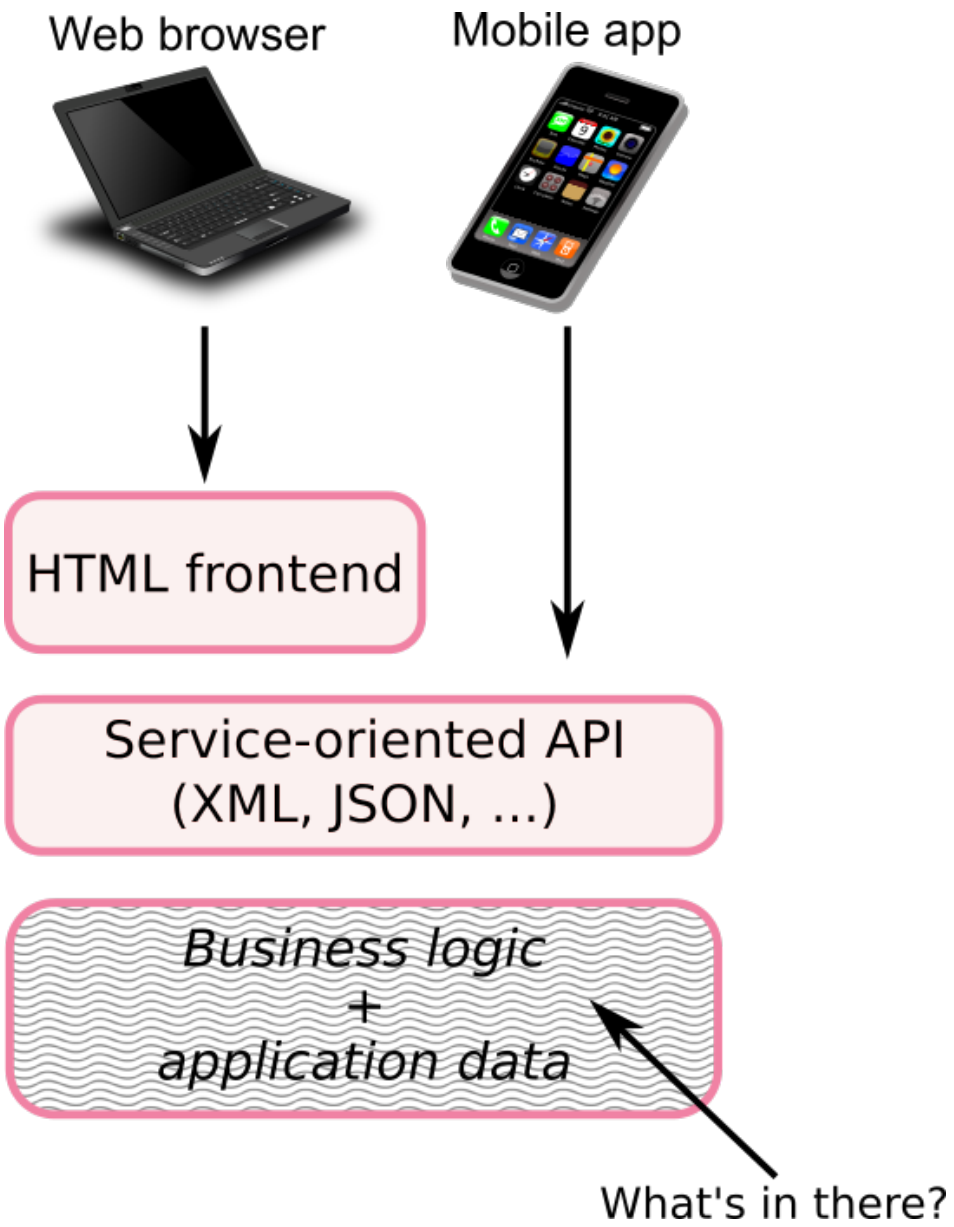
Développement logiciel pour le Cloud (TLC)

2. Micro-services

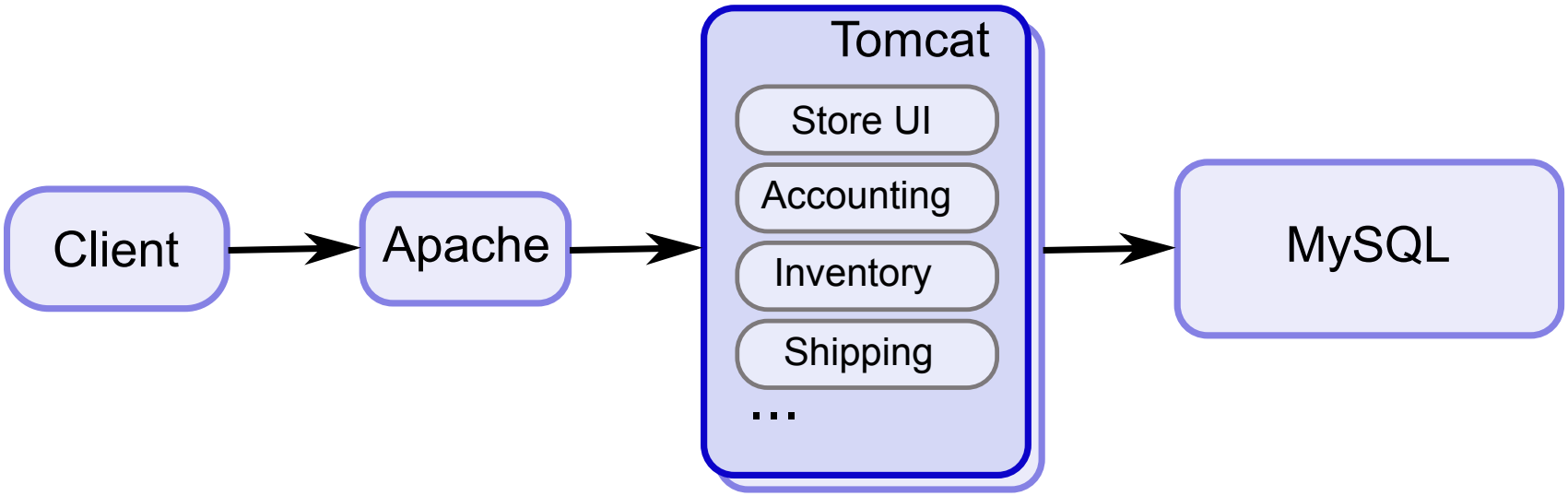
Quentin Dufour (credits: Guillaume Pierre)

Table of Contents

Web/mobile applications



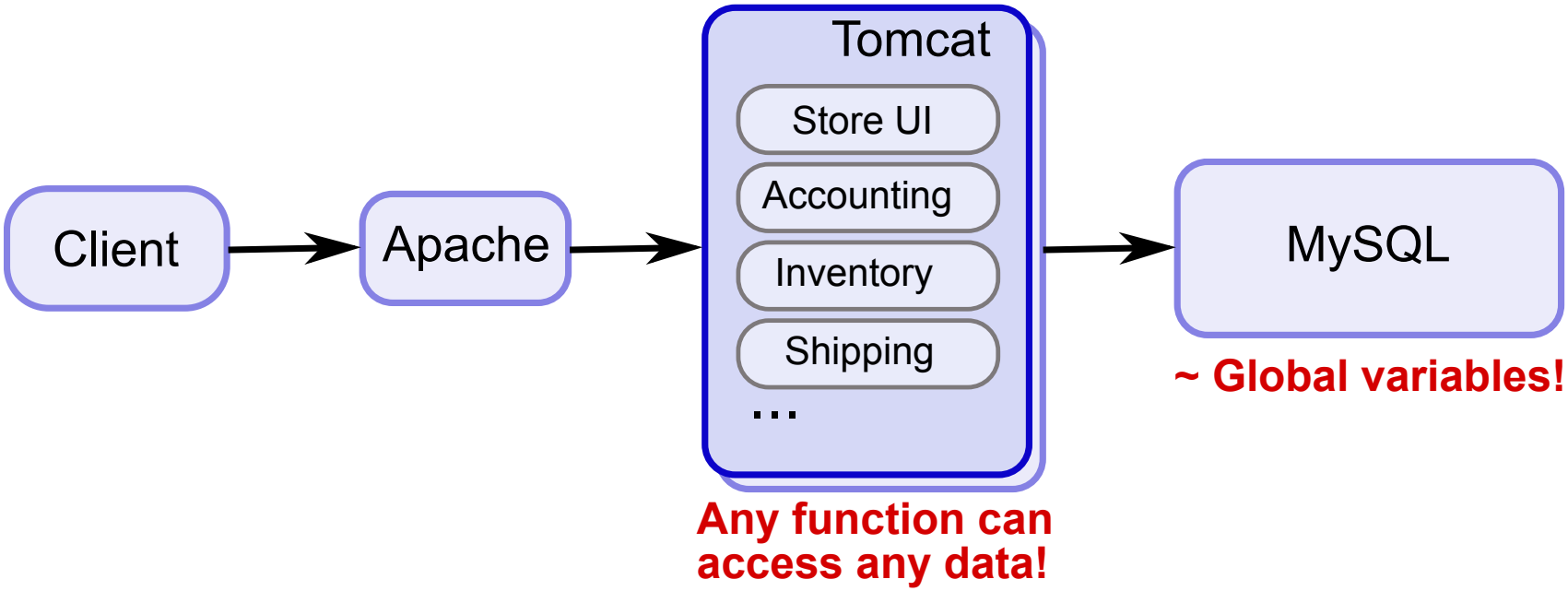
Traditional Web application architecture



Simple to:

- Develop
- Test
- Deploy

Traditional Web application architecture



Simple to:

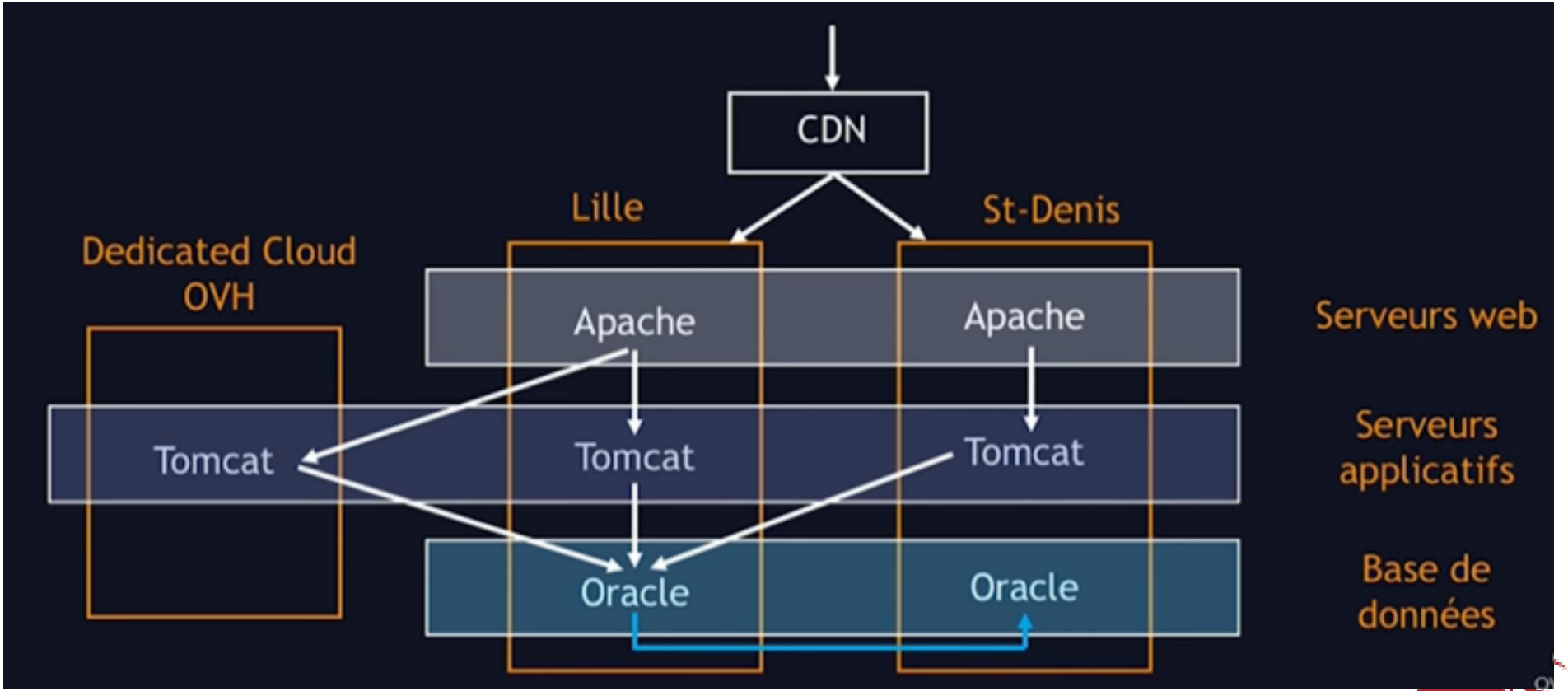
- Develop
- Test
- Deploy

But:

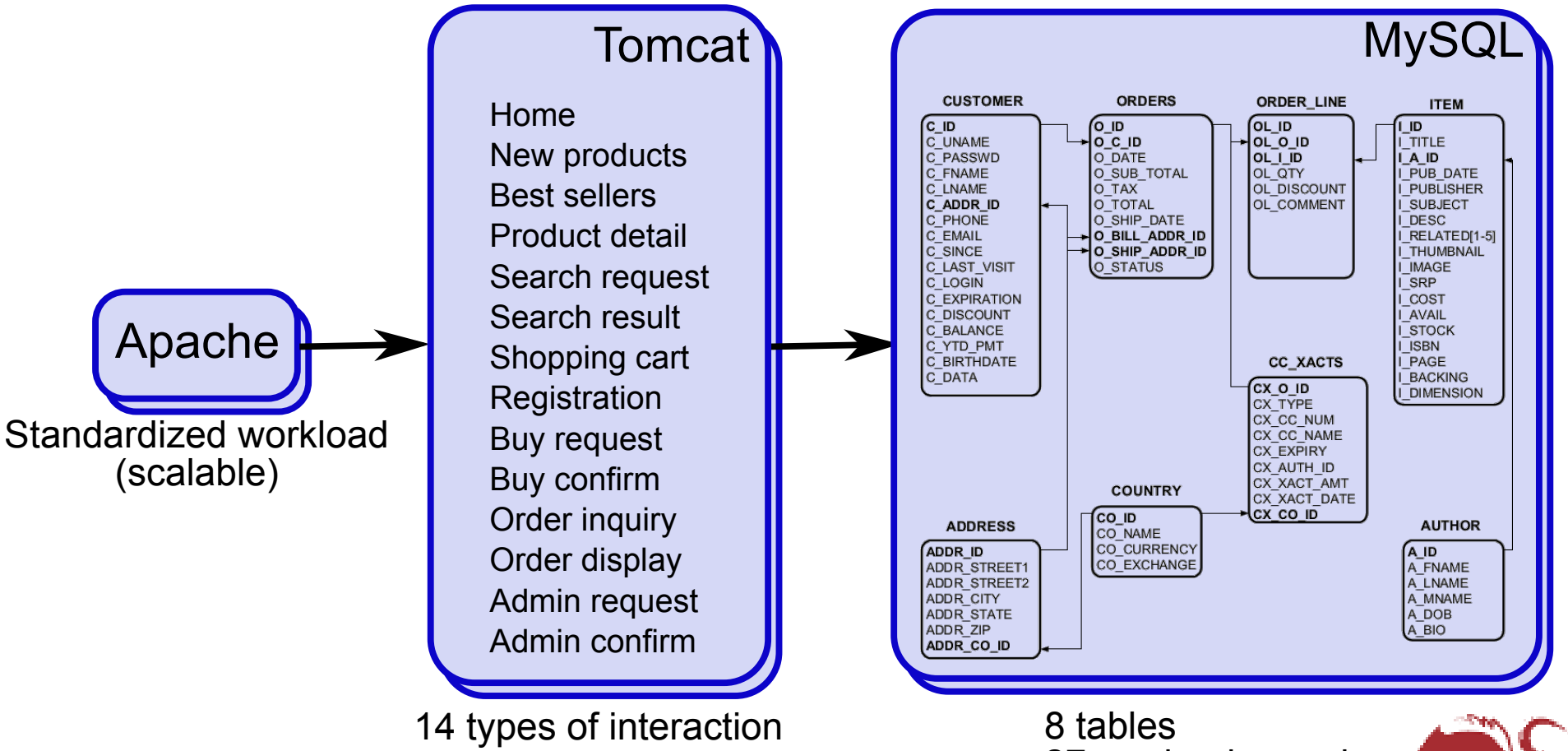
- No real data access control
- Any bug in one module can screw up the others
- Hard to scale...

Traditional Web application architecture

A real life application: Voyages-SNCF (now OuiSNCF)



TPC-W: a standard Web application + workload



8 tables
27 read-only queries
6 read-write queries
8 transactions

Scaling TPC-W

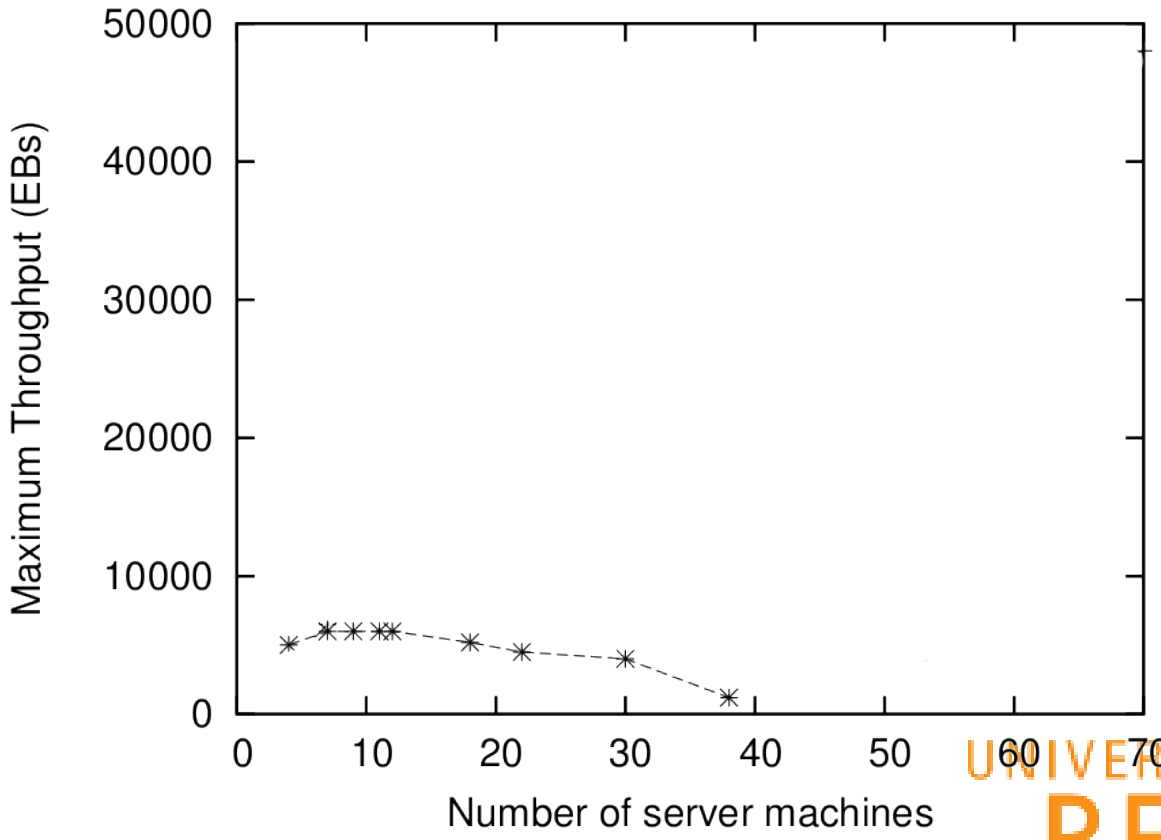
When the workload increases we need more machines to serve the traffic:

