Cloud Computing, Part 1

Distributed Systems, 3rd Semester, BSc

Christian Marius Lillelund
cl@eng.au.dk

Section of Electrical and Computer Engineering
Department of Engineering
Aarhus University

Revised on November 12, 2020

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

What is cloud computing?

The cloud

- A metaphor for the Internet. Something that is remote.

Cloud computing

- The delivery of online computing services
- Most often, these include servers, storage, databases software and analytics
- Management is done by third-party
- Services are often at an enterprise-level

Benefits

- Lower costs
- Accessibility
- Productivity
- Scalability
- Updates

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Cloud computing: Background

Origin

- Around 1970, the concept of virtual machine's (VMs) was invented
- In 1999, Salesforce.com delivered applications to users using a simple website
- In 2002, Amazon provided the first cloud service
- In 2009, Google Apps saw the light of day
- In 2009, Microsoft launched Azure

Most known providers

- Google Cloud (Search, Gmail)
- Amazon's AWS (Online marketplace)
- Microsoft Azure (OneDrive)

Deployment models

- Private cloud
- Public cloud (e.g. Amazon AWS, Google Cloud)

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Monolithic vs. microservices

Monolithic

- Everything in one place
- Tightly coupled, runs as a single service
- Developed and scaled as one
- Hard to maintain

SoA-architecture

- Service-oriented architecture (SOA)
- Splits software application in smaller units
- Units communicate over network, but functions separately

What microservices are

- Modern version of the SoA-style
- Small, independent services that do one thing
- Highly maintainable, testable, loosely coupled
- "Instead of having one machine build a whole car, get multiple factories to work at the same time.

Monolitic vs. microservices

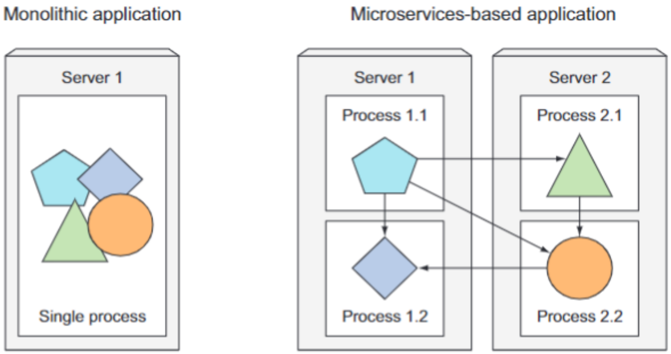


Figure: Fig. by courtesy of Marko Luksa[1]

Monolitic vs. microservices

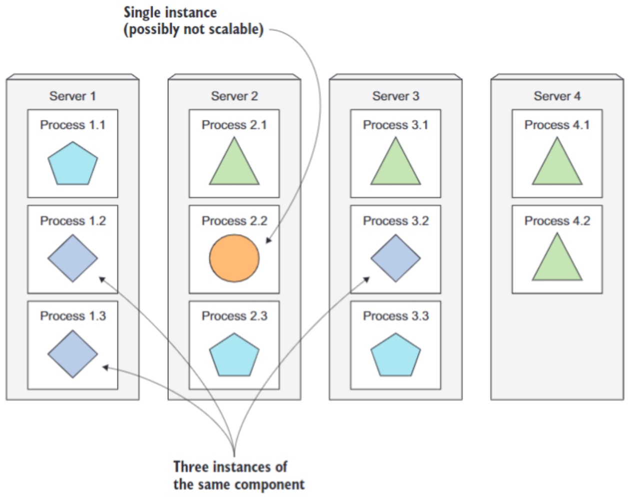


Figure: Fig. by courtesy of Marko Luksa[1]

