Scale out, a mówili, że się nie da

Arkadiusz Borek
LJUG

JDD 2016

@arekborek

arek.borek@gmail.com

github.com/arekborek

arekborek.blogspot.com/

Microservices?

You probably heard a lot already!

AGENDA

So many services

Deploy

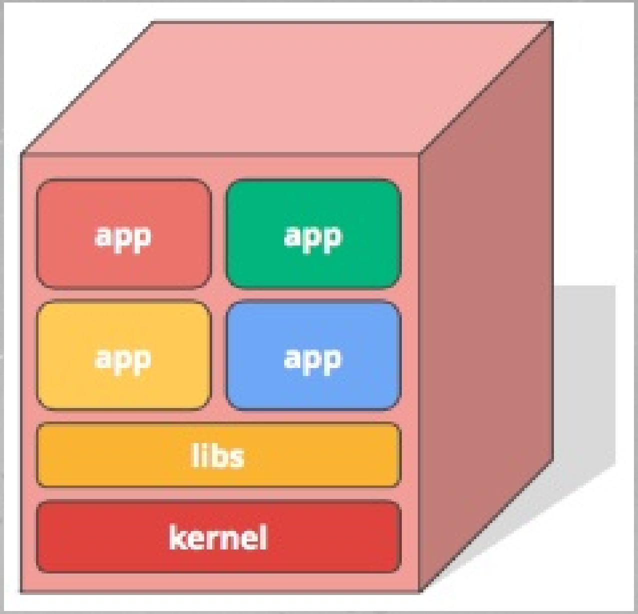
Manage

Discovery

How?

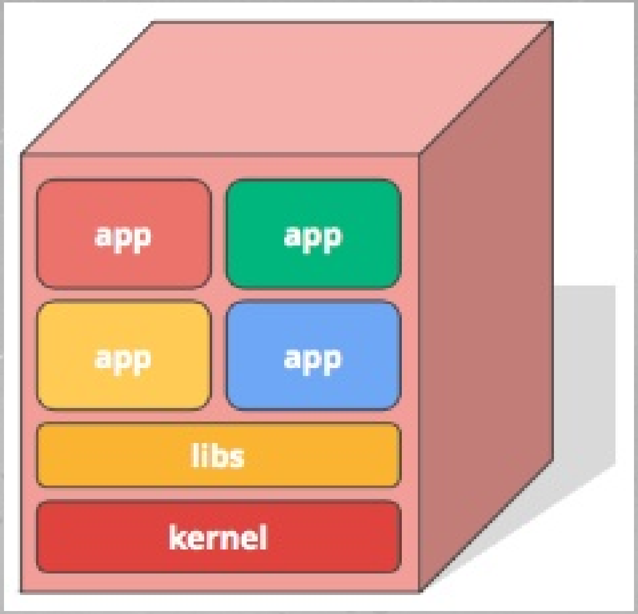
Old way: shared machines

- No isolation
- No namespacing
- Common libs
- High coupled apps and OS



Old way: shared app servers

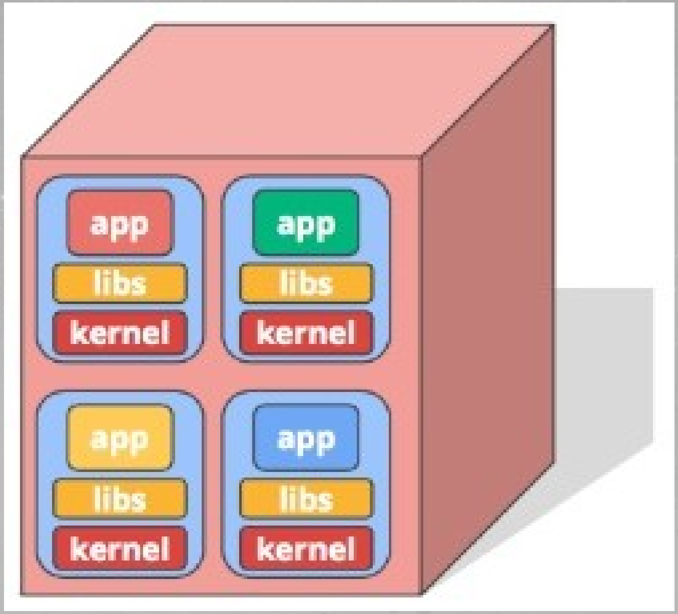
- No isolation
- No namespacing
- Common libs