- Add more Apache servers
- Add more Tomcat servers
- Add more MySQL replicas

Scaling TPC-W

When the workload increases we need more machines to serve the traffic:

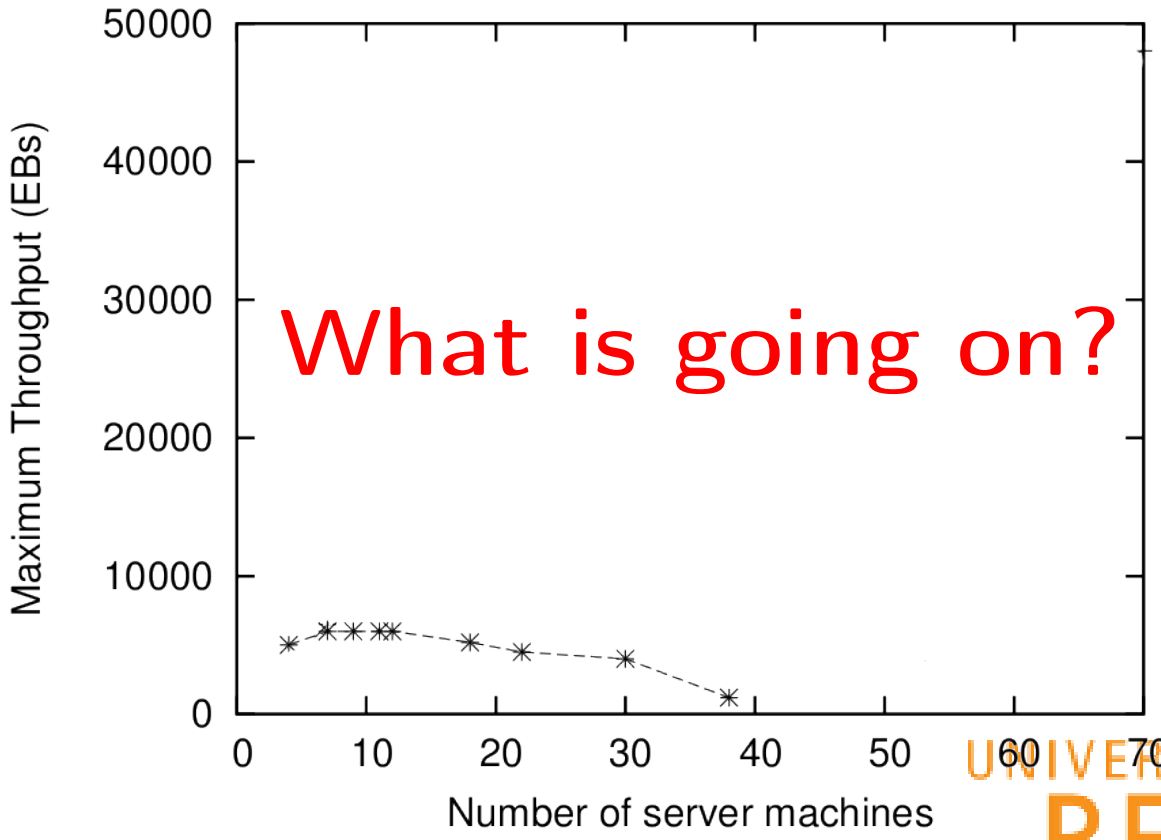
- Add more Apache servers
- Add more Tomcat servers
- Add more MySQL replicas



Scaling TPC-W

When the workload increases we need more machines to serve the traffic:

- Add more Apache servers
- Add more Tomcat servers
- Add more MySQL replicas



Scalability bottlenecks

- The Apache server layer is stateless
  - Several identical servers next to each other
  - No communication between them
  - Each request can be processed by any server

  $\Rightarrow$ No scalability issue

Scalability bottlenecks

- The Apache server layer is stateless
  - Several identical servers next to each other
  - No communication between them
  - Each request can be processed by any server

  $\Rightarrow$ No scalability issue
- The Tomcat server layer is also stateless
  - Identical Tomcat servers run independently from each other

  $\Rightarrow$ No scalability issue

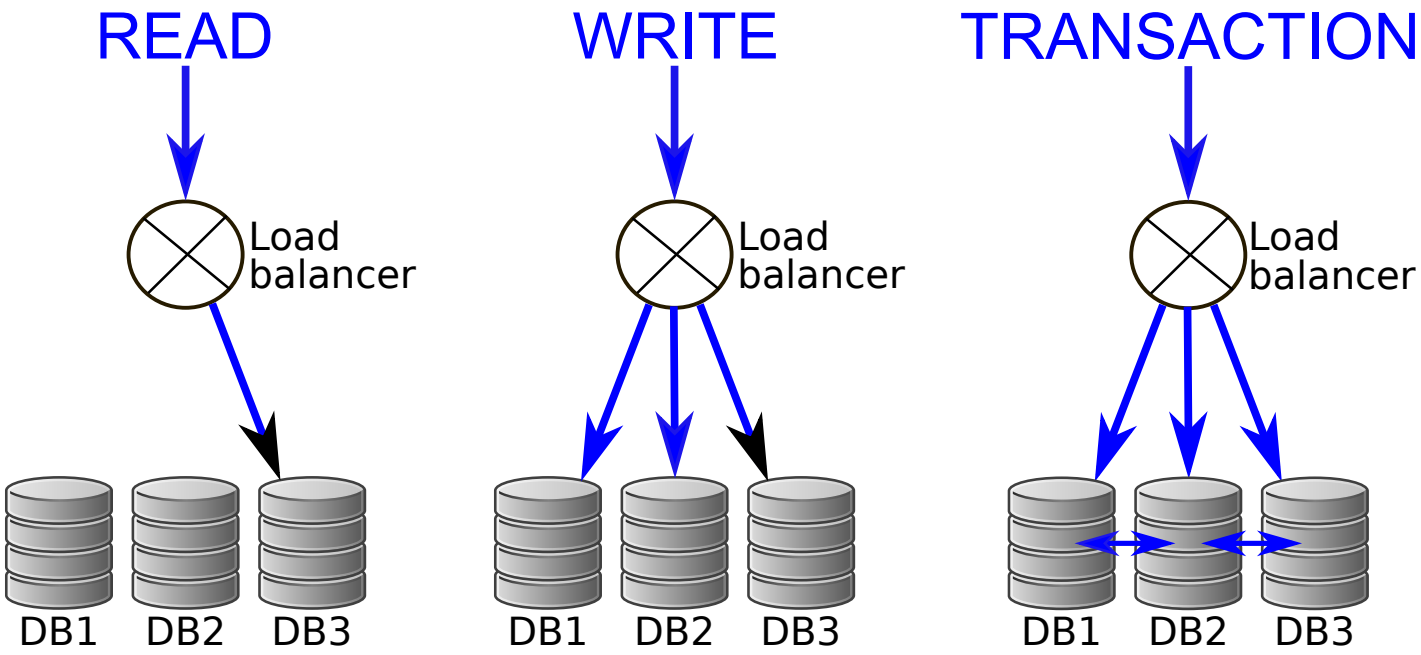
Scalability bottlenecks

- The Apache server layer is stateless
  - Several identical servers next to each other
  - No communication between them
  - Each request can be processed by any server

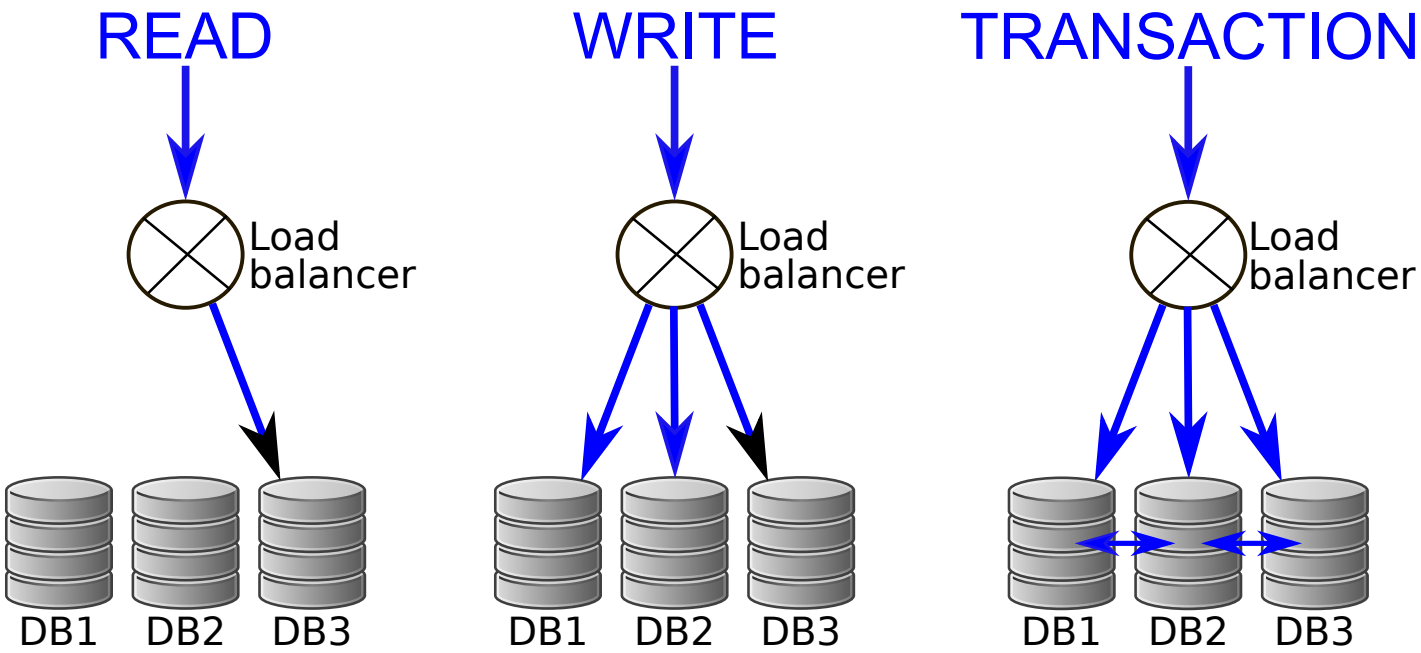
  $\Rightarrow$ No scalability issue
- The Tomcat server layer is also stateless
  - Identical Tomcat servers run independently from each other

  $\Rightarrow$ No scalability issue
- The MySQL server layer is stateful
  - Multiple database servers store replicas of the application's state
  - READ-ONLY queries can be processed by any server
  - READ-WRITE queries must be processed by every server
  - TRANSACTIONS require complex algorithms across all servers

Scalability issues



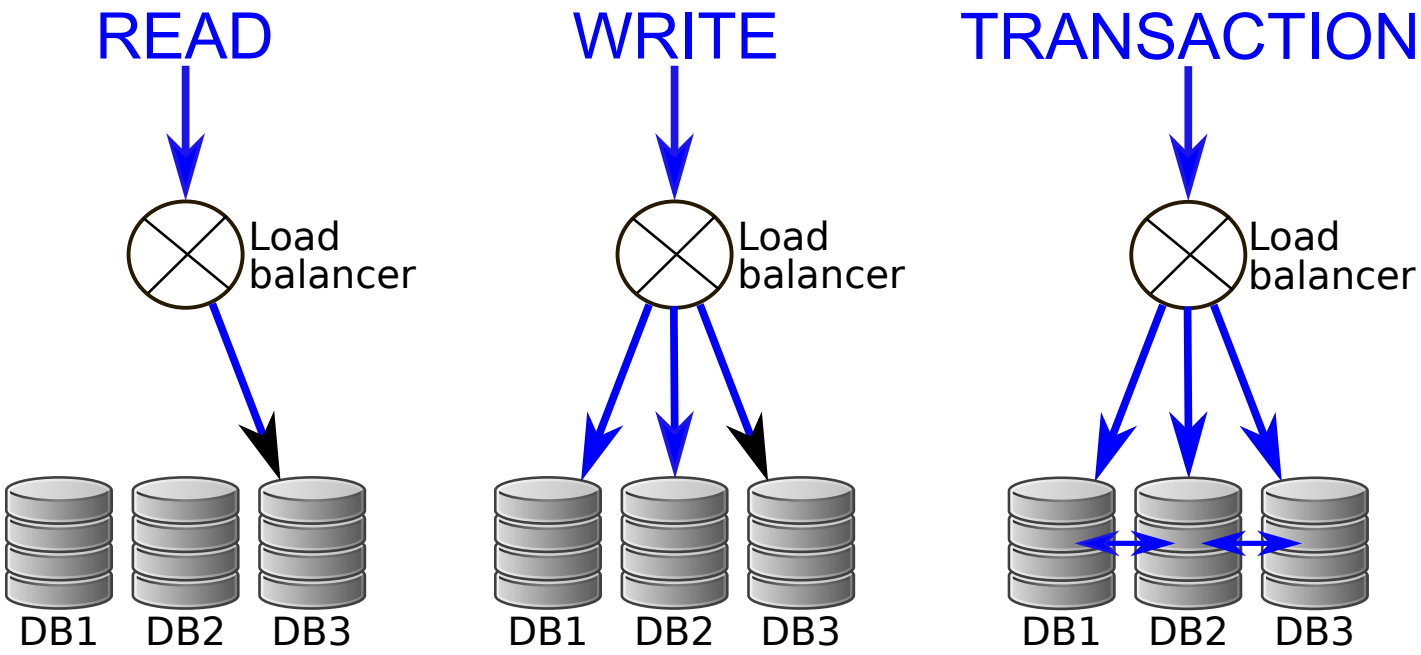
Scalability issues



If we have n database servers, then each server processes:

$$Single\_DB\_load = \frac{READS}{n} + WRITES + TRANSACTIONS$$

Scalability issues

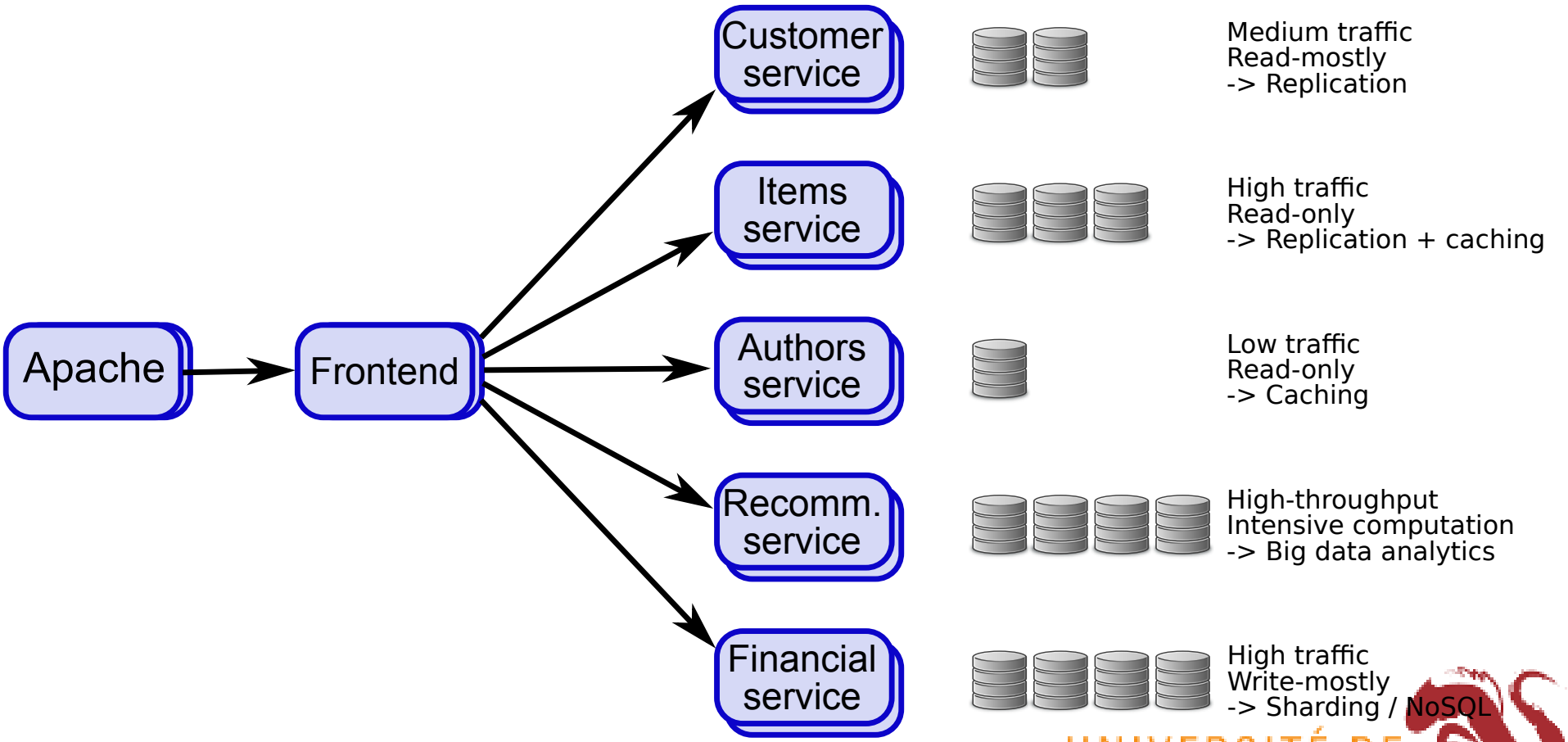


If we have n database servers, then each server processes:

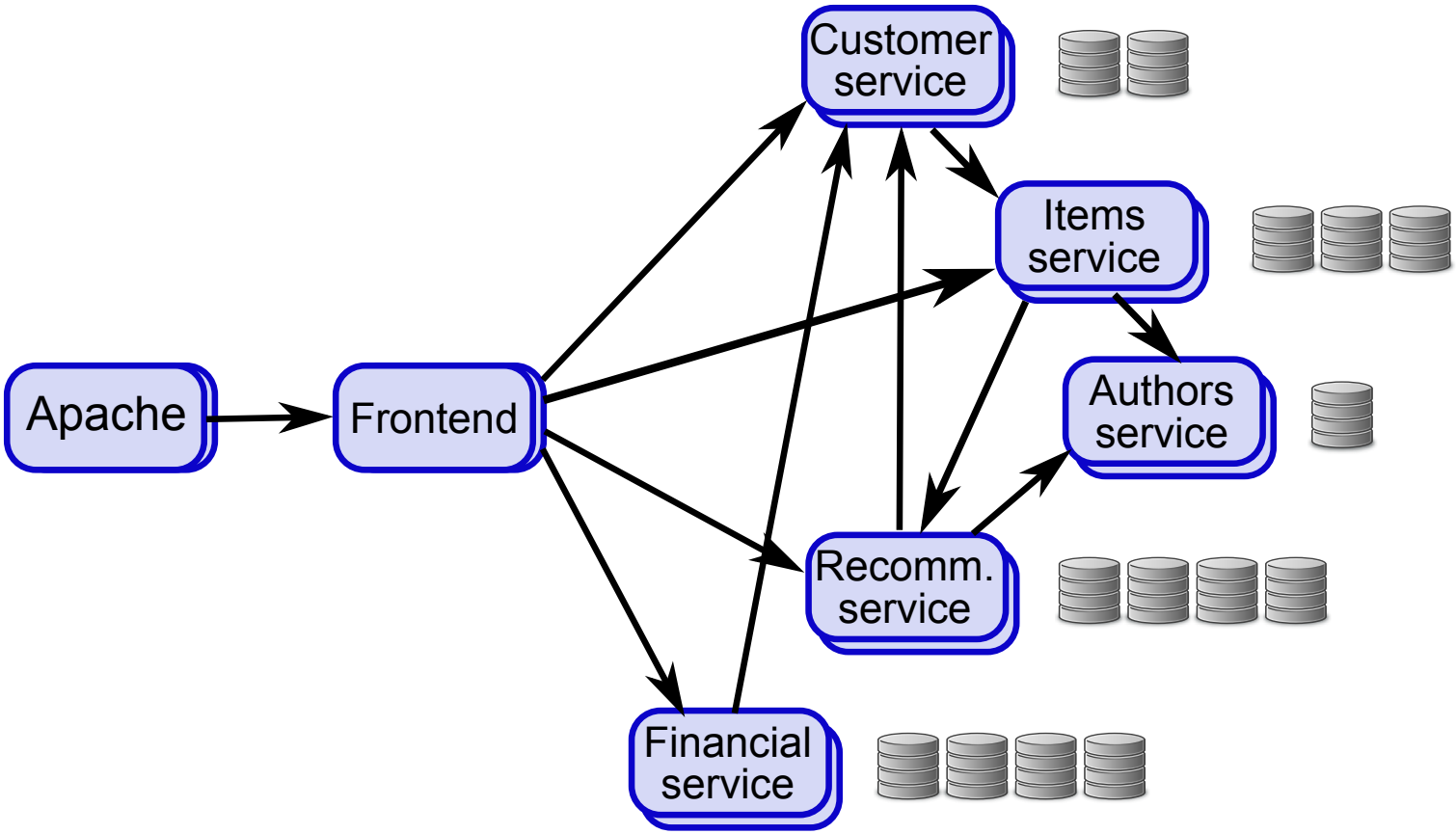
$$Single\_DB\_load = \frac{READS}{n} + WRITES + TRANSACTIONS$$