Monolitic vs. microservices

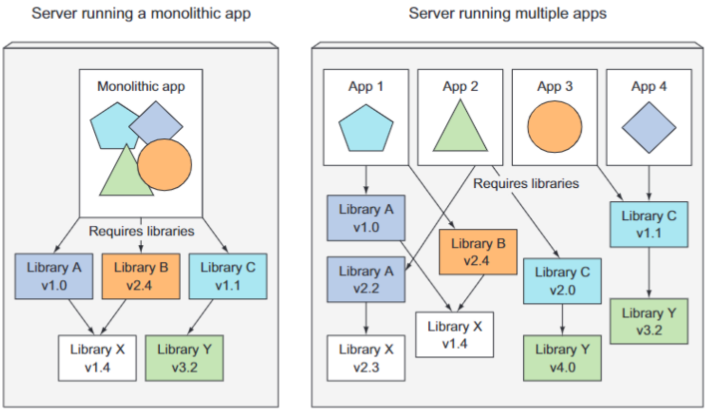


Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Linux containers

Motivation

- We can't give every software component its own VM
- VM's are manual. We need automation
- Changes once place should not impact others

What Linux containers are

- Containers are isolated software environments
- Application and dependencies bundled inside
- A lightweight version of VM's

A normal VM vs. containers

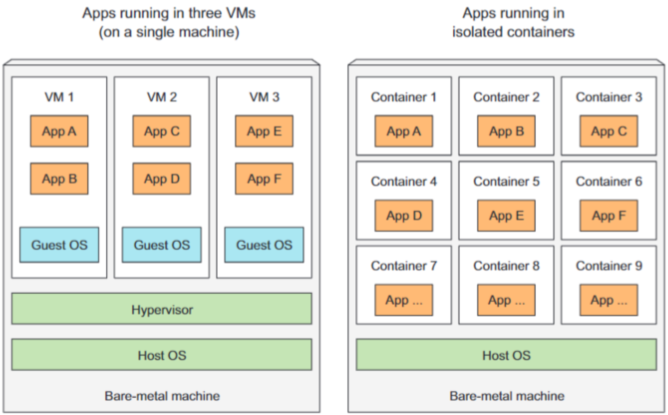


Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Docker

Motivation

- We need suitable tooling to create containers
- It should be automated, predictable and fast
- It should run everywhere, be modular and scale well

What Docker is

- Docker is a container tool that can run and create containers
- Containers only see their exact file system
- Similar to VM's, but less overhead
- Consists of reusable layers
- Uses a Dockerfile

Important concepts

- Images. Something you package your application into
- Registries. A repository to store your image
- Containers. Like normal Linux container

The Docker build process

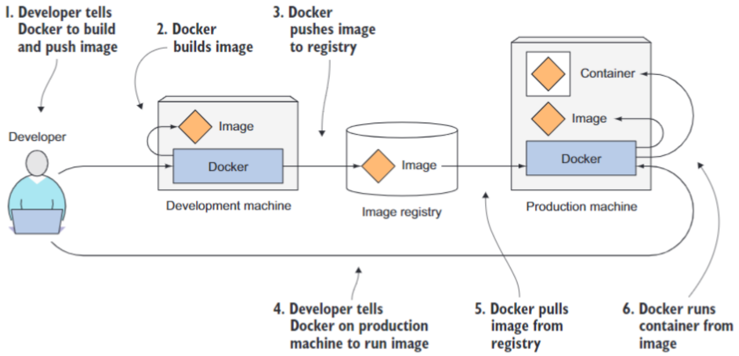
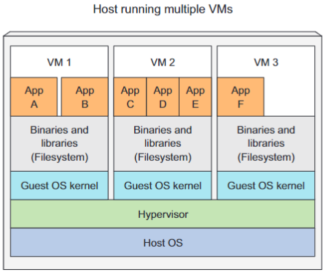

1. Developer tells Docker to build and push image
2. Docker builds image
3. Docker pushes image to registry
Developer
Image
Docker
Development machine
Image
Image registry
Container
Image
Docker
Production machine
4. Developer tells Docker on production machine to run image
5. Docker pulls image from registry
6. Docker runs container from image

Figure: Fig. by courtesy of Marko Luksa[1]

Running apps in VM's vs. containers


Host running multiple VMs
VM 1
VM 2
VM 3
App A
App B
App C
App D
App E
App F
Binaries and libraries (Filesystem)
Binaries and libraries (Filesystem)
Binaries and libraries (Filesystem)
Guest OS kernel
Guest OS kernel
Guest OS kernel
Hypervisor
Host OS

