Docker, Kubernetes

symflower
AUTOMATING QUALITY ASSURANCE

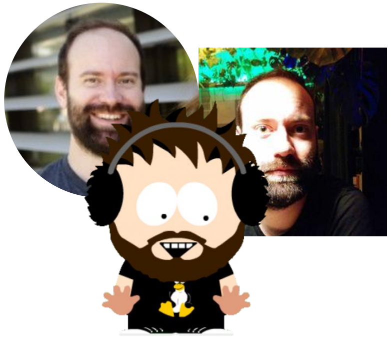
Motivation, Basics, Hands-on

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

Agenda

- **Introduction**
- VMs vs Containers
- Docker
- Kubernetes
- Q&A (... for bigger questions)

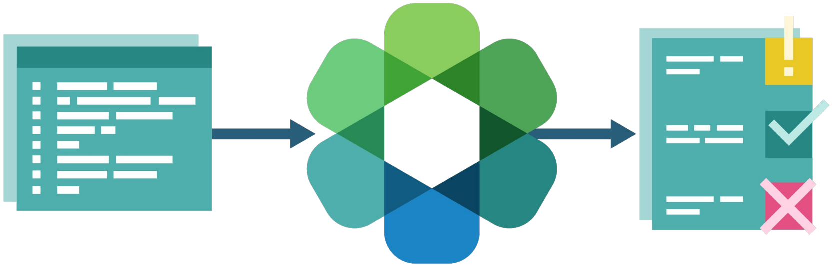
Who Am I?



- Markus Zimmermann https://github.com/zimmski
- Writing software since primary school
- Using
  - Docker full-time since 2014
  - Kubernetes full-time since 2016
- Work (mainly “project consulting”)
  - Lots of “enterprise” applications, web services, tooling, software infrastructure, distributed apps and clustering, software testing
  - **Now: Software testing and verification @Symflower**

What is Symflower?

Symflower completely **automatically writes** and runs unit tests revealing bugs, security issues and performance problems.



→ Reduce development and maintenance time

→ Increase quality of your software and tests

Example: Find the Problem...

```
func match(s1, s2 []byte) bool {
    for i := 0; i < len(s1); i++ {
        c1 := s1[i]
        c2 := s2[i]
        if c1 != c2 {
            c1 |= 'a' - 'A'
            c2 |= 'a' - 'A'
            if c1 != c2 || c1 < 'a' || c1 > 'z' {
                return false
            }
        }
    }
    return true
}
```

Code from https://golang.org/src/time/format.go

Found the Problem!

```go
func match(s1, s2 []byte) bool {
    for i := 0; i < len(s1); i++ {
        c1 := s1[i]
        c2 := s2[i]
        if c1 != c2 {
            c1 |= 'a' - 'A'
            c2 |= 'a' - 'A'
            if c1 != c2 || c1 < 'a' || c1 > 'z' {
                return false
            }
        }
    }
    return true
}
```

Index out of range!

Code from https://golang.org/src/time/format.go

Generated Unit Tests

```
func TestMatch117(t *testing.T) {
	var s1 []byte = []byte{'\x00'}
	var s2 []byte = nil

	match(s1, s2) // Panic!
}

func TestMatch119(t *testing.T) {
	var s1 []byte = []byte{'\x00'}
	var s2 []byte = []byte{'\x00'}

	actual := match(s1, s2)

	var expected bool = true
	assert.Equal(t, expected, actual)
}
```

```
func TestMatch123(t *testing.T) {
	var s1 []byte = []byte{'c'}
	var s2 []byte = []byte{'C'}

	actual := match(s1, s2)

	var expected bool = true
	assert.Equal(t, expected, actual)
}
```

....

Full MC/DC and problem coverage

If you find that interesting: talk to me or write to hello@symflower.com

Docker/Kubernetes at Symflower

- We use Docker and Kubernetes **everywhere**
- Services (e.g. issue tracker, mail, websites)
- CI/CD
  - ➔ Every pipeline run has at least 3 deployments
  - ➔ Every deployment is a Kubernetes deployment
- Symflower the product
  - ➔ Every instance is a Kubernetes cluster
  - ➔ Every service must be isolated and must scale
  - ➔ Every test case is isolated & executed multiple times

Who are you?