If WRITES+TRANSACTIONS is large enough to saturate one DB server, then **increasing n will not help!**

TPC-W's potential microservice architecture



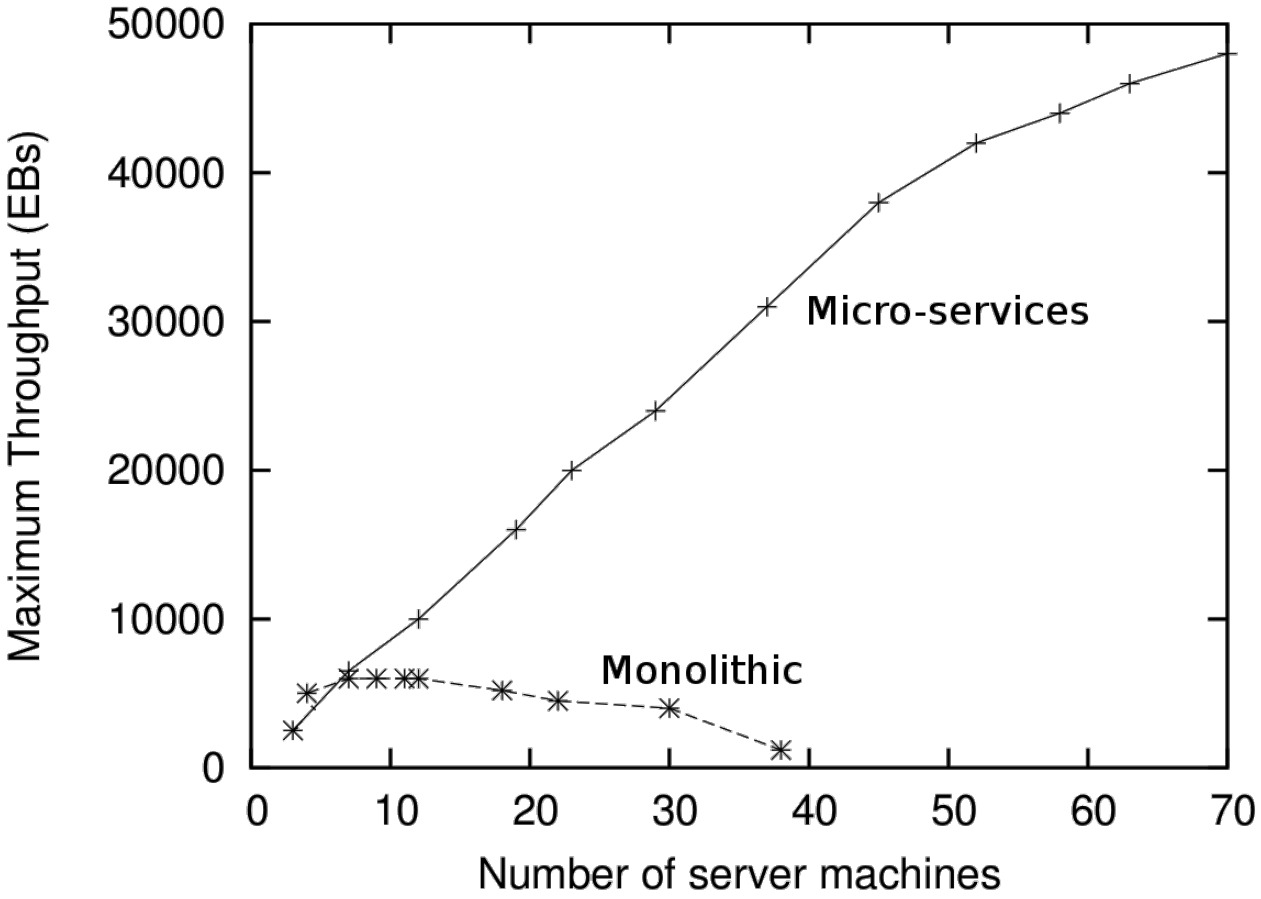
TPC-W's potential microservice architecture



"If you hit the Amazon.com gateway page, the application calls more than 100 services to collect data and construct the page for you."
— Werner Vogels, Amazon.com CTO.

TPC-W's potential microservice architecture

If you do things right:



http://www.globule.org/publi/SODDSWA_www2008.html

Table of Contents

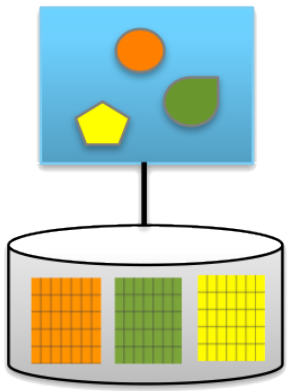

All you have to do is start your little project with five microservices.
Daniel Stori {turnoff.us}

Microservices philosophy:

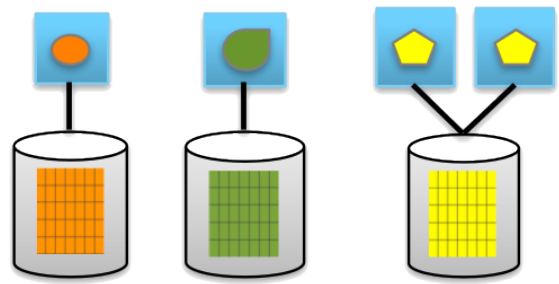
"Each microservice should do a single thing, but do it really well"

(very similar to the classic Unix philosophy)

Monolithic Applications



- All functionality in single process
- Change cycles tied together
- Inefficient scaling



Microservices

- Broken into decoupled services
- Communication via self contained APIs
- Decentralized data

Microservice architectural principles

Any microservice should be:

1. **Elastic:** *A microservice must be able to scale, up or down, independently of other services in the same application.*

Microservice architectural principles

Any microservice should be:

1. **Elastic:** *A microservice must be able to scale, up or down, independently of other services in the same application.*
2. **Resilient:** *A microservice must fail without impacting other services in the same application.*

Microservice architectural principles

Any microservice should be:

1. **Elastic:** *A microservice must be able to scale, up or down, independently of other services in the same application.*
2. **Resilient:** *A microservice must fail without impacting other services in the same application.*
3. **Composable:** *A microservice must offer an interface that is uniform and is designed to support service composition.*

Microservice architectural principles

Any microservice should be:

1. **Elastic:** *A microservice must be able to scale, up or down, independently of other services in the same application.*
2. **Resilient:** *A microservice must fail without impacting other services in the same application.*
3. **Composable:** *A microservice must offer an interface that is uniform and is designed to support service composition.*
4. **Minimal:** *A microservice must only contain highly cohesive entities.*

Microservice architectural principles

Any microservice should be:

1. **Elastic:** *A microservice must be able to scale, up or down, independently of other services in the same application.*
2. **Resilient:** *A microservice must fail without impacting other services in the same application.*
3. **Composable:** *A microservice must offer an interface that is uniform and is designed to support service composition.*
4. **Minimal:** *A microservice must only contain highly cohesive entities.*
5. **Complete:** *A microservice must be functionally complete.*

http://www.nirmata.com/2015/02/microservices-five-architectural-constraints/

THE EVOLUTION OF

SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED
ARCHITECTURE
(aka Copy & Paste)

2000's

LASAGNA-ORIENTED
ARCHITECTURE
(aka Layered Monolith)

2010's

RAVIOLI-ORIENTED
ARCHITECTURE
(aka Microservices)

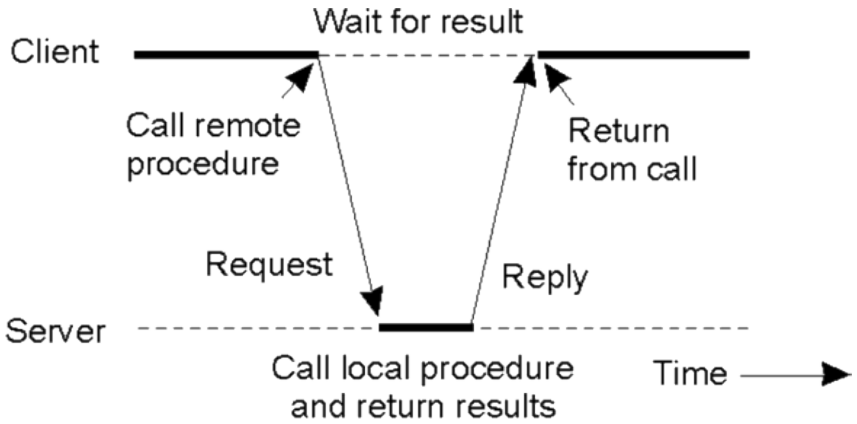
WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE

UNIVERSITÉ DE RENNES 1

Inter-microservice communication

Remote procedure call:

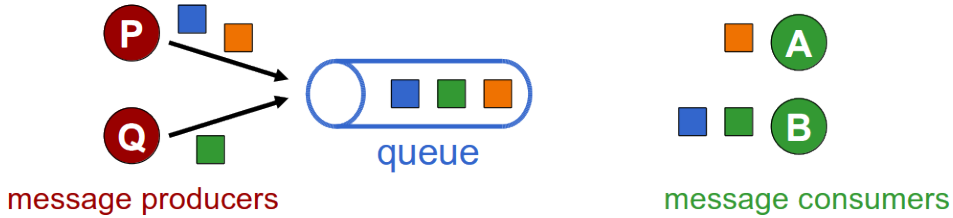


Service2 must expose a well-defined address (IP address + port, URL, ...)

- ☹ What if Service2 wants to add/remove resources?
- ☹ What if it fails?

Message-oriented middlewares

One-to-one communication



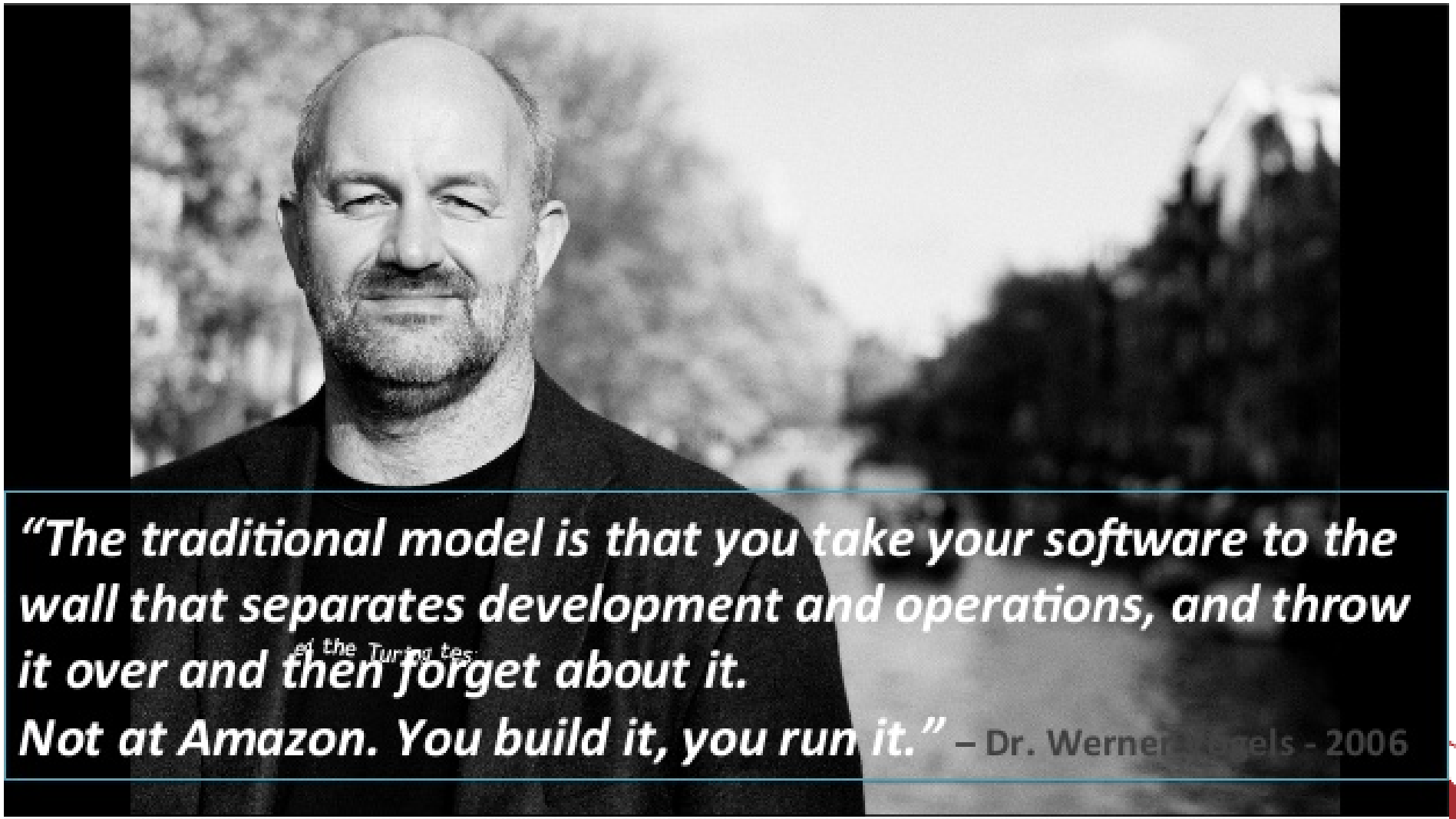
One-to-many communication

- ☺ Producers don't need to know where consumers are
- ☺ Producers are not blocked while their messages are processed
- ☺ The receiving service can scale up/down without bothering its clients

RabbitMQ
Open Source Enterprise Messaging

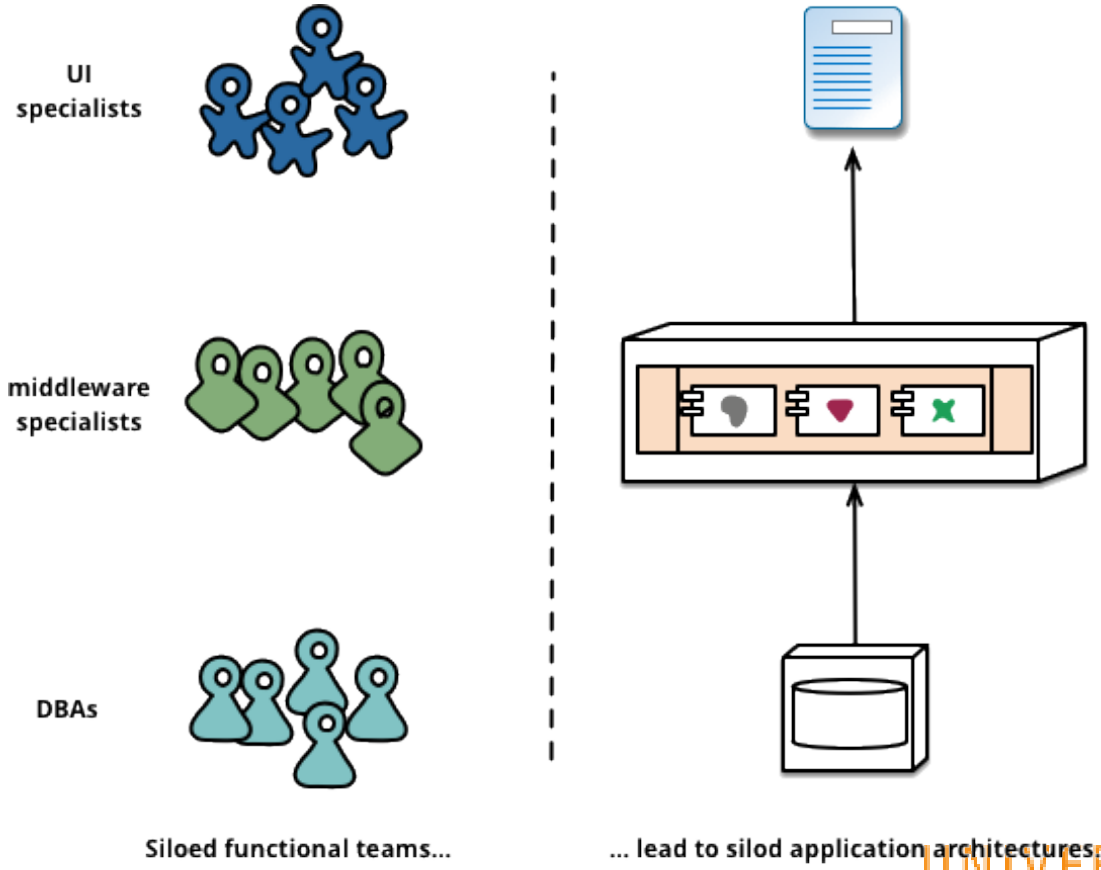
Apache Kafka
A high-throughput distributed messaging system.

Table of Contents


"The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it.
Not at Amazon. You build it, you run it." – Dr. Werner

Conway's law

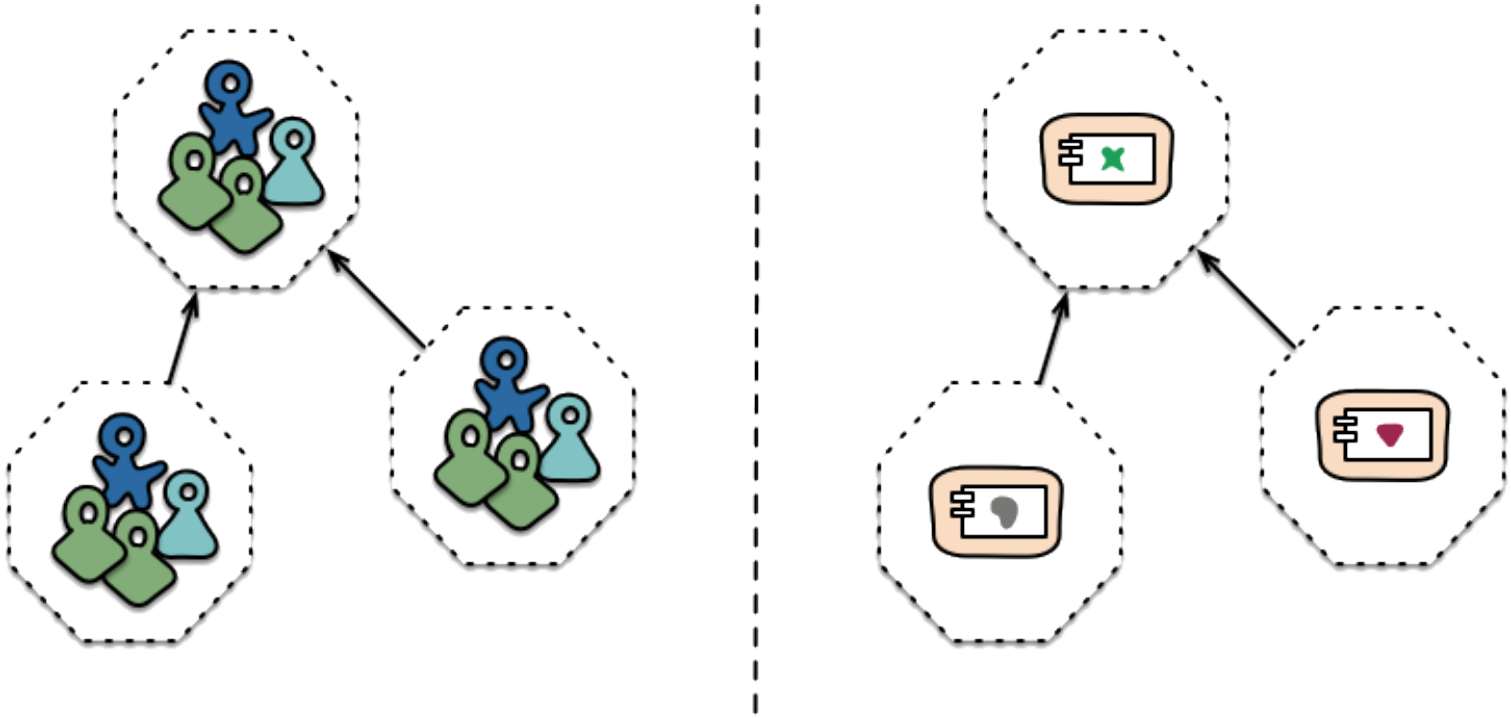
"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." — Melvyn Conway, 1967.



http://martinfowler.com/articles/microservices.html

Conway's law

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." — Melvyn Conway, 1967.