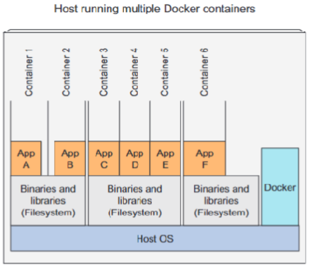

Host running multiple Docker containers
Container 1
Container 2
Container 3
Container 4
Container 5
Container 6
App A
App B
App C
App D
App E
App F
Binaries and libraries (Filesystem)
Binaries and libraries (Filesystem)
Binaries and libraries (Filesystem)
Docker
Host OS

Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Introduction

Background

Microservices

Linux containers

Docker

Kubernetes

Hands on

Kubernetes

Motivation

- We need something to manage our containers
- Should be reliable, automated and scalable
- Should ease the process of deploying containers

What Kubernetes is

- Open-source container orchestration engine
- Made by Google, open-sourced in 2014
- Google needed to better utilize their resources
- Enables easy deployment, scaling and managing
- Exposes whole datacenter as a single platform.

Running apps in VM's vs. containers

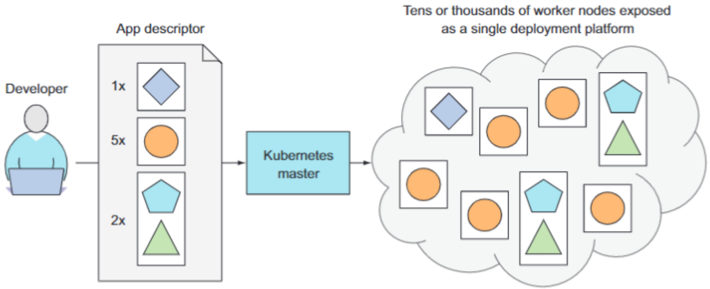

Tens or thousands of worker nodes exposed as a single deployment platform
App descriptor
Developer
1x
5x
2x
Kubernetes master

Figure: Fig. by courtesy of Marko Luksa[1]

Kubernetes cont.

Main features of Kubernetes

- Keep containers running.
- Scaling copies.
- Hitting a moving target.

Benefits of using Kubernetes

- Simplifying application deployment
- Achieve better utilization
- Health checking and self-healing
- Automatic scaling
- Access to services via API/DNS

Enterprise-use

- Often seen as a PaaS (OpenShift)

Kubernetes cont.

The master node hosts the Control Plane and worker nodes run deployments. The master contains

- API Server, which you and the Control Plane components communicate with
- The Scheduler, which schedules your apps
- The Controller Manager, keeps track of workers among others
- etcd, a distributed db that stores cluster configuration

The nodes contain

- Docker, rtk or another container runtime
- Kubelet, which talks to the API server and manages containers
- Kube-proxy, which load-balances network traffic

The components in Kubernetes

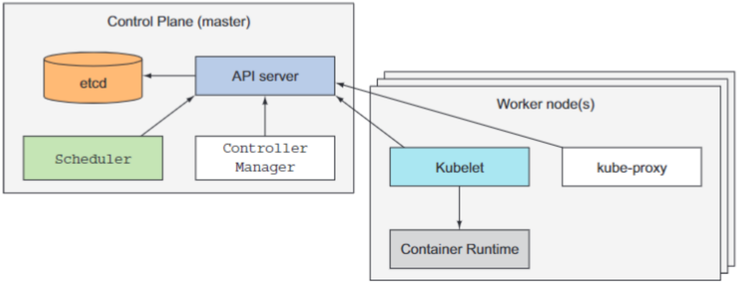


Figure: Fig. by courtesy of Marko Luksa[1]