- Who are you?
- What have you done before?
- What are you doing right now?
- **What do you want to achieve today?**

Agenda

- Introduction
- **VMs vs Containers**
- Docker
- Kubernetes
- Q&A (... for bigger questions)

1

VMs vs Containers

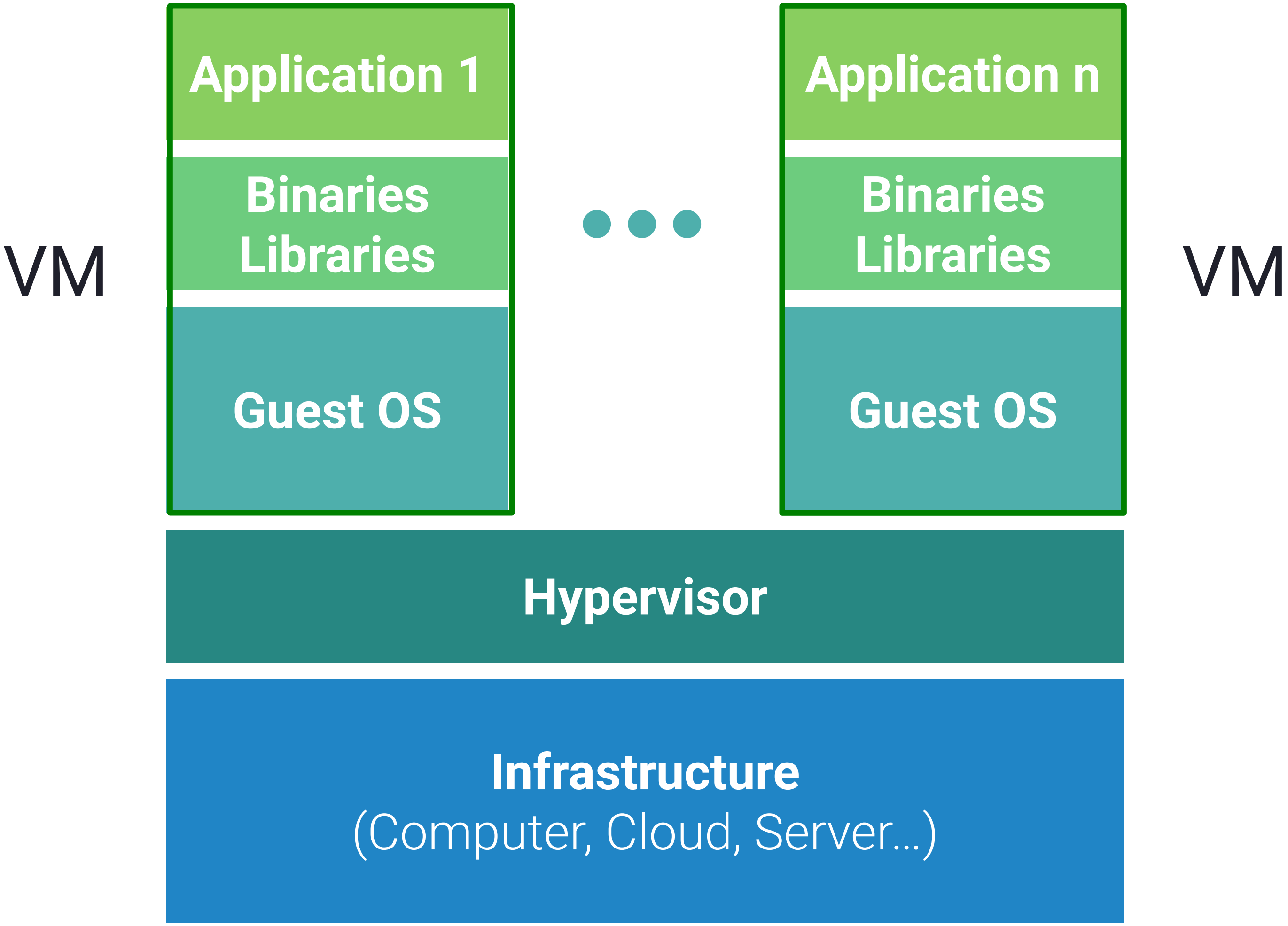
They seem to be kind of similar

What is a VM?

> *A **virtual machine** (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer.*

wikipedia.org

What is a VM?


Application 1
Binaries
Libraries
Guest OS
Application n
Binaries
Libraries
Guest OS
VM
VM
Hypervisor
Infrastructure
(Computer, Cloud, Server...)

What is a Container?

”

> ***<u>OS-level virtualization</u>*** *refer to an operating system paradigm in which the kernel allows the existence of multiple isolated user-space* ***instances (e.g. a container)****. Containers may look like real computers from the point of view of programs running in them.*