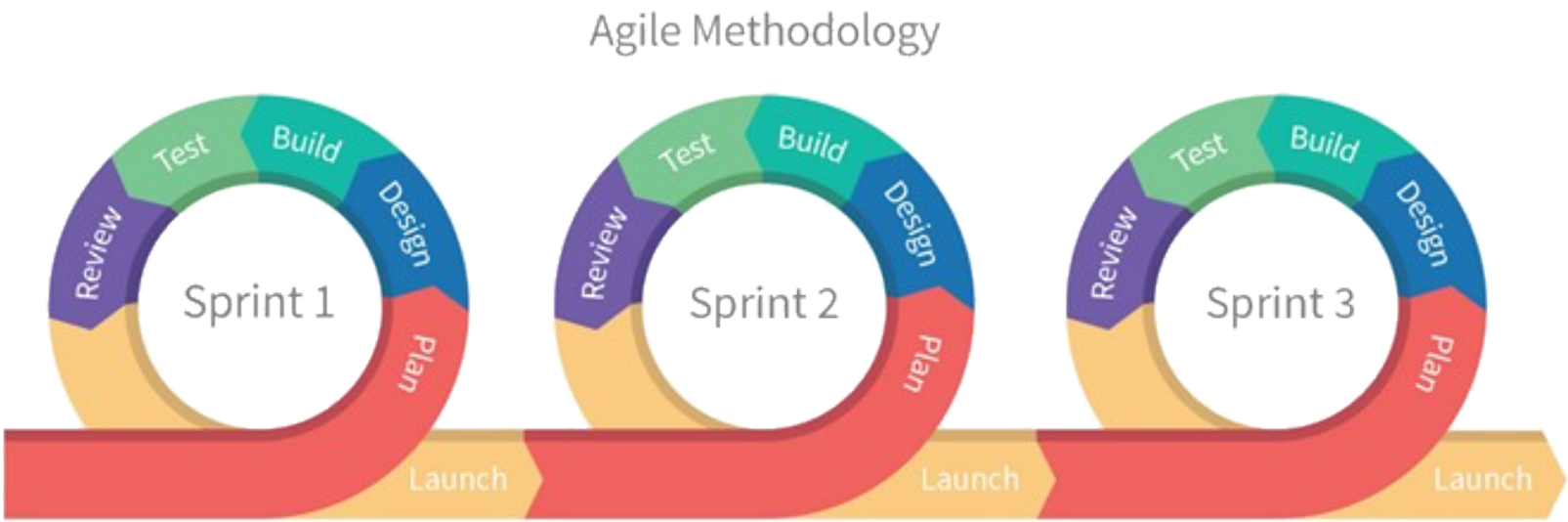
Cross-functional teams...

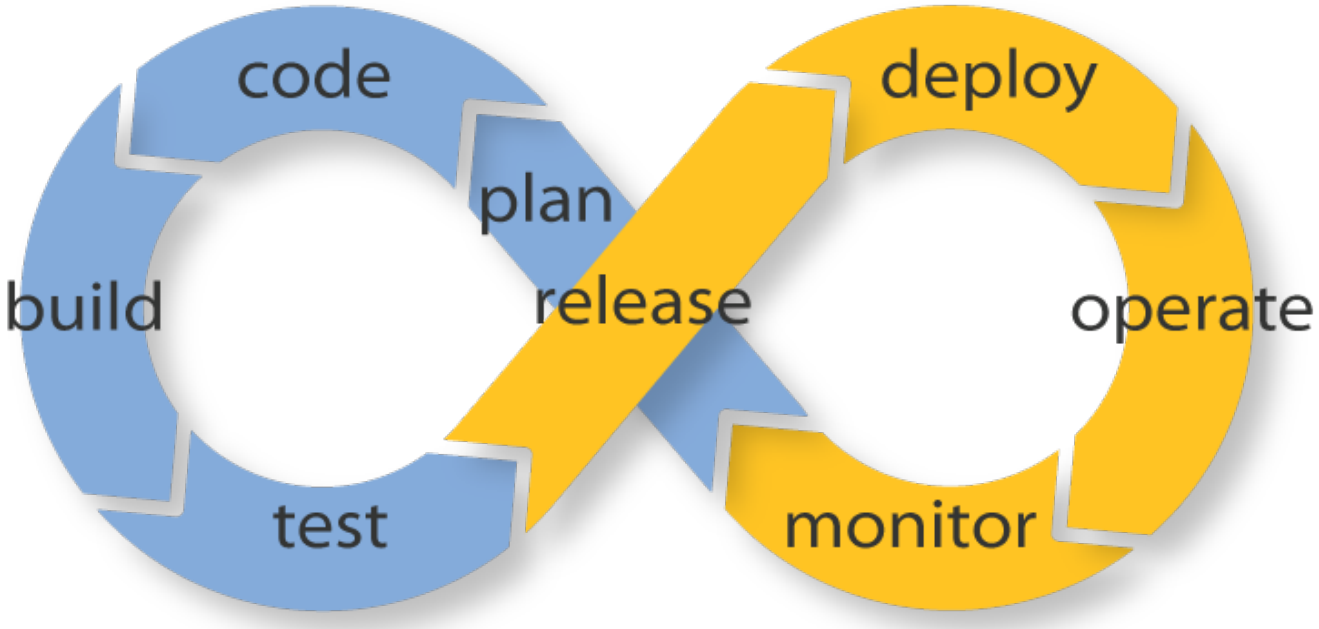
... organised around capabilities

Because Conway's Law

http://martinfowler.com/articles/microservices.html

Agile DevOps methodologies





Endless Possibilities: DevOps can create an infinite loop of release and feedback for all your code and deployment targets.

How big should a micro-service be?

Amazon's two-pizza principle

Teams shouldn't be larger than what two pizzas can feed.

If you're still hungry after sharing these two pizzas during a lunch meeting, then the team must split (and the micro-service as well).

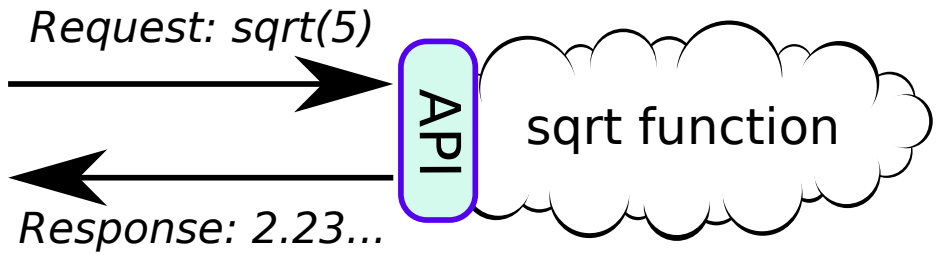
(and, yes, they also standardized the maximum pizza size. . . ☺ *)*

http://blog.idonethis.com/two-pizza-team/

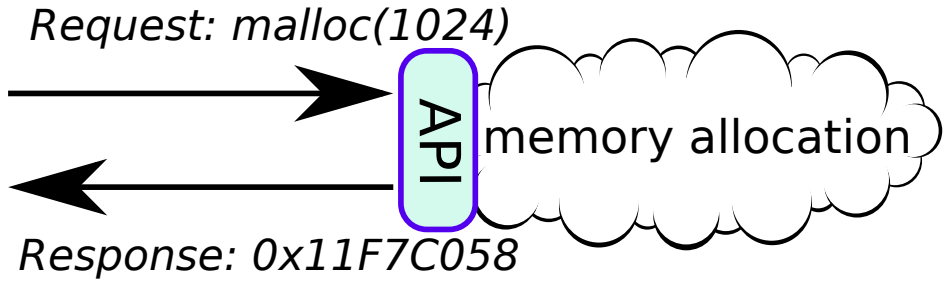
Table of Contents

Infrastructure-as-a-Service is mostly an API

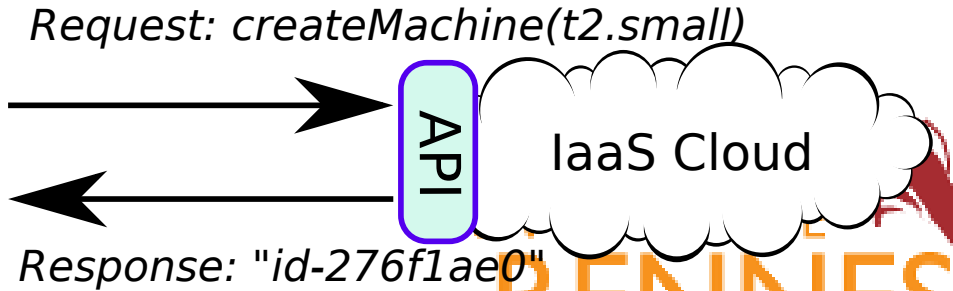
float x = sqrt(5);



char *x = malloc(1024);



id = createMachine(t2.small);



OpenStack: **the** open-source IaaS implementation

1991:  + 

2011: 

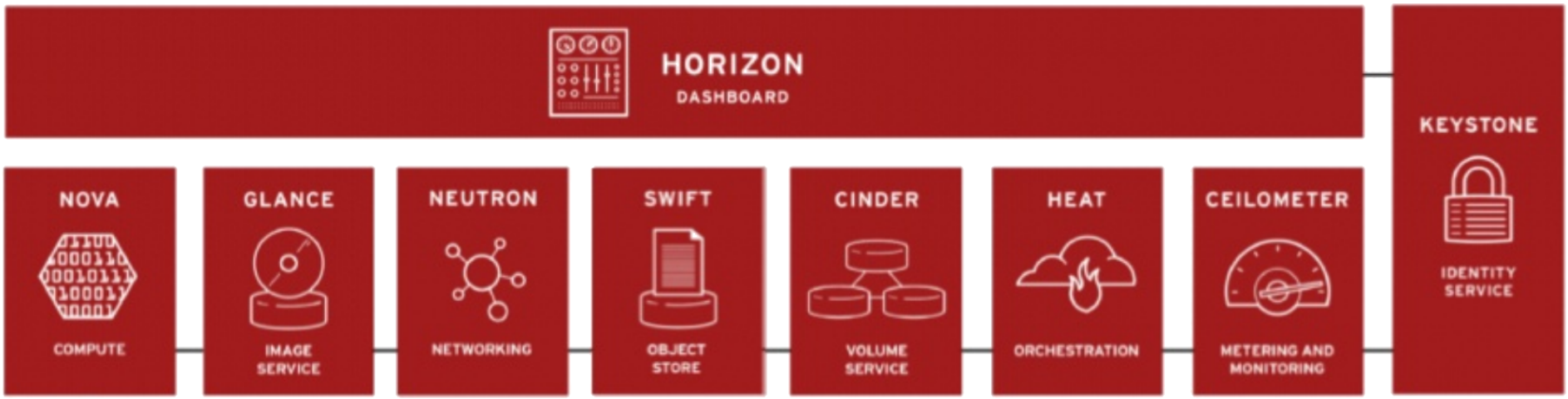
+ 

- "OpenStack is a community of open source developers, participating organizations and users building and running the open source cloud operating system."
- "OpenStack is a Cloud Orchestration layer"
- "OpenStack is a Cloud Operating System"

OpenStack's common architecture

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

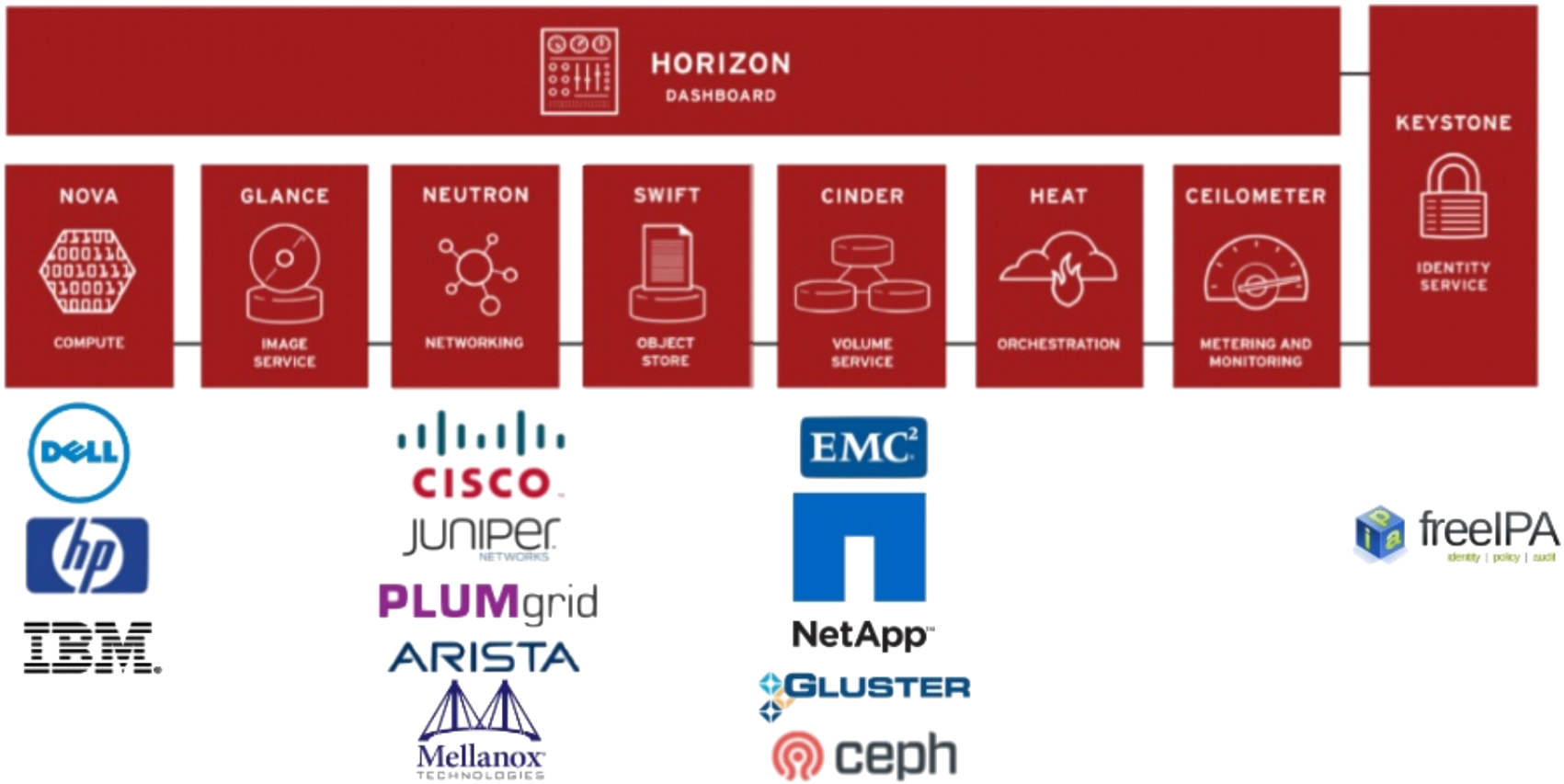
OpenStack's components



- A single user-facing API + dashboard
- Internal services for identity management, compute, networking, etc.

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

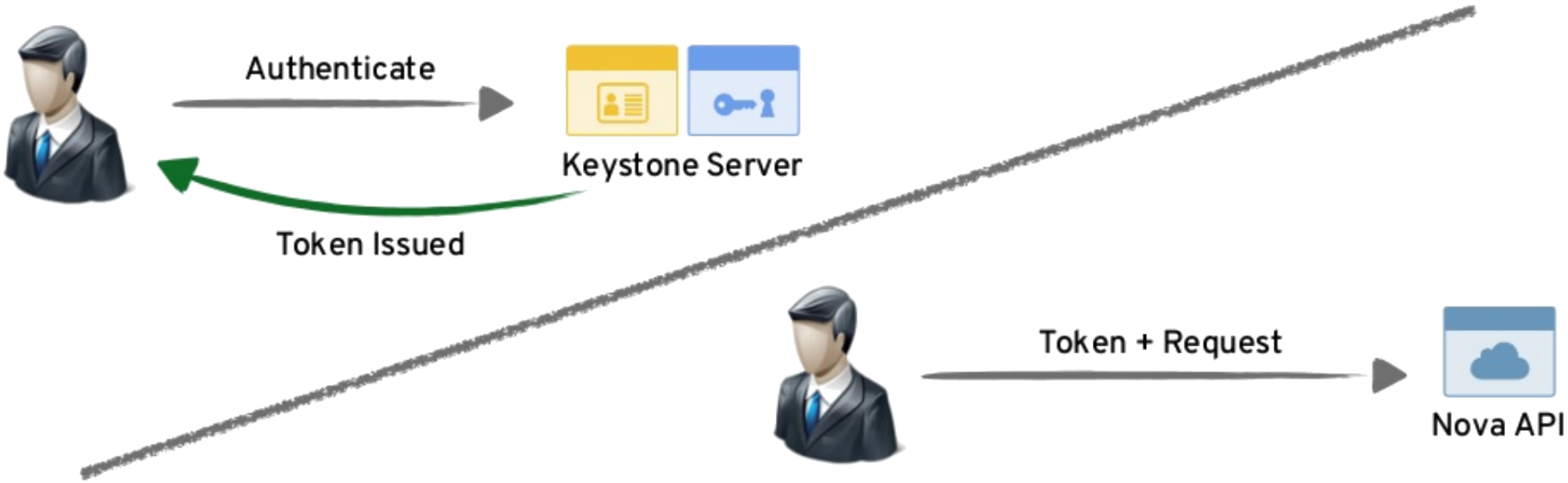
OpenStack's components



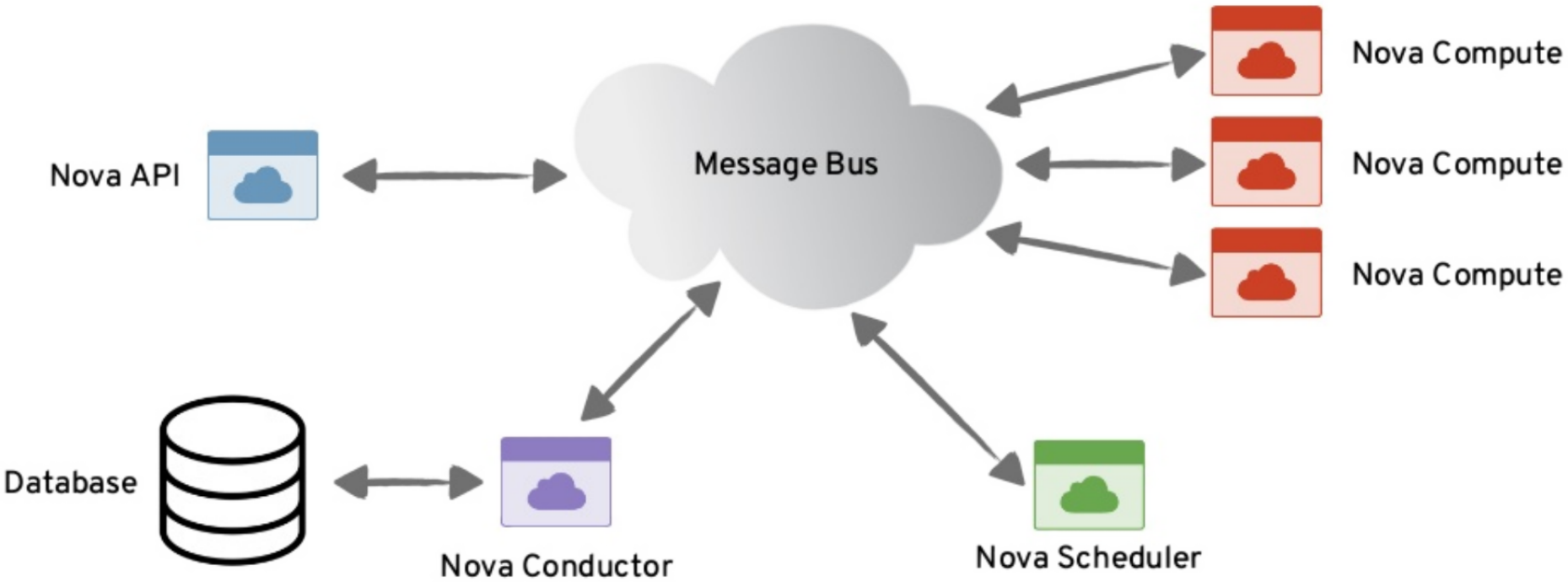
- Each service implements a standard internal API
- Each service supports vendor-specific plugins

