

Микросервисная архитектура на базе CoreOS и Kubernetes

Денис Измайлов

STARTUP^{MAKERS}

13 июля 2016



Денис Измайлова



- 16 лет опыта разработки ПО и web
- Последние 6 лет посвятил Front-end, Node.js и архитектуре
- Более 15 проектов SPA, React.js и high-load
- Коммиты в Redux, webpack и koa
- Спикер HighLoad++ 2015, AgileDays 2016, RIT 2016, DevConf 2016, MoscowJS, React Amsterdam Meetup
- Статьи на Habrahabr и Medium
- Два года назад основал Startup Makers

STARTUP^{MAKERS}



- Мы делаем web, мобильные приложения и DevOps для наших клиентов
- Мы используем самые передовые и эффективные технологии
- Более 20 талантливых разработчиков

Микросервисы

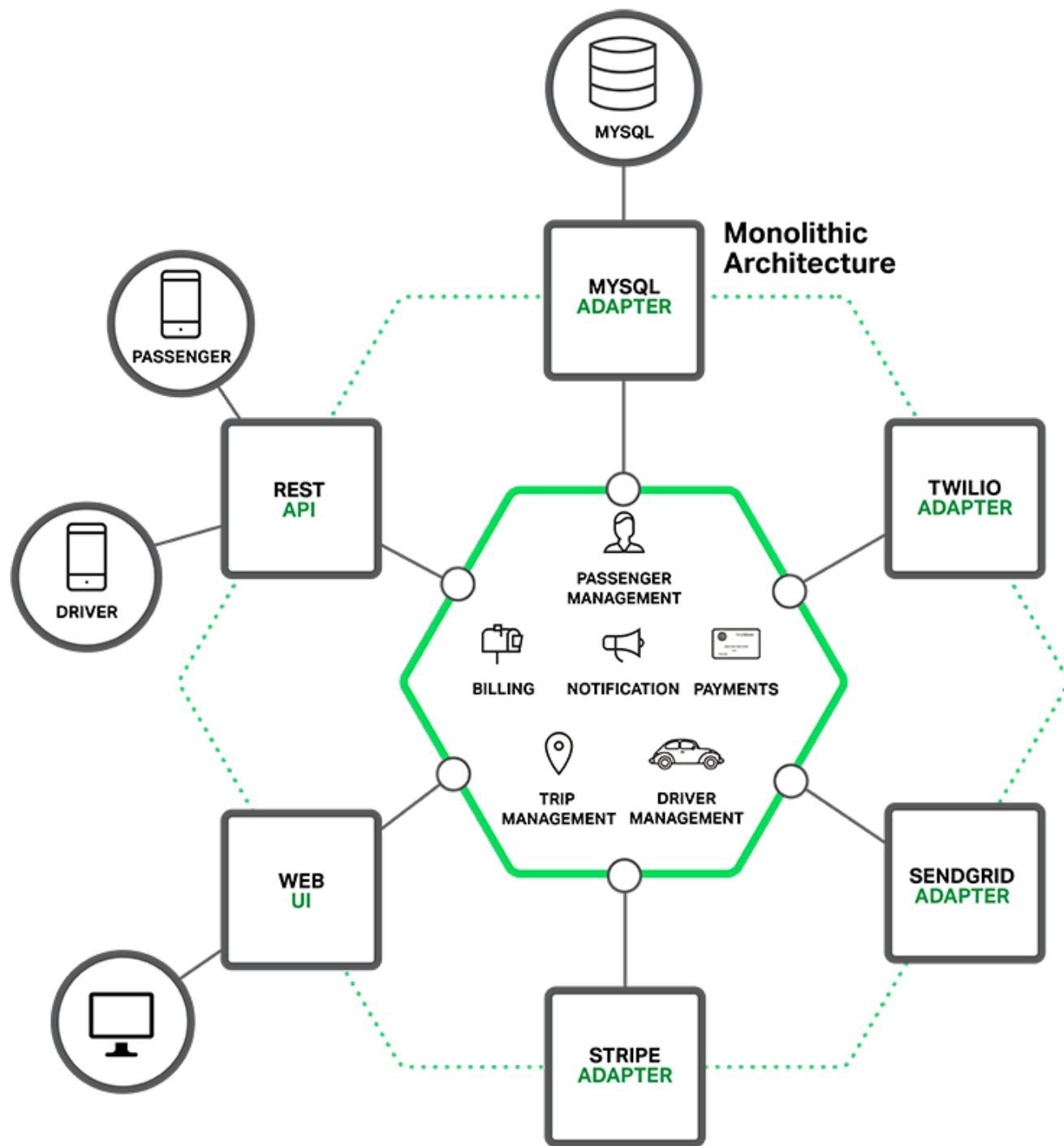
“Microservices allow engineering teams to move quickly to grow a product... assuming they don’t get bogged down by the complexity of operating a distributed system”

**Микросервисы не
решают проблем**

**Микросервисы создают
новый уровень проблем**

Так ли это?

Монолитная архитектура



Хрупкость

Если падает, то падает всё

Хрупкость

Если падает, то падает всё

С учётом высокой связанности, падает
всё часто

Низкая скорость поставки

Тяжелые процессы сборки и запуск

Необходимость выкатывать всё даже
при небольших изменениях

Ментальная сложность изменений

- Slow IDE
- Ментальная нагрузка
- Высокая ответственность каждого
- Что я не должен делать?
- Высокие риски провала

Ментальная сложность изменений



**Типичный монолит -
человечное тело**

Организационная сложность

Необходимость согласований и
координации

Затягивание сроков

Дорогой технологический стэк

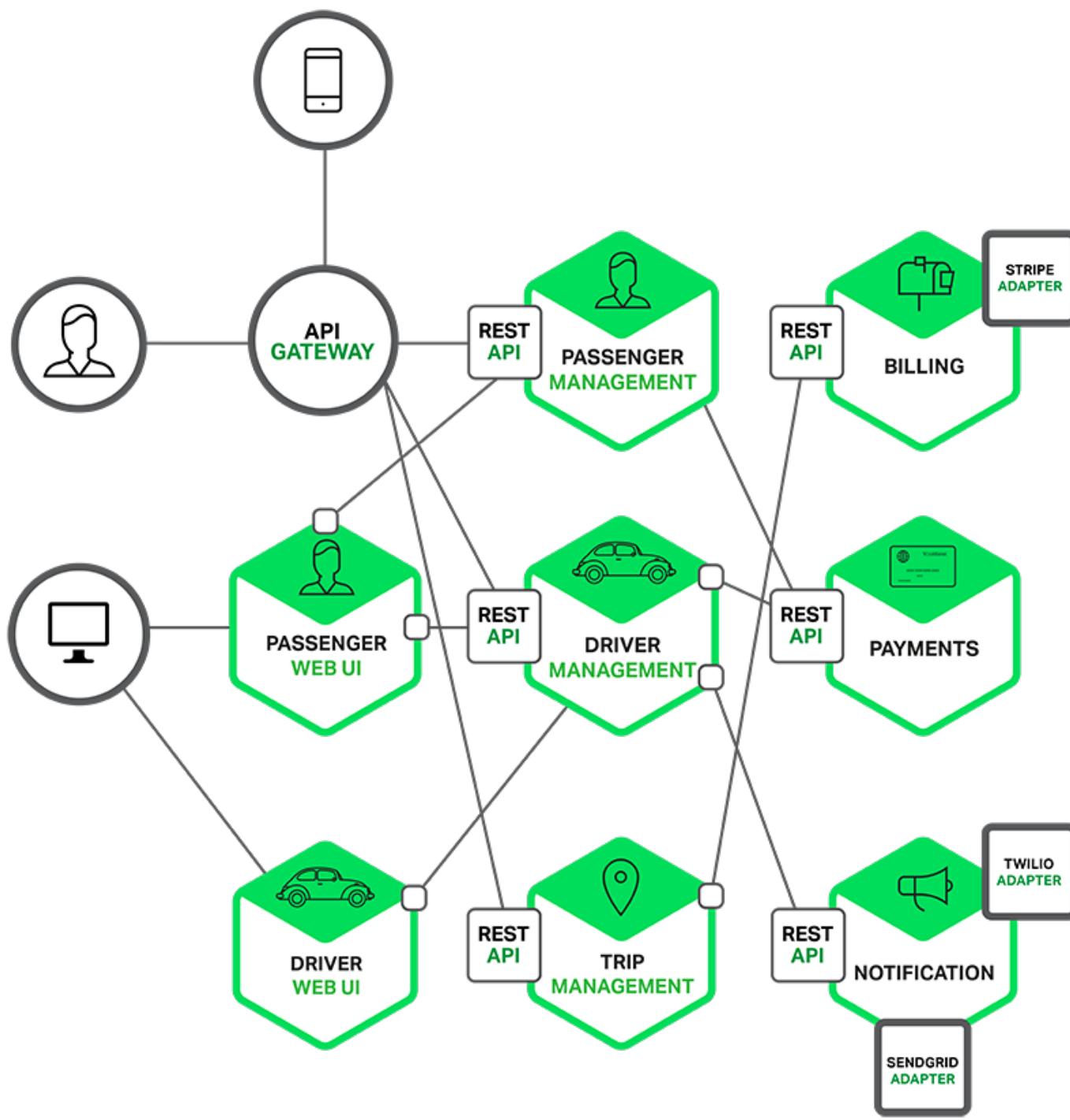
- Мы не можем его менять
- Необходимо брать с запасом, а это all-in-one и дорого
- Небольшой выбор средств
- Vendor lock-in

**Медленно. Дорого.
Высокие риски.**

Монолитная архитектура

- Хрупкость
- Низкая скорость доставки
- Ментальная сложность изменений
- Организационная сложность
- Дорогой технологический стук
- Медленно. Дорого. Высокие риски.

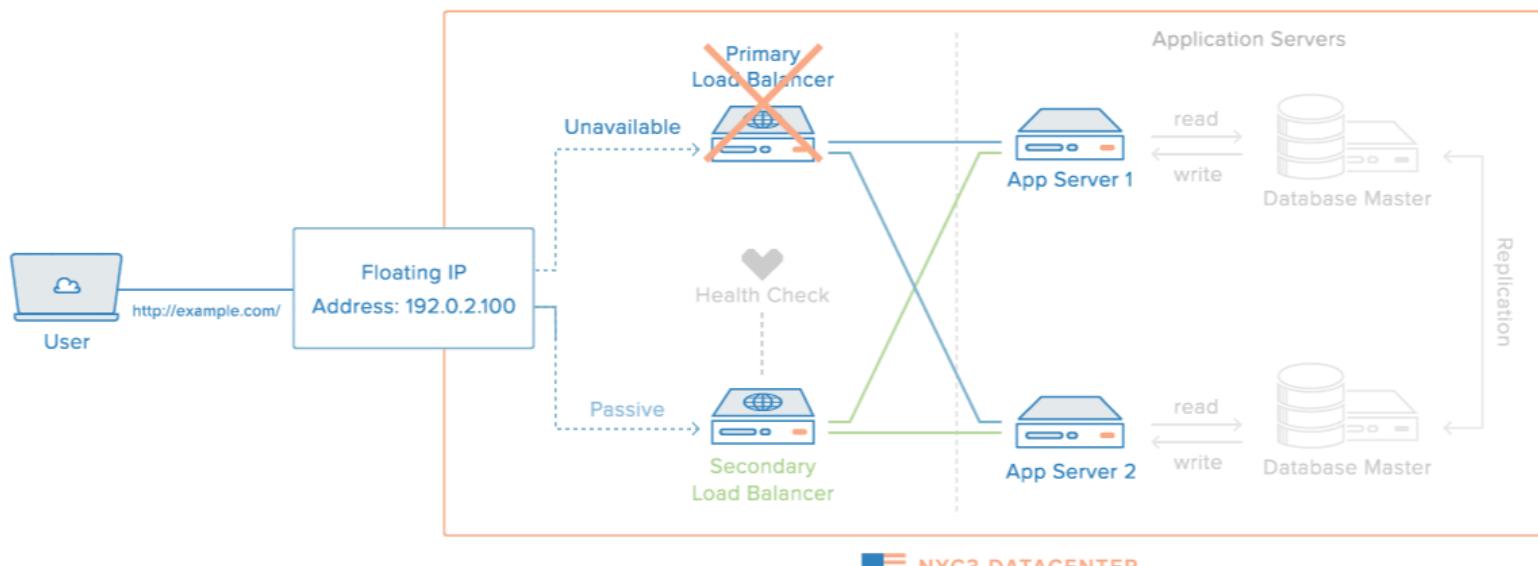
Микросервисная архитектура



Устойчивость

Если падает один сервис, система продолжает работать

Fail-over. Сервисы взаимозаменяемы.
Переключение.



