback my-service
```

---

# Возможности Docker Swarm

- **Восстановление при отказе узлов**

Рабочие ноды постоянно контролируются управляющей нодой и, если какая-либо нода сбоит, то **новые задачи запускаются на других рабочих нодах** с целью обеспечения заявленного (желаемого) количество реплик.

Docker Swarm также позволяет **создавать несколько управляющих нод для предотвращения поломки кластера** в случае выхода из строя единственной управляющей ноды.

---

# Возможности Docker Swarm

- Обновления с задержкой (rolling updates)

Обновление сервисов может **применяться постепенно**.

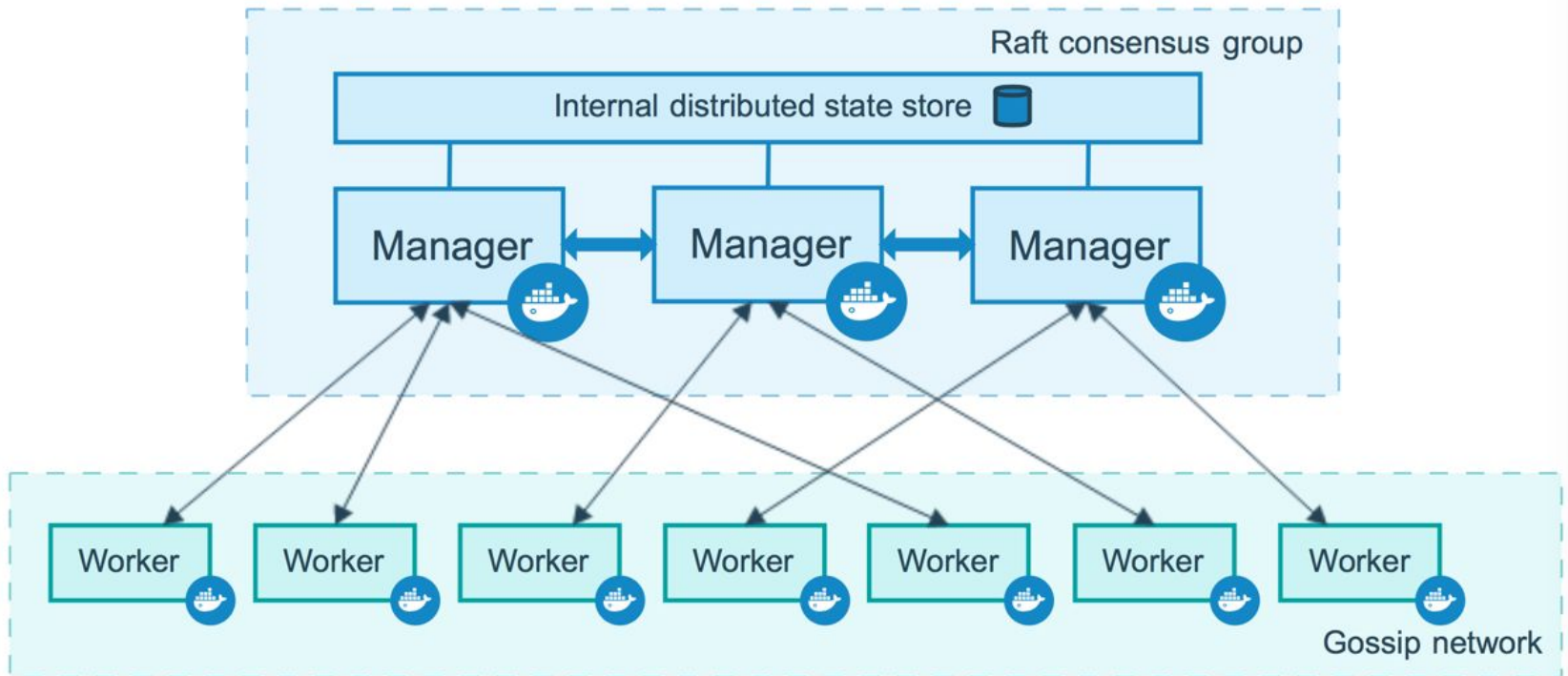
Например, если у нас есть 10 реплик, и мы хотим внести изменения (обновить версию нашего сервиса), мы можем **определить задержку между развертыванием** для каждой реплики.

В таком случае, когда что-то пойдет не так, **процесс обновления автоматически прерывается**, тем самым защищая нас от ситуации, когда в кластере не останется рабочих реплик.

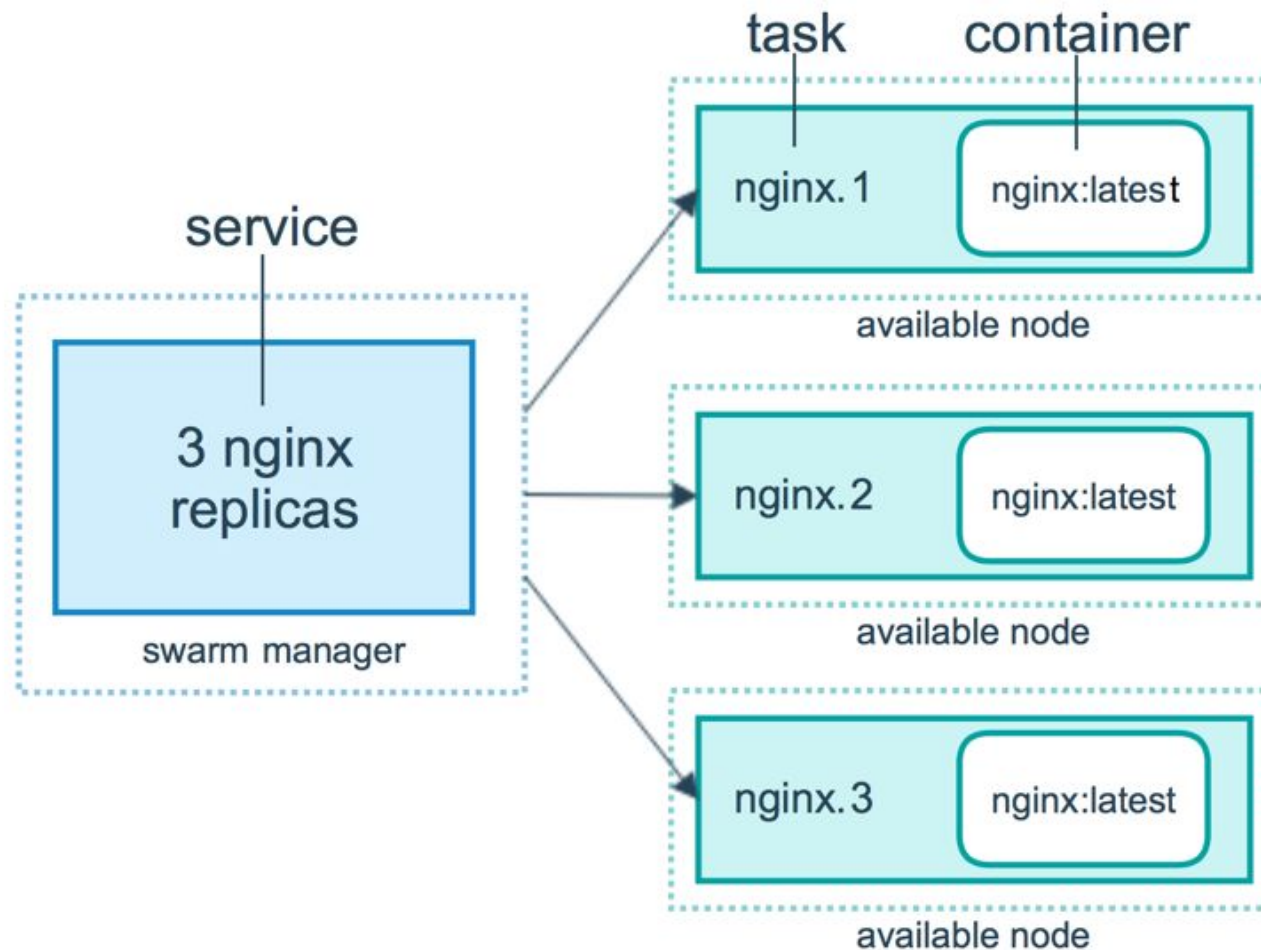