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

Keystone: the identity service



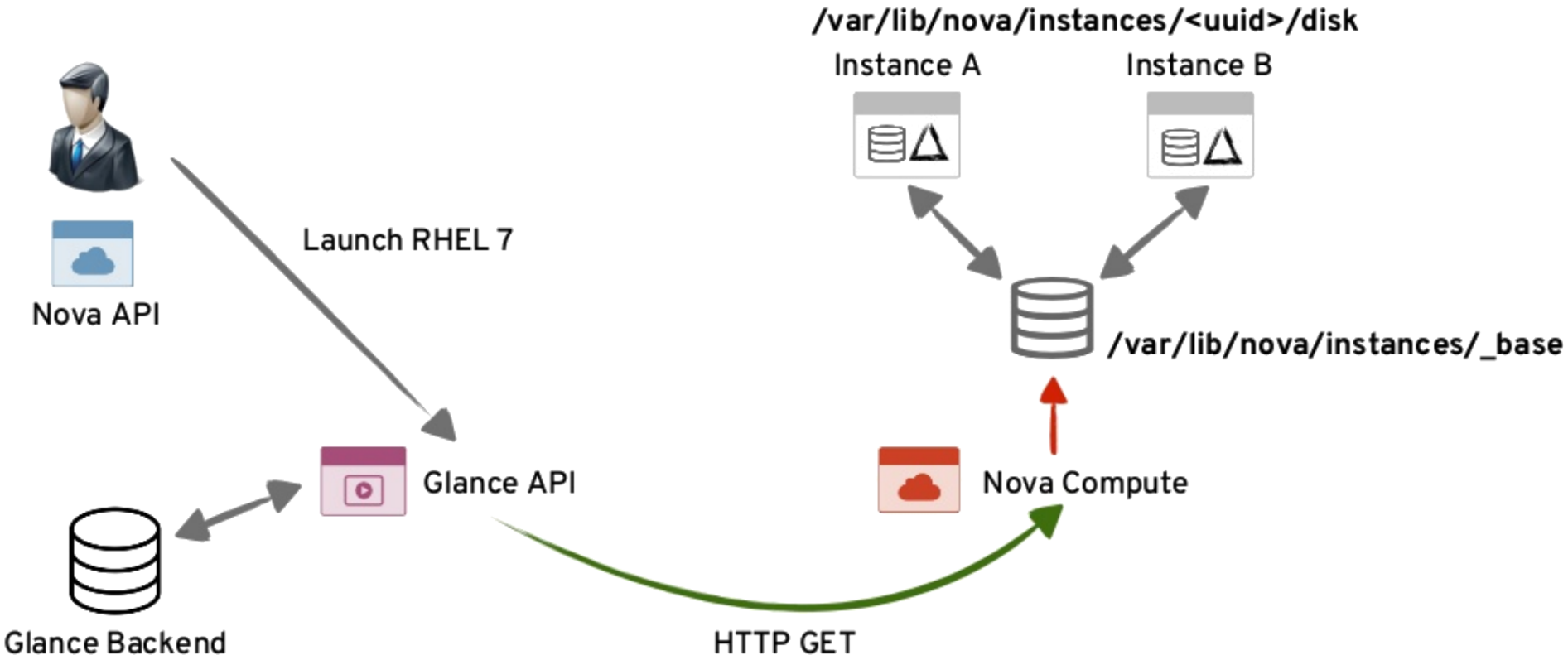
Nova: the compute service



"The service which starts VMs"

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

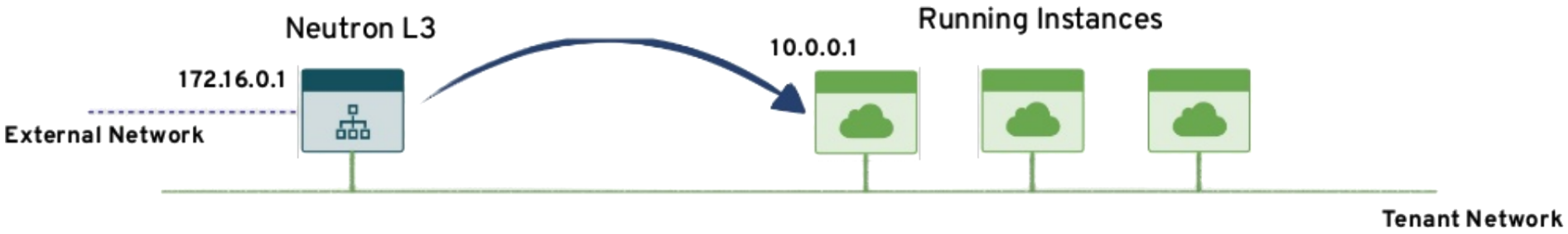
Glance: the image service



"image" == content of a virtual hard drive that a VM can boot from

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

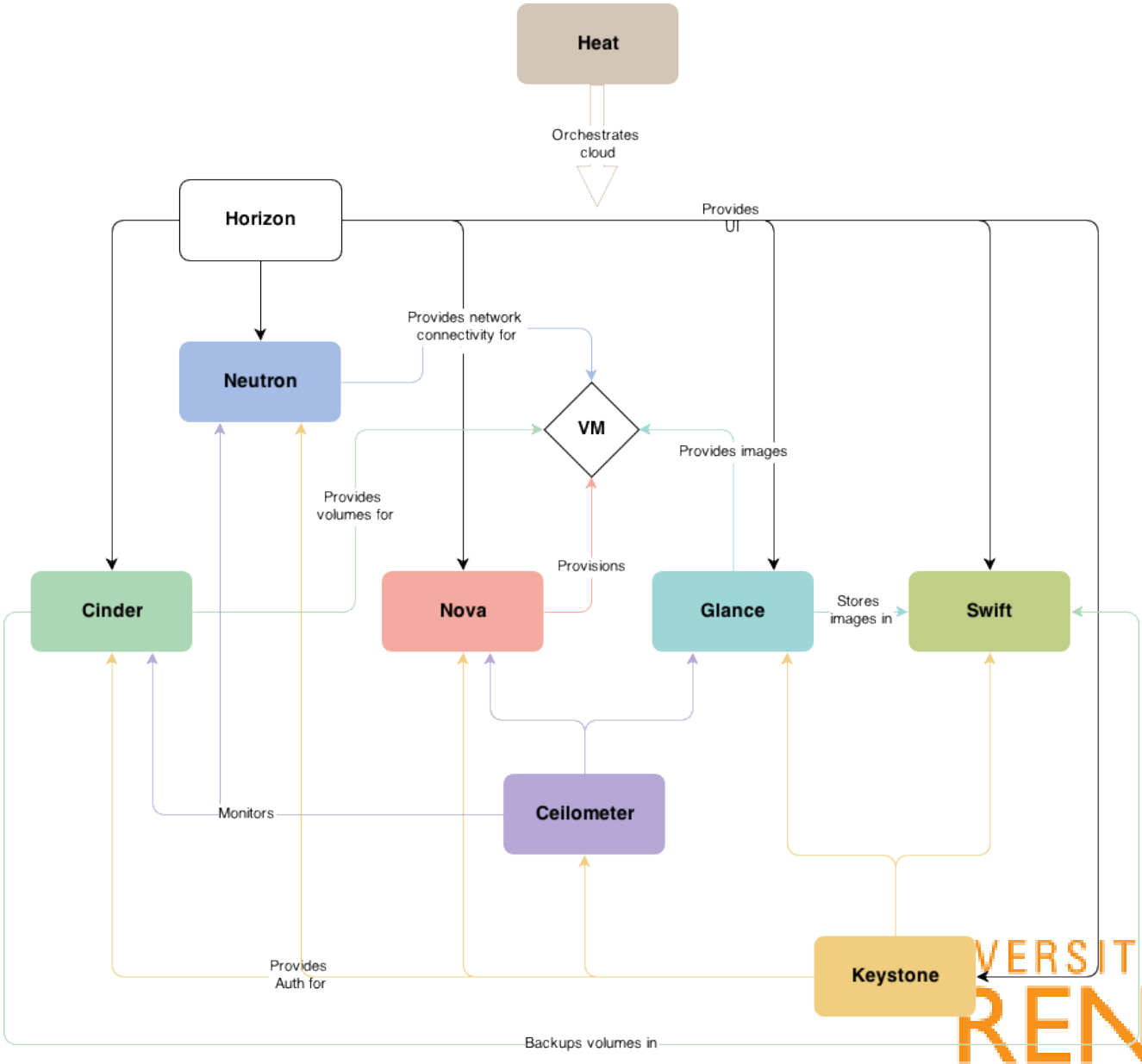
Neutron: the networking service



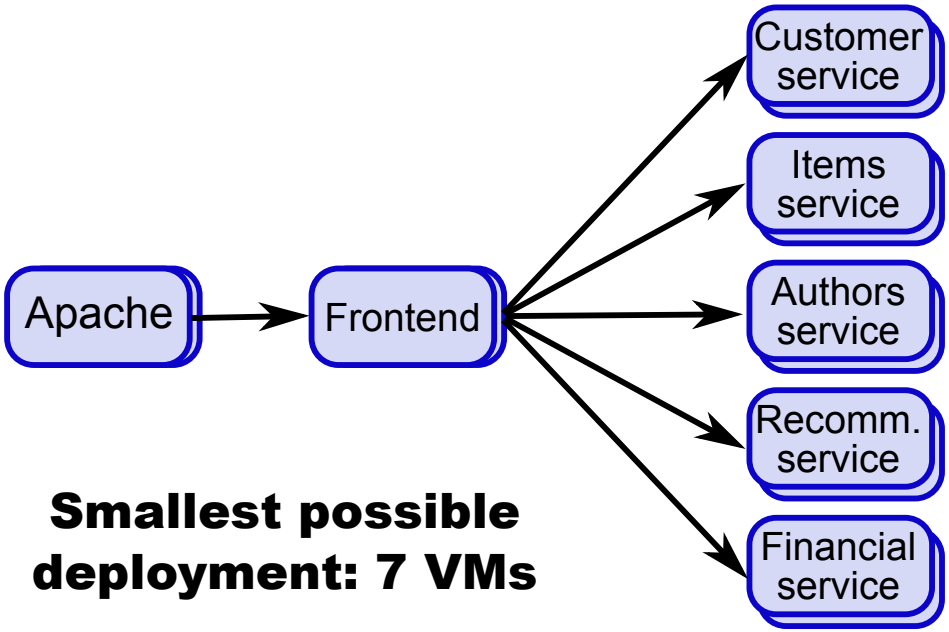
- Manage a pool of IP addresses
- Give an IP address to each new VM
- Routing, firewall, private networks, etc.

http://www.slideshare.net/alessandrovozza/cloud-architect-alliance-15-openstack

In reality...

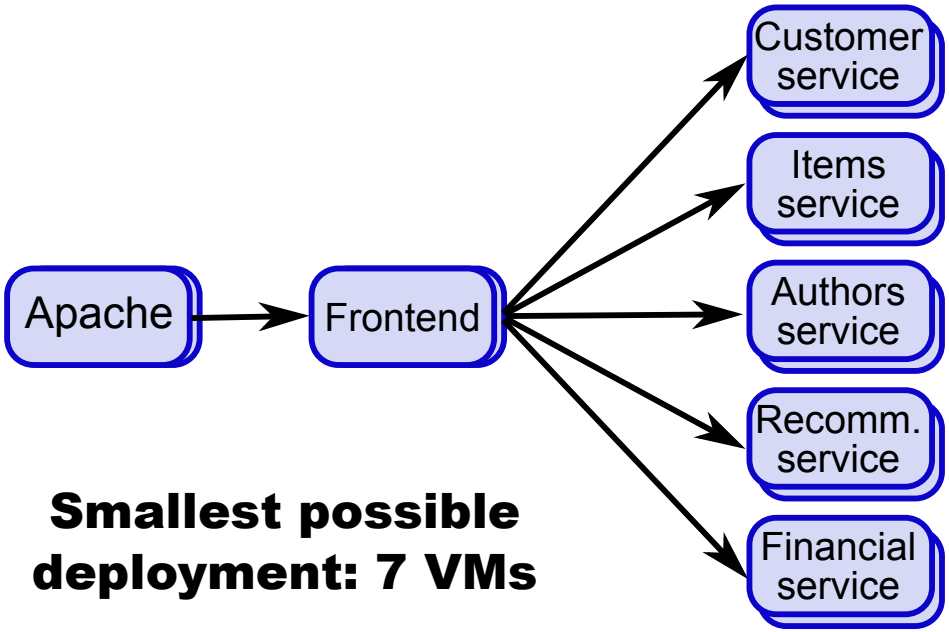

Heat
Orchestrates cloud
Horizon
Provides UI
Provides network connectivity for
Neutron
VM
Provides images
Provides volumes for
Provisions
Cinder
Nova
Glance
Stores images in
Swift
Monitors
Ceilometer
Provides Auth for
Keystone
Backups volumes in

Back to our microservice architecture



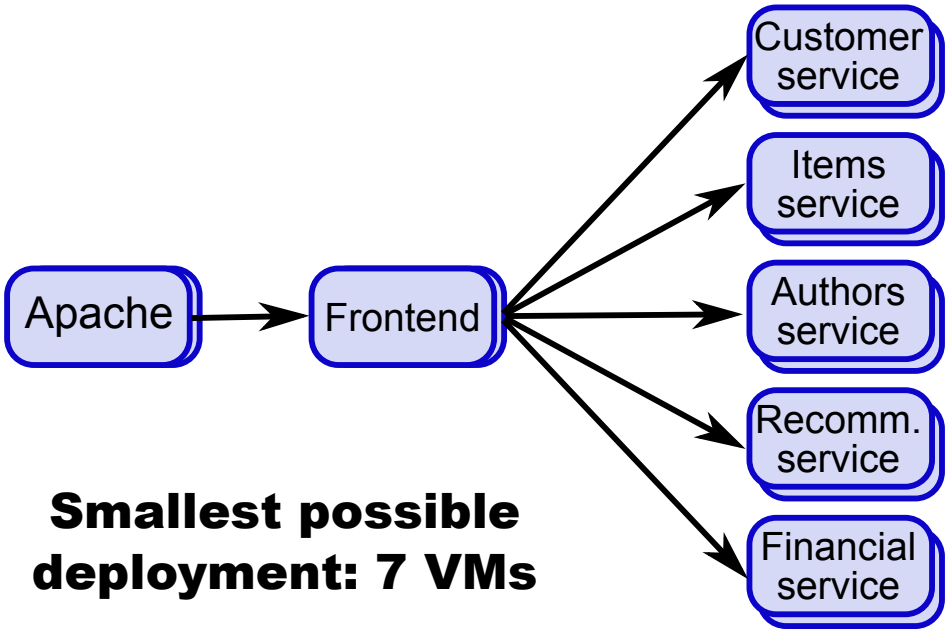
Smallest possible deployment: 7 VMs

Back to our microservice architecture



- The smallest VM type at Amazon Web services: t2.nano
  - 1 CPU core, 500 MB RAM, no disk (to be purchased separately)
  - Price: \$0.0065/hour + VAT
  - Storage: \$0.045/GB-month + VAT

Back to our microservice architecture



- The smallest VM type at Amazon Web services: t2.nano
  - 1 CPU core, 500 MB RAM, no disk (to be purchased separately)
  - Price: \$0.0065/hour + VAT
  - Storage: \$0.045/GB-month + VAT
- If we give a small 50 GB disk to each VM:
  The smallest possible deployment costs **\$587/year + VAT!!!**

Microservices ♡♡ Containers

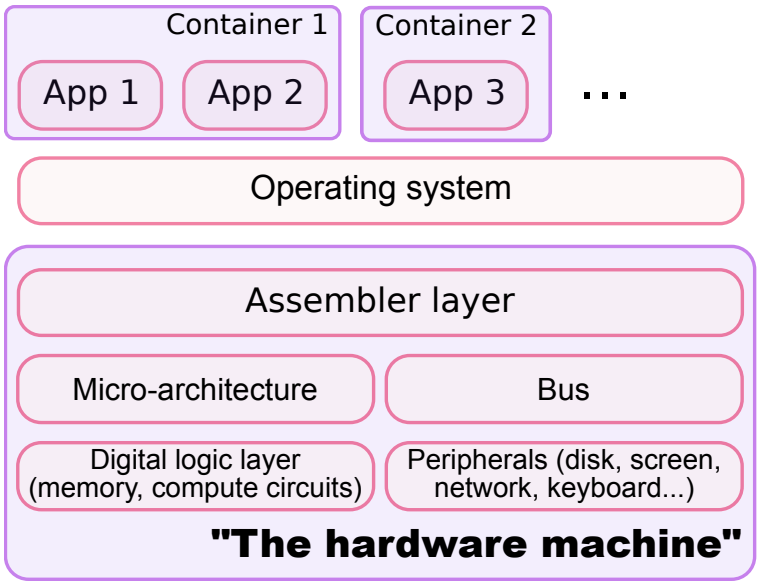
- Using VMs, our little microservice application cannot use less than:
  - 7 cores, 3.5 GB RAM, 350 GB disk...
  - This will keep growing if we add more microservices to the application!
- The problem with VMs+microservices:
  - Each VM needs its own operating system, libraries, deamons, programs, etc.
  - This takes lots of resources!
  - And it is not always necessary

Microservices ♡♡ Containers

- Using VMs, our little microservice application cannot use less than:
  - 7 cores, 3.5 GB RAM, 350 GB disk...
  - This will keep growing if we add more microservices to the application!
- The problem with VMs+microservices:
  - Each VM needs its own operating system, libraries, deamons, programs, etc.
  - This takes lots of resources!
  - And it is not always necessary
- Containers are much more lightweight
  - Only one OS for the whole machine (kernel, deamons, etc.)
  - Each container includes only what's really necessary for their execution (a webserver, a DB server, etc.)
  - We can easily run hundreds of containers on a medium-grade machine

Table of Contents

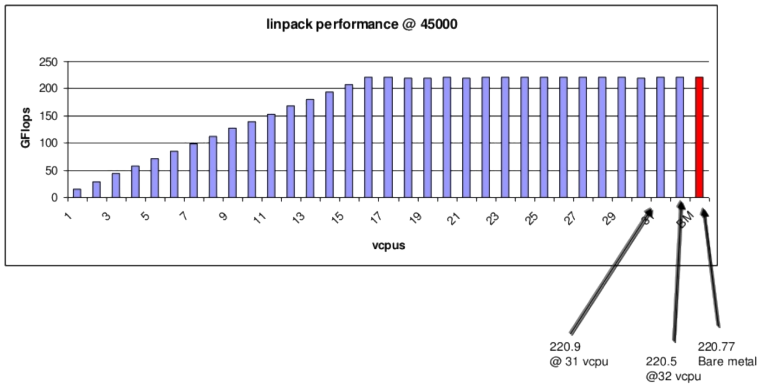
LXC Containers



- All containers in a machine share:
  - OS kernel
  - Kernel configuration & modules
  - Background deamons
  - Programs & libraries (where appropriate)
- Each container may specialize:
  - Files/programs/libraries
  - Network configuration (NAT, IP address, firewall, routing...)
  - Resource limits /priorization / accounting (CPU, memory, bandwidth...)
- Every container receives:
  - Isolation (processes, file systems, network, users...)