- 1 Active/Passive Cluster is healthy
- 2 Primary node fails
- 3 Floating IP is assigned to Secondary node

NYC3 DATACENTER

Высокая скорость поставки

Быстрая сборка и запуск

Разработка и независимые
выкатка сервисов

Прогрессивное обновление UI

Ментальная легкость изменений

- Изолированность сервиса
- Низкие риски провала
- Точечные изменения
- Fast IDE
- Rollback

Свободный технологический стэк

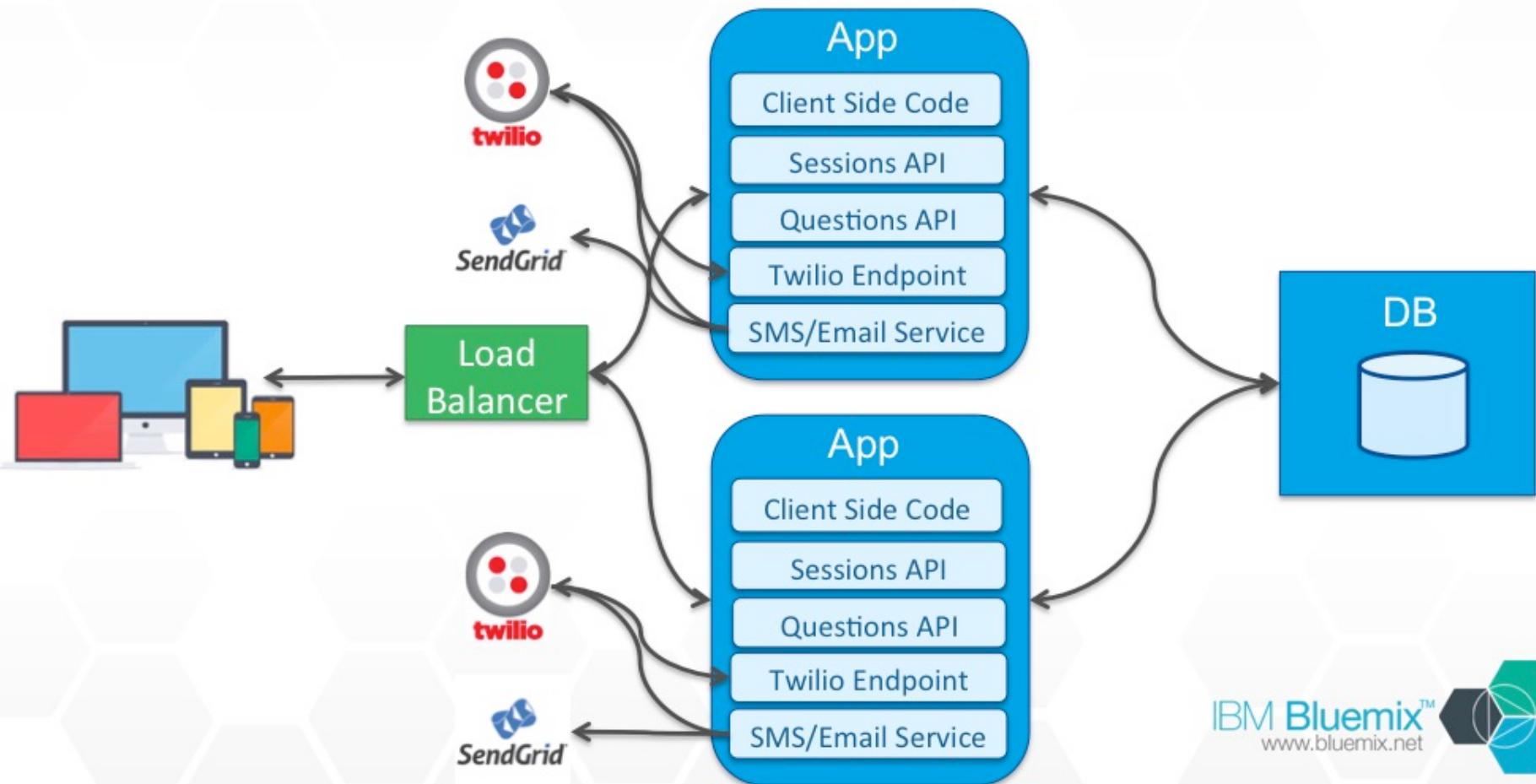
- Agile Way
- Оптимальный для каждого сервиса инструмент
- Подходящий под экспертизу команды и каждого разработчика
- Open-Source Software
- SaaS

**Быстро. Надёжно.
Эффективно.**

Микросервисная архитектура

- Устойчивость
- Высокая скорость поставки
- Ментальная легкость изменений
- Свободный технологический стэк
- Быстро. Надёжно. Эффективно.