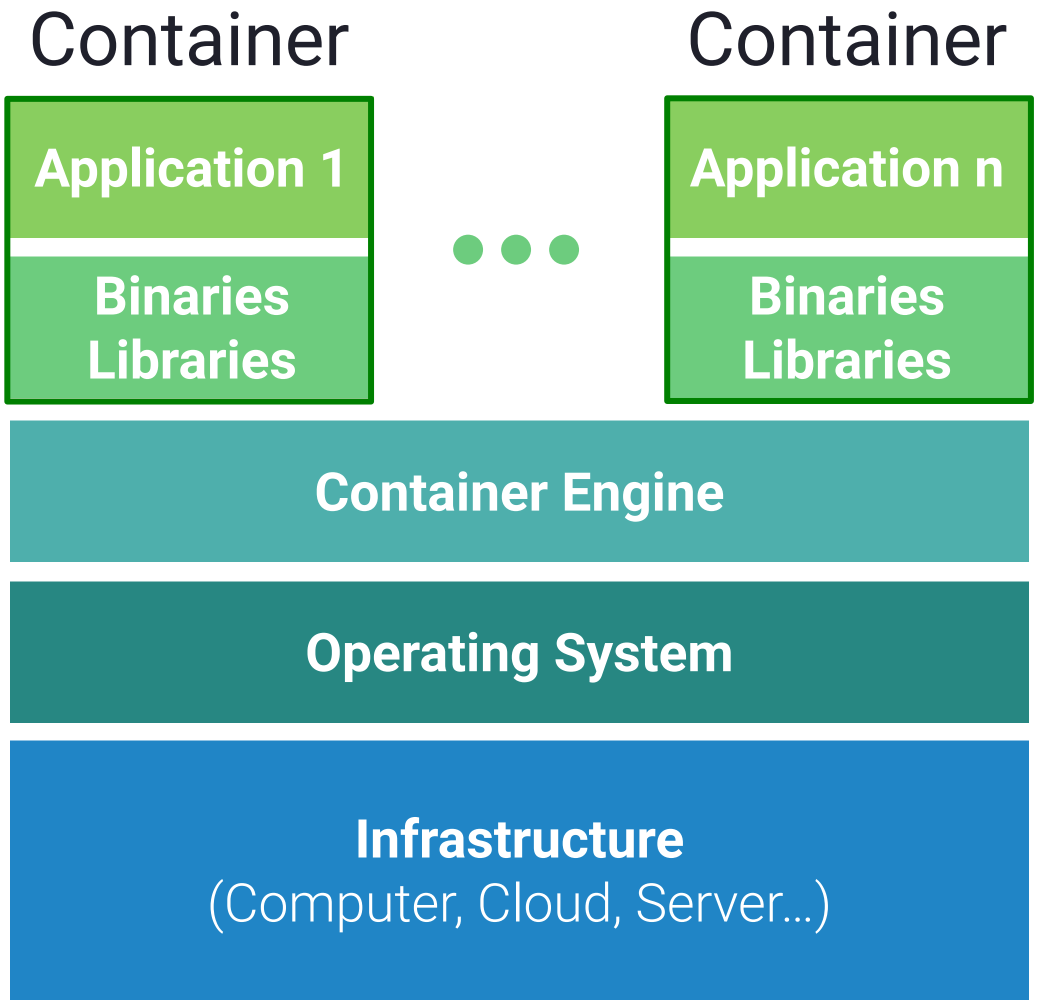
wikipedia.org

What is a Container?

> ”
>
> *Think of a container*
> *as a single process and its children*
> *that run isolated*
> *with limited resources.*

Markus Zimmermann

What is a Container?