source@kubernetes.io

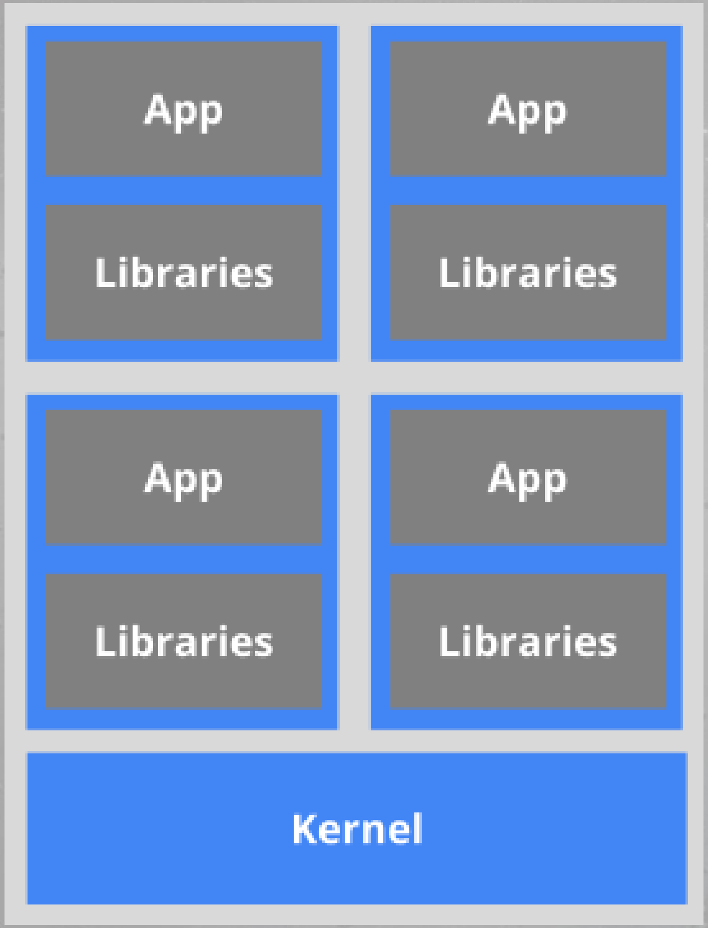
Old way: many virtual machines

- Some isolation
- Expensive and inefficient
- Still highly coupled
  to the guest OS
- Hard to manage



source@kubernetes.io

New way: containers



source@kubernetes.io

Containers: Process isolation

- CPU (PIDs)
- Memory
- Network interfaces
- Filesystem

Containers: Security

- Chroots determines what parts of the filesystem a use can see
- Control - CPU, Memory, IO via cgroups
- Limits what a user can do: mount, kill, chown
- No access to host !!!

Containers: Other features

- New approach to packaging
- Lightweight
- Runs on any machine
  Physical, virtual, cloud
- Write once – run everywhere!
  Development, Test, Staging, Production

docker

Let’s Containerize

Containerize Option #1

Docker file

Containerize Option #2
Maven plug-in made by spotify

Containerize Option #3
Docker hub / GitHub

Example of docker

Docker file

/hellonode

- Dockerfile
- server.js

server.js

```
var http = require('http');
var handleRequest = function(request, response) {
  console.log('Received request for URL: ' + request.url);
  response.writeHead(200);
  response.end('Hello World (' + process.env.HOSTNAME + ':V1) !');
};
var www = http.createServer(handleRequest);
www.listen(8080);
```

Dockerfile

```
FROM node:4.4
EXPOSE 8080
COPY server.js .
CMD node server.js
```

Build Docker Container

```
docker build \
-t gcr.io/$PROJECT_ID/hello-jdd:v1 \
.
```

Run Docker Container

```
docker run \
-d \
-p 8080:8080 \
--name hello-jdd \
gcr.io/$PROJECT_ID/hello-jdd:v1
```

Test

```
for ((;;)) do
  curl http://localhost:8080;
  printf '\n';
  sleep 1;
done
```

What next ??