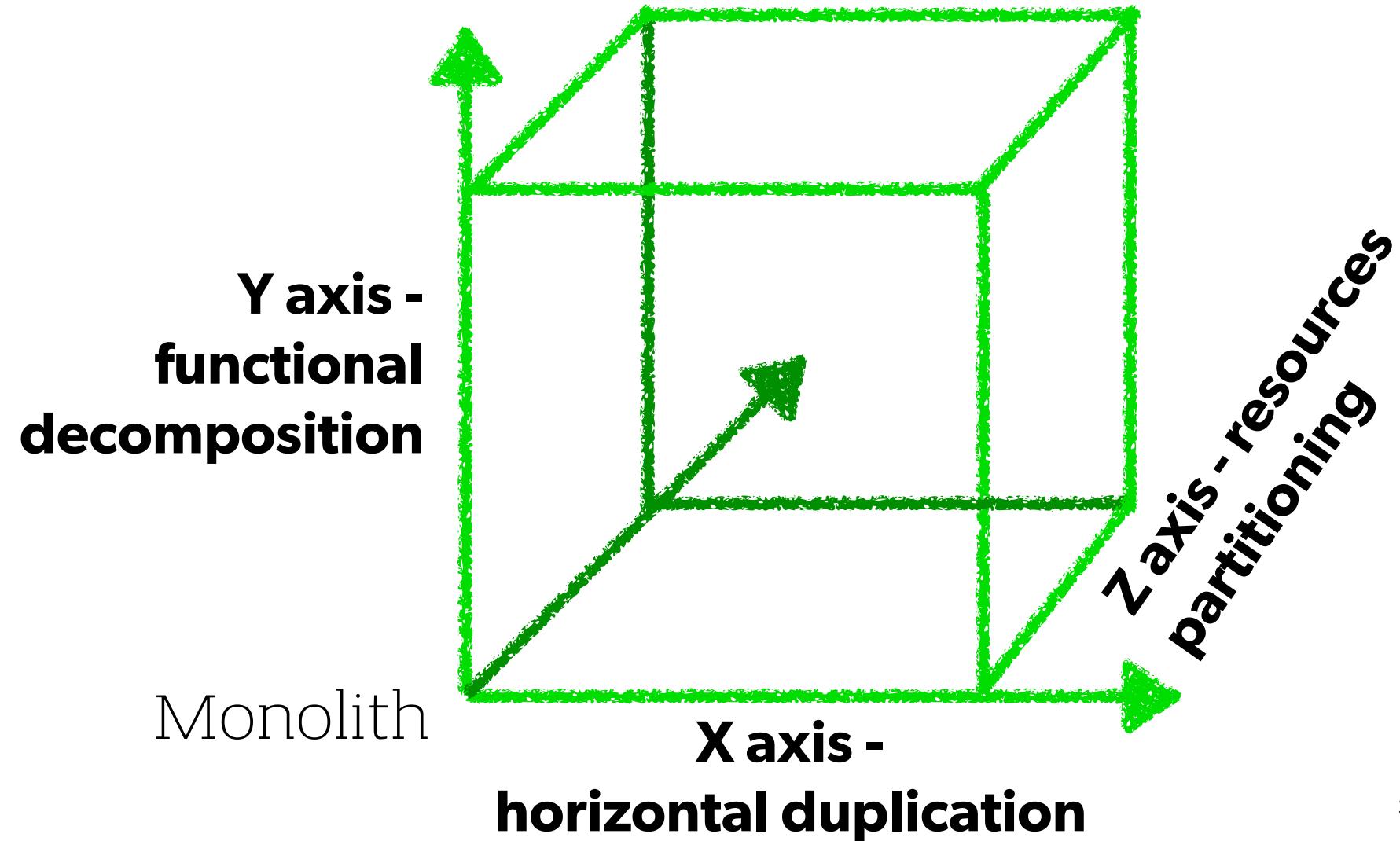
Scaled Monolith



The Art Of Scalability

Scale Cube

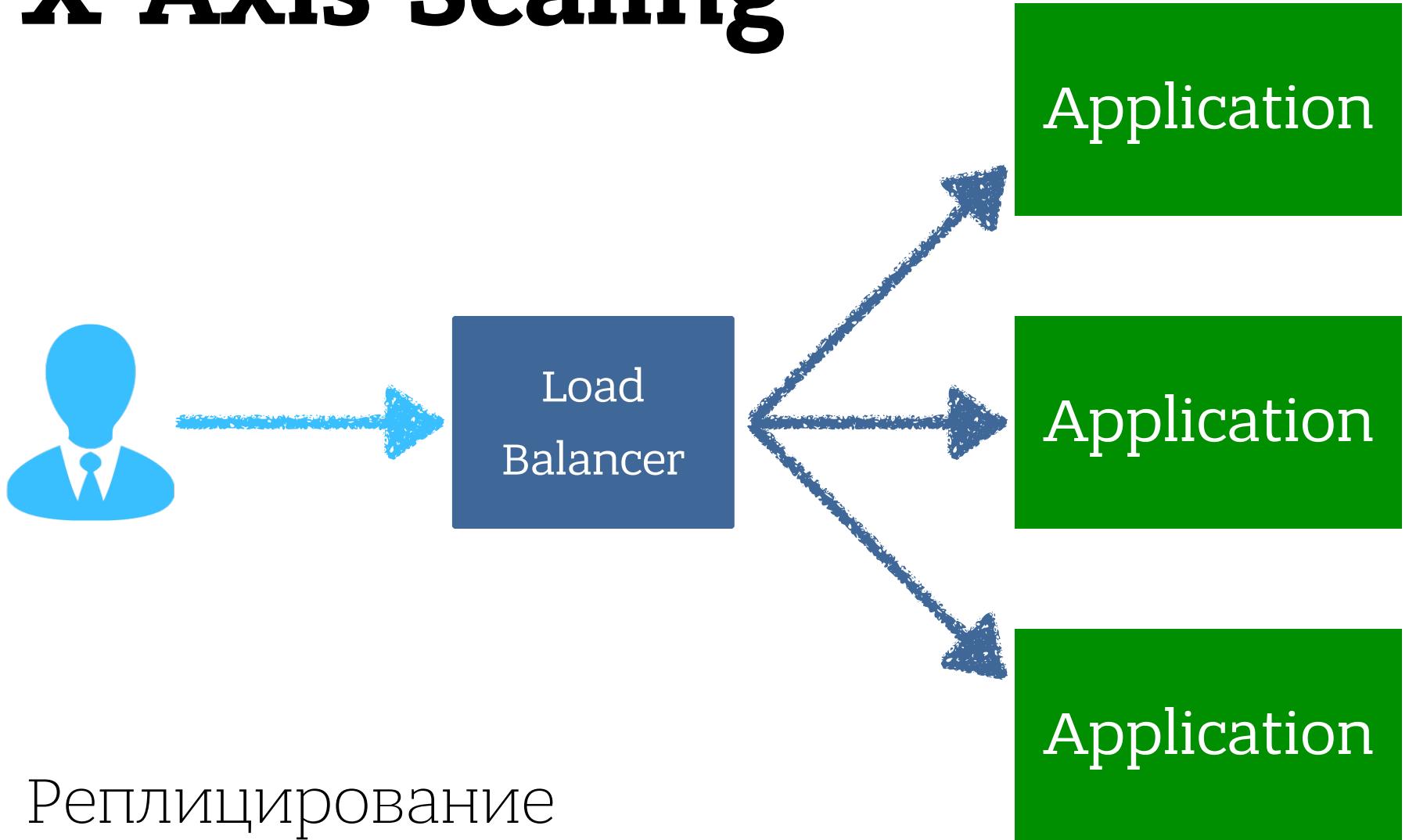
Infinite Scaling



The Scale Cube

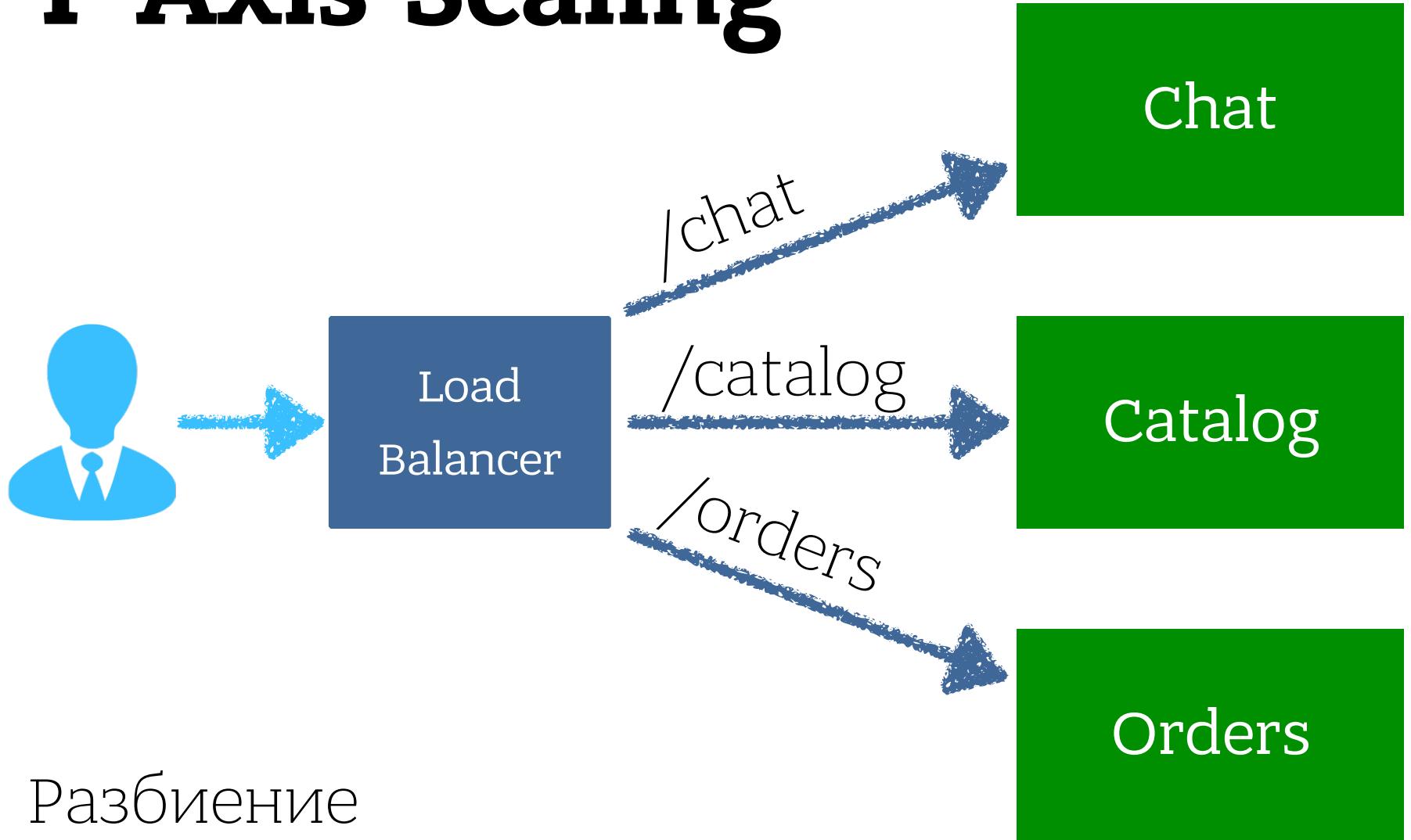
- X - добавление новых копий приложения
- Y - разделение приложение на сервисы
- Z - разделение данных или ресурсов (например, для платных пользователей)

X-Axis Scaling



Реплицирование
приложения

Y-Axis Scaling

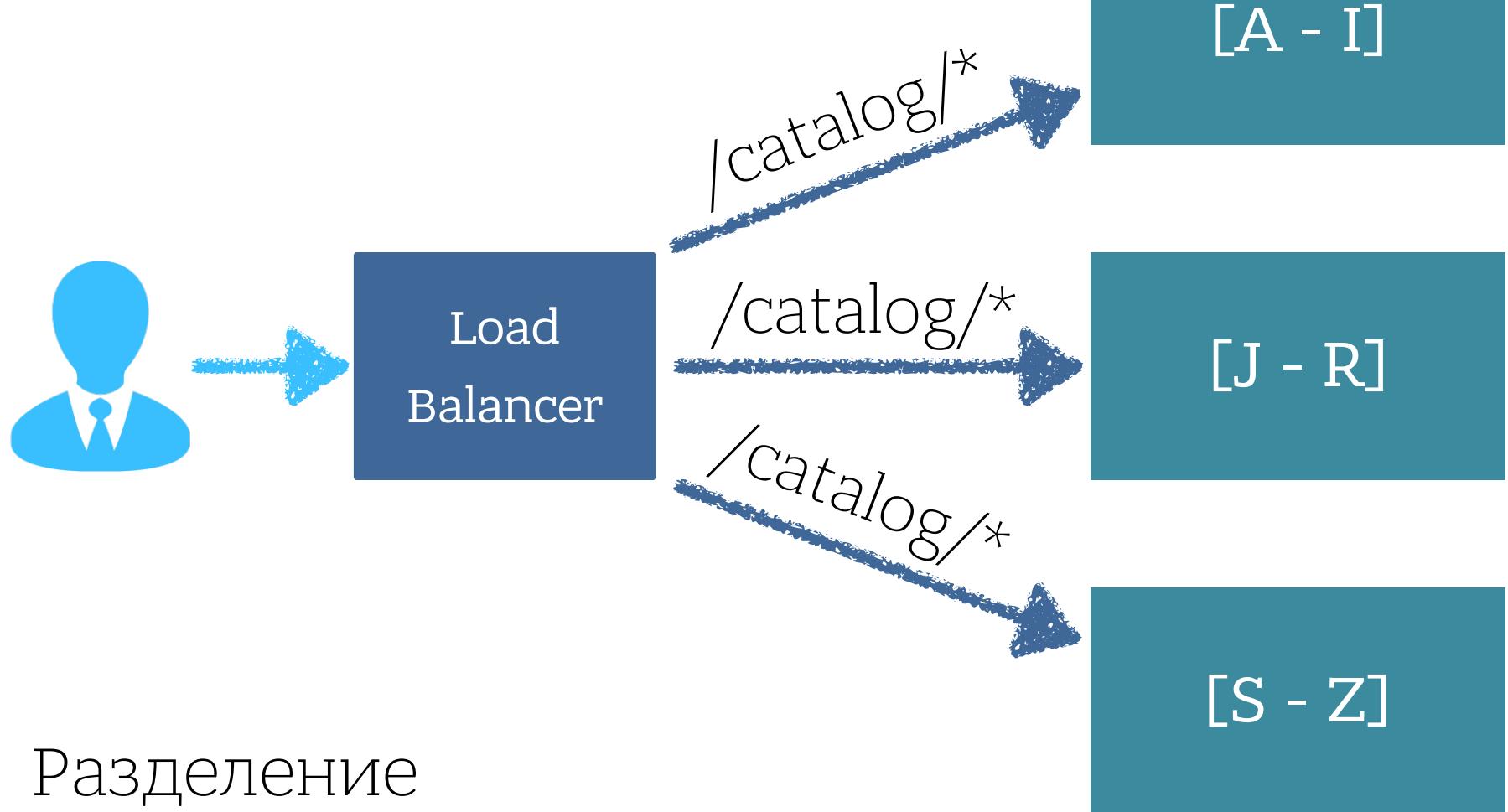


Разбиение
приложения

Y-Axis Scaling

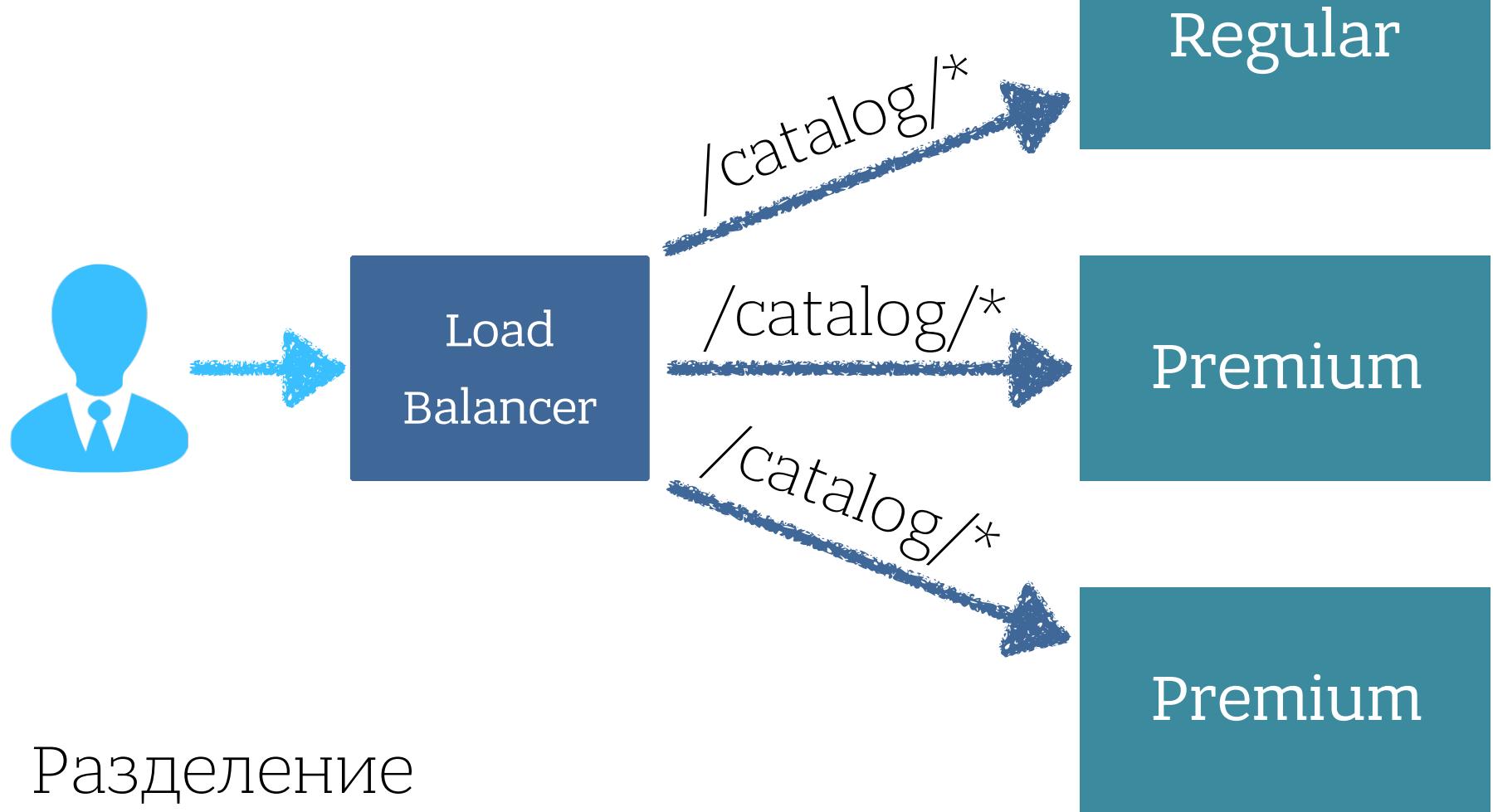
- **За какую функцию отвечает?** Что делает? - управление формами, отправка e-mail, мониторинг
- **С какими данными работает?** Кем используется?
 - пользовательский UI, менеджеры, отдел поддержки
- Комбинированный способ
- Single Responsibility Principle