Container

Application 1

Binaries
Libraries

Container

Application n

Binaries
Libraries

Container Engine

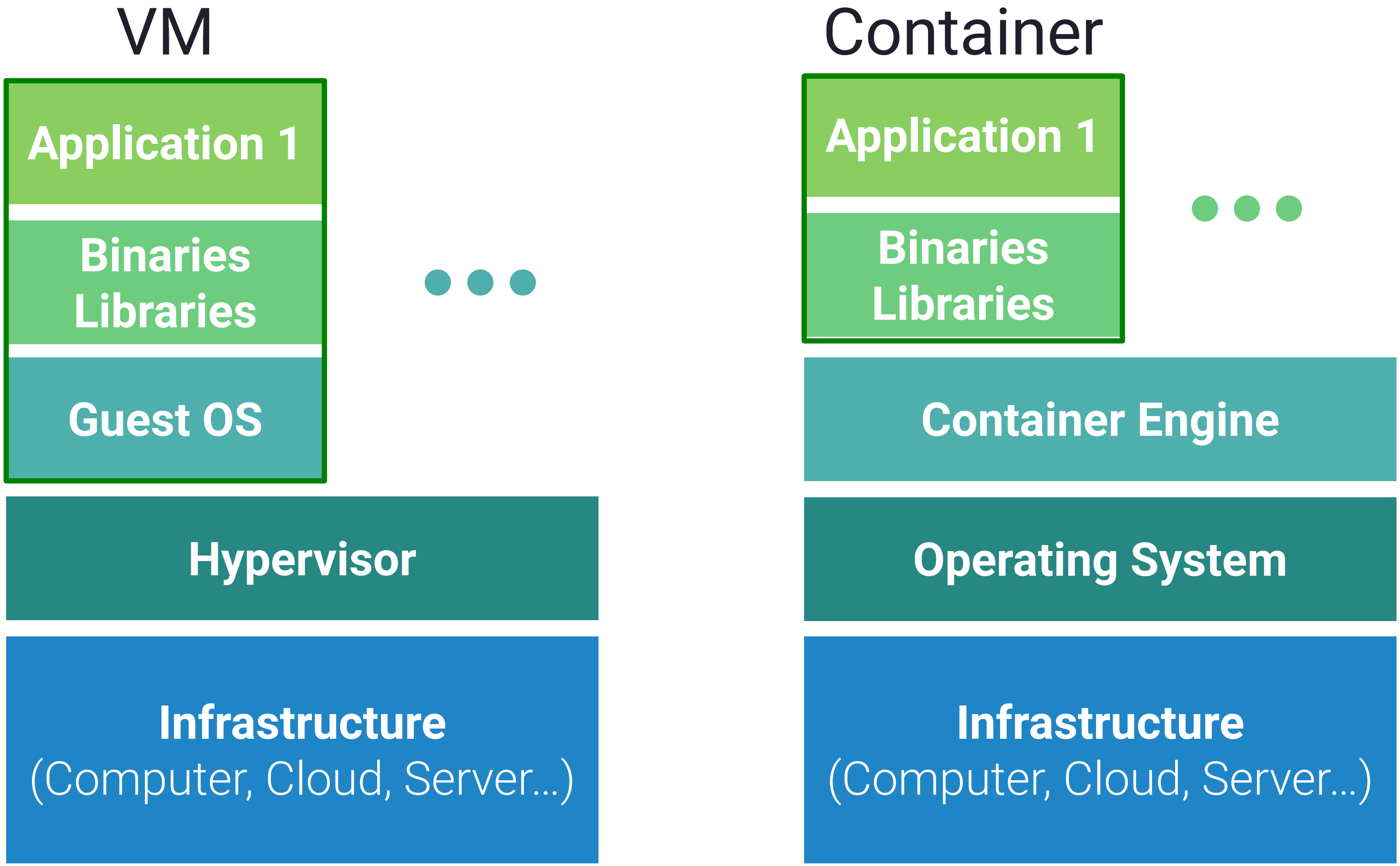
Operating System

Infrastructure
(Computer, Cloud, Server...)

What is a Container?

- Nowadays there is a complete ecosystem
- Typical container technology but also …
- Binaries are replaced with “images”
  - Contain not just files
- Handling of volumes
- Handling of network
- Handling of devices
- Handling of images/life cycle/...

VMs vs Container?



VMs vs Container?

VM	Container
Heavyweight	Lightweight
Limited performance	Native performance
Own OS	Shared OS
Startup time in minutes	Startup time in seconds
Fully isolated	Process-level isolation

VMs vs Container?

- When to use one over the other?
  - Use VMs when you do not want to share resources
  - Use VMs when you absolutely do not trust X
  - Use VMs for legacy applications (do not touch a …)
  - Use containers when you need to maximize the number of applications running on a server
  - Use containers if you need to scale horizontally
  - But in general...

Should you use containers?

- My honest opinion: ***YES***
- There are four major reasons
  - Isolation
  - Consistency
  - Reproducibility
  - Utilization
- Using container forces you to think about configurations and to document your decisions

Btw: Is Container == Docker?

- **No** there are a lot of different technologies
- You can choose e.g.
  - Docker
  - CRI-O
  - Kata Container
  - gVisor
  - ….
- Open Container Initiative (OCI) establishes standards for container technology

Btw: What about “the Cloud”?

- Do not be fooled: “the cloud” can be on-premise too
- Do not just think about the infrastructure
- It is mainly about modernization, e.g.
  - 12-factor applications
  - Reproducible and reusable
  - DevOps trend
  - “Infrastructure as Code”
  - Rapidly updated applications
  - Efficient resource usage

Agenda

- Introduction
- VMs vs Containers
- **Docker**
- Kubernetes
- Q&A (... for bigger questions)

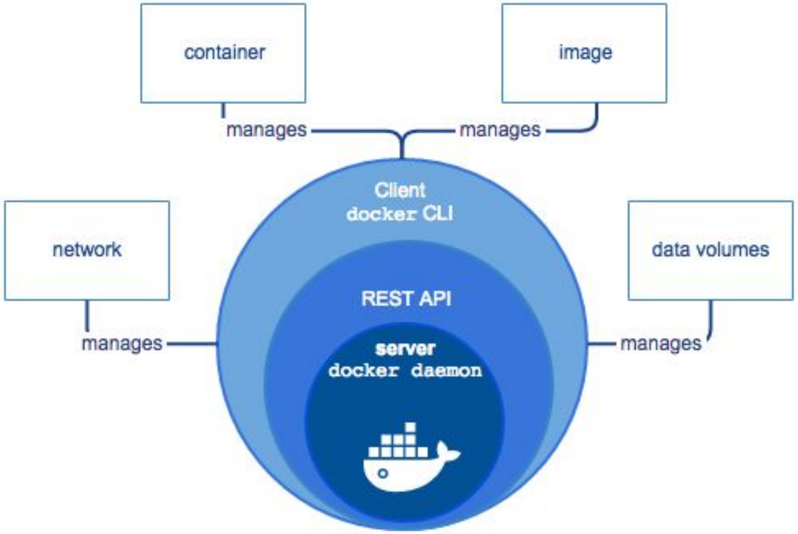
2

Docker

The safe and easy way of doing containers

What is Docker?

