# Web 2.0

## Lecture 3: Microservices and Cloud Native

**doc. Ing. Tomáš Vitvar, Ph.D.**

tomas@vitvar.com • @TomasVitvar • http://vitvar.com

Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/w20

EVROPSKÁ UNIE

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

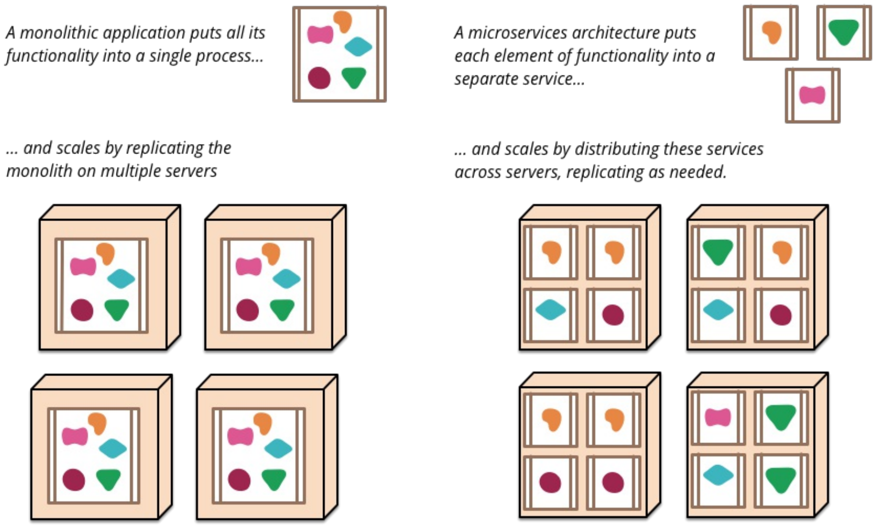
Modified: Sun Mar 01 2020, 22:34:50
Humla v0.3

## Overview

- Microservices Architecture
- Docker
- Kubernetes

# Overview

- Emerging software architecture
  - *monolithic vs. decoupled applications*
  - *applications as independenly deployable services*



*A monolithic application puts all its functionality into a single process...*

*A microservices architecture puts each element of functionality into a separate service...*

*... and scales by replicating the monolith on multiple servers*

*... and scales by distributing these services across servers, replicating as needed.*

# Major Characteristics

- Loosely coupled
  - *Integrated using well-defined interfaces*
- Technology-agnostic protocols
  - *HTTP, they use REST architecture*
- Independently deployable and easy to replace
  - *A change in small part requires to redeploy only that part*
- Organized around capabilities
  - *such as accounting, billing, recommendation, etc.*
- Impplemented using different technologies
  - *polyglot – programming languages, databases*

## Overview

- Microservices Architecture
- Docker
  - *Overview*
  - *Image Layering*
  - *Working with Docker*
  - *Swarm*
- Kubernetes