Z-Axis Scaling



Разделение
данных

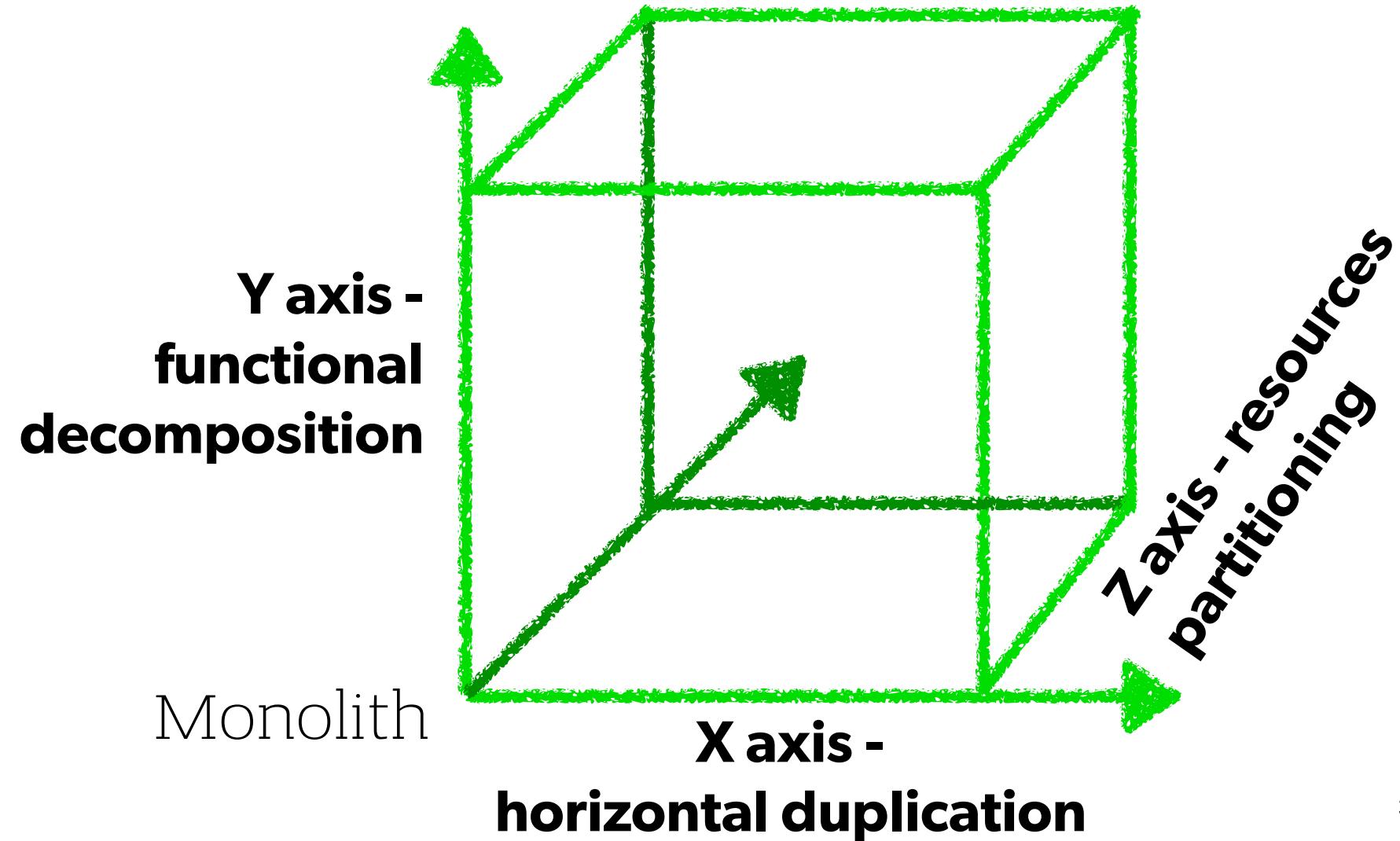
Z-Axis Scaling



Разделение
ресурсов

Scale Cube

Infinite Scaling

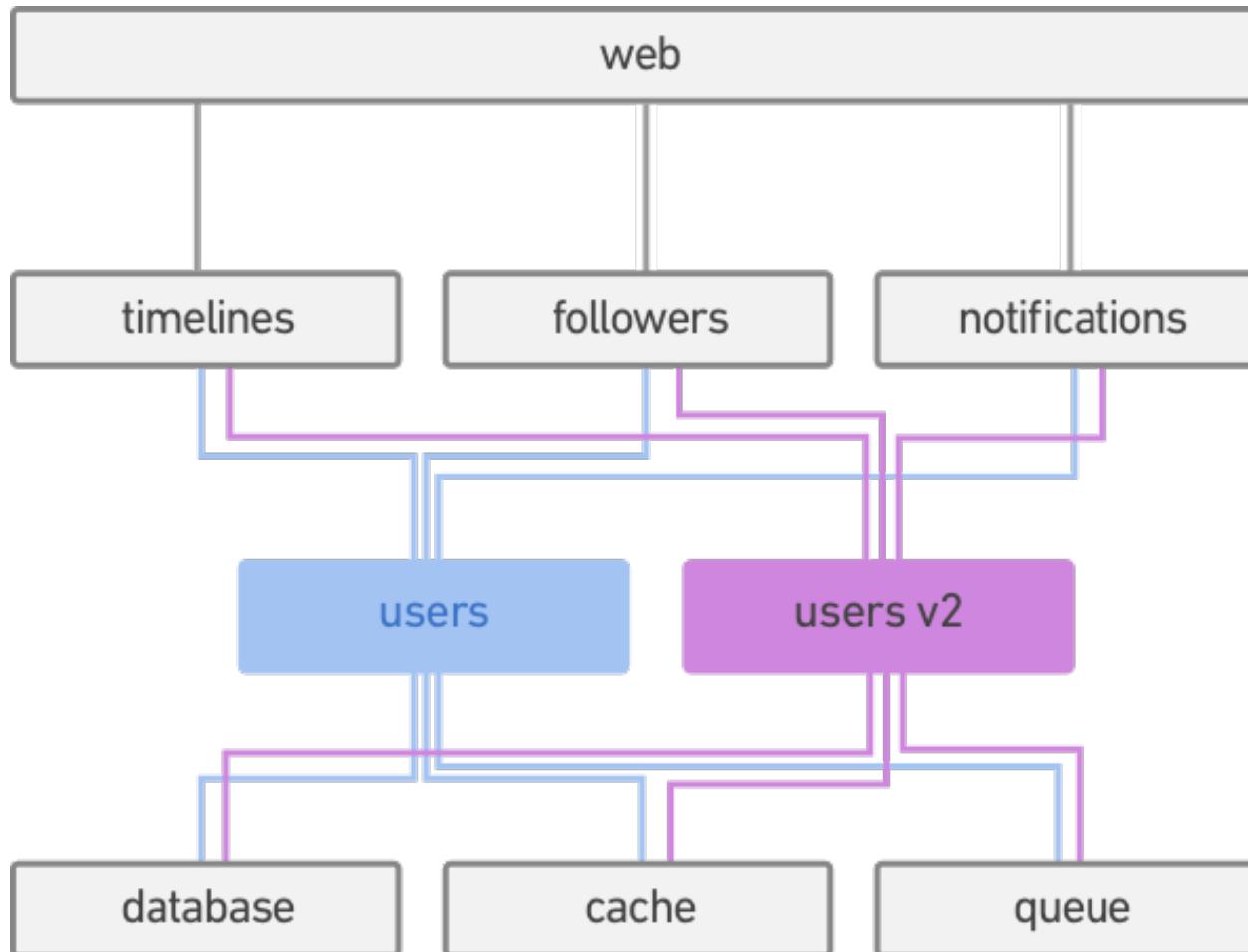


- ОК, Google. Как с этим
двигаться?

Процесс роста

- Новая версия - новый сервис
-

Новая версия - новый сервис



Blue-Green Deployment

Процесс роста

- Новая версия - новый сервис
- Effortless data model versioning for Javascript and Node.js

[https://github.com/TechnologyAdvice/
Vers](https://github.com/TechnologyAdvice/Vers)

Vers

build passing

code climate 4.0

coverage 100%

then

Effortless data model versioning for Javascript and Node.js

Why do I need model versioning?

- Support versioned REST APIs without cluttering your API code with an endpoint for every individual version
- Only code against the latest version of your data -- no messy if-statements throughout your codebase to check for earlier versions or properties in different places
- Update your backends and your frontends on their own schedules
- Roll out new SQL schemas and update your code for the new changes independently, without coordinating a precise rollout, and with no downtime
- For the love of all things holy *stop trying to update every record in your noSQL store every time you update your data model*. Every time you pull a new record from your database, just call `toLatest` on it. Done. If you save it, it saves as the new version. If you don't, it stays the old version and saves you the bandwidth/request time. Your code only ever deals with the latest version and you never scan through and update your entire database ever again. Doesn't that feel better?

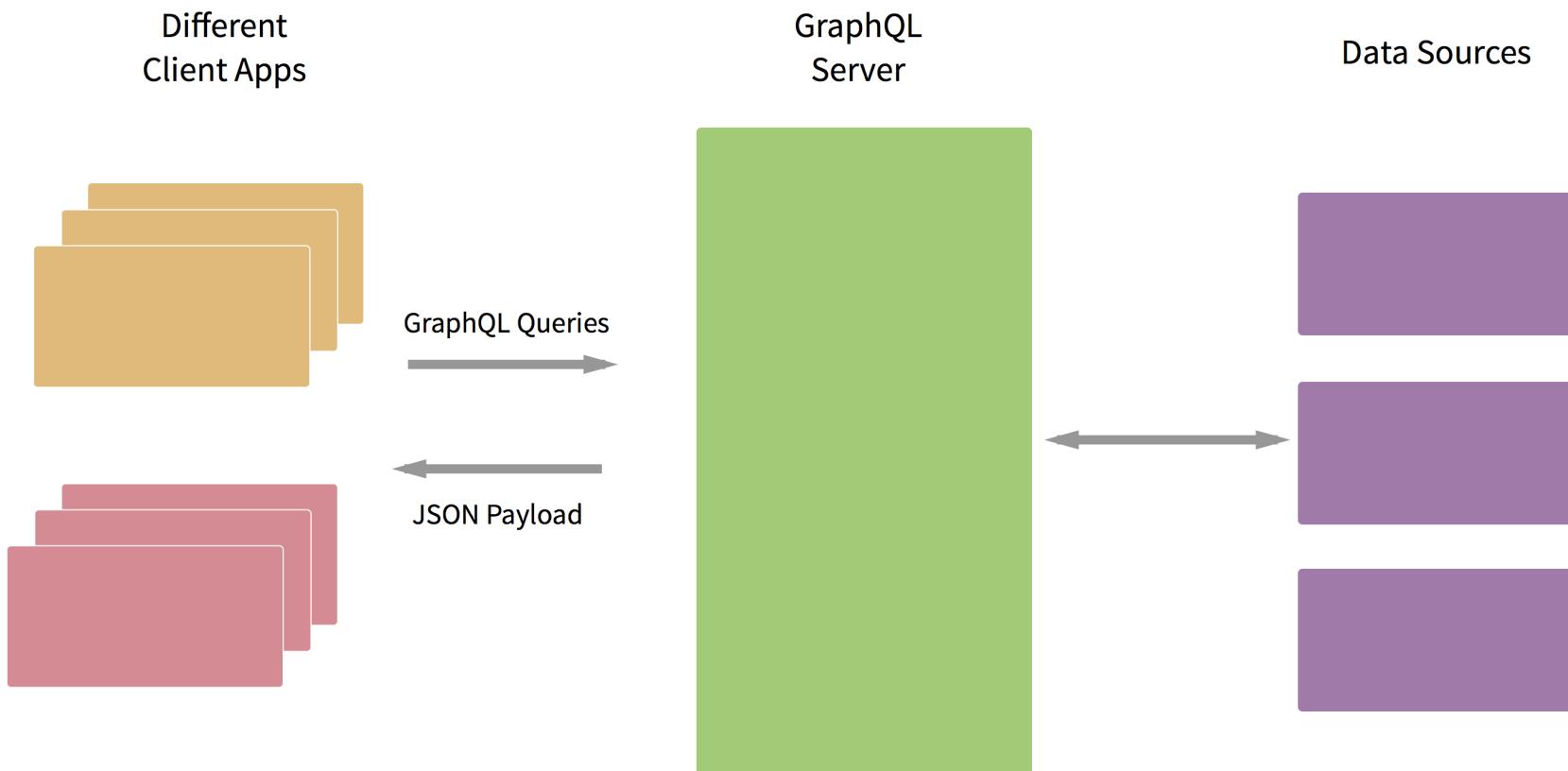
Quick start

```
var vers = require('vers')();
var user = {
  version: 1,
  firstName: 'Doug',
  lastName: 'Funnie'
};
```

Процесс роста

- Новая версия - новый сервис
- Effortless data model versioning for Javascript and Node.js
[https://github.com/TechnologyAdvice/
Vers](https://github.com/TechnologyAdvice/Vers)
- GraphQL

GraphQL



Distributed SQL Query Engine for Big Data

[GET STARTED](#)

```
$ presto
presto:default> describe nation;
  Column | Type | Null | Partition Key
-----+-----+-----+-----
  n_nationkey | bigint | true | false
  n_name | varchar | true | false
  n_regionkey | bigint | true | false
  n_comment | varchar | true | false
(4 rows)

Query 20131105_005529_00080_ee7y3, FINISHED, 2 nodes
Splits: 2 total, 2 done (100.00%)
0:00 [8 rows, 446B] [23 rows/s, 1.29KB/s]

presto:default> select n_nationkey, n_name from nat
```

[\\$://showterm](#)  slow fast stop ? []

› WHAT IS PRESTO?

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

Presto was designed and written from the ground up for interactive analytics and approaches the speed of commercial data warehouses while scaling to the size of organizations like Facebook.

› WHAT CAN IT DO?