- “**Docker** is a set of platform-as-a-service (PaaS) products that use **OS-level virtualization** to deliver software in packages called containers.”



https://docs.docker.com/engine/docker-overview/

Basic Docker Wording 1/2

- Image
  - Basis of a container
  - Basically “the content” e.g. files and defaults
- Registry
  - Stores images
  - Basically an intelligent filesystem for images
- Container
  - One **running** image (not necessarily one service)

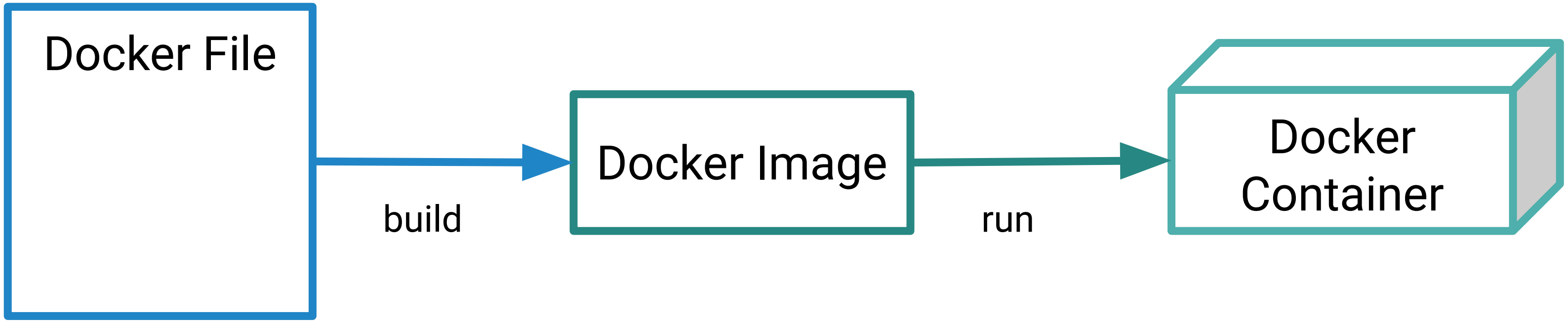
Basic Docker Wording 2/2

- Engine
  - “API” to interact with containers
  - Usually takes care of networking and volumes
- Orchestration (sometimes “control-plane”)
  - Handles life cycle of containers
  - Usually over multiple nodes (hosts)

What is Moby?

- “Project group” of Docker
- To assemble container based systems
  - And not invent the wheel all over again
- Provides a “Lego-Set” of standard components
  - Orchestration, image management, secret management, networking, provisioning
- Basically:
  - Docker split into Moby
  - Docker is now assembled by Moby projects

Docker Basics



- The docker file specifies all properties of the container
- Docker files are built into docker images
- Docker images are run in docker containers
- Docker Hub:
  - **THE** common library for container images

Docker Container Life Cycle

- Created: A created container that has not started yet
- Restarting: A container currently restarting
- Started: A running container
- Paused: A container with paused processes
- Exited: A container that finished its work
- Dead: A container that has been killed

Docker Namespaces

- Docker namespaces are used for isolation
- Each container has a set of namespaces
- Each aspect of a container runs in a dedicated namespace, its access is limited to this namespace
- E.g. NET, PID, MNT, USER, …
- Cgroups: used for limiting and isolation resource usage

Basic Commands

- `docker`
- `docker info`
- `docker build --tag=$tag-name .`
- `docker run $image-name`
- `docker image ls (or: docker images)`
- `docker container ls (or: docker ps)`

Agenda

- Introduction
- VMs vs Containers
- Docker
- **Kubernetes**
- Q&A (... for bigger questions)

3
Kubernetes

Infrastructure and deployments made simple

”

Kubernetes (k8s) is an ***open-source container-orchestration*** *system for automating deployment, scaling and management of containerized applications.*

wikipedia.org

Different Perspectives

Developer

The framework for deployment and infrastructure

Admin

The OS to manage the infrastructure

Manager

Let one person perform like ten.

Standardization + Knowledge of hundreds of experts

Fundamental Concepts of K8s 1/2

- Automate everything
  - Resource management
  - Deployments (Rollouts and Rollbacks)
  - Scaling
  - Provisioning
  - Monitoring/Healing/Debugging/...
- Declarative (generic) configuration
  - No explicit host usage
  - No SSH, no scripts -> see Ansible/Puppet/Salt

Fundamental Concepts of K8s 2/2

- “Infrastructure as Code”
  - Generic configuration and container images
  - Variables define specifics, e.g. user/password of DB
  - Everything is reproducible and reusable
- Everything is disposable (best practice)
  - Pet vs Cattle (see next slide)

Pet vs. Cattle

- Pets are given names
- They are unique lovingly raised and cared for
- When they get ill, you nurse them back to health

- Cattle is given numbers
- They are identical to one another
- When they get ill, you get another one

When to (not) use K8s

- Do you have some kind of job/service/server?
  - If not: Sorry, no Kubernetes for you …
- Single job/service/server
  - Usually a container is fine.
- Multiple ... -> **yes**
- K8s right from the start?
  - Is there k8s experience?
  - Time spent on maintaining
  - ...