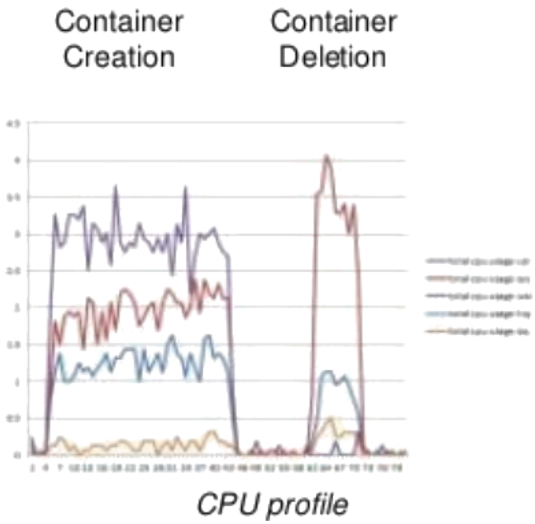
LXC containers

Performance



Performance is extremely close to bare-metal

Agility

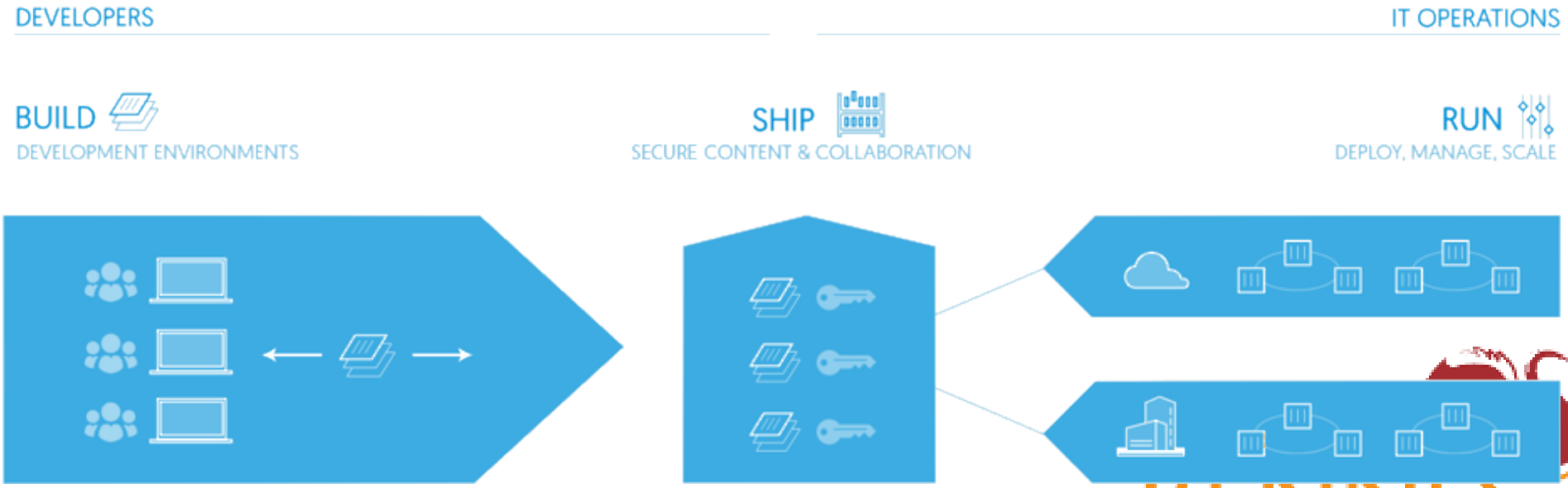


CPU profile

- Starting 150 containers w/ Apache
  - Total time: 36 s (240 ms/container)
  - Consumes about 2% of CPU
  - Memory usage: $\sim$10 MB/container
- Stopping 150 containers:
  - Total time: 9 seconds

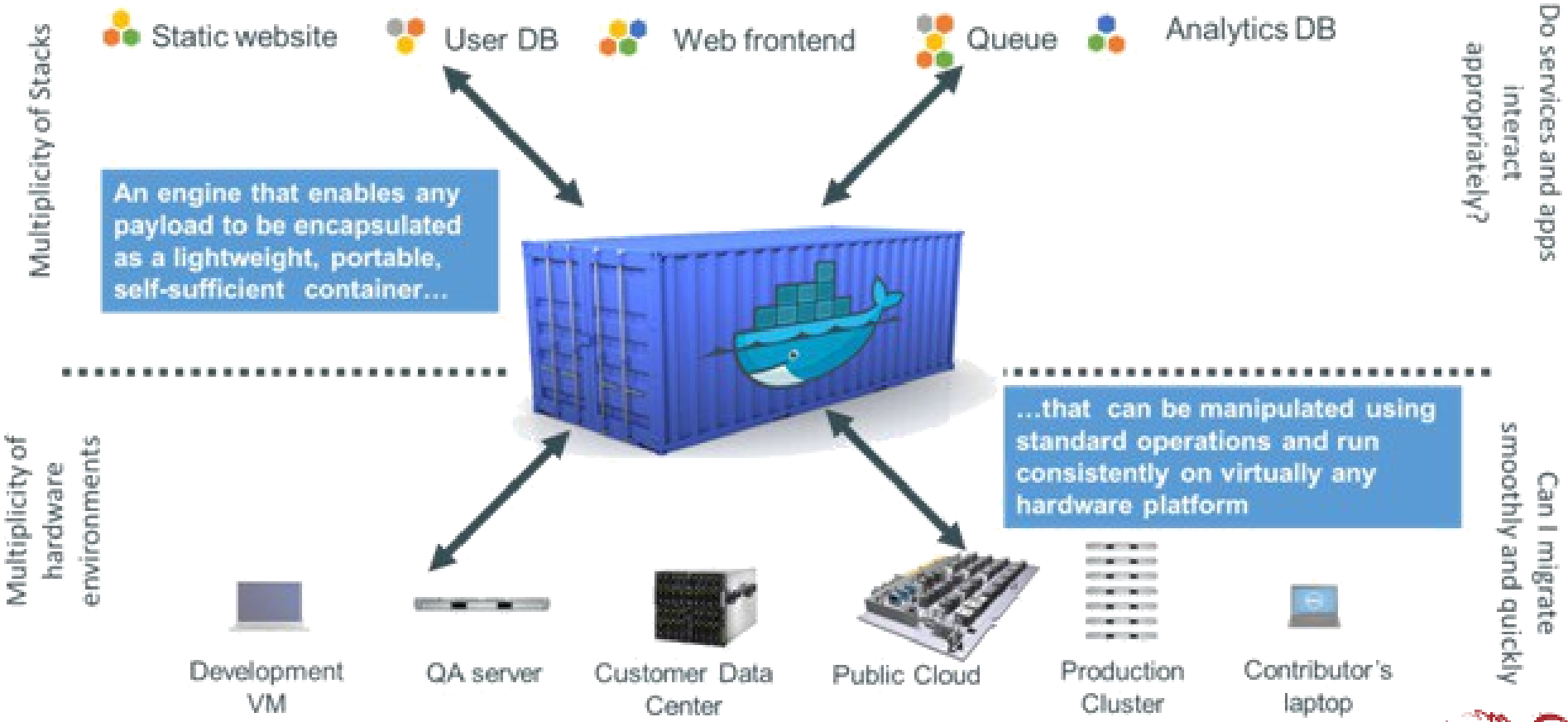
http://www.slideshare.net/BodenRussell/realizing-linux-containerslxc

- Initially a French company! ☺
- Open-source software layer which makes LXC containers easy to use



https://www.docker.com/enterprise

Docker is a shipping container system for code



https://impythonist.wordpress.com/2015/06/21/

docker-the-future-of-virtualization-for-your-django-web-development/

Typical Docker workflow

1. **Using a ready-made image:**

```
$ docker run docker/whalesay cowsay hello world
```

Image name: `docker/whalesay` — Parameters: `cowsay hello world`

2. **Customizing a ready-made image:**

Choose base image

Install additional software in the image

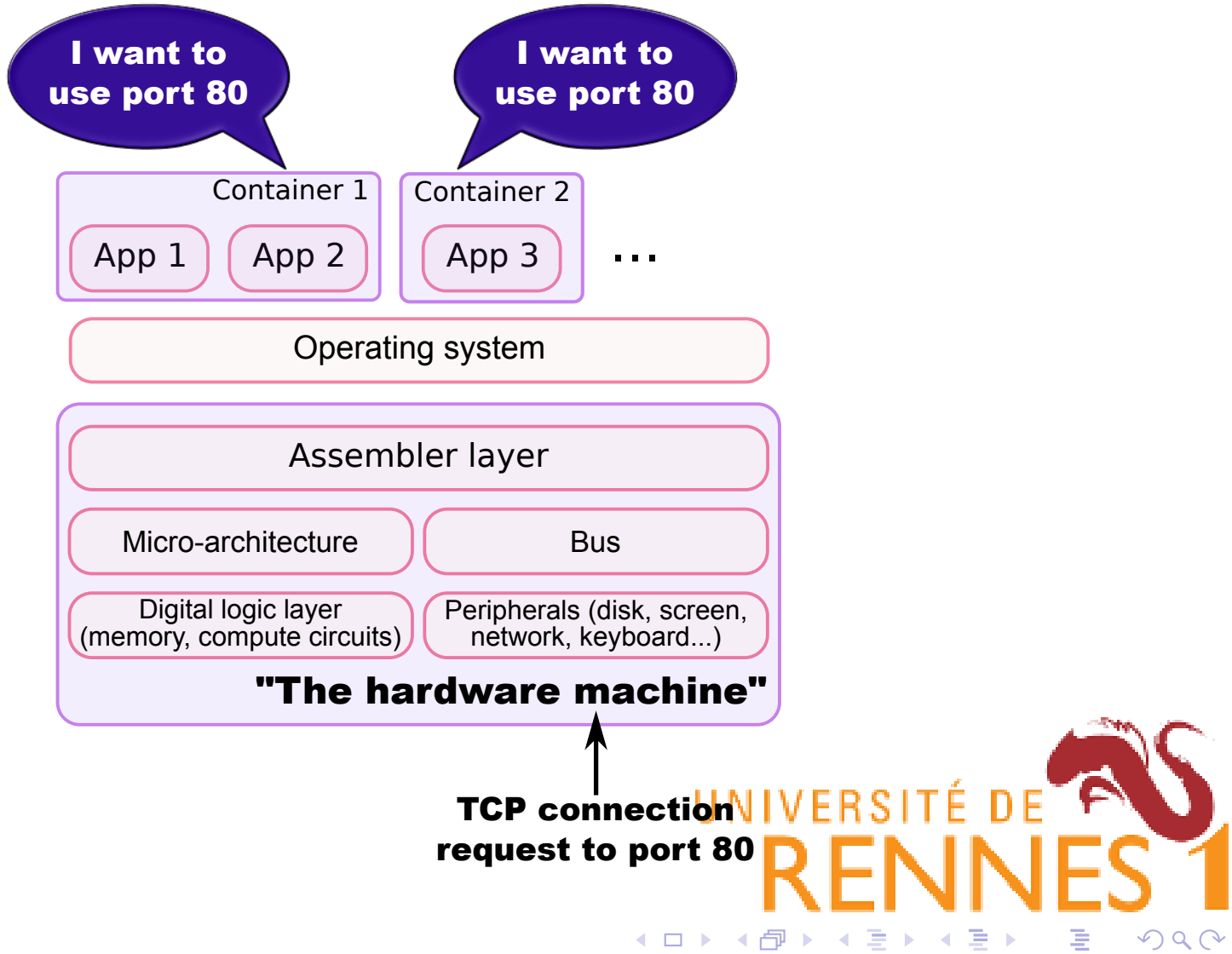
```
$ cat Dockerfile
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
$
$
$
$
$
$ docker build -t docker-whale .
 Sending build context to Docker daemon 2.048 kB
 ...snip...
 Removing intermediate container a8e6faa88df3
 Successfully built 7d9495d03763
$
```

Build a new image

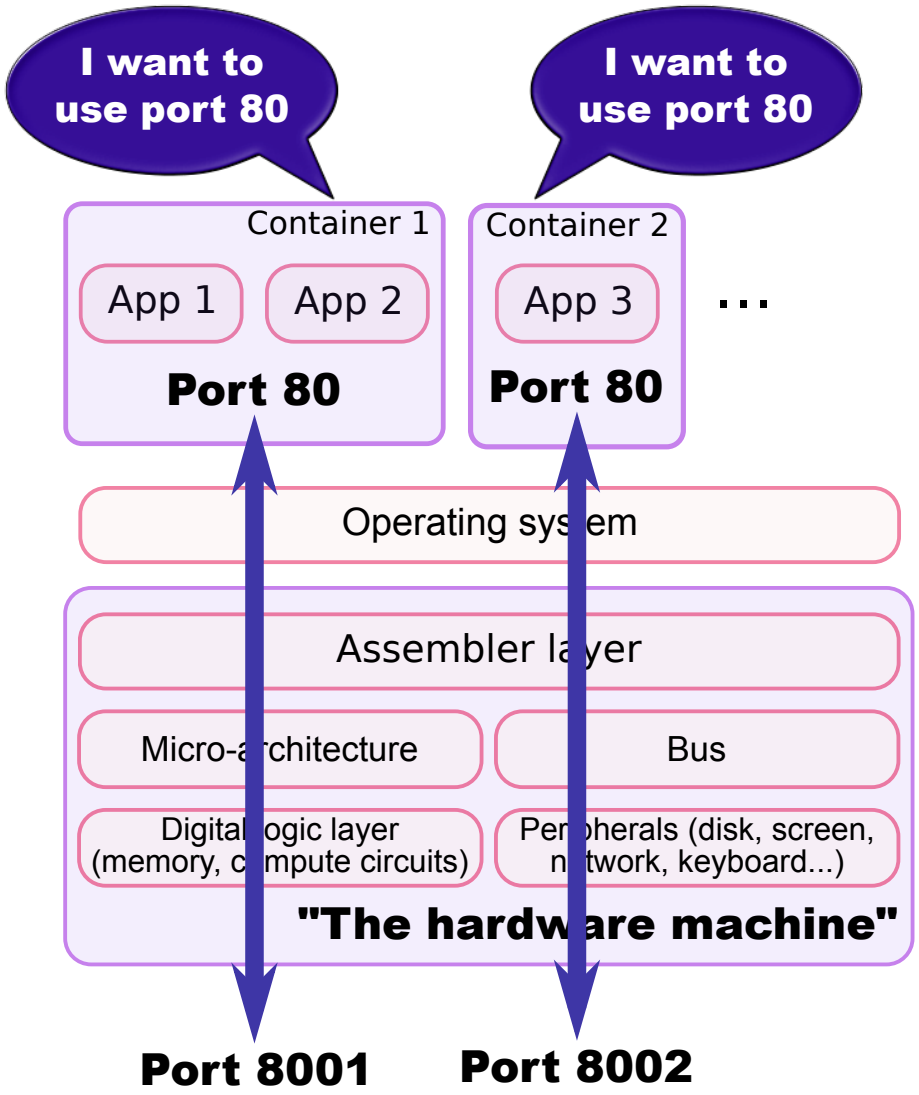
Define a startup command

Networking issues

Two containers want to run independent Web servers...



Container port mapping




```
$ docker run -p 8001:80 image1
$ docker run -p 8002:80 image2
$
```

Docker Compose

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

docker-compose.yml

- Define a complex set of containers in a single file
  - Container configurations and relationships
- Simple commands for manipulating the entire application
  - Start, stop and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service

Docker Compose

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

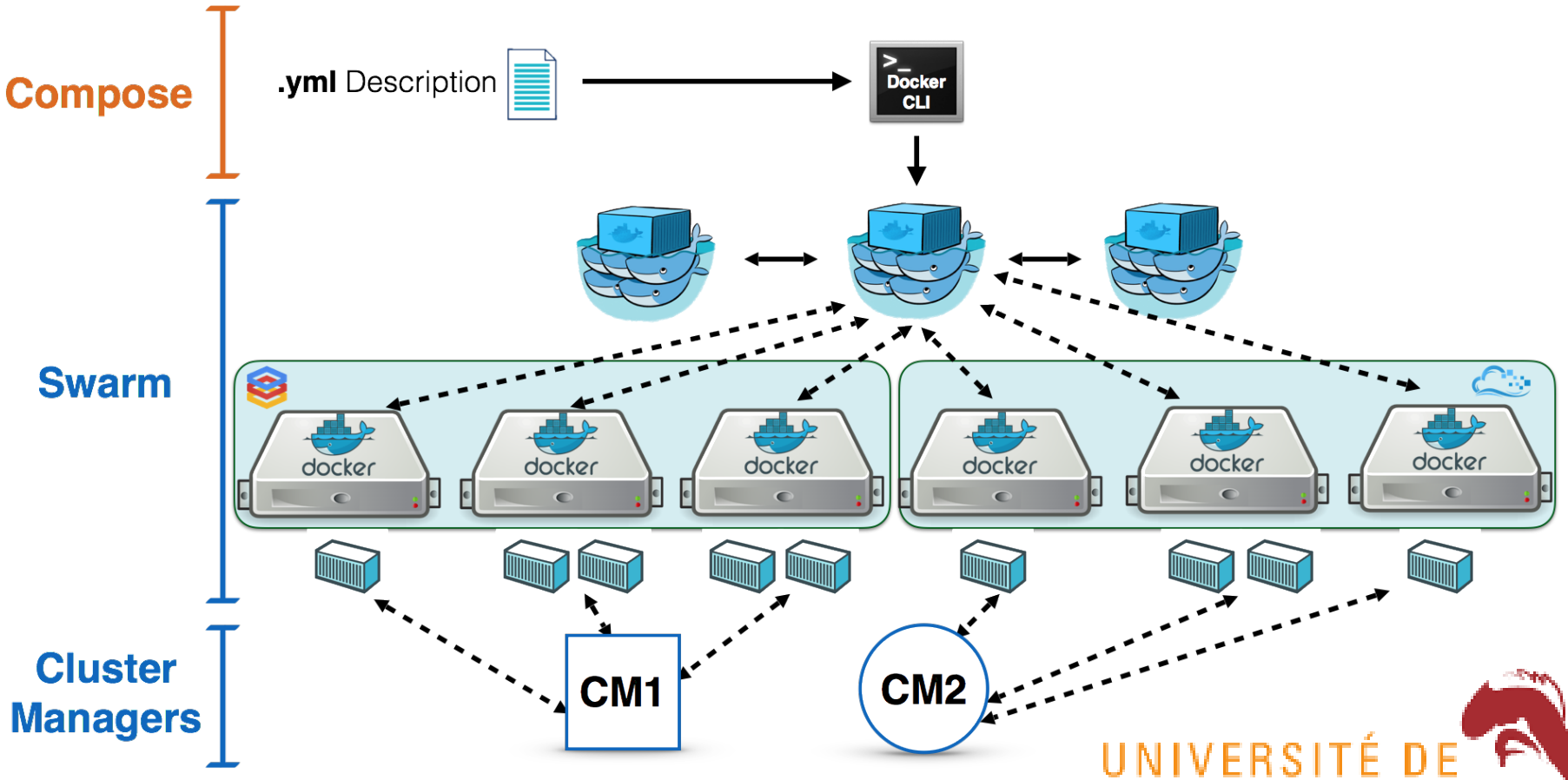
docker-compose.yml

- Define a complex set of containers in a single file
  - Container configurations and relationships
- Simple commands for manipulating the entire application
  - Start, stop and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service

```
$ docker-compose up
Pulling image redis...
Building web...
Starting composetest_redis_1...
Starting composetest_web_1...
redis_1 | [8] 02 Jan 18:43:35.576 # Server started, Redis version 2.8.3
web_1   |  * Running on http://0.0.0.0:5000/
web_1   |  * Restarting with stat
```

Docker Swarm

- Docker was initially designed for single machines
- Docker Swarm extends it to clusters of machines



https://blog.docker.com/2015/11/deploy-manage-cluster-docker-swarm/

Table of Contents

Kubernetes: Google's container orchestration system

- Google runs everything in containers:
  - Gmail, Web Search, Maps...
  - Mapreduce, batch, GFS, ...
  - Google's Cloud Platform: even VMs run in containers!