Presto allows querying data where it lives, including Hive, Cassandra, relational

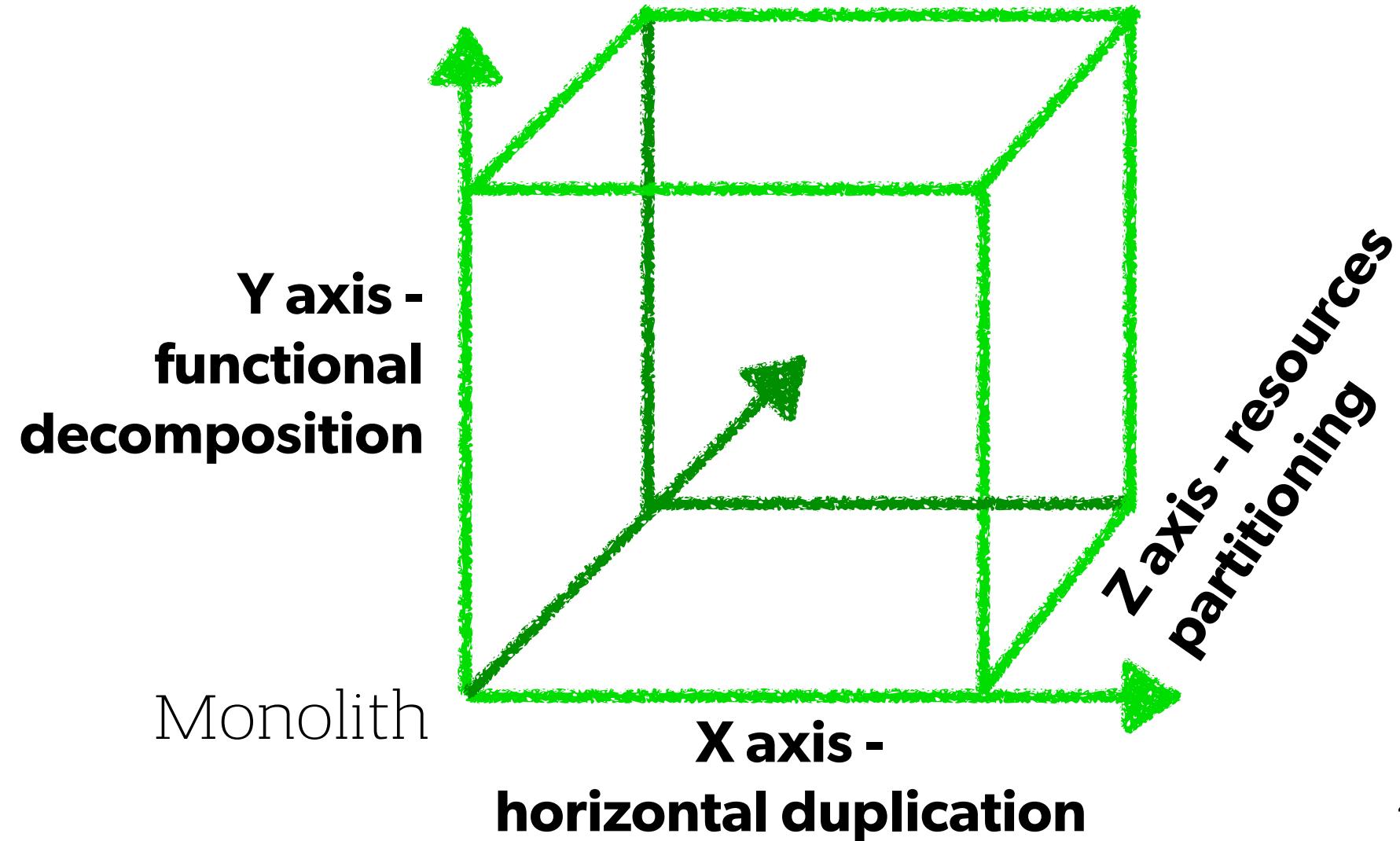
Quick links

- [Overview](#)
- [Documentation](#)
- [FAQ](#)
- [Community](#)
- [Resources](#)
- [Source](#)

Community

Scale Cube

Infinite Scaling



- С чего начать?

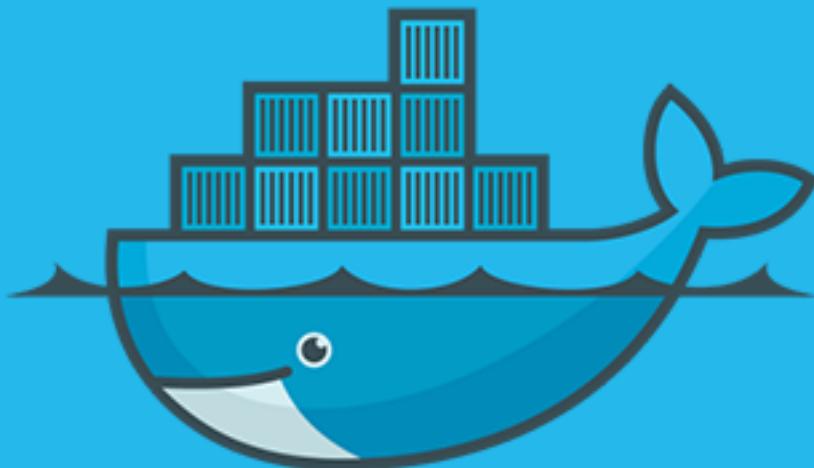


Разбиваем монолит

1. Сборка и поставка сервисов
2. Взаимодействие сервисов
3. Масштабирование

Сборка и поставка сервисов

Платформа Docker

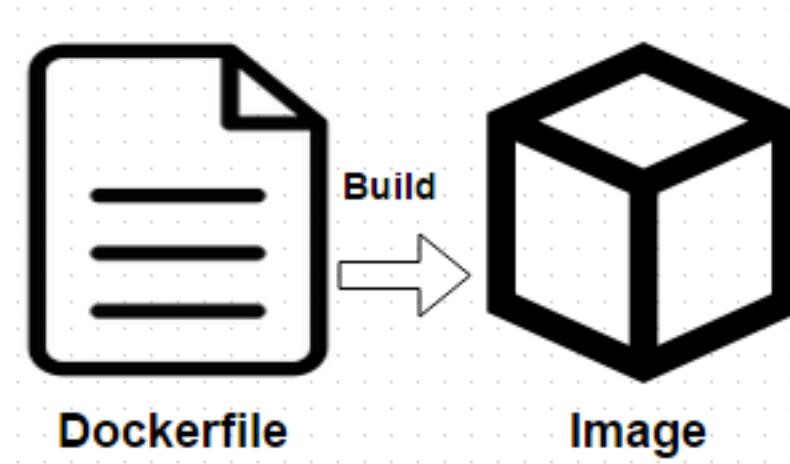


docker



**Docker allows you to package an application
with all of its dependencies into a
standardized unit for software development.**

Build



Приложение +
Зависимости = Image

Run

1. Image via LXC
2. Network
3. Environment

Dockerfile

```
FROM      centos:centos6

# Enable Extra Packages for Enterprise Linux (EPEL) for CentOS
RUN      yum install -y epel-release
# Install Node.js and npm
RUN      yum install -y nodejs npm

# Install app dependencies
COPY package.json /src/package.json
RUN cd /src; npm install --production

# Bundle app source
COPY . /src

EXPOSE 8080
CMD [ "node", "/src/index.js" ]
```

Dockerfile

```
1  FROM node:5
2
3  ENV NODE_ENV=production
4
5  # Create app directory
6  RUN mkdir -p /usr/src/app
7  WORKDIR /usr/src/app
8
9  # Install app dependencies
10 COPY package.json /usr/src/app/
11 RUN npm install
12
13 # Build the bundle
14 COPY . /usr/src/app/
15 RUN npm run build
16
17 EXPOSE 3000
18 CMD [ "npm", "start" ]
```

Dockerfile

```
1 FROM node:5
2
3 ENV NODE_ENV=production ← Input
4
5 # Create app directory
6 RUN mkdir -p /usr/src/app
7 WORKDIR /usr/src/app
8
9 # Install app dependencies
10 COPY package.json /usr/src/app/
11 RUN npm install
12
13 # Build the bundle
14 COPY . /usr/src/app/
15 RUN npm run build
16
17 EXPOSE 3000 ← Output
18 CMD [ "npm", "start" ]
```

Dockerfile

```
1 FROM node:5
2
3 ENV NODE_ENV=production ← Input
4
5 # Create app directory
6 RUN mkdir -p /usr/src/app
7 WORKDIR /usr/src/app
8
9 # Install app dependencies
10 COPY package.json /usr/src/app/
11 RUN npm install
12
13 # Build the bundle
14 COPY . /usr/src/app/
15 RUN npm run build
16
17 EXPOSE 3000 ← Output
18 CMD [ "npm", "start" ]
```

container = f(env)

Output

Docker Image

... in different environments!

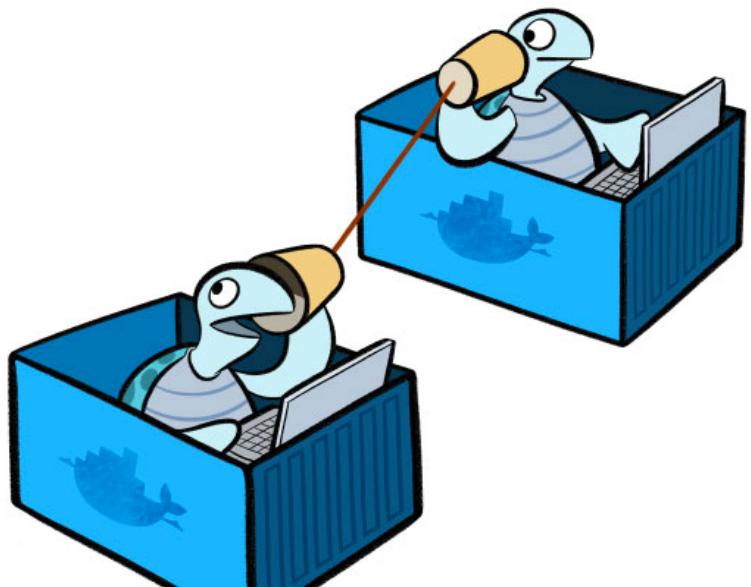


Dockerfile described the Docker image with its dependencies and Dev tested his code in that Docker image



Docker Image

- Запускается в отдельном контейнере
- Иногда - ещё и Volumes
- Взаимодействует с другими контейнерами через сетевой интерфейс

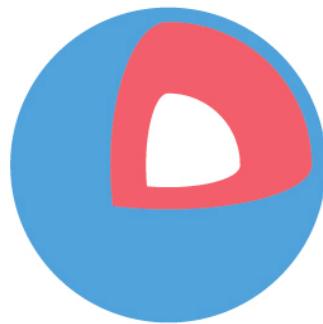


Взаимодействие сервисов

Микросервисное взаимодействие

- RESTful API, JSON API
- OAuth, JWT
- AMQ, WebSocket, JSON RPC, WAMP, etc
- API Based Collaboration
- Database

Масштабирование



Core OS

Open Source Projects for Linux Containers

CoreOS

- Скоро будет 3 года
- Мини-дистрибутив на базе Chrome OS
- Нет пакетного менеджера
- Все приложения - через Docker
- Cloud Config
- Автоматические обновления
- Инфраструктура для кластеров



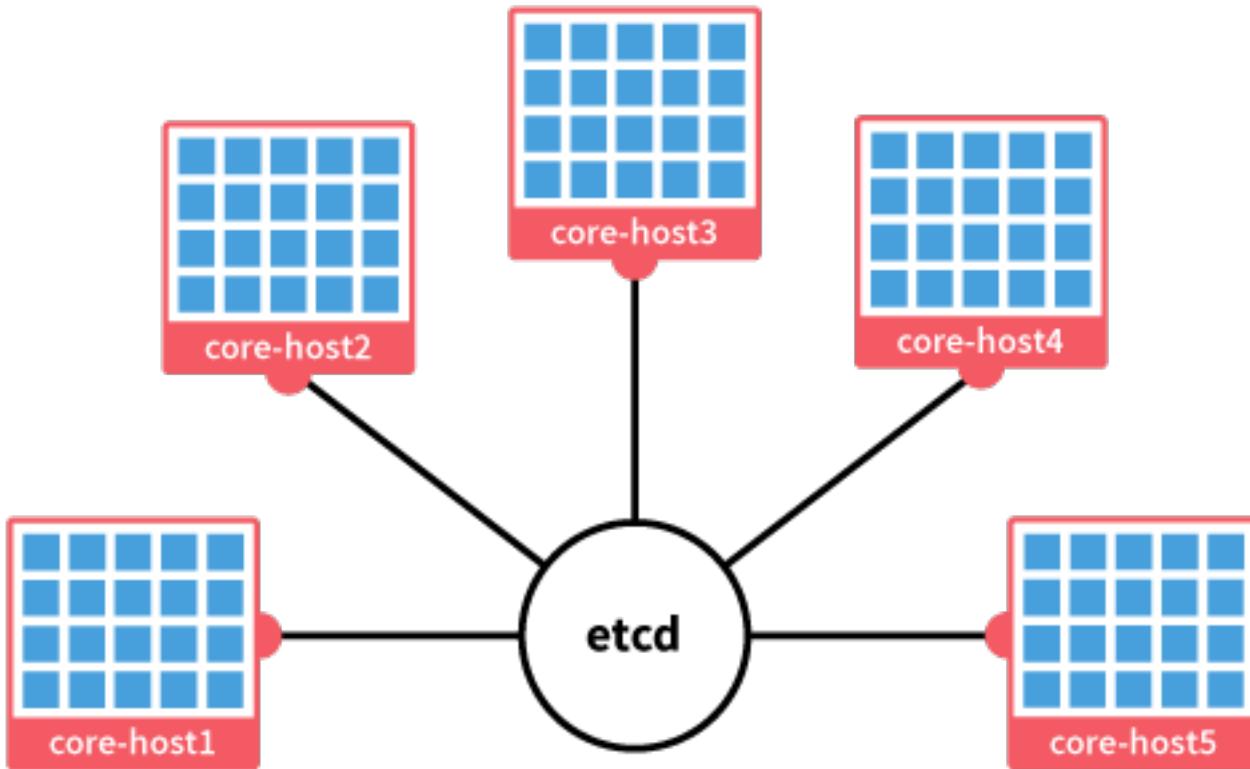
CoreOS

- Etcd
- Fleet
- Flannel

Etcd

- Распределённое key-value хранилище
- Назначение:
 - Общая конфигурация
 - Service Discovery
 - Блокировка ресурсов