A practical example: CI/CD

- Continuous Integration (CI) is perfect for k8s
  - Goal: Develop, test and release fast
  - Automate builds, checks, deployments
  - Integrate changes multiple times a day
- Continuous Deployment (CD)
  - Automatically deploy each change
- @Symflower
  - Our setup would be almost impossible without k8s
  - Let’s take a look

Installation of K8s

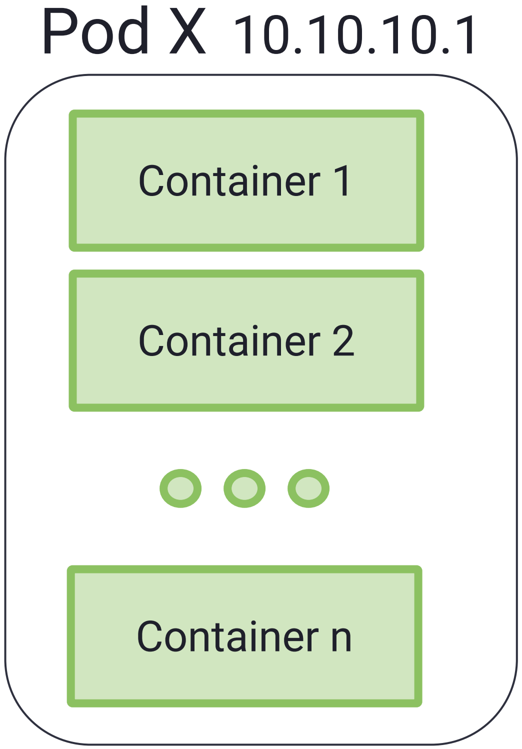
- How and where should I run k8s?
  - Lots of possibilities
- Your notebook
  - minikube
  - Vagrant e.g. with kubespray
- Managed: almost every cloud provider
- On-premise (self-hosted)?
  - WARNING: getting easier every day
    but still knowledge intensive in production

Basics Concepts of K8s

- We will only look at a minimum
  - **What do you need to deploy an application?**
- What is a k8s …
  - … pod
  - … node
  - … cluster
  - … deployment
  - … service

K8s Pods

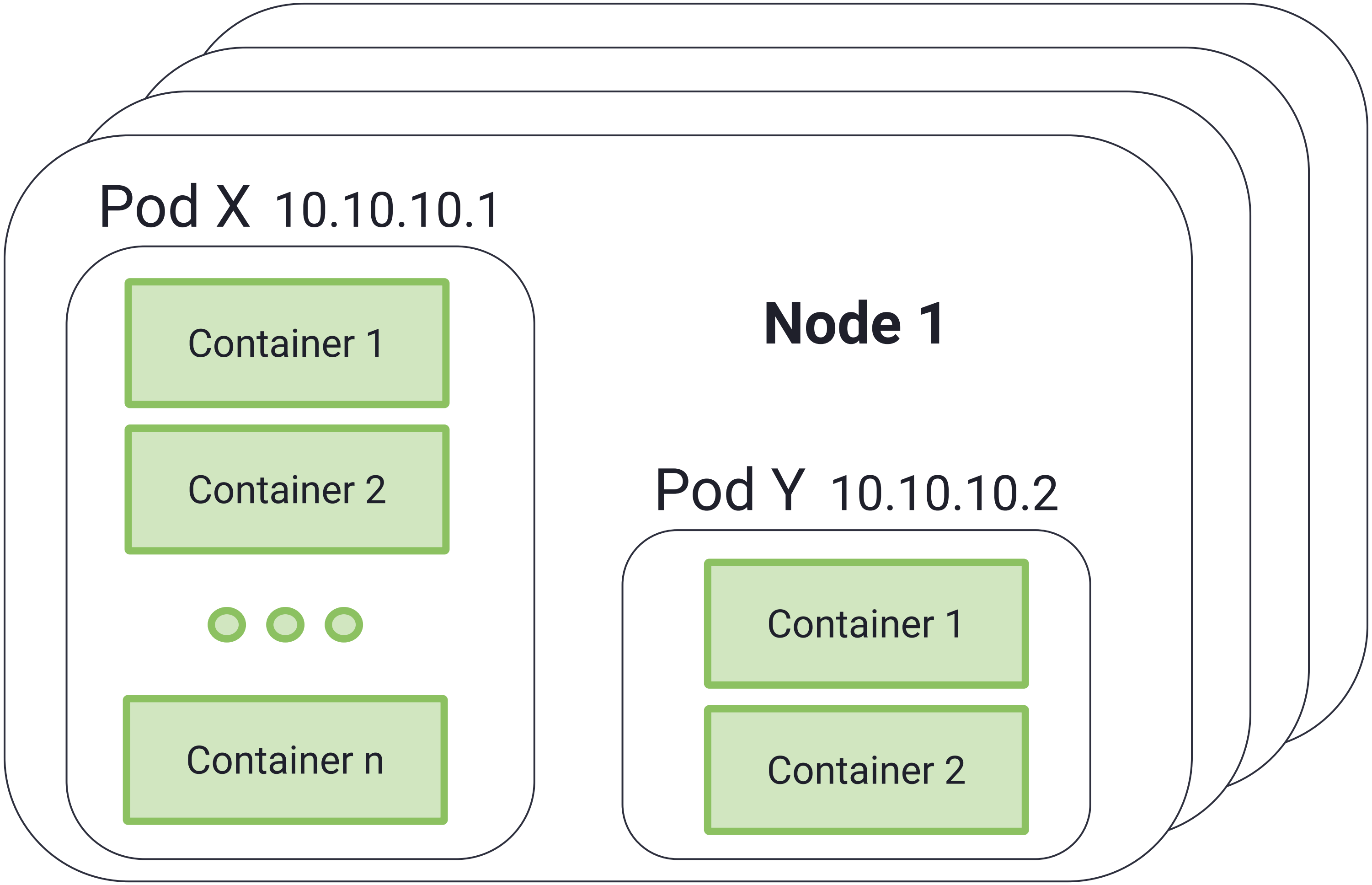
- One or more containers
- Shares resources between its containers
- Needs all of its containers to live
- Why use different containers?
- Which containers should share a pod?