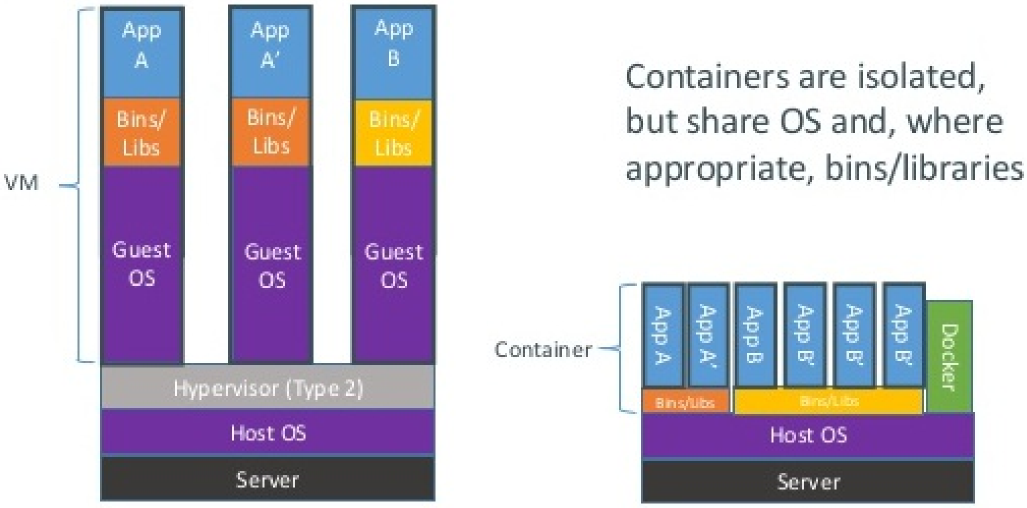
## Overview

- Linux Containers
  - *Introduced in 2008*
  - *Allow to run a process tree in a isolated system-level "virtualization"*
  - *Use much less resources and disk space than traditional virtualization*
- Implementations
  - *LXC – default implementation in Linux*
  - ***Docker Containers***
    - → *Builds on Linux namespaces and union file system (OverlayFS, AUFS, etc.)*
    - → *A way to build, commit and share images*
    - → *Build images using a description file called Dockerfile*
    - → *Large number of available base and re-usable images*

# Linux Namespaces

- "Isolation" of Linux processes
- There are 7 namespaces
  – *Mount – isolae filesystem mount points*
  – *UTS – isolate hostname and domainname*
  – *IPC – isolate interprocess communication resources*
  – *PID – isolate PID number space*
  – *Network – isolate network interfaces*
  – *User – isolate UID/GID number spaces*
  – *cgroup – isolate cgroup root directory*

# VM vs. Docker Containers

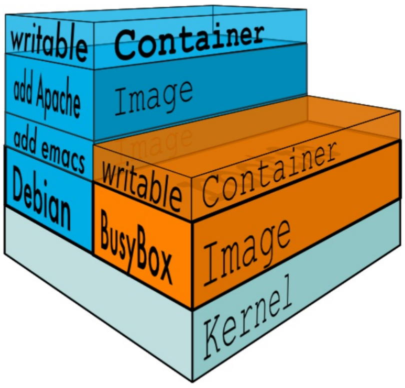


Containers are isolated, but share OS and, where appropriate, bins/libraries

# Docker Basic Terms

- Image
  - *Bssis for containers.*
  - *An image contains a union of layered filesystems stacked on top of each other.*
  - *An image does not have state and it never changes.*
- Container
  - *A runtime instance of a Docker image, a standard to "ship software".*
- Docker Engine
  - *The core process providing the Docker capabilities on a host.*
- Docker Client
  - *Interface that integrates with docker engine.*
- Registry
  - *A hosted service containing repository of images.*
  - *A registry provides a registry API to search, pull and push images.*
  - *Docker Hub is the default Docker registry.*
- Swarm
  - *A cluster of one or more docker engines.*

# Docker Images



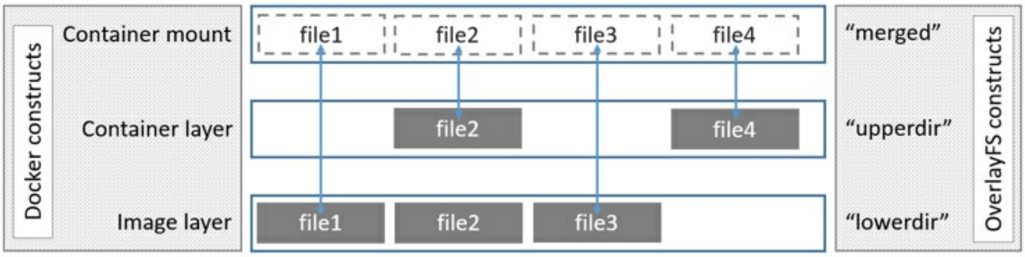
- *Containers are made up of R/O layers via a storage driver (OverlayFS, AUFS, etc.)*
- *Containers are designed to support a single application*
- *Instances are ephemeral, persistent data is stored in bind mounts or data volume containers.*

## Overview

- Microservices Architecture
- Docker
  - *Overview*
  - *Image Layering*
  - *Working with Docker*
  - *Swarm*
- Kubernetes