Etcdb



Fleet

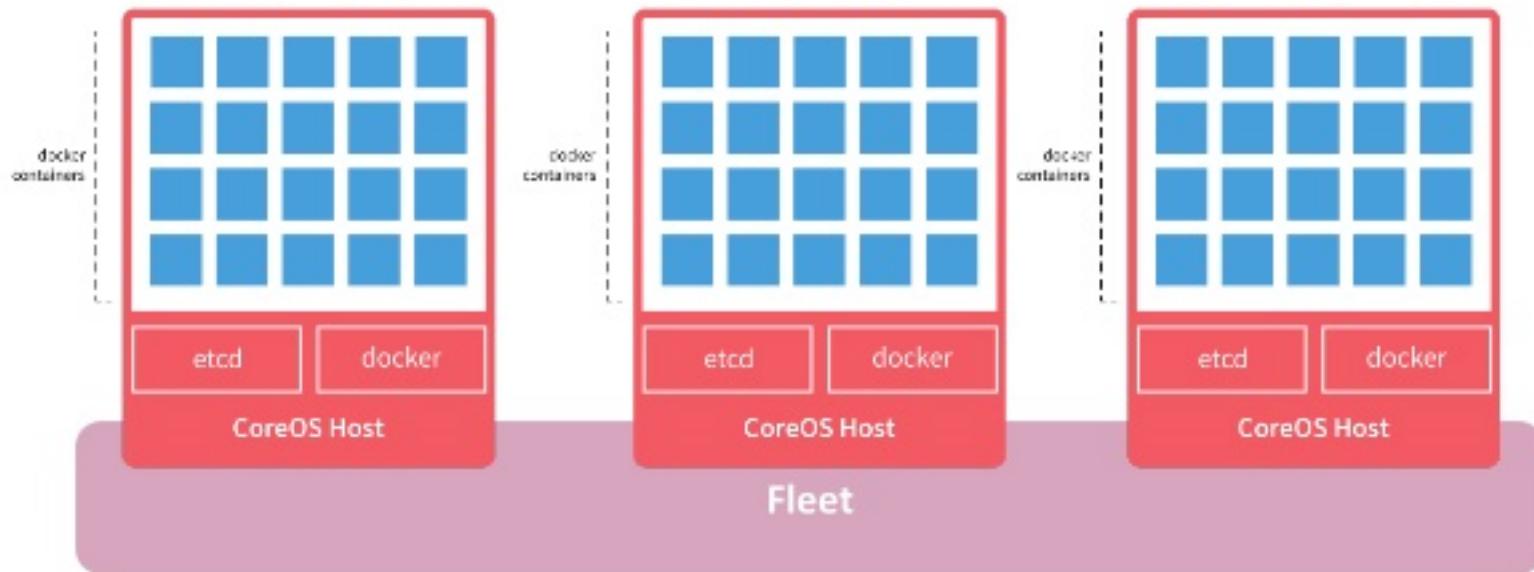
- Распределённый systemd
- Набор Unit-файлов
- Назначение:
 - Запуск Docker-контейнеров
 - Распределение их в кластере
 - Планирование запуска

Fleet - планирование

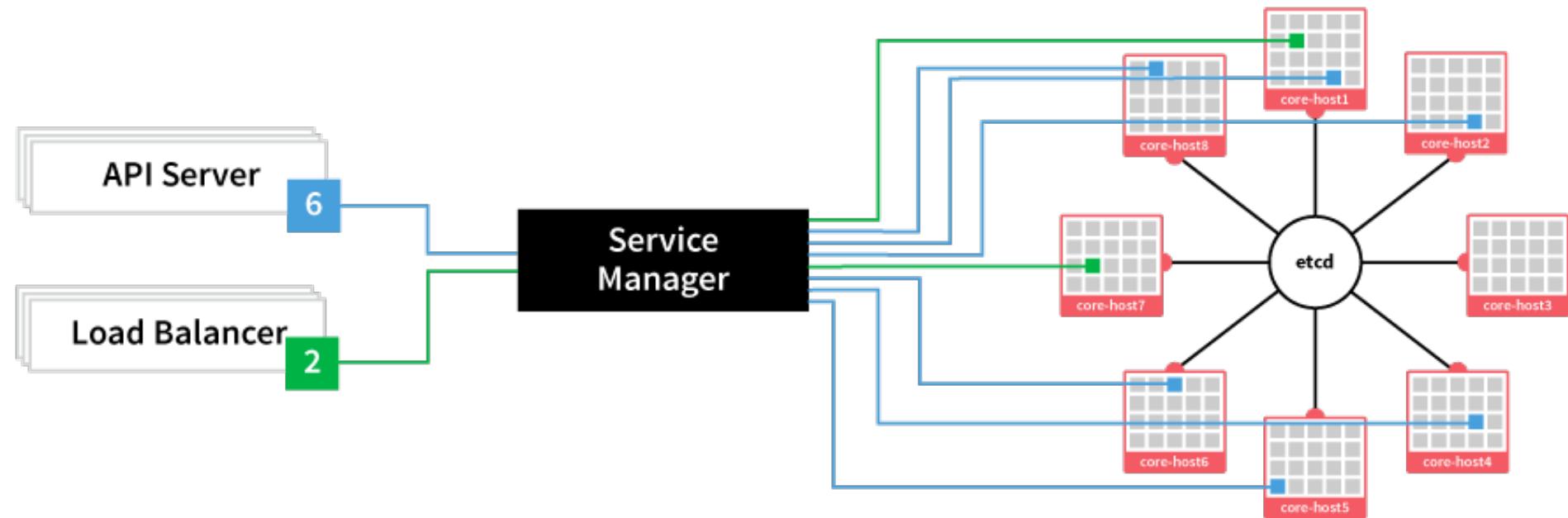
- **Global** (можно запускать в любом месте кластера)
- **MachineMetadata** (только на определённых машинах)
- **Conflicts** (для избежания конфликта с коллоцированием)
- **MachineOf** (запускать только там, где запущены определённые Unit)

```
1 [Unit]
2 Description=startup-makers-lb%i
3 After=docker.service
4 Requires=docker.service
5 After=etcd2.service
6 Requires=etcd2.service
7
8 [Service]
9 TimeoutStartSec=0
10 KillMode=none
11 EnvironmentFile=/etc/environment
12 ExecStartPre=-/usr/bin/docker kill startup-makers-lb%i
13 ExecStartPre=-/usr/bin/docker rm startup-makers-lb%i
14 ExecStartPre=/usr/bin/docker pull startup-makers/nginx-lb
15 ExecStart=/usr/bin/sh -c "/usr/bin/docker run --name startup-
    makers-lb%i --rm -p 443:443 -p 80:80 -e SERVICE_NAME=startup-
    makers -e ETCD=$(ifconfig docker0 | awk '/<inet>/ {
        print $2 }):2379" findx-docker-secured-lb"
16 ExecStop=/usr/bin/docker stop startup-makers-lb%i
17
18 [X-Fleet]
19 Conflicts=startup-makers-lb@*
20 MachineMetadata=loadbalancer=true
```

Fleet



Fleet



**Как Load Balancer узнает про
адрес и порт контейнеров?**

Service Discovery

- Fleet Unit
- MachineOf
- Определяет IP и порт целевого контейнера
- Записывает в etcd
- Через 5 секунд повторяет
- = Sidekick Pattern

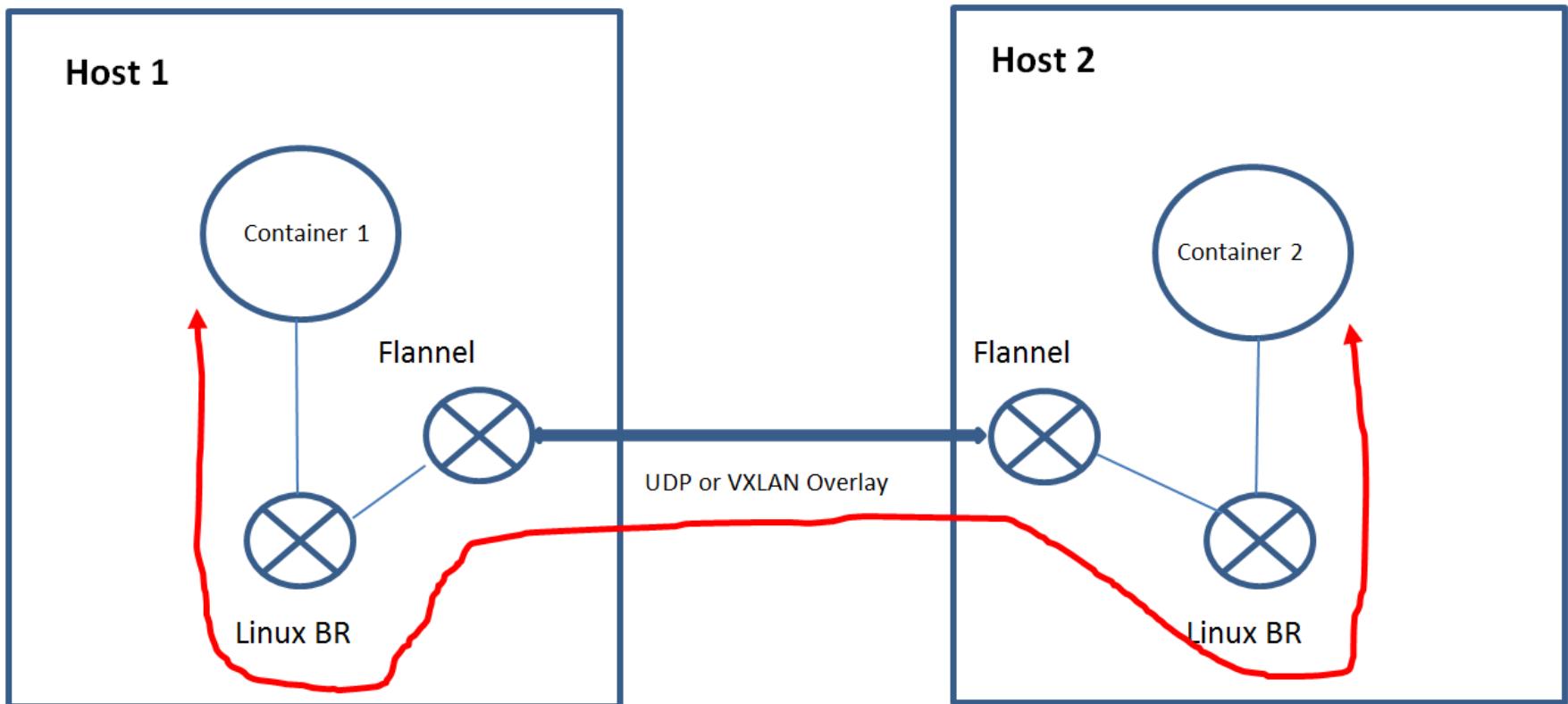
Service Discovery

- Fleet Unit
 - MachineOf
 - Определяет IP и порт целевого контейнера
 - Записывает в etcd
 - Через 5 секунд повторяет
 - = Sidekick Pattern
- 

```
1 [Unit]
2 Description=Announce findx-web%i
3 BindsTo=findx-web@%i.service
4 After=findx-web@%i.service
5 ....
6
7 [Service]
8 ExecStart=/bin/sh -c "while true; do etcdctl set /services/findx-web/upstream/%i \"$(sleep 5 && docker inspect -f '{{.NetworkSettings.IPAddress}}' findx-web%i):3000\" --ttl 60; sleep 45; done"
9 ExecStop=/usr/bin/etcdctl rm /services/findx-web/upstream/%i
10
11 [X-Fleet]
12 MachineOf=findx-web@%i.service
```

Как объединить Docker-контейнеры в одну сеть?