K8s Nodes and Clusters 1/2

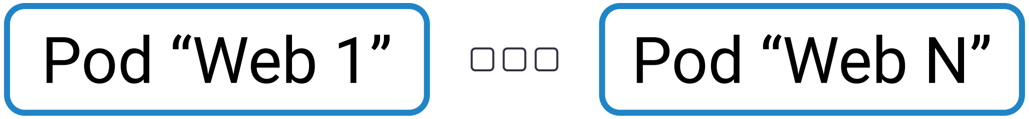
- A node is a “worker machine”
- A cluster consists of one or more nodes
  - Does a single node cluster make sense?
- Nodes provide resources to pods
- A specific pod lives in exactly one node
- Assignment of pods to nodes and their life cycle are managed by k8s
  - What might be good assignment decisions?

K8s Nodes and Clusters 2/2



K8s Deployment

- A deployment manages a “replica set” of pods
- **Guarantees** the **availability** of a **specific number** of **identical pods**
- Handles scaling and rollout/rollback of pods

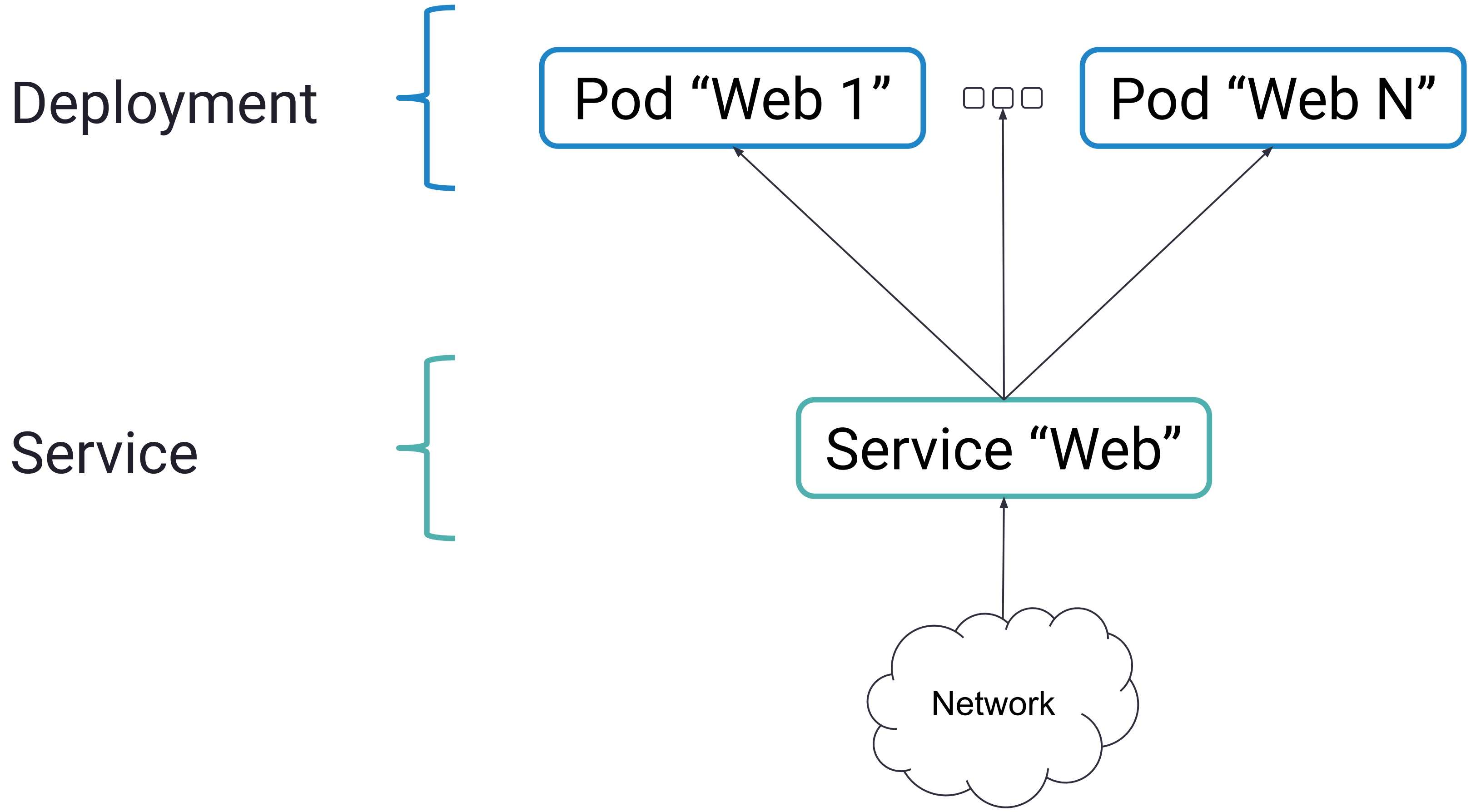


Pod “Web 1” □□□ Pod “Web N”

K8s Service 1/2

- Exposes a set of pods as a network service
  - E.g. deployment “web” with domain “api”
- Not necessarily just one deployment
- Basically an internal load-balancer
  - Pods can be born and die -> routes can change
  - Service is always reachable and forwards

K8s Service 2/2



Deployment

Pod “Web 1”

Pod “Web N”

Service

Service “Web”