# Архитектура Docker Swarm

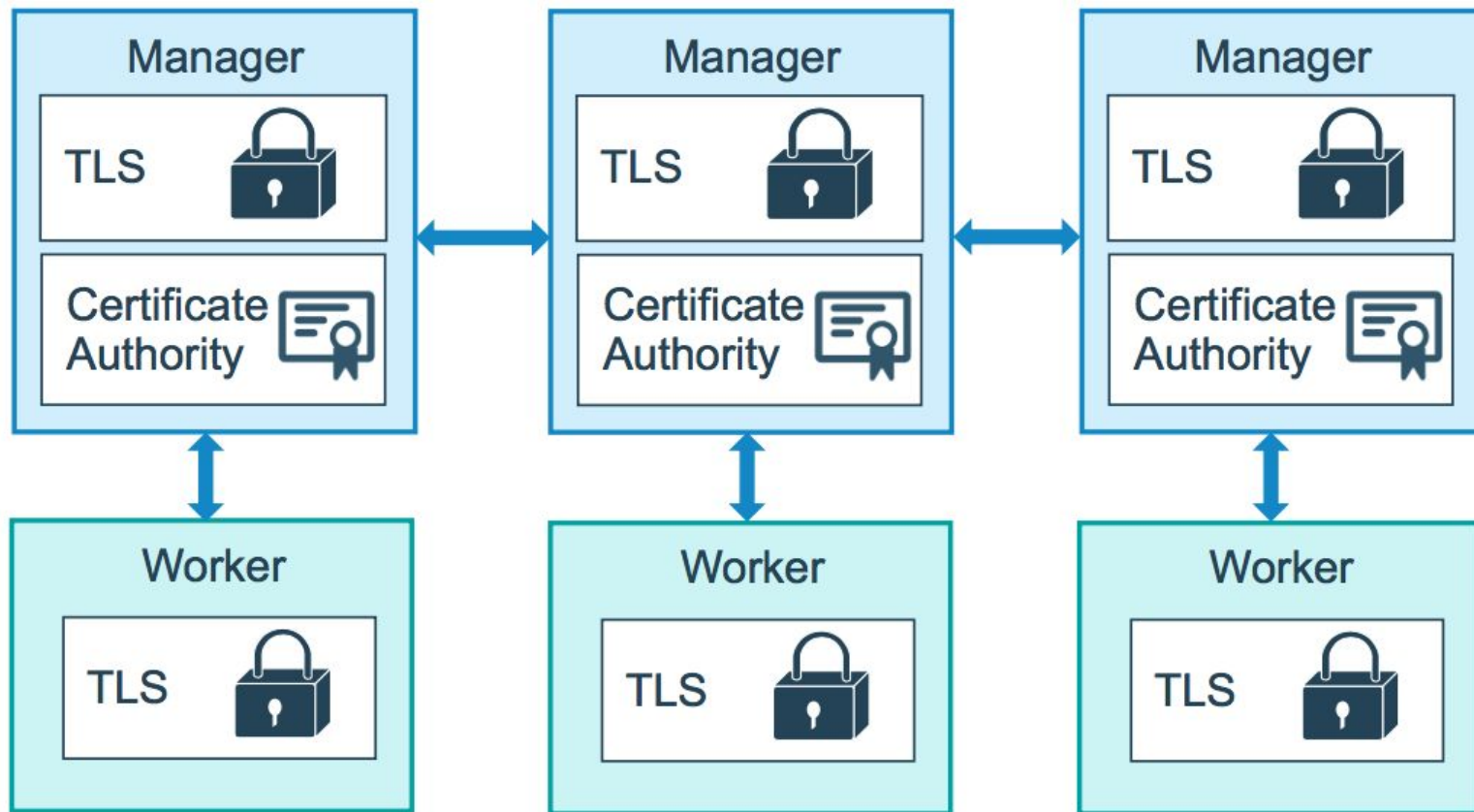
# Архитектура Docker Swarm



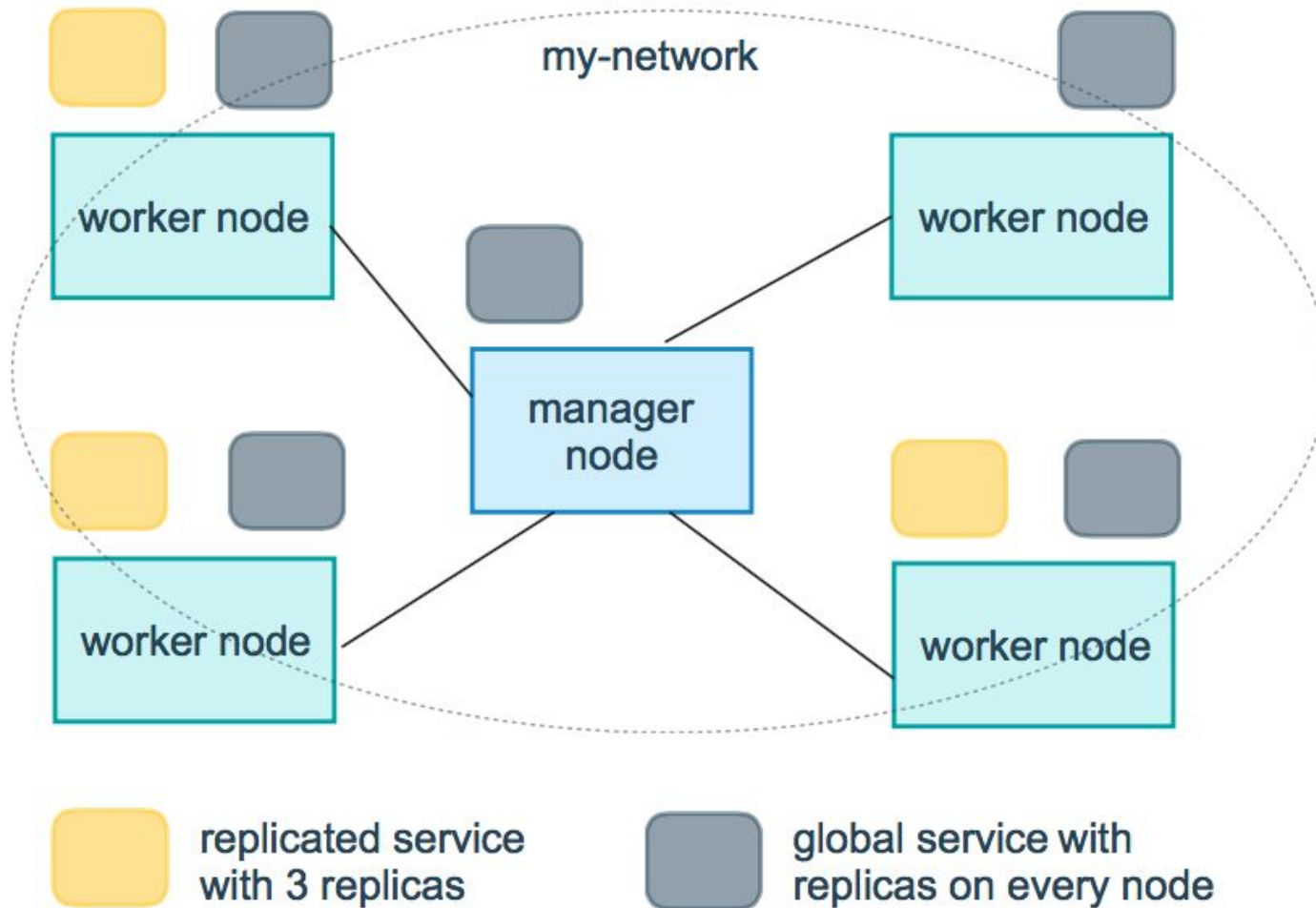
# Архитектура Docker Swarm



# Архитектура Docker Swarm



# Архитектура Docker Swarm





---

# Сети в кластере Docker Swarm

При разворачивании Swarm кластера на VM с публичными IP-адресами хорошей практикой является настройка правил брандмауэра для разрешения трафика Docker Swarm на каждом сервере перед созданием кластера.

Для успешного создания кластера необходимо, чтобы каждая VM могла связаться друг с другом по следующим протоколам и портам:

- TCP порт 2377 для обеспечения связи с целью управления кластером;
- TCP и UDP порт 7946 для связи между нодами;
- UDP порт 4789 для трафика overlay-сети.



# **Базовые команды Docker Swarm**

# Базовые команды Docker Swarm

- **docker swarm init** — инициализация кластера. Кластер будет инициализирован, как **single-mode instance**. Так же этой ноде будет автоматически присвоена роль **manager**. [Подробнее](#)