Flannel



CoreOS

- Etcd - распределённое хранилище данных

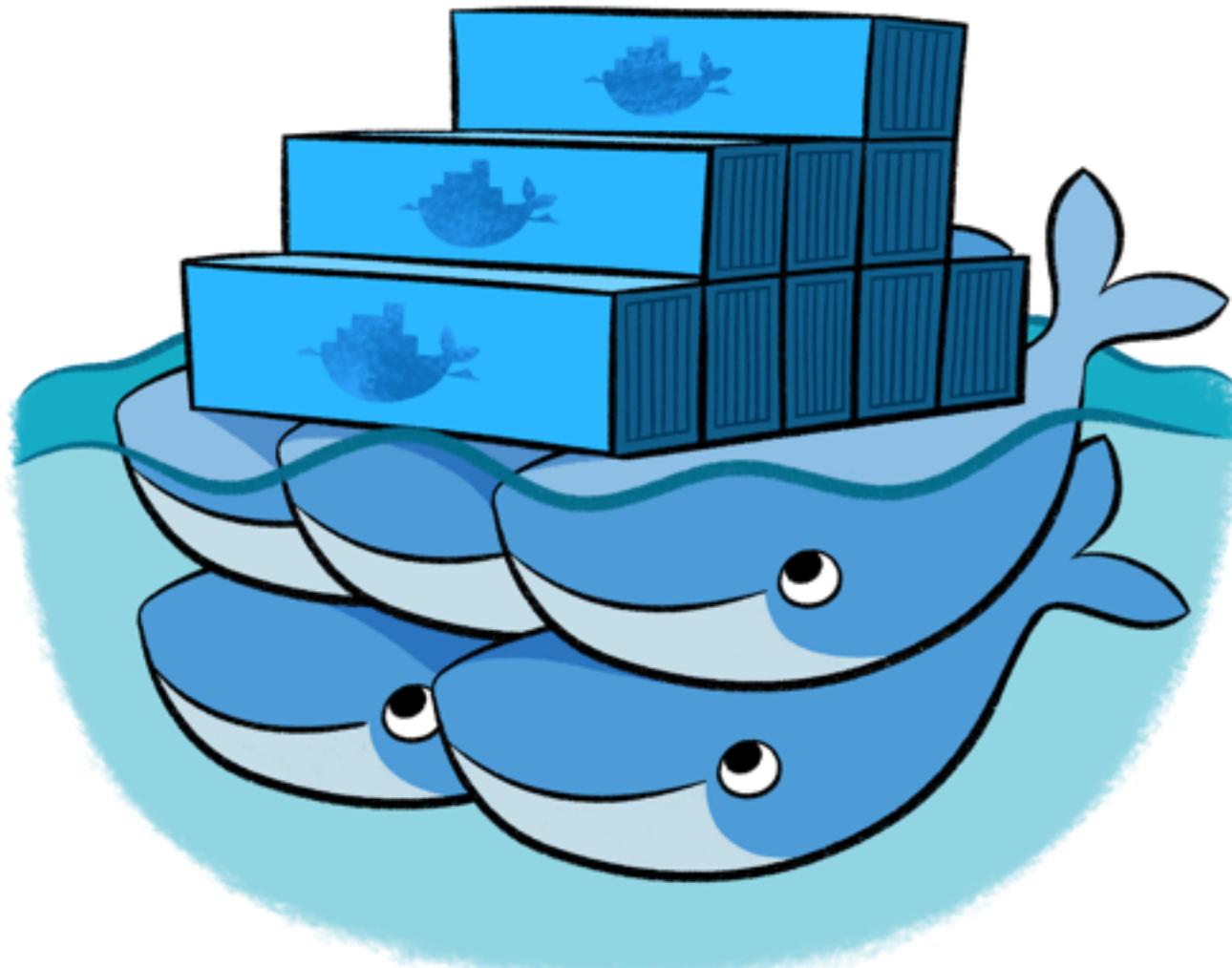
CoreOS

- Etcd - распределённое хранилище данных
- Fleet - распределённый запуск

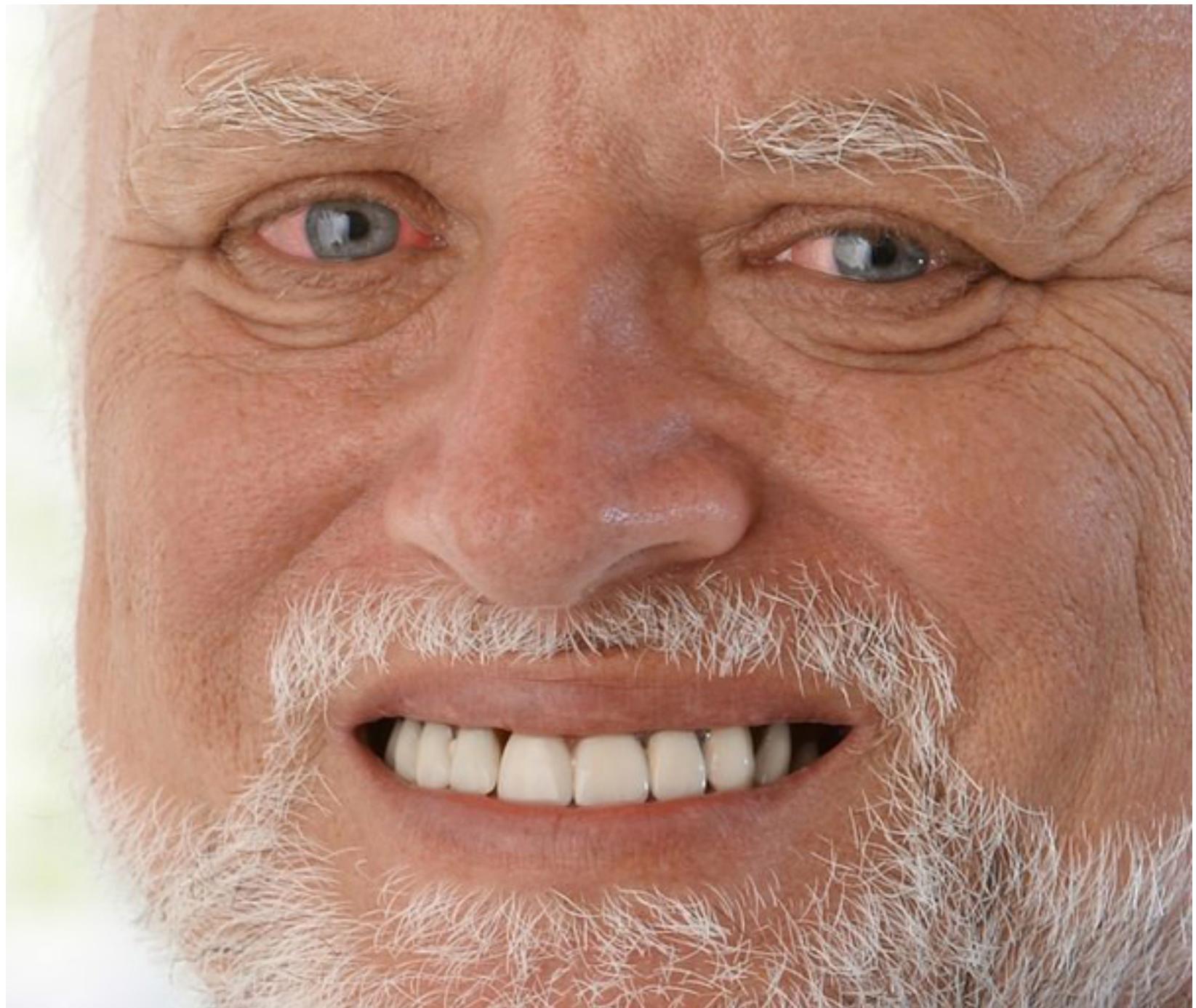
CoreOS

- Etcd - распределённое хранилище данных
- Fleet - распределённый запуск
- Flannel - виртуальная сеть

CoreOS



Real World Example



Реальный проект

- Несколько сервисов (Y)
- Сервис + Service Discovery
- Несколько экземпляров (X)
- Несколько окружений: dev, stage, production

Реальный проект

- 3 Несколько сервисов (Y)
- 2 Сервис + Service Discovery
- 6 Несколько экземпляров (X)
- 3 Несколько окружений: dev, stage, production

Реальный проект

3

x

2

x

6

x

3

= 108 контейнеров



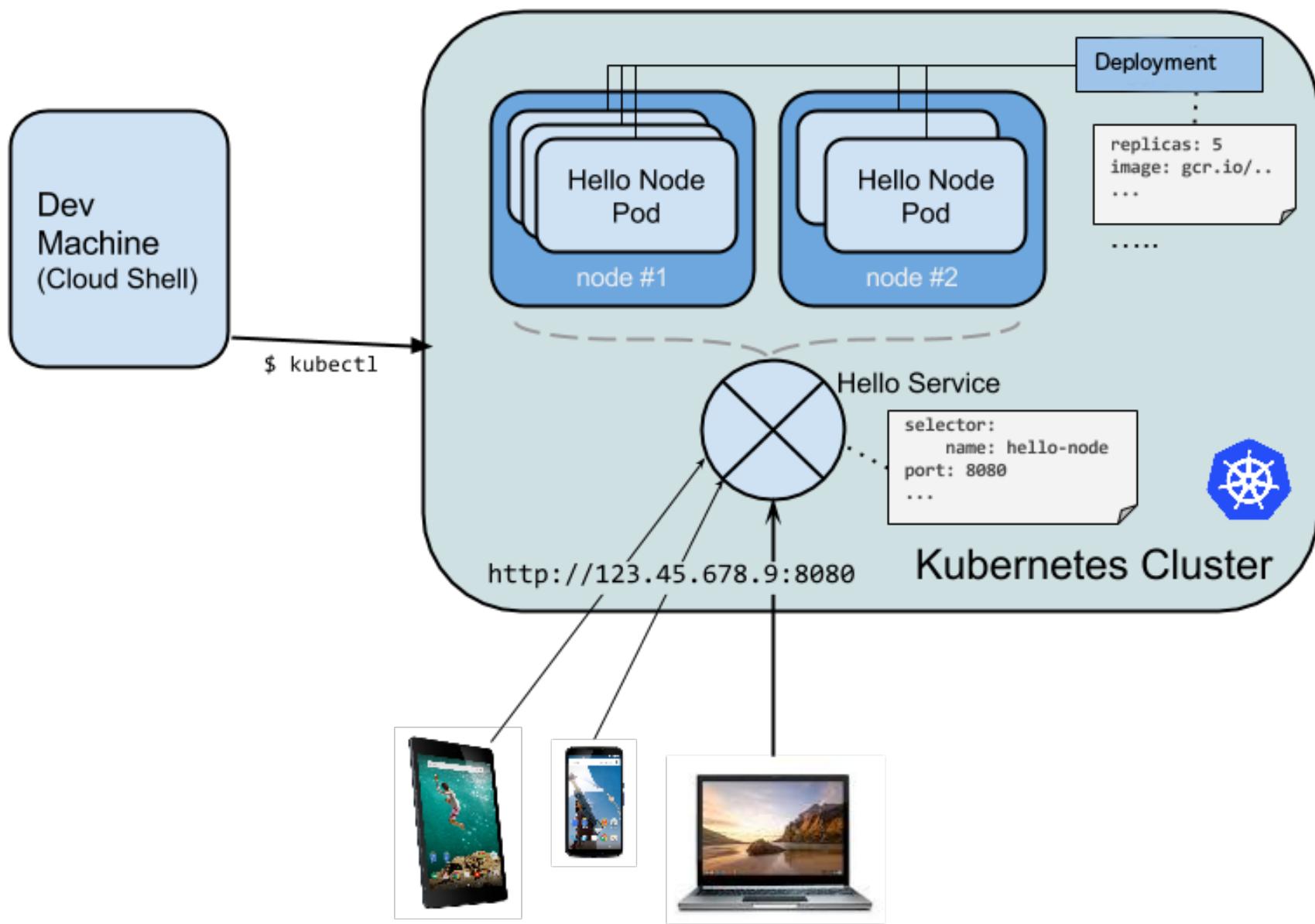


kubernetes

by Google®

Kubernetes

- Следующий уровень абстракции
- **Pods** - группы контейнеров
- **Labels** - тэги для идентификации Pods
- **ReplicationController** - количество копий
- **Deployment** - roll-out, roll-back, BGD
- **cAdvisor** - анализ производительности и потребления ресурсов контейнеров
- UI и прочее - про это всё в следующий раз

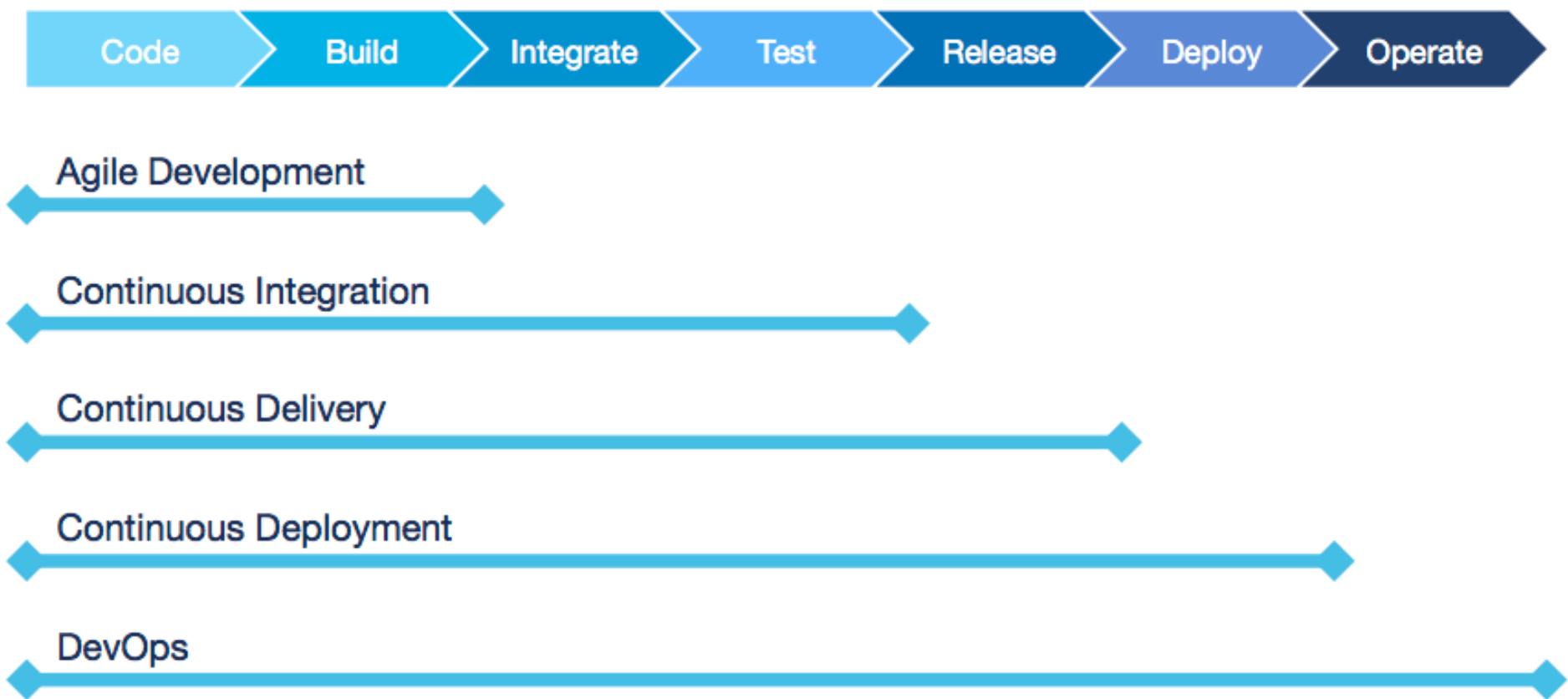


Stay tuned



О чём не рассказал?