## Image Layering with OverlayFS

- OverlayFS
  - *A filesystem service implementing a* ***union mount*** *for other file systems.*
  - *Docker uses* `overlay` *and* `overlay2` *storage drivers to build and manage on-disk structures of images and containers.*
- Image Layering
  - *OverlayFS takes two directories on a single Linux host, layers one on top of the other, and provides a single unified view.*
  - *Only works for two layers, in multi-layered images hard links are used to reference data shared with lower layers.*



# Image Layers Example

- Pulling out the image from the registry

```
$ sudo docker pull ubuntu

Using default tag: latest
latest: Pulling from library/ubuntu

5ba4f30e5bea: Pull complete
9d7d19c9dc56: Pull complete
ac6ad7efd0f9: Pull complete
e7491a747824: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:46fb5d001b88ad904c5c732b086b596b92cfb4a4840a3abd0e35dbb6870585e4
Status: Downloaded newer image for ubuntu:latest
```

  – *Each image layer has its own directory under* `/var/lib/docker/overlay/`.
  – *This is where the contents of each image layer are stored.*

- Directories on the file system

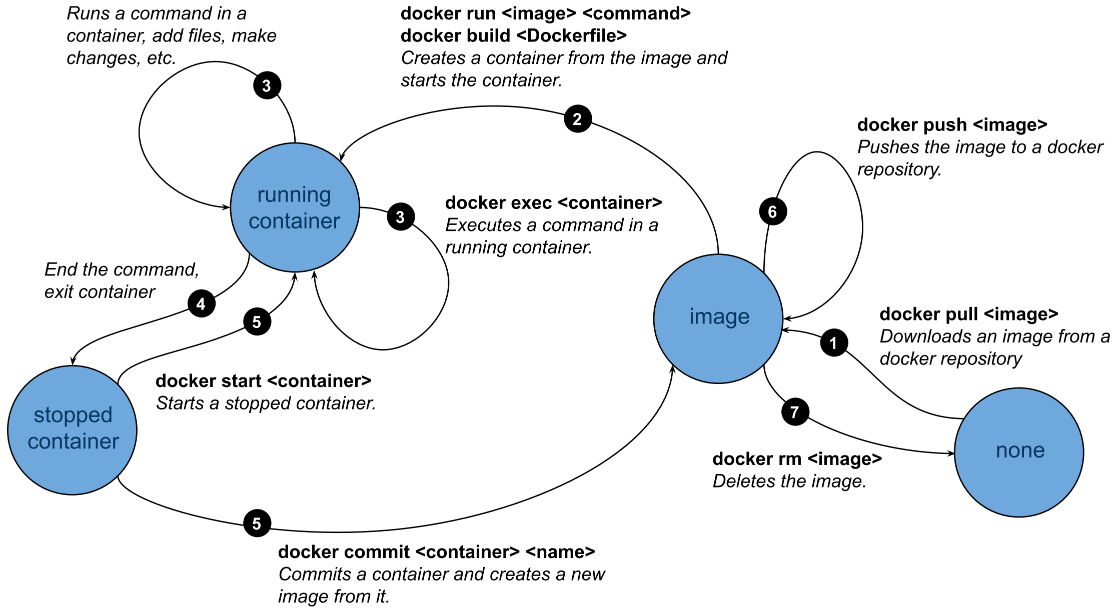
```
$ ls -l /var/lib/docker/overlay/

total 20
drwx------ 3 root root 4096 Jun 20 16:11 38f3ed2eac129654acef11c32670b534670c3a06e483fce313d72e3e
drwx------ 3 root root 4096 Jun 20 16:11 55f1e14c361b90570df46371b20ce6d480c434981cbda5fd68c6ff61
drwx------ 3 root root 4096 Jun 20 16:11 824c8a961a4f5e8fe4f4243dab57c5be798e7fd195f6d88ab06aea92
drwx------ 3 root root 4096 Jun 20 16:11 ad0fe55125ebf599da124da175174a4b8c1878afe6907bf7c7857034
drwx------ 3 root root 4096 Jun 20 16:11 edab9b5e5bf73f2997524eebeac1de4cf9c8b904fa8ad3ec43b35041
```