- What if there are more containers?
- Scheduling – where should my containers run ?
- Lifecycle and health: keep my containers running
- Discovery: where are my container now ?
- Monitoring: What’s happening with my container ?
- Scaling: Making job bigger or smaller

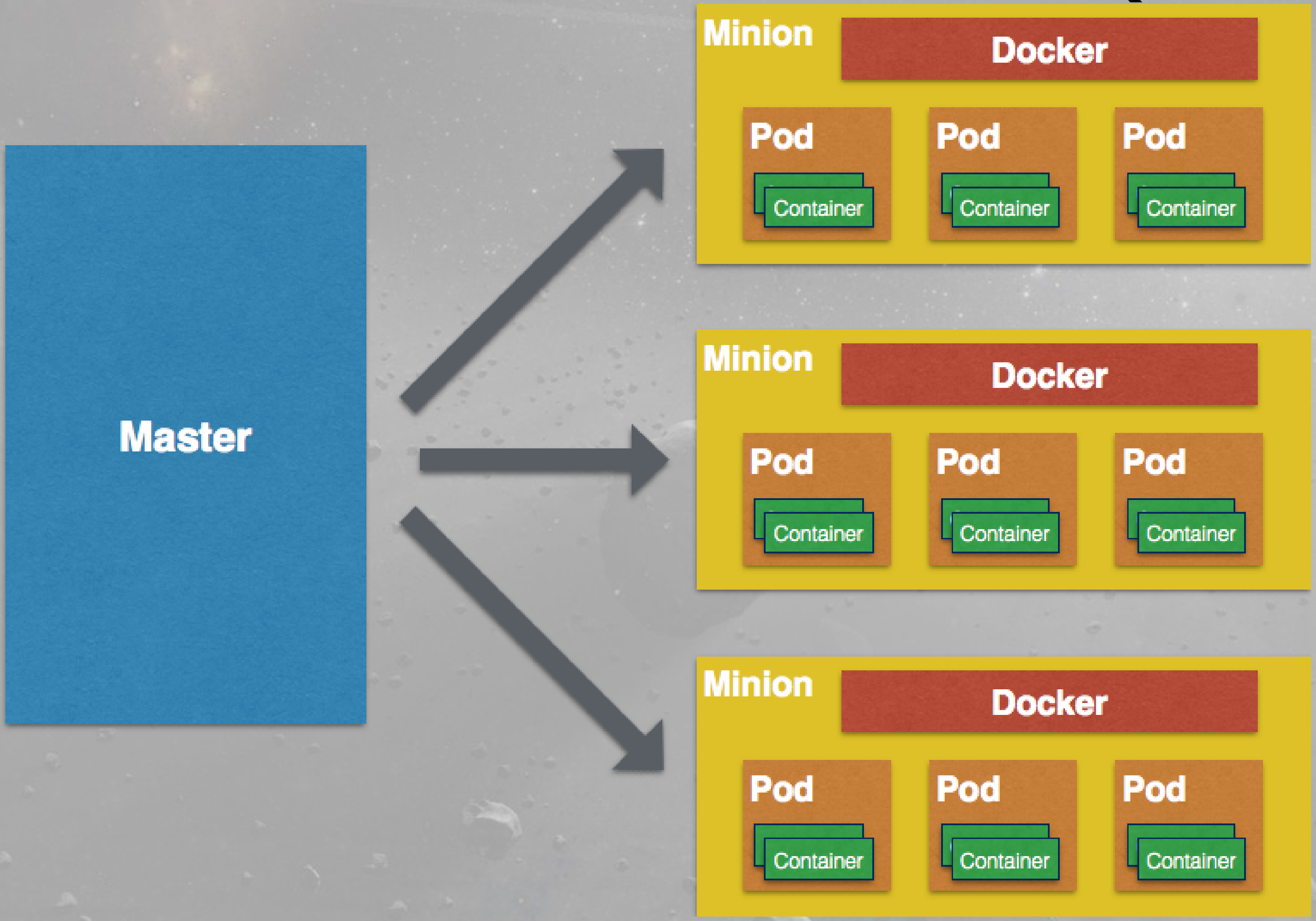
How to handle it ?