Example: Web Server + Database

- We need all our Docker knowledge but ...
- … this time we write “infrastructure as code”

- Bonus concept: k8s namespaces
  - A pod has exactly one namespace
  - A namespace is isolated from other namespaces

Additional K8s Concepts

- ConfigMaps / Secrets
  - Inject configurations
- Persistent Volumes
  - **Persist** data to some storage
- Ingress (Controller)
  - Manage external access to services
- Pod/Network/… Policies
  - Restrict access to pods/functionality/…
- …

Where do we go from here?

- Lots to learn https://kubernetes.io/docs/home/
- Migrating legacy applications?
- Let applications scale?
- Creating a multi-node cluster?
- Creating a cluster of clusters?
- Manage VMs with k8s?
- Manage GPUs with k8s?
- …

Agenda

- Introduction
- VMs vs Containers
- Docker
- Kubernetes
- **Q&A (... for bigger questions)**

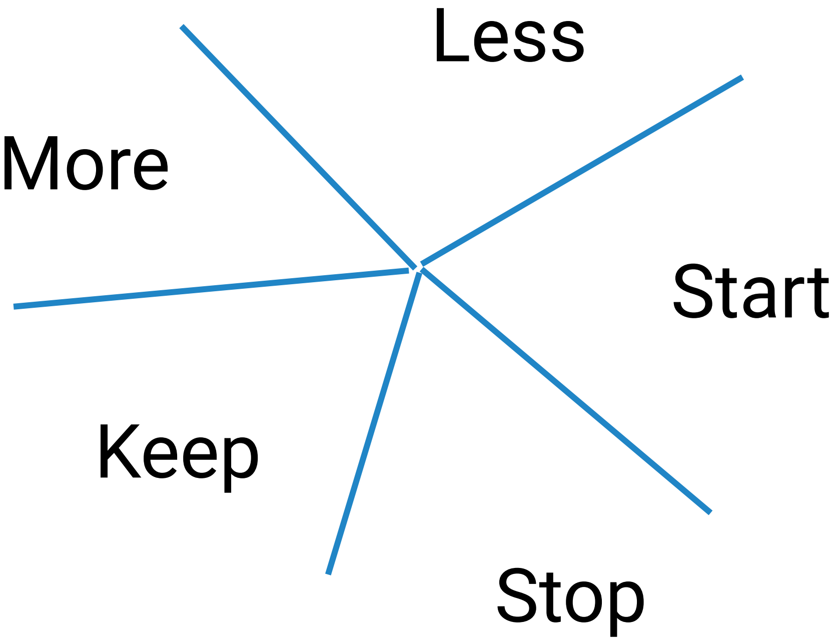
4
Q&A

symflower

AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

Feedback



Less

More

Start

Keep

Stop