  – *The organization of files allows for efficient use of disk space.*
  – *There are* ***files unique to every layer*** *and* ***hard links to files*** *shared with lower layers*

# Overview

- Microservices Architecture
- Docker
  – *Overview*
  – *Image Layering*
  – *Working with Docker*
  – *Swarm*
- Kubernetes

# Docker Container Lifecycle State Diagram



*Runs a command in a container, add files, make changes, etc.*

**docker run <image> <command>**
**docker build <Dockerfile>**
*Creates a container from the image and starts the container.*

**docker push <image>**
*Pushes the image to a docker repository.*

**docker exec <container>**
*Executes a command in a running container.*

*End the command, exit container*

**docker pull <image>**
*Downloads an image from a docker repository*

**docker start <container>**
*Starts a stopped container.*

**docker rm <image>**
*Deletes the image.*

**docker commit <container> <name>**
*Commits a container and creates a new image from it.*

# Commands (1)

`docker version`

*list current version of docker engine and client*

`docker search <image>`

*search for an image in the registry*

`docker pull <image[:version]>`

*download an image of a specific version from the registry*
*if the version is not provided, the latest version will be downloaded*

`docker images`

*list all local images*

`docker run -it <image[:version]> <command>`

*start the image and run the command inside the image*
*if the image is not found locally, it will be downloaded from the registry*
*option `-i` starts the container in interactive mode*
*option `-t` allocates a pseudo TTY*

`docker ps [-as]`

*list all running containers*
*option `-a` will list all containers including the stopped ones.*
*option `-s` will list the container's size.*

# Commands (2)

`docker rm <container>`
*remove the container*

`docker rmi <image>`
*remove the image*

`docker commit <container> <name[:version]>`
*create an image from the container with the name and the version*

`docker history <image>`
*display the image history*

# Networking and Linking

- There are 3 docker networks by default
  - ***bridge*** *– container can access host's network (default)*
    - *→ Docker creates subnet* `172.17.0.0/16` *and gateway to the network*
    - *→ When a container is started, it is automatically added to this network*
    - *→ All containers in this network can communicate with each other*
  - ***host*** *– all host's network interfaces will be available in the container.*
  - ***none*** *– container will be placed on its own network and no network interfaces will be configured.*
- Custom Network configuration
  - *You can create a new network and add containers to it*
  - *Containers in the new network can communicate with each other but the network will be isolated from the host network*
- Linking containers (legacy)

```
$ docker run -d --name redmine-db postgres
$ docker run -it --link redmine-db:db postgres /bin/bash
root@c4b12143ebe8:/# psql -h db -U postgres
psql (9.6.1)
Type "help" for help.
postgres=# SELECT inet_server_addr();
postgres=# SELECT * FROM pg_stat_activity \x\g\x
```

# Networking Commands

```
docker network ls
```

*lists all available networks*

```
docker network inspect <network-id>
```

*Returns the details of specific network*

```
docker network create --driver bridge isolated_nw
```

*creates a new isolated network*

```
docker run -it --network=isolated_nw ubuntu bin/bash
```

*starts the container ubuntu and attaches it to the isolated network*

# Data Volumes

- Data Volume
  - *A directory that bypass the union file system*
  - *Data volumes can be shared and reused among containers*
  - *Data volume persists even if the container is deleted*
  - *It is possible to mount a shared sotrage volume as a data volue by using a volume plugin to mount e.g. NFS*
- Adding a data volume

  ```
  docker run -d -v /webapp training/webapp python app.py
  ```

  *will create a new value with name* `webapp`*,*

  *the location of the volume can be determined by using* `docker inspect`*.*
- Mount a host directory as a data volume

  ```
  docker run -d -v /src/webapp:/webapp training/webapp python app.py
  ```

  *if the path exists in the container, it will be overlayed (not removed),*

  *if the host directory does not exist, the docker engine creates it.*
- Data volume container
  - *Persistent data to be shared among two or more containers*

  ```
  docker create -v /dbdata --name dbstore training/postgres /bin/true
  docker run -d --volumes-from dbstore --name db1 training/postgres
  docker run -d --volumes-from dbstore --name db2 training/postgres
  ```