Let’s kubectrl-ing world!

Enter Kubernetes (k8s)

- Greek for “Helmsman”, also the root of the word “Governor”
- Container orchestrator, run containers
- Support multiple cloud and bare-metal environments
- Inspired and informed by Google’s experiences and internal systems
- Open source, written in Go

Enter Kubernetes (k8s)



Enter Kubernetes (k8s)

- Kubernetes master – order others
- Kublet – ask how many resources you have if enough run machine
- Distributed process scheduler
- Cluster of machines as one
- All your machines via kubernetes can be treats as one single machine

k8s basis concepts

k8s basis concepts

- **Pod**

Pod

- Group of containers
- One atomic resource that can be managed by Kubernetes
- Live and die together
- Shared network interface

k8s basis concepts

- Pod
- **Pod networking**

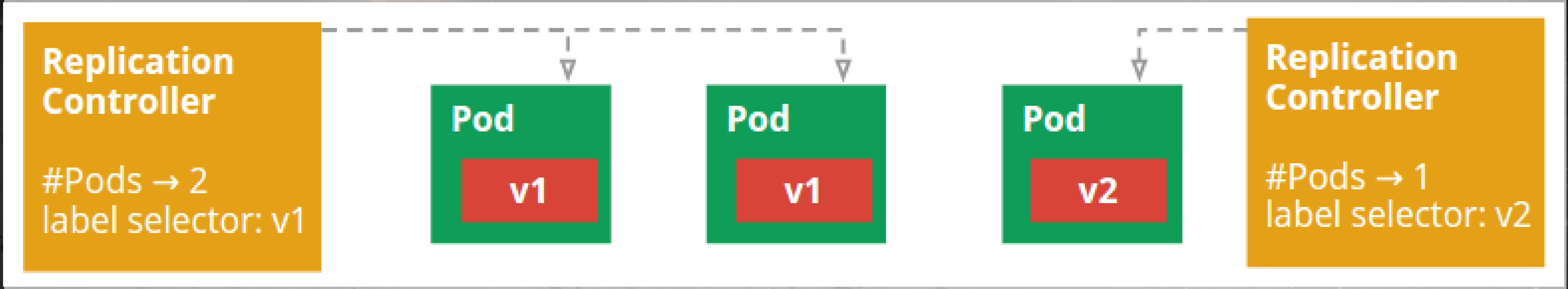
Pod networking

- Pod IPs routable, docker default is private IP
- Pods can reach each other without NAT, even across nodes
- No brokering of port numbers
- This is fundamental requirement, several SDN solutions

k8s basis concepts

- Pod
- Pod networking
- **Replication Controllers**

Replication Controller



- Allow start pods multiple times
- Check state if one of you pod died, RC run next instance of container for you