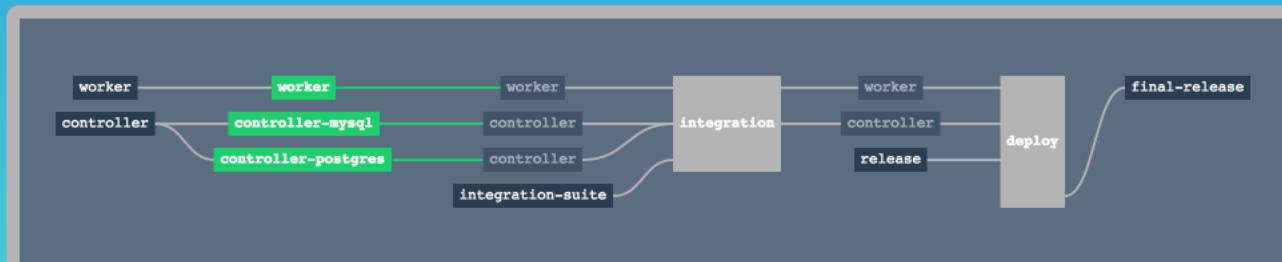
Deployment Automation



Concourse CI

[Documentation](#)[Community](#)[Downloads](#)[GitHub](#)

CI that scales with your project.

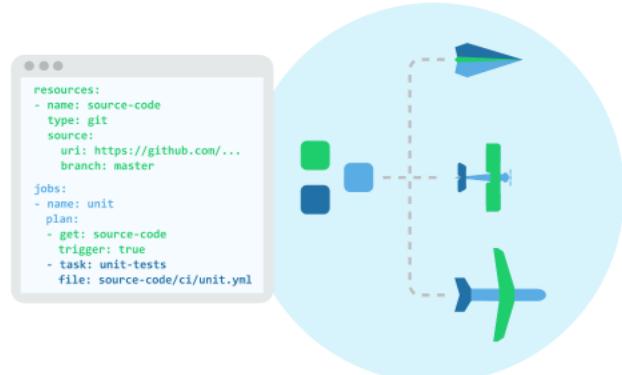


Simple and Scalable

Rather than a myriad of checkboxes, pipelines are defined as a single declarative config file, composing together just [three core concepts](#).

As your project grows, your pipeline will grow with it, and remain understandable.

```
...  
resources:  
- name: source-code  
type: git  
source:  
uri: https://github.com/...  
branch: master  
  
jobs:  
- name: unit  
plan:  
- get: source-code  
trigger: true  
- task: unit-tests  
file: source-code/ci/unit.yml
```



Resource Type

```
1 resource_types:
2   - name: k8s-resource
3     type: docker-image
4     source:
5       repository: aaa.bbb.com:5000/k8s-resource
6       username: {{docker_registry_login}}
7       password: {{docker_registry_password}}
8     ca_certs:
9       - domain: aaa.bbb.com:5000
10      cert: {{docker_registry_cert}}
```

Resource

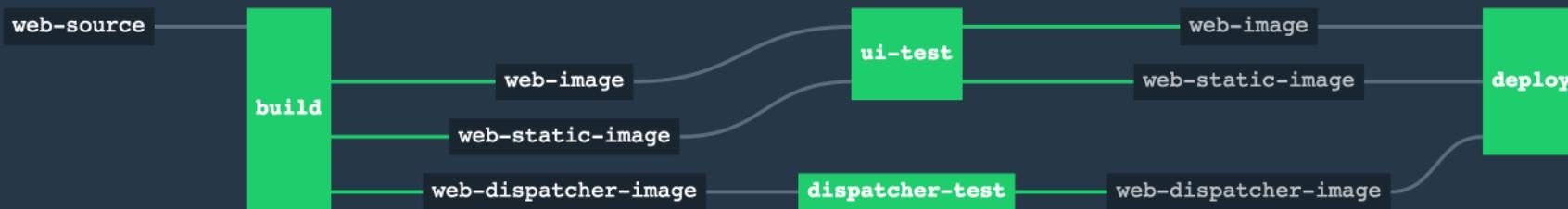
```
12 ~   resources:
13 ~     - name: k8s-bbb-release
14 ~       type: k8s-resource
15 ~       source:
16 ~         url: https://12.34.56.78:443
17 ~         cert_data: {{k8s_cert_data}}
18 ~         key_data: {{k8s_key_data}}
19 ~         ca_data: {{k8s_ca_data}}
20 ~       namespace: release
```

Resource

```
22 - name: web-source
23   type: git
24   source:
25     uri: git@github.com:ccc/bbb-web.git
26     branch: release
27     private_key: {{github_private_key}}
```

Concourse Pipeline

☰ ⌂



www.concourse.ci

Мониторинг производительности и ошибок



Уведомления при недоступности сервисов

- Интеграционные тесты
- Synthetic monitoring
- Отправка E-mail и SMS

Логирование и агрегация логов

Kibana

Discover Visualize Dashboard Settings June 1st 2011, 00:16:26.010 to May 7th 2013, 14:29:14.931

+article
+article

Selected Fields

_source
Fields
Popular fields
agent
bytes
geo.city
geo.coordinates
geo.country_code
@timestamp
_id
_index
_type
creation_time
destination
geo.region
referrer
request
timezone
user

Count June 1st 2011, 00:16:26.010 - May 7th 2013, 14:29:14.931

30k
20k
10k
0

2011-06-30 2011-09-30 2011-12-31 2012-03-31 2012-06-30 2012-09-30 2012-12-31 2013-03-31

@timestamp per week

Time ▾ _source

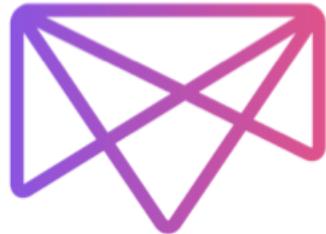
May 7th 2013, 14:29:02.000
referrer: http://www.spk.usace.army.mil/Media/NewsReleases/tabid/1034/Article/13480/corps-releases-inspection-rating-sacramento-river-west-bank-levees.aspx @timestamp: May 7th 2013, 14:29:02.000 creation_time: May 7th 2012, 14:15:25.000 agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0 bytes: 37990 geo.country_code: US geo.city: Fort Huachuca geo.region: AZ geo.coordinates: {"lat":31.5273, "lon":-110.360703} timezone: America/Phoenix request: http://bit.ly/JMGCW destination: http://www.spk.usace.army.mil/M

May 7th 2013, 14:28:50.000
destination: http://www.spk.usace.army.mil/Media/NewsReleases/tabid/1034/Article/13480/corps-releases-inspection-rating-sacramento-river-west-bank-levees.aspx @timestamp: May 7th 2013, 14:28:50.000 creation_time: May 3rd 2013, 12:01:44.000 agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0 bytes: 303 geo.country_code: US geo.city: Fort Huachuca geo.region: AZ geo.coordinates: {"lat":31.5273, "lon":-110.360703} timezone: America/Phoenix request: http://1.usa.gov/181kFnB user: usacesacramento referrer: direct

Table JSON Link to /usagov/log/AUuEF4d3npiyxElindpkv

Timestamp May 7th 2013, 14:28:50.000

DC/OS



MESOSPHERE



DC/OS

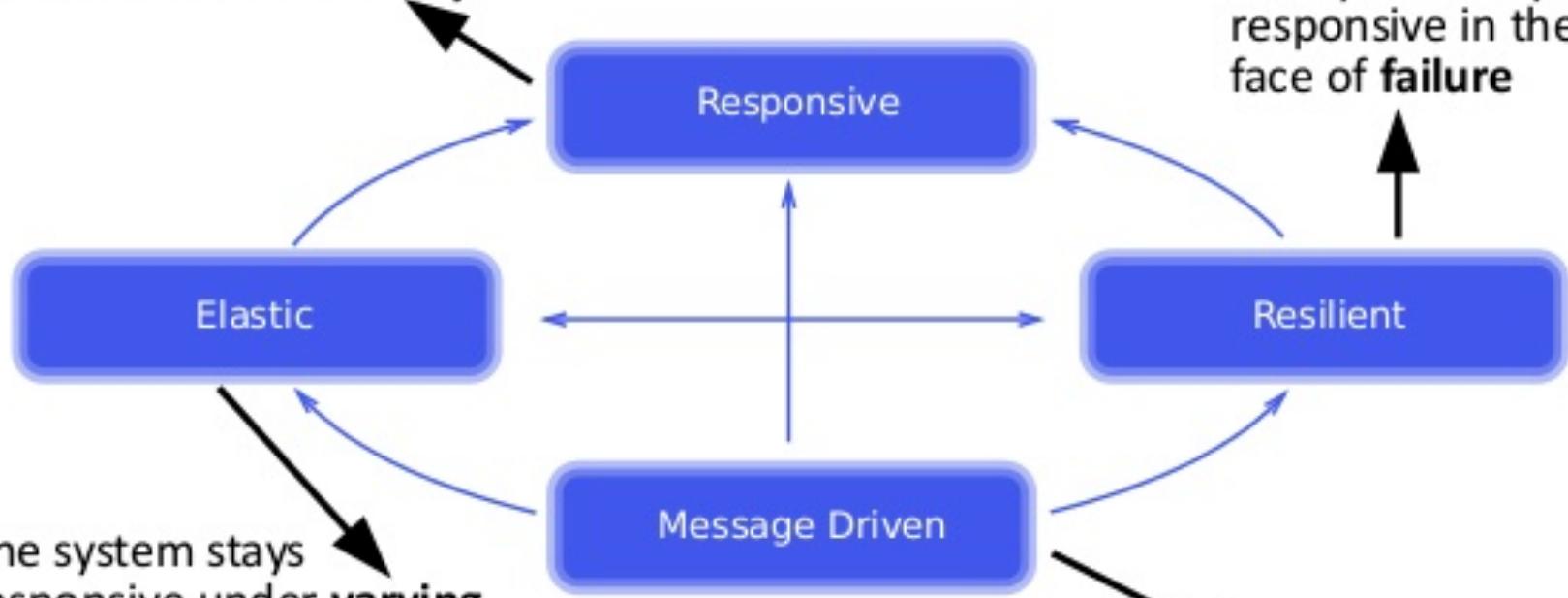
Мы рассмотрели

- Монолитная архитектура
- Микросервисная архитектура
- Модель Scale Cube
- Docker
- CoreOS
- Kubernetes
- Concourse CI

The Reactive Manifesto

We Are Reactive

The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of **usability**



The system stays responsive under **varying workload**. It can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs

The system rely on **asynchronous message passing** to establish a boundary between components that ensures loose coupling, isolation and location transparency



THE TWELVE-FACTOR APP

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Config

Store config in the environment

IV. Backing services

Treat backing services as attached resources

V. Build, release, run

Strictly separate build and run stages

VI. Processes

Execute the app as one or more stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep development, staging, and production as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin/management tasks as one-off processes

Спасибо за внимание

Денис Измайлова

izmaylov.dm@gmail.com

 <https://github.com/DenisIzmaylov>

 @DenisIzmaylov

 denis_izmaylov



STARTUP  **MAKERS**

www.startup-makers.com

Полезные ссылки

- <http://www.reactivemanifesto.org/>
- <http://12factor.net/>
- <https://www.infoq.com/articles/seven-uservices-antipatterns>
- <https://www.quora.com/In-the-future-of-data-center-architecture-who-will-be-the-main-orchestrator-controller-of-containers-Mesos-or-Kubernetes-What-will-be-the-division-of-responsibilities>