```
# Инициализация кластера Docker Swarm
```

```
$ docker swarm init --advertise-addr <ip address>
```

```
Swarm initialized: current node (bvz81updecsj6wjz393c09vti) is now a manager.
```

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

```
docker swarm join \ --token
```

```
SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znp26u2lkl-1awxwuwd  
3z9j1z3puu7rcgdbx \ <ip address>:2377
```

```
To add a manager to this swarm, run 'docker swarm join-token  
manager' and follow the instructions.
```

# Базовые команды Docker Swarm

- **docker swarm join** — добавление в кластер новых серверов.  
**ВАЖНО!!!** В зависимости от того, какой **ключ** указать вводимый в кластер сервер получит либо **роль worker**, либо **роль manager**.
- **docker swarm join-token** — вывод актуальных ключей для добавления нод в кластер. [Подробнее](#)

```
# Добавление ноды в кластер Docker Swarm
```

```
$ docker swarm join --token
```

```
SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfc8p2is99znp26u2lkl-1awxwud3z9j1z3puu7rcgdbx \
```

```
<ip address>:2377
```

```
$ docker swarm join-token -q worker
```

```
SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfc8p2is99znp26u2lkl-1awxwud3z9j1z3puu7rcgdbx
```

```
$ docker swarm join-token -q manager
```

```
SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfc8p2is99znp26u2lkl-7p73s1dx5in4tatdymyhg9hu2
```

# Базовые команды Docker Swarm

- **docker swarm ca** — просмотр и обновление сертификатов кластера. Кластер по-умолчанию **при инициализации создает цепочку сертификатов** для безопасной коммуникации и передачи данных между нодами. [Подробнее](#)

```
# Просмотр и обновление сертификатов кластера Docker Swarm
$ docker swarm ca
-----BEGIN CERTIFICATE----- .... -----END CERTIFICATE-----
$ docker swarm ca --rotate
desired root digest: sha256:05da740cf2577a25224...
rotated TLS certificates: [=====> ] 1/3 nodes
rotated CA certificates: [> ] 0/3
nodes
```

---

# Базовые команды Docker Swarm

- **docker swarm leave** — удаление ноды из кластера.  
**ВАЖНО!!!** Перед удалением ноды из кластера, во избежание простоев работающих сервисов, **нужно очистить ноду** от запущенных на ней сервисов. [Подробнее](#)

```
# Очистка Docker Swarm ноды перед удалением из кластера
```

```
$ docker node update --availability drain node01
```

```
node01
```

```
# Удаление Docker Swarm ноды из кластера
```

```
$ docker swarm leave
```

```
Node left the default swarm.
```

# Базовые команды Docker Swarm

- **docker node** — набор команд для управления свойствами, ролями, атрибутами нод Docker Swarm кластера. Доступны команды: `ls`, `promote`, `demote`, `inspect`, `ps`, `rm`, `update`. [Подробнее](#)

```
# Добавление роли manager для 2-х Docker Swarm нод в работающем кластере
```

```
$ docker node promote node02 node03
```

```
Node node02 promoted to a manager in the swarm.  
Node node03 promoted to a manager in the swarm.
```

```
# Удаление роли manager для 2-х Docker Swarm нод в работающем кластере
```

```
$ docker node demote node02 node03
```

```
Node node02 demoted to a manager in the swarm.  
Node node03 demoted to a manager in the swarm.
```

# Базовые команды Docker Swarm

- **docker service** — набор команд для управления сервисами и их свойствами, работающими в Docker Swarm кластере.  
Доступны команды: create, inspect, logs, ps, ls, rollback, rm, scale, update. [Подробнее](#)

```
# Добавление сервиса nginx в количестве 3-х реплик и определение критериев для целевых нод
```

```
$ docker service create \
  --name web \
  --replicas 3 \
  --replicas-max-per-node 1 \
  --constraint node.platform.linux==linux \
  nginx:alpine
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
b6lww17hrr4e	web	replicated	3/3	nginx:alpine	