k8s basis concepts

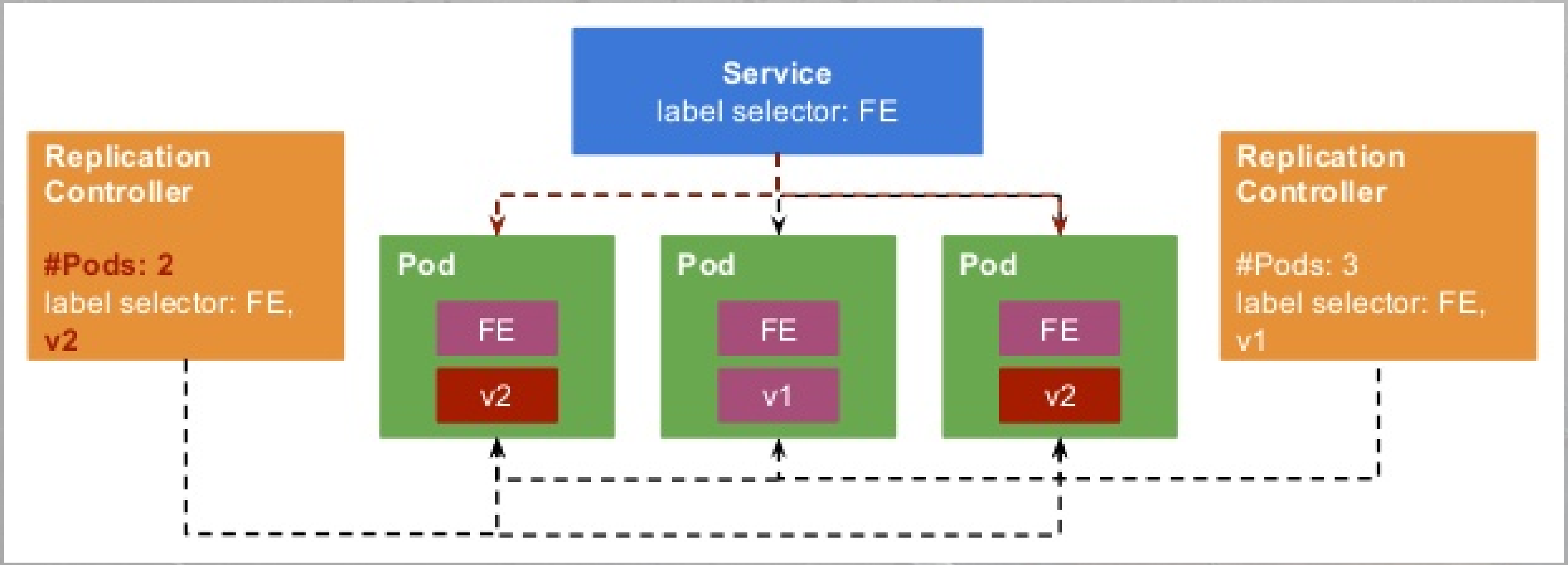
- Pod
- Pod networking
- Replication Controllers
- **Service**

k8s basis concepts

- Pod
- Pod networking
- Replication Controllers
- Service
- **Labels & Selectors**

Labels & Selectors

- Label anything
- Name-value pair
- Make your own



Example of kube

Push image

```
gcloud docker push \
gcr.io/$PROJECT_ID/hello-jdd:v1
```

Replication Controller

```
kubectl create -f rc-v1.yaml
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: hello-jdd-v1
spec:
  replicas: 1
  template:
    metadata:
      name: hello-jdd-v1
      labels:
        app: hello-jdd-v1
        env: jdd
        tier: backend
        version: v1
        visualize: "true"
    spec:
      containers:
      - name: hello-jdd
        image: "gcr.io/strong-art-145220/hello-jdd:v1"
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
```

Service

Allow external traffic

kubectl create -f service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    env: jdd
    visualize: "true"
  name: hello-jdd
spec:
  ports:
    - port: 8080 #The port that this service should serve on.
  selector: #Label keys and values connect RC and Service
    env: jdd
    tier: backend
  type: LoadBalancer
```

Test :)

```
for ((;;)) do \
curl http://localhost:8080; \
printf '\n'; \
sleep 1; \
done
```

Scale out

```
kubectl scale rc hello-jdd-v1 --replicas=2
```

Roll out an upgrade

```
gcloud docker push \
gcr.io/$PROJECT_ID/hello-jdd:v2


kubectl create -f rc-v2.yaml
kubectl scale rc hello-jdd-v1 –replicas=1
kubectl scale rc hello-jdd-v2 –replicas=2
kubectl scale rc hello-jdd-v1 --replicas=0
```

--rollback

```
kubectl scale rc hello-jdd-v1 --replicas=1
kubectl scale rc hello-jdd-v2 --replicas=1
kubectl scale rc hello-jdd-v1 --replicas=2
kubectl scale rc hello-jdd-v2 --replicas=0
```

Some useful cmds

```
kubectl get rc
kubectl get pods
kubectl logs <POD_NAME>
kubectl cluster-info
kubectl config view
kubectl describe rc <RC_NAME>
kubectl get -o json pod
```

Some useful ...

Staging vs. production
Use label – deploy in the same infrastructure

Service discovery
Kubernetes API Or… DNS Lookups!

Use Docker Machine
In the cloud – faster network to download images

Some useful ...

Don’t Log to Container Filesystem!
Log to a volume… docker -v /tmp/log:/log Or
Send it elsewhere!

Clean up disk spaces
Every image, layer and even container litters

Use Spring Profile
One container – run on Docker Machine,
Kubernetes, and App Engine!

THE END !!