  $\Rightarrow$ They claim to launch over 2 billion containers per week

- Kubernetes extends Docker toward planetary scale
  - Container grouping, load balancing, auto-healing, scaling
  - 100% open-source

kubernetes

by Google

UNIVERSITÉ DE RENNES 1

Design principles

1. (μ)-services are more interesting than containers
   - Applications just *happen* to run in containers
2. Declarative statements are better than imperative programs
   - Explain what you want to obtain, not how to make it happen
3. Control loops
   - Observe, rectify, repeat
4. KISS
   - **K**eep **I**t **S**imple, **S**tupid!
5. Legacy-compatible
   - Requiring apps to change is a **non-starter**
6. Keep it open
   - Open-source software, standards, REST, JSON, etc.

Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

Queryable by **selectors**

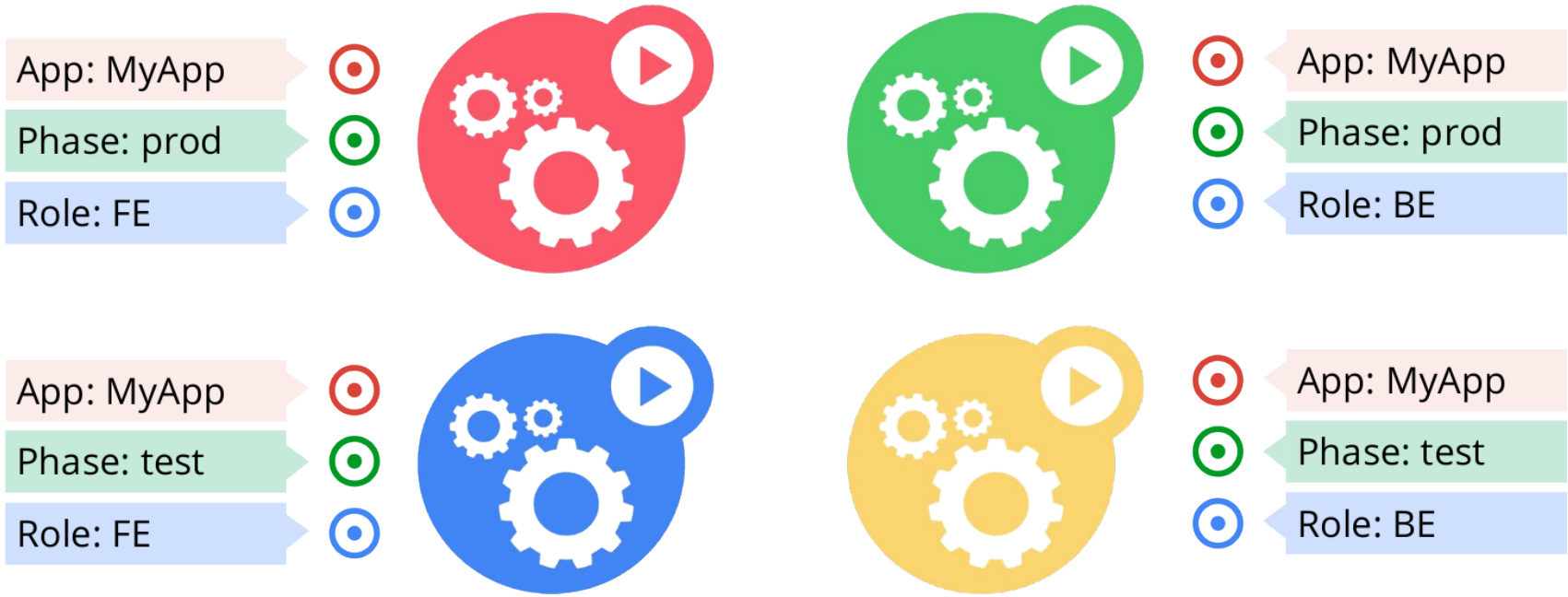
- think SQL '*select ... where ...*'

The **only** grouping mechanism

- pods under a ReplicationController
- pods in a Service
- capabilities of a node (constraints)

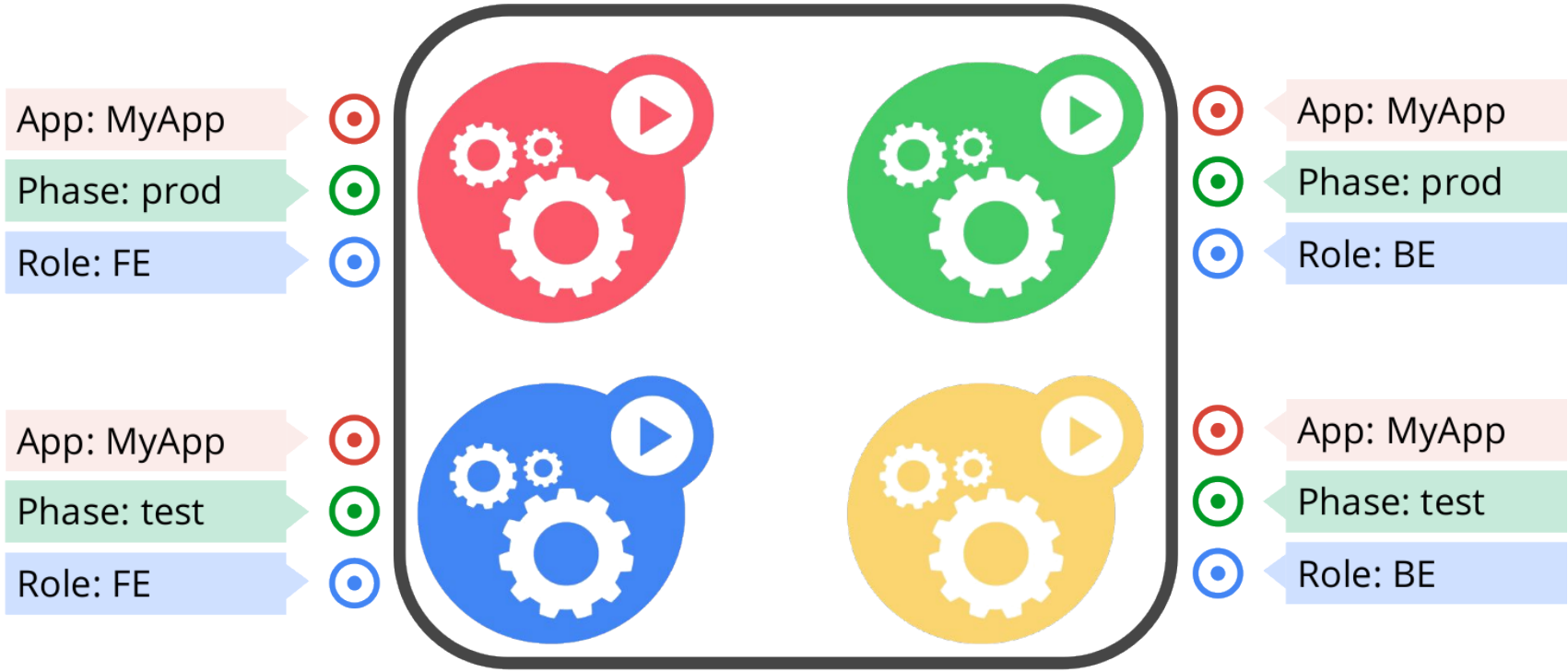
http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

Labels



http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

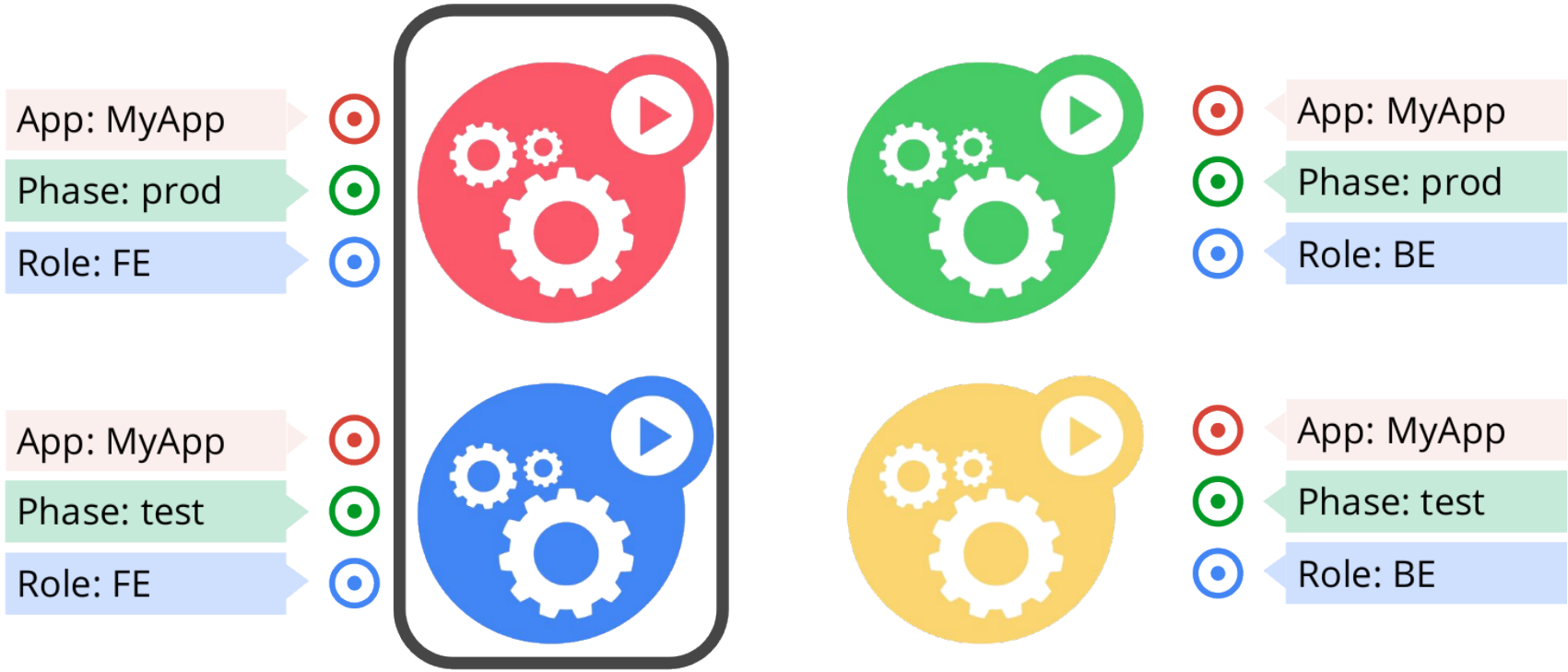
Labels



App = MyApp

http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

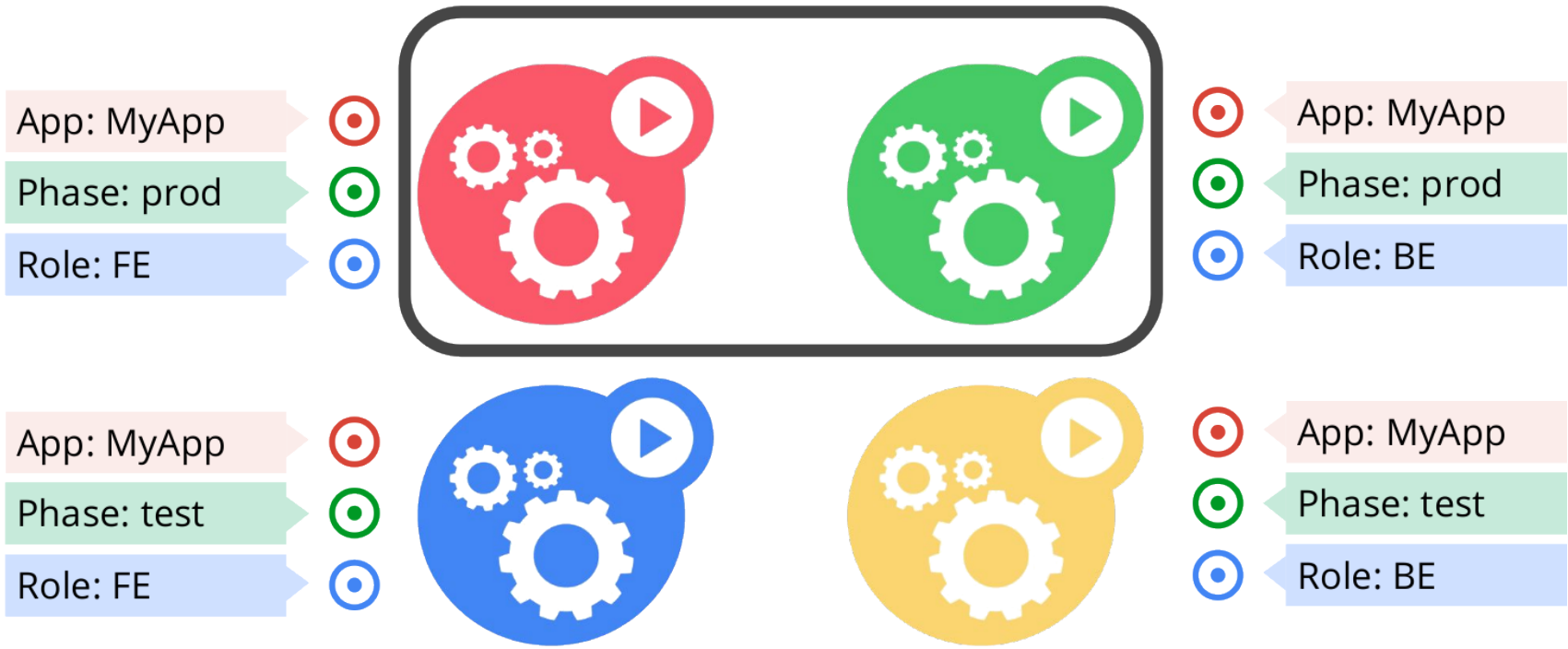
Labels



App = MyApp, Role = FE

http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

Labels



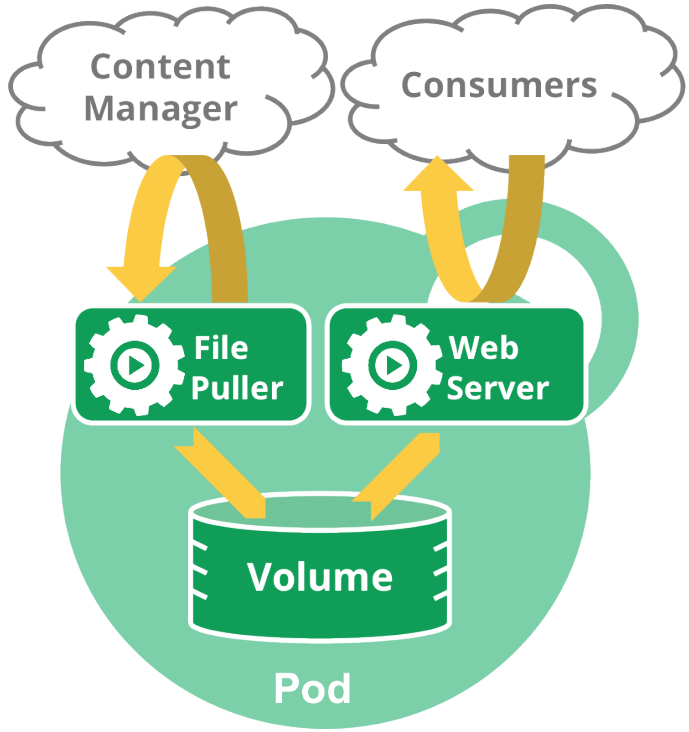
App = MyApp, Phase = prod

http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

The basic computational unit: pods

A pod is a small set of containers working together

- Example: file puller + storage volume + web server
- A pod's containers are tightly coupled
  - They are always placed together in the same server
  - If one container dies, Kubernetes kills the others
- Each pod has its own IP address
  - Containers of a pod share the same IP address
  - Different pods always have different IP addresses
  - No need to play complex games with port numbers!



http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$
```

Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$ kubectl create -f pod.yaml
pod "mysmallpod" created
$
```

Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$ kubectl create -f pod.yaml
pod "mysmallpod" created
$ kubectl get pods
NAME            READY     STATUS              RESTARTS   AGE
mysmallpod      0/2       ContainerCreating   0          8s
$
```

Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$ kubectl create -f pod.yaml
pod "mysmallpod" created
$ kubectl get pods
NAME            READY     STATUS              RESTARTS   AGE
mysmallpod      0/2       ContainerCreating   0          8s
$ kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
mysmallpod      2/2       Running   0          30s
$
```

Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$ kubectl create -f pod.yaml
pod "mysmallpod" created
$ kubectl get pods
NAME            READY     STATUS              RESTARTS   AGE
mysmallpod      0/2       ContainerCreating   0          8s
$ kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
mysmallpod      2/2       Running   0          30s
$ kubectl describe pod mysmallpod
Name:           mysmallpod
Namespace:      default
Node:           my-pc/192.168.1.37
Start Time:     Sun, 16 Oct 2016 17:07:47 +0200
Labels:         app=web
Status:         Running
IP:             10.32.0.11
(...)
$
```

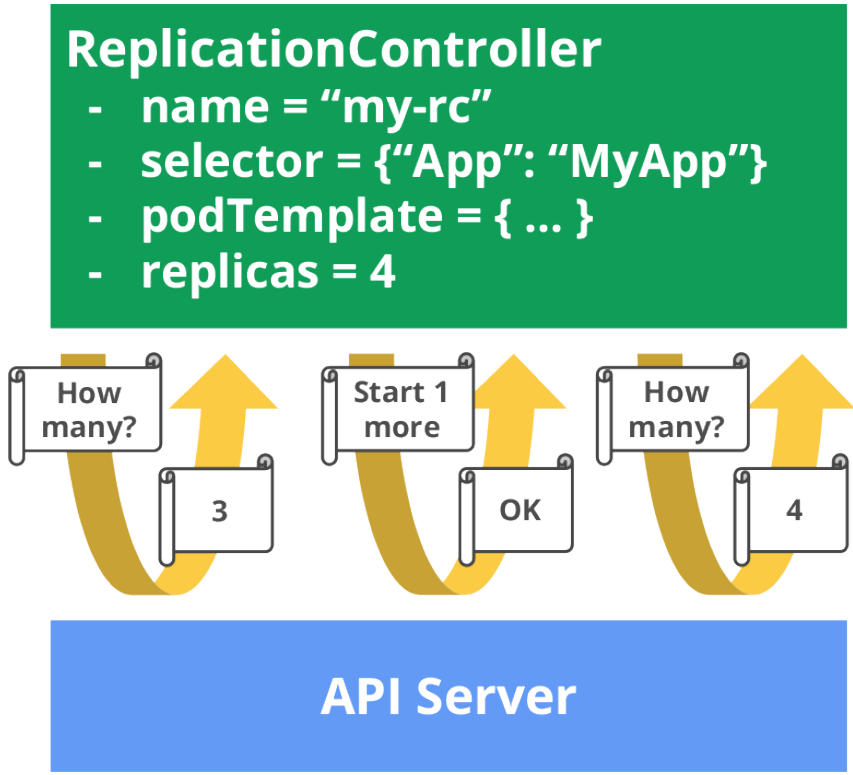
Example: a pod with two containers