---

# Базовые команды Docker Swarm

- **docker stack** — набор команд для управления сервисами и их свойствами, в формате идентичном Docker Compose, но для Docker Swarm кластера. [Подробнее](#)  
Доступны команды: `deploy`, `ls`, `ps`, `rm`, `services`.

```
# Деплой сервиса nginx с использованием конфигурационного Compose файла
```

```
$ docker stack deploy --compose-file docker-compose.yml nginx
```

```
Creating network nginx_nginx
Creating network nginx_default
Creating service nginx_nginx
```

```
$ docker stack rm netology
Removing service nginx_nginx
Removing network nginx_default
Removing network nginx_nginx
```



# Теорема CAP

---

# Теорема CAP

Теорема CAP (известная также как теорема Брюера) — эвристическое утверждение о том, что в любой реализации распределенных вычислений возможно обеспечить не более двух из трёх следующих свойств:

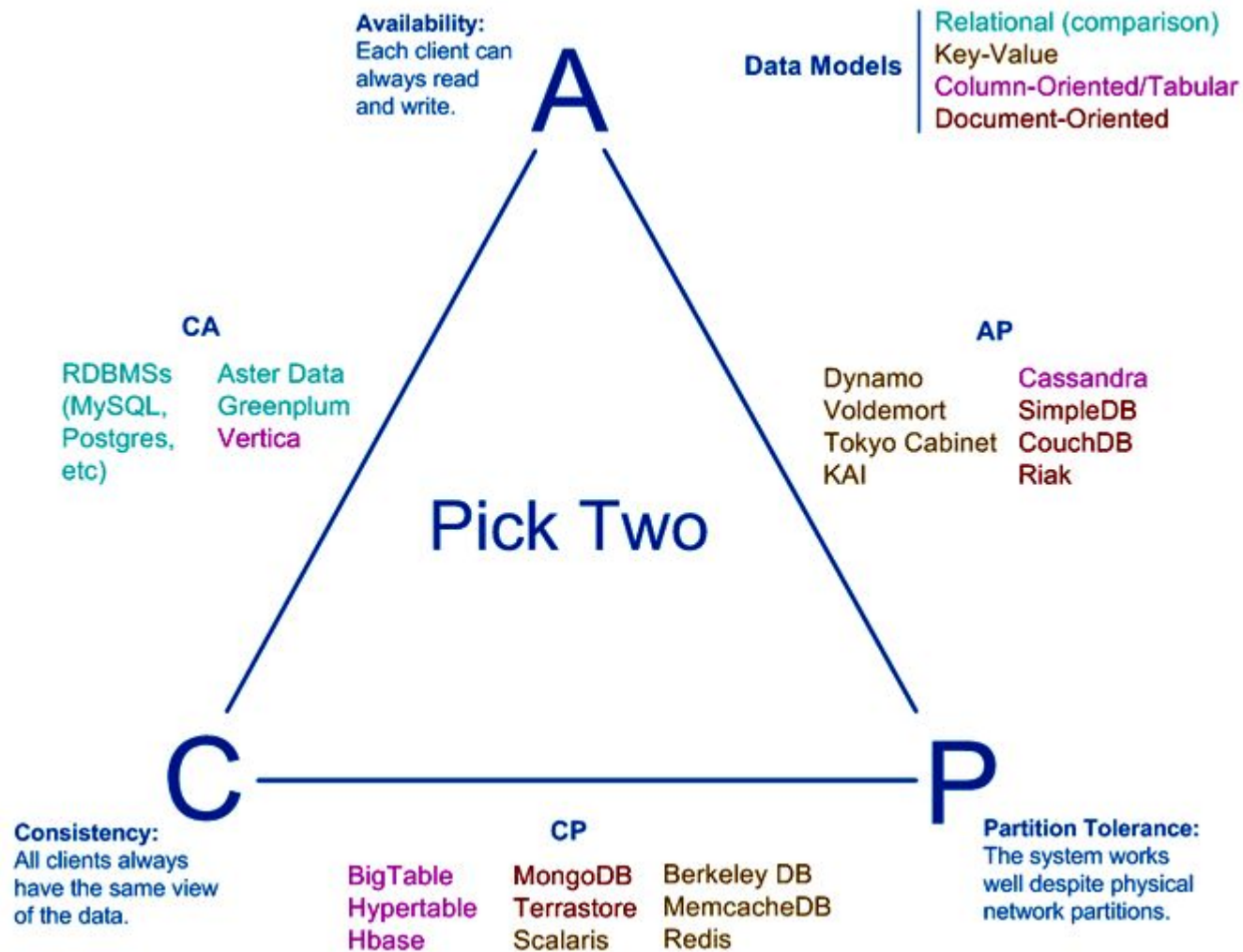
- Consistency (согласованность данных);
- Availability (доступность);
- Partition tolerance (устойчивость к разделению).