# Dockerfile

- Dockerfile is a script that creates a new image

```
# This is a comment
FROM oraclelinux:7
MAINTAINER Tomas Vitvar <tomas@vitvar.com>
RUN yum install -q -y httpd
EXPOSE 80
CMD httpd -X
```

- A line in the Dockerfile will create an intermediary layer

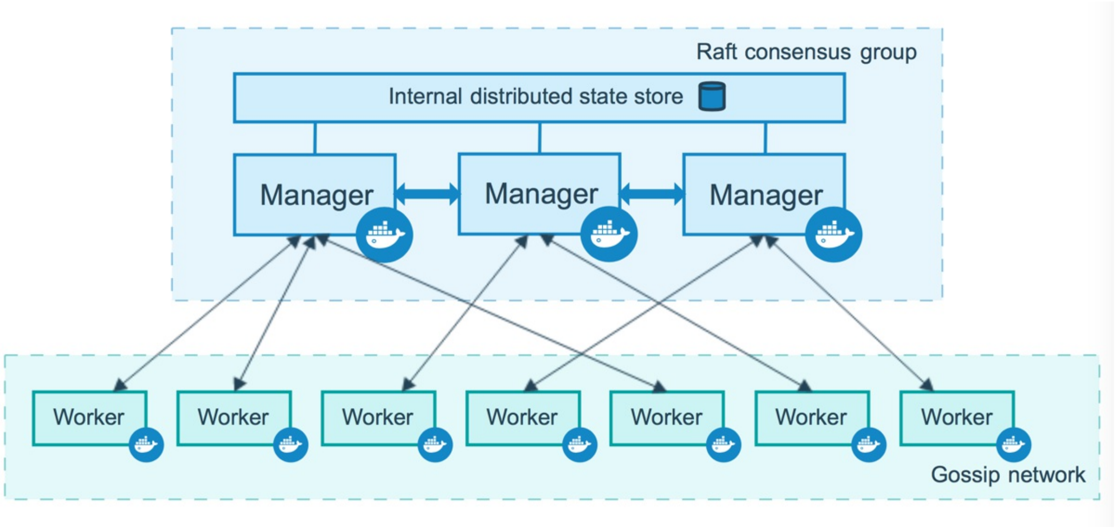
```
$ docker build -t tomvit/httpd:v1 .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM oraclelinux:7
 ---> 4c357c6e421e
Step 2 : MAINTAINER Tomas Vitvar <tomas@vitvar.com>
 ---> Running in 35feebb2ffab
 ---> 95b35d5d793e
Removing intermediate container 35feebb2ffab
Step 3 : RUN yum install -q -y httpd
 ---> Running in 3b9aee3c3ef1
 ---> 888c49141af9
Removing intermediate container 3b9aee3c3ef1
Step 4 : EXPOSE 80
 ---> Running in 03e1ef9bf875
 ---> c28545e3580c
Removing intermediate container 03e1ef9bf875
Step 5 : CMD httpd -X
 ---> Running in 3c1c0273a1ef
```

*If processing fails at some step, all preceeding steps will be loaded from the cache on the next run.*

# Overview

- Microservices Architecture
- Docker
  - *Overview*
  - *Image Layering*
  - *Working with Docker*
  - *Swarm*
- Kubernetes