```
$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysmallpod
  labels:
    app: web
spec:
  containers:
    - name: www
      image: nginx
      ports:
        - containerPort: 80
    - name: keyvaluestore
      image: redis
      ports:
        - containerPort: 6379
$ kubectl create -f pod.yaml
pod "mysmallpod" created
$ kubectl get pods
NAME            READY     STATUS              RESTARTS   AGE
mysmallpod      0/2       ContainerCreating   0          8s
$ kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
mysmallpod      2/2       Running   0          30s
$ kubectl describe pod mysmallpod
Name:           mysmallpod
Namespace:      default
Node:           my-pc/192.168.1.37
Start Time:     Sun, 16 Oct 2016 17:07:47 +0200
Labels:         app=web
Status:         Running
IP:             10.32.0.11
(...)
$ kubectl delete pod mysmallpod
pod "mysmallpod" deleted
$
```

Replication Controllers

Replication controllers ensure that pods are replicated according to your demands

- Specify how many pods you want
- Replication Controllers ensure this number is achieved
  - If too few, start new ones
  - If too many, stop some



http://www.eitdigital.eu/fileadmin/files/2015/events/symposium/Filip-Grzadkowski.pdf

Replication Controller example [1/2]

```
$ cat rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
    name: my-rc
spec:
    replicas: 4
    selector:
        app: mywebapp
    template:
      # Same as a single pod's description
      # but embedded in the RC's description
      metadata:
        name: mywebapp-pod
        labels:
          app: mywebapp
      spec:
        containers:
          - name: frontend
            image: nginx
            ports:
               - containerPort: 80
          - name: keyvaluestore
            image: redis
            ports:
               - containerPort: 6379
$
```

Replication Controller example [1/2]

```
$ cat rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
    name: my-rc
spec:
    replicas: 4
    selector:
        app: mywebapp
    template:
      # Same as a single pod's description
      # but embedded in the RC's description
      metadata:
        name: mywebapp-pod
        labels:
          app: mywebapp
      spec:
        containers:
          - name: frontend
            image: nginx
            ports:
                - containerPort: 80
          - name: keyvaluestore
            image: redis
            ports:
                - containerPort: 6379
$ kubectl create -f rc.yaml
replicationcontroller "my-rc" created
$
```

Replication Controller example [1/2]

```
$ cat rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
    name: my-rc
spec:
    replicas: 4
    selector:
        app: mywebapp
    template:
      # Same as a single pod's description
      # but embedded in the RC's description
      metadata:
        name: mywebapp-pod
        labels:
          app: mywebapp
      spec:
        containers:
          - name: frontend
            image: nginx
            ports:
                - containerPort: 80
          - name: keyvaluestore
            image: redis
            ports:
                - containerPort: 6379
$ kubectl create -f rc.yaml
replicationcontroller "my-rc" created
$ kubectl get rc
NAME      DESIRED   CURRENT   READY     AGE
my-rc     4         4         0         4s
$
```

Replication Controller example [2/2]

```
$ kubectl describe rc my-rc
Name:           my-rc
Namespace:      default
Image(s):       nginx,redis
Selector:       app=mywebapp
Labels:         app=mywebapp
Replicas:       4 current / 4 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen   LastSeen   Count   From                          Message
  ---------   --------   -----   ----                          -------
  2m          2m         1       {replication-controller }     Created pod: my-rc-p051p
  2m          2m         1       {replication-controller }     Created pod: my-rc-iuded
  2m          2m         1       {replication-controller }     Created pod: my-rc-has56
  2m          2m         1       {replication-controller }     Created pod: my-rc-m726m
$
```

Replication Controller example [2/2]

```
$ kubectl describe rc my-rc
Name:           my-rc
Namespace:      default
Image(s):       nginx,redis
Selector:       app=mywebapp
Labels:         app=mywebapp
Replicas:       4 current / 4 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen   LastSeen   Count   From                          Message
  ---------   --------   -----   ----                          -------
  2m          2m         1       {replication-controller }     Created pod: my-rc-p051p
  2m          2m         1       {replication-controller }     Created pod: my-rc-iuded
  2m          2m         1       {replication-controller }     Created pod: my-rc-has56
  2m          2m         1       {replication-controller }     Created pod: my-rc-m726m
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
my-rc-has56   2/2       Running   0          3m
my-rc-iuded   2/2       Running   0          3m
my-rc-m726m   2/2       Running   0          3m
my-rc-p051p   2/2       Running   0          3m
$
```

Replication Controller example [2/2]

```
$ kubectl describe rc my-rc
Name:           my-rc
Namespace:      default
Image(s):       nginx,redis
Selector:       app=mywebapp
Labels:         app=mywebapp
Replicas:       4 current / 4 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen   LastSeen   Count   From                          Message
  ---------   --------   -----   ----                          -------
  2m          2m         1       {replication-controller }     Created pod: my-rc-p051p
  2m          2m         1       {replication-controller }     Created pod: my-rc-iuded
  2m          2m         1       {replication-controller }     Created pod: my-rc-has56
  2m          2m         1       {replication-controller }     Created pod: my-rc-m726m
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
my-rc-has56   2/2       Running   0          3m
my-rc-iuded   2/2       Running   0          3m
my-rc-m726m   2/2       Running   0          3m
my-rc-p051p   2/2       Running   0          3m
$ kubectl scale rc my-rc --replicas=2
replicationcontroller "my-rc" scaled
$
```

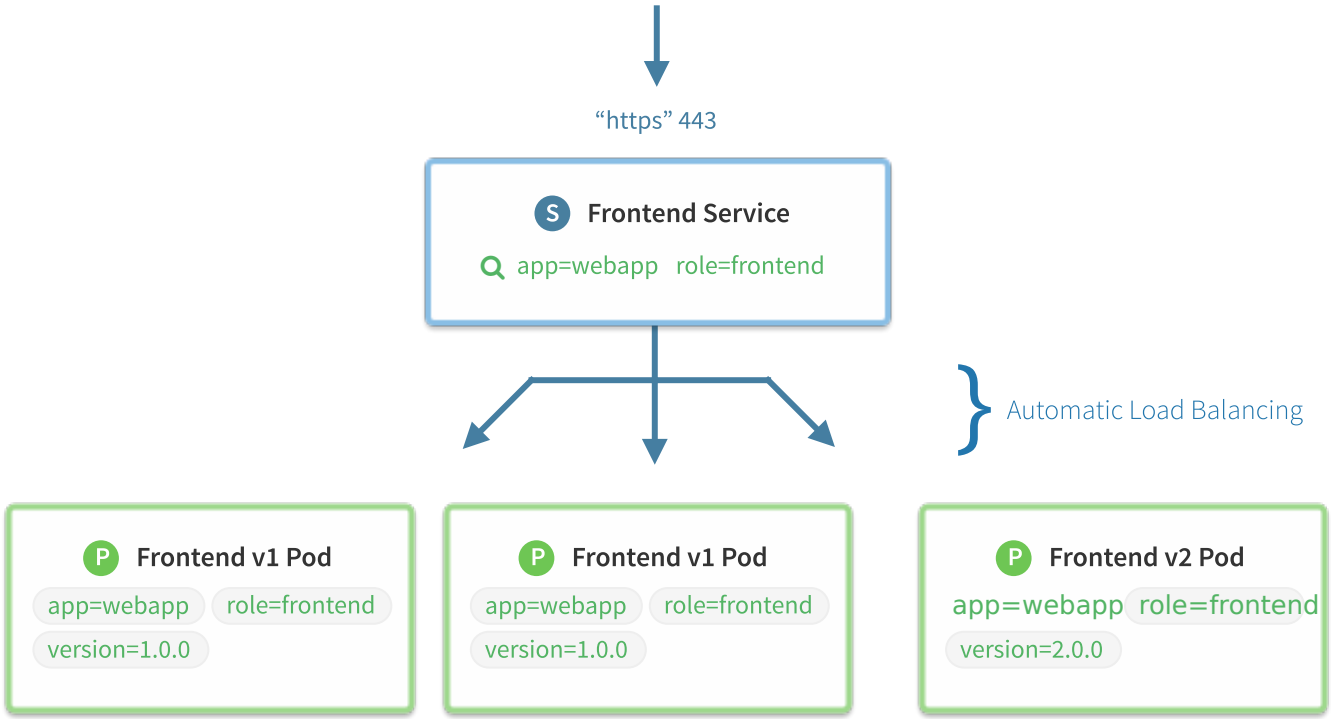
Replication Controller example [2/2]

```
$ kubectl describe rc my-rc
Name:           my-rc
Namespace:      default
Image(s):       nginx,redis
Selector:       app=mywebapp
Labels:         app=mywebapp
Replicas:       4 current / 4 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen   LastSeen   Count   From                          Message
  ---------   --------   -----   ----                          -------
  2m          2m         1       {replication-controller }     Created pod: my-rc-p051p
  2m          2m         1       {replication-controller }     Created pod: my-rc-iuded
  2m          2m         1       {replication-controller }     Created pod: my-rc-has56
  2m          2m         1       {replication-controller }     Created pod: my-rc-m726m
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
my-rc-has56   2/2       Running   0          3m
my-rc-iuded   2/2       Running   0          3m
my-rc-m726m   2/2       Running   0          3m
my-rc-p051p   2/2       Running   0          3m
$ kubectl scale rc my-rc --replicas=2
replicationcontroller "my-rc" scaled
$ kubectl get rc
NAME      DESIRED   CURRENT   READY     AGE
my-rc     2         2         2         9m
$
```

Replication Controller example [2/2]

```
$ kubectl describe rc my-rc
Name:           my-rc
Namespace:      default
Image(s):       nginx,redis
Selector:       app=mywebapp
Labels:         app=mywebapp
Replicas:       4 current / 4 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen   LastSeen   Count   From                          Message
  ---------   --------   -----   ----                          -------
  2m          2m         1       {replication-controller }     Created pod: my-rc-p051p
  2m          2m         1       {replication-controller }     Created pod: my-rc-iuded
  2m          2m         1       {replication-controller }     Created pod: my-rc-has56
  2m          2m         1       {replication-controller }     Created pod: my-rc-m726m
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
my-rc-has56   2/2       Running   0          3m
my-rc-iuded   2/2       Running   0          3m
my-rc-m726m   2/2       Running   0          3m
my-rc-p051p   2/2       Running   0          3m
$ kubectl scale rc my-rc --replicas=2
replicationcontroller "my-rc" scaled
$ kubectl get rc
NAME      DESIRED   CURRENT   READY     AGE
my-rc     2         2         2         9m
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
my-rc-has56   2/2       Running   0          8m
my-rc-m726m   2/2       Running   0          8m
$
```

Services



A service is a load-balanced set of pods

- Examples:
  - Several replicated pods (e.g., controlled by a Replication Controller)
  - Slightly different pods (e.g., for testing a new service version)
- A service has a stable name and IP address
  - Regardless of the set of pods it contains

https://coreos.com/kubernetes/docs/latest/services.html

Service example

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$
```

Service example

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$ kubectl create -f service.yaml
service "my-service" created
$
```

Service example

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$ kubectl create -f service.yaml
service "my-service" created
$ kubectl get services
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   100.64.0.1       <none>        443/TCP   1h
my-service   100.67.142.254   <none>        80/TCP    6s
$
```

Service example

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$ kubectl create -f service.yaml
service "my-service" created
$ kubectl get services
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   100.64.0.1       <none>        443/TCP    1h
my-service   100.67.142.254   <none>        80/TCP     6s
$ kubectl describe service my-service
Name:                   my-service
Namespace:              default
Labels:                 <none>
Selector:               app=mywebapp
Type:                   ClusterIP
IP:                     100.67.142.254
Port:                   <unset> 80/TCP
Endpoints:              10.32.0.11:80,10.32.0.12:80
Session Affinity:       None
No events.
$
```

Service example

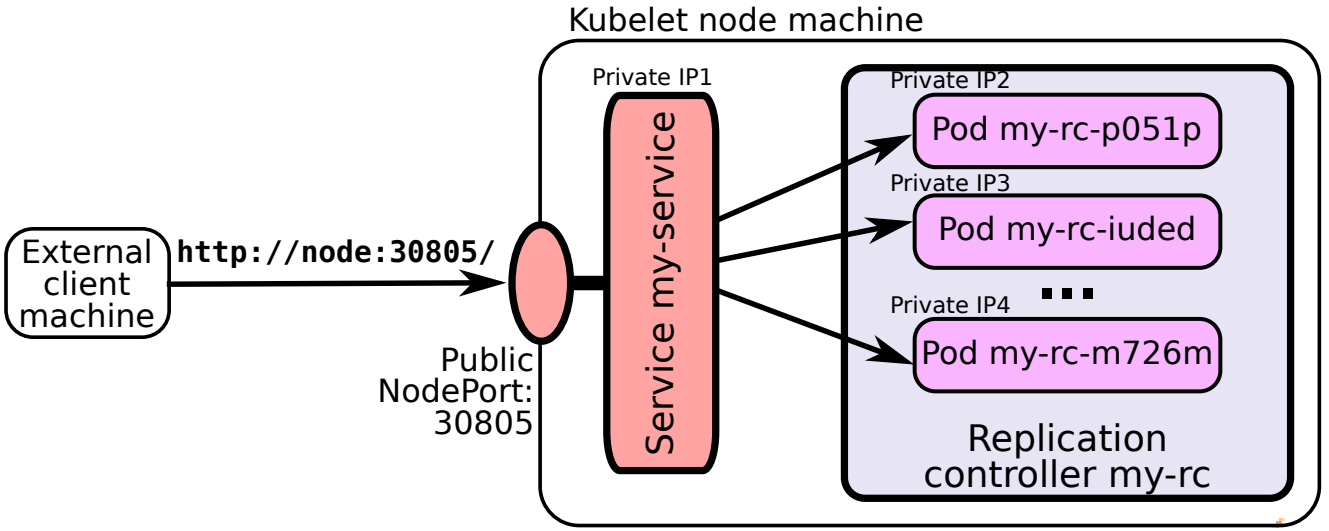
```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$ kubectl create -f service.yaml
service "my-service" created
$ kubectl get services
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   100.64.0.1       <none>        443/TCP    1h
my-service   100.67.142.254   <none>        80/TCP     6s
$ kubectl describe service my-service
Name:                   my-service
Namespace:              default
Labels:                 <none>
Selector:               app=mywebapp
Type:                   ClusterIP
IP:                     100.67.142.254
Port:                   <unset> 80/TCP
Endpoints:              10.32.0.11:80,10.32.0.12:80
Session Affinity:       None
No events.
$ kubectl scale rc my-rc --replicas=5
replicationcontroller "my-rc" scaled
$
```

Service example

```
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
$ kubectl create -f service.yaml
service "my-service" created
$ kubectl get services
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   100.64.0.1       <none>        443/TCP    1h
my-service   100.67.142.254   <none>        80/TCP     6s
$ kubectl describe service my-service
Name:                   my-service
Namespace:              default
Labels:                 <none>
Selector:               app=mywebapp
Type:                   ClusterIP
IP:                     100.67.142.254
Port:                   <unset> 80/TCP
Endpoints:              10.32.0.11:80,10.32.0.12:80
Session Affinity:       None
No events.
$ kubectl scale rc my-rc --replicas=5
replicationcontroller "my-rc" scaled
$ kubectl describe service my-service
Name:                   my-service
(...)
Endpoints:              10.32.0.11:80,10.32.0.12:80,10.32.0.13:80 + 2 more...
(...)
$
```

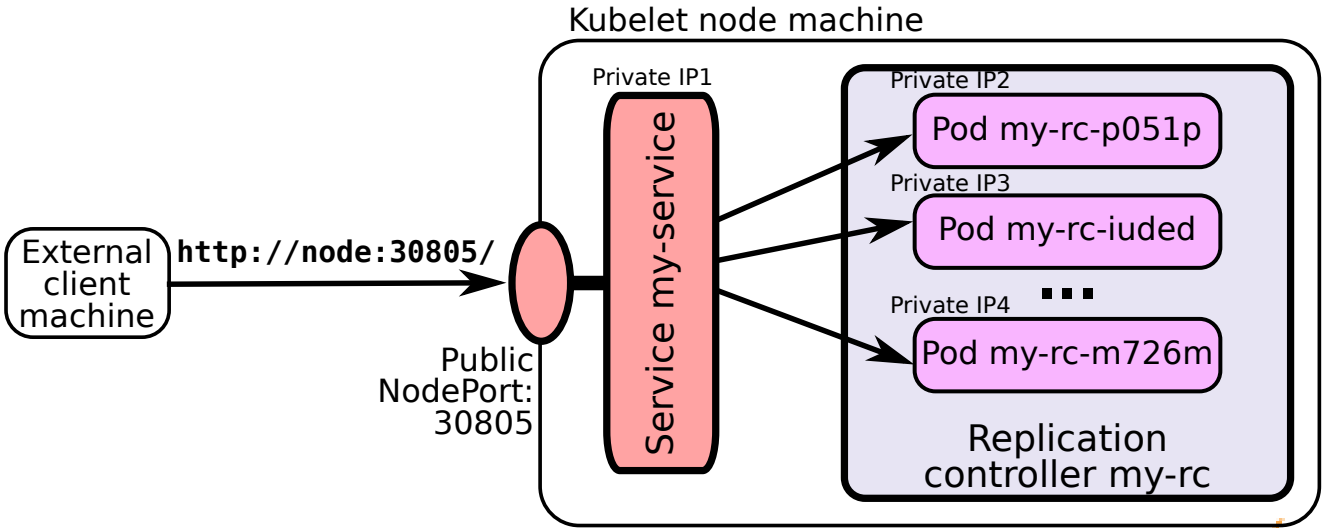
Exposing services to the outside world

- All pods' and services' IP addresses are private addresses
  - They allow pod-to-pod and service-to-pod communication
  - But they cannot be accessed from the outside world!
- To expose a service to the outside world:
  - Add "`type: NodePort`" in the service description
  - Kubernetes will assign a port number on the local machine which routes traffic to the service
  - External users can access the service at machine.name:NodePort



Exposing services to the outside world

- All pods' and services' IP addresses are private addresses
  - They allow pod-to-pod and service-to-pod communication
  - But they cannot be accessed from the outside world!
- To expose a service to the outside world:
  - Add "`type: NodePort`" in the service description
  - Kubernetes will assign a port number on the local machine which routes traffic to the service
  - External users can access the service at machine.name:NodePort



- Note: it is also possible to assign a public IP address to the service
  - But this requires SDN network support...

Exposing services to the outside world

```
$ cat service-public.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
    type: NodePort
$
```

Exposing services to the outside world

```
$ cat service-public.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
    type: NodePort
$ kubectl create -f service-public.yaml
service "my-service" created
$
```

Exposing services to the outside world

```
$ cat service-public.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
    type: NodePort
$ kubectl create -f service-public.yaml
service "my-service" created
$ kubectl describe service my-service
Name:                   my-service
Namespace:              default
Labels:                 <none>
Selector:               app=mywebapp
Type:                   NodePort
IP:                     100.74.93.20
Port:                   <unset> 8 0/TCP
NodePort:               <unset>  30805/TCP
Endpoints:              10.32.0.33:80,10.32.0.34:80,10.32.0.35:80 + 1 more...
Session Affinity:       None
No events.
$
```

Exposing services to the outside world

```
$ cat service-public.yaml
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: mywebapp
    ports:
        - port: 80
          targetPort: 80
    type: NodePort
$ kubectl create -f service-public.yaml
service "my-service" created
$ kubectl describe service my-service
Name:                   my-service
Namespace:              default
Labels:                 <none>
Selector:               app=mywebapp
Type:                   NodePort
IP:                     100.74.93.20
Port:                   <unset> 8 0/TCP
NodePort:               <unset>  30805/TCP
Endpoints:              10.32.0.33:80,10.32.0.34:80,10.32.0.35:80 + 1 more...
Session Affinity:       None
No events.
$ wget -nv http://my.machine.com:30805/
2016-10-17 11:44:58 URL:http://my.machine.com:30805/ 612/612 -> "index.html" 1
$
```

Bibliography

- "A conversation with Werner Vogels." ACM Queue, June 2006. `http://queue.acm.org/detail.cfm?id=1142065`
- "The Hidden Dividends of Microservices (microservices aren't for every company, and the journey isn't easy)." ACM Queue, June 2016. `http://queue.acm.org/detail.cfm?id=2956643`