An overview of Kubernetes's architecture

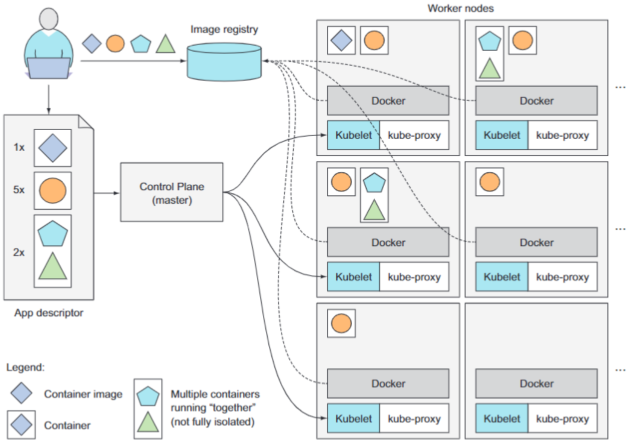


Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Introduction

Background

Microservices

Linux containers

Docker

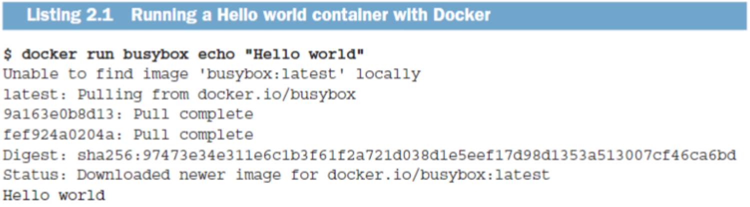
Kubernetes

Hands on

Running the busybox image

Installing Docker and running a Hello World container

- Busybox is a single executable with many UNIX tools.


Listing 2.1 Running a Hello world container with Docker

```
$ docker run busybox echo "Hello world"
Unable to find image 'busybox:latest' locally
latest: Pulling from docker.io/busybox
9a163e0b8d13: Pull complete
fef924a0204a: Pull complete
Digest: sha256:97473e34e311e6c1b3f61f2a721d038d1e5eef17d98d1353a513007cf46ca6bd
Status: Downloaded newer image for docker.io/busybox:latest
Hello world
```

Figure: Listing. by courtesy of Marko Luksa[1]

```
$ docker run busybox echo "Hello world"
$ docker run <image>
$ docker run <image>:<tag>
```

Running echo "Hello world" in a container

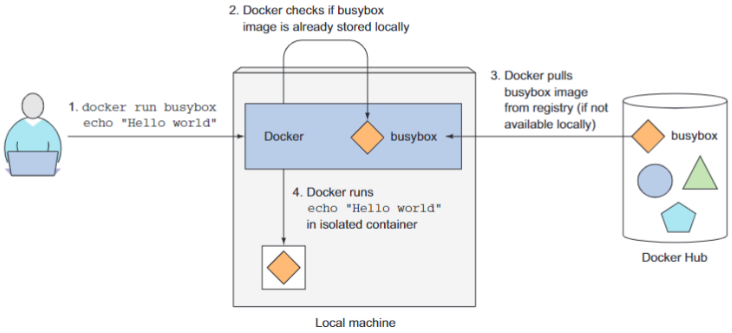
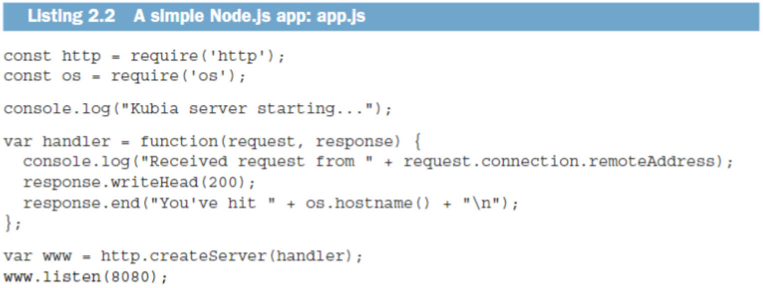

2. Docker checks if busybox image is already stored locally
1. docker run busybox echo "Hello world"
Docker
busybox
3. Docker pulls busybox image from registry (if not available locally)
busybox
4. Docker runs echo "Hello world" in isolated container
Docker Hub
Local machine

Figure: Fig. by courtesy of Marko Luksa[1]

Creating a Node.js app

- We make a simple HTTP app that can receive and reply to requests with its hostname
- Using Node.js and JavaScript