---

# Теорема CAP

- **Согласованность данных** (*англ. consistency*) — во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
- **Доступность** (*англ. availability*) — любой запрос к распределенной системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;
- **Устойчивость к разделению** (*англ. partition tolerance*) — расщепление распределенной системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

# Теорема CAP



---

# Теорема CAP

**CA** (Availability + Consistency – Parition tolerance), когда **данные во всех узлах кластера согласованы и доступны, но не устойчивы к разделению.**

Это означает, что реплики одной и той же информации, распределенные по разным серверам по отношению друг к другу, не противоречат друг другу и любой запрос к распределенной системе завершается корректным откликом.

---

# Теорема CAP

**CA** (Availability + Consistency – Partition tolerance).

Такие системы возможны при поддержке **ACID**-требований к транзакциям (Атомарность, Согласованность, Изоляция, Долговечность) и абсолютной надежности сети.

На практике таких решений на основе кластерных систем управления базами данных **почти не существует**.

# Теорема CAP

**CP (Consistency + Partition tolerance – Availability)** в каждый момент обеспечивает целостность данных и способна работать в условиях распада в ущерб доступности, не выдавая отклик на запрос.

Устойчивость к разделению требует дублирования изменений во всех узлах системы, что реализуется с помощью распределенных пессимистических блокировок для сохранения целостности.



---

# Теорема CAP

**CP** (Consistency + Partition tolerance – Availability).

По сути, CP – это **система с несколькими синхронно обновляемыми мастер-базами**. Она всегда корректна, отработывая транзакцию, только в том случае, если изменения удалось распространить по всем серверам.

---

# Теорема CAP

**AP** (Availability + Partition tolerance – Consistency) не гарантирует целостность данных, обеспечивая их доступность и устойчивость к разделению, например, как в распределенных веб-кэшах и DNS.


Считается, что большинство NoSQL-СУБД относятся к этому классу систем, обеспечивая лишь некоторый уровень согласованности данных в конечном счете (eventually consistent).

---

# Теорема CAP

**AP** (Availability + Partition tolerance – Consistency).

Таким образом, AP-система может быть представлена кластером из нескольких узлов, каждый из которых **может принимать данные, но не обязуется в тот же момент распространять их** на другие сервера.



# **Развертывание стека микросервисов в Docker Swarm кластере**

---

## Практическая часть

1. Авторизуемся в Yandex.Cloud.
2. Создаём сеть и подсеть, чтобы собрать образ ОС с помощью Packer и запускаем сборку образа.
3. Удаляем подсеть и сеть, которую использовали для сборки образа ОС.
4. Создаём 6 виртуальных машин с помощью Terraform.
5. Создаём Docker Swarm кластер из виртуальных машин, созданных на предыдущем шаге.
6. Запускаем деплой стека приложений.
7. Проводим стресс тест Docker Swarm кластера.
8. Удаляем всё, чтобы не тратить деньги!



# Итоги

---

# Итоги

Сегодня мы:

- узнали о преимуществах технологии Docker Swarm;
- рассмотрели архитектуру Swarm кластера;
- научились создавать Docker Swarm;
- научились создавать простейший pipeline деплоя используя Terraform и Ansible;
- создали кластер высокой доступности на основе Docker Swarm и развернули в нём стек микросервисов, устойчивый к отказам виртуальных машин.

---

## Полезные материалы

- [In Search of an Understandable Consensus Algorithm \(Extended Version\)](#)
- [Raft \(визуализация\)](#)
- [Gossip](#)



---

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Олег Букатчук**