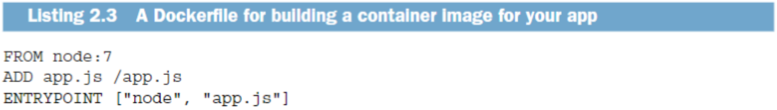
Listing 2.2 A simple Node.js app: app.js

```
const http = require('http');
const os = require('os');

console.log("Kubia server starting...");

var handler = function(request, response) {
  console.log("Received request from " + request.connection.remoteAddress);
  response.writeHead(200);
  response.end("You've hit " + os.hostname() + "\n");
};

var www = http.createServer(handler);
www.listen(8080);
```

Figure: Listing. by courtesy of Marko Luksa[1]

Creating a Dockerfile for the image

- We need a Dockerfile to create an image
- It describes the app and its dependencies



Listing 2.3 A Dockerfile for building a container image for your app

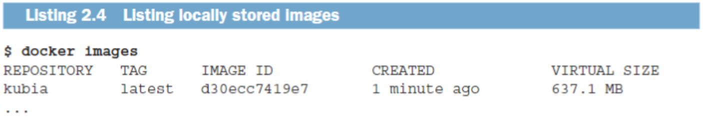
```
FROM node:7
ADD app.js /app.js
ENTRYPOINT ["node", "app.js"]
```

Figure: Listing. by courtesy of Marko Luksa[1]

Building the container image

The Docker daemon builds the image.

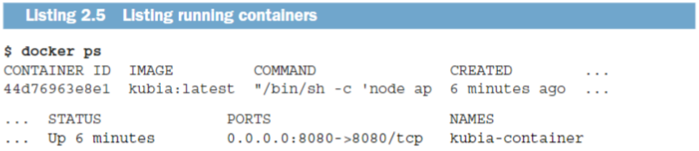
```
$ docker build -t kubia .
```



Listing 2.4 Listing locally stored images

```
$ docker images
REPOSITORY    TAG       IMAGE ID         CREATED            VIRTUAL SIZE
kubia         latest    d30ecc7419e7     1 minute ago       637.1 MB
...
```

Figure: Listing. by courtesy of Marko Luksa[1]



Listing 2.5 Listing running containers

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND                CREATED        ...
44d76963e8e1  kubia:latest  "/bin/sh -c 'node ap   6 minutes ago  ...

...  STATUS            PORTS                     NAMES
...  Up 6 minutes      0.0.0.0:8080->8080/tcp    kubia-container
```

Figure: Listing. by courtesy of Marko Luksa[1]

Building a new container image from a Dockerfile

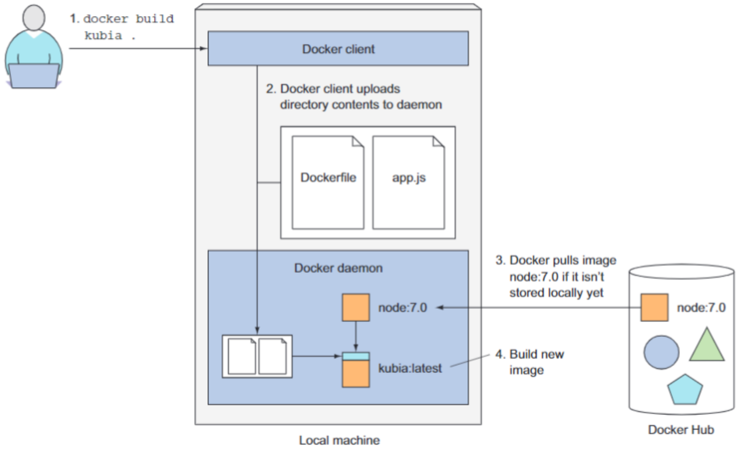


Figure: Fig. by courtesy of Marko Luksa[1]

The layers of a container image

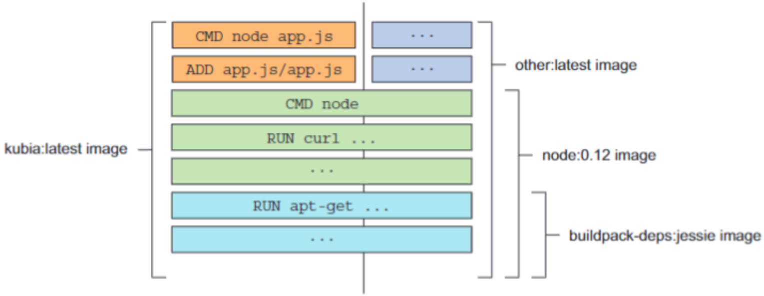


Figure: Fig. by courtesy of Marko Luksa[1]

More container commands

Running the container image

```
$ docker run --name kubia-container -p 8080:8080
$ curl localhost:8080
```

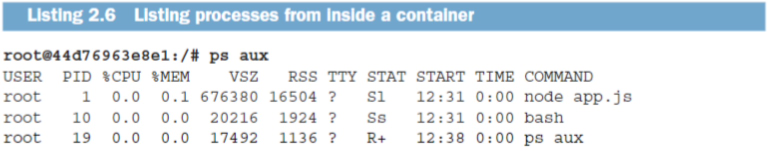
Exploring the inside of a running container

```
$ docker exec -it kubia-container bash
$ ps aux
```

Stopping and removing a container

```
$ docker stop kubia-container
$ docker rm kubia-container
```

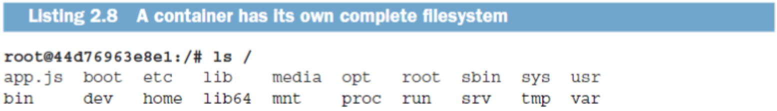
Output of container commands



Listing 2.6 Listing processes from inside a container

```
root@44d76963e8e1:/# ps aux
USER   PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
root     1  0.0  0.1 676380 16504 ?   Sl   12:31 0:00 node app.js
root    10  0.0  0.0  20216  1924 ?   Ss   12:31 0:00 bash
root    19  0.0  0.0  17492  1136 ?   R+   12:38 0:00 ps aux
```

Figure: Listing. by courtesy of Marko Luksa[1]



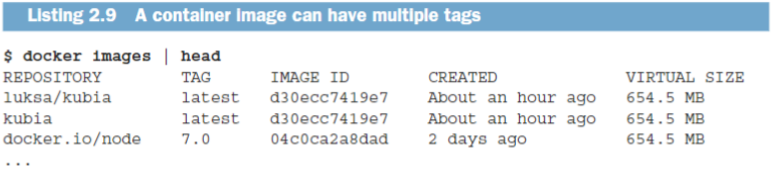
Listing 2.8 A container has its own complete filesystem

```
root@44d76963e8e1:/# ls /
app.js  boot  etc   lib    media  opt   root  sbin  sys  usr
bin     dev   home  lib64  mnt    proc  run   srv   tmp  var
```

Figure: Listing. by courtesy of Marko Luksa[1]

Pushing the image to the registry

```
$ docker tag kubia <dockerid>/kubia
```



Listing 2.9 A container image can have multiple tags

```
$ docker images | head
REPOSITORY        TAG      IMAGE ID       CREATED              VIRTUAL SIZE
luksa/kubia       latest   d30ecc7419e7   About an hour ago    654.5 MB
kubia             latest   d30ecc7419e7   About an hour ago    654.5 MB
docker.io/node    7.0      04c0ca2a8dad   2 days ago           654.5 MB
...
```

Figure: Listing. by courtesy of Marko Luksa[1]

```
$ docker push <dockerid>/kubia
$ docker run -p 8080:8080 -d <dockerid>/kubia
```

Setting up a Kubernetes cluster with Minikube

What is minikube

- A tool that enables us to run Kubernetes locally
- Implements a local single node cluster
- Runs on macOS, Linux and Windows
- Used for Kubernetes development and prototyping
- Exercises and project make use of minikube

You can also order a hosted Kubernetes cluster at Google, Amazon, Microsoft Azure etc.

```
$ minikube start
$ kubectl get nodes
$ kubectl describe node <nodeid>
```

Introducing pods

Before we can deploy our app, we need to know what pod is

- A pod is a group of one or more tightly related containers that run together and share the same Linux namespace
- Each pod is like a separate logical machine.
- All containers in a pod will appear to be running on the same logical machine.

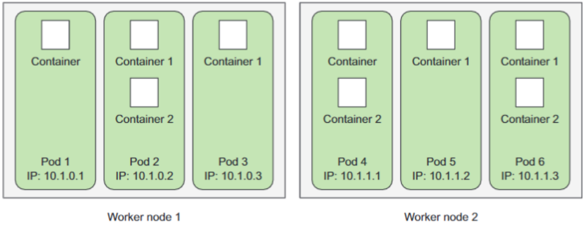
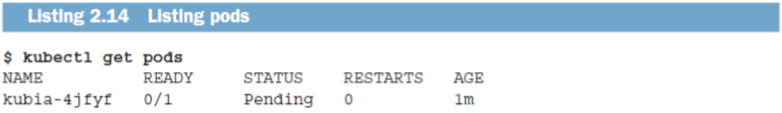


Figure: Fig. by courtesy of Marko Luksa[1]

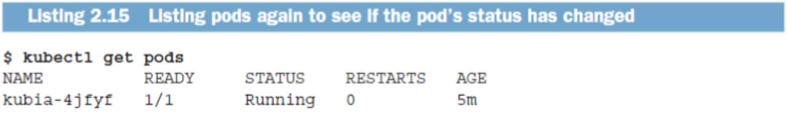
Pod commands



Listing 2.14 Listing pods

```
$ kubectl get pods
NAME          READY     STATUS     RESTARTS    AGE
kubia-4jfyf   0/1       Pending    0           1m
```

Figure: Listing. by courtesy of Marko Luksa[1]



Listing 2.15 Listing pods again to see if the pod's status has changed

```
$ kubectl get pods
NAME          READY     STATUS     RESTARTS    AGE
kubia-4jfyf   1/1       Running    0           5m
```

Figure: Listing. by courtesy of Marko Luksa[1]

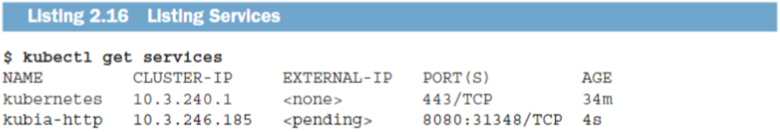
Running our first app on Kubernetes

```
$ kubectl run kubia --image=<dockerid>/kubia \\
 --port=8080 --generator=run-pod/v1
pod/kubia created
```

Accessing our application

We can create a service that exposes our pod to us

```
$ kubectl expose pod kubia --type=NodePort \\
 --name kubia-http
```


Listing 2.16 Listing Services

```
$ kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
kubernetes   10.3.240.1     <none>        443/TCP         34m
kubia-http   10.3.246.185   <pending>     8080:31348/TCP  4s
```

Figure: Listing. by courtesy of Marko Luksa[1]

Accessing our application cont.

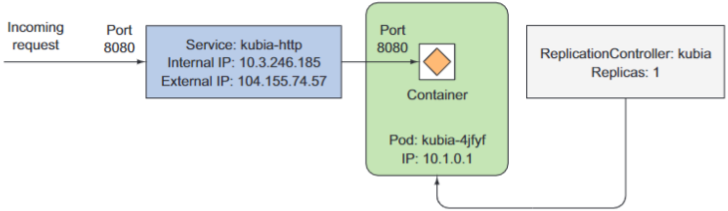


Figure: Fig. by courtesy of Marko Luksa[1]

Accessing our application cont.

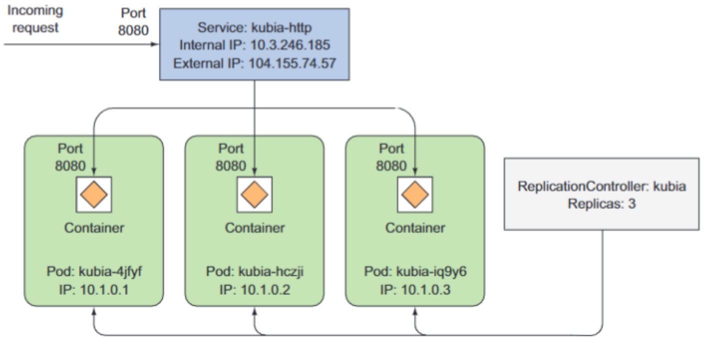


Figure: Fig. by courtesy of Marko Luksa[1]

References I

[1] Luksa, M. (2018). *Kubernetes in Action*. Manning Publications Co.