# Swarm

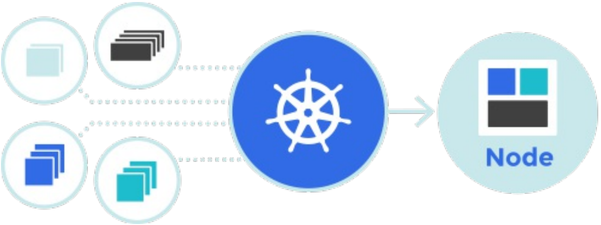


# Overview

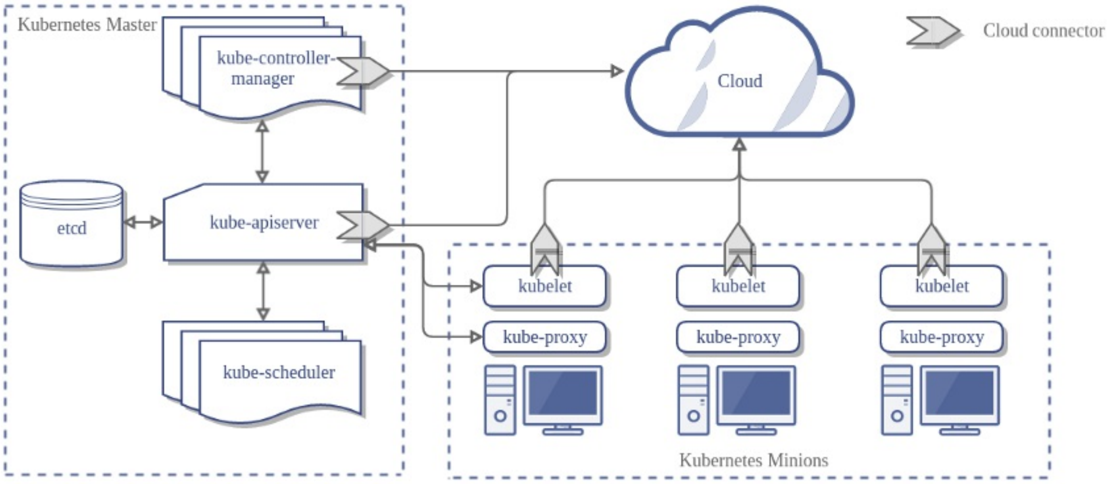
- Microservices Architecture
- Docker
- Kubernetes

# Overview

- In your architecture...
  - *Containers are atomic pieces of application architecture*
  - *Containers can be linked (e.g. web server, DB)*
  - *Containers access shared resources (e.g. disk volumes)*
- Kubernetes
  - *Automation of deployments, scaling, management of containerized applications across number of nodes*
  - *Based on Borg, a parent project from Goolge*



# System Architecture



# Major Terms

- Node
  – *a worker machine in Kubernetes, previously known as a minion (a VM or physical machine). It uses* `kubelet` *and* `kube-proxy` *to communicate with the master and other nodes/services.*
- Master
  – *A node that manages the cluster of nodes.*
- Pod
  – *The basic building block of Kubernetes, one or more dependant containers.*
- Service
  – *A set of pods with rules allowing pods to talk to each other, such as:*
  → `NodePort` *exposes the pod under a cluster IP.*
  → `LoadBalancer` *exposes the pod for load balancing by external load balancer*
- Controllers
  – *Worker units to ensure a desired state, such as:*
  → `ReplicaSet` *ensures that a specified number of pod replicas are running.*
  → `Deployment` *manages ReplicaSets, provides declarative updates to pods.*
  → `StatefulSet` *manages deployment and scaling of a set of Pods.*

# Features

- Automatic binpacking
  – *Automatically places containers onto nodes based on their resource requirements and other constraints.*
- Horizontal scaling
  – *Scales your application up and down with a simple command, with a UI, or automatically based on CPU usage.*
- Automated rollouts and rollbacks
  – *Progressive rollout out of changes to application/configuration, monitoring application health and rollback when something goes wrong.*
- Storage orchestration
  – *Automatically mounts the storage system (local or in the cloud)*
- Self-healing
  – *Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to user-defined health checks.*
- Service discovery and load balancing
  – *Gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.*

# Demo

- Environment Setup

  `minikube` – *a local virtual machine (running a master and a single node)*

  `kubectl` – *CLI to access Kubernetes cluster*

- Steps

  1. *create* `hello-node` *app in node.js and test it [see server.js]*

     ```
     node server.js
     ```

  2. *create docker image for the app [see Dockerfile]*

     ```
     docker build -t hello-node:v1 .
     ```

  3. *deploy the app to Kubernetes by using* `kubectl`

     ```
     kubectl run hello-node --image=hello-node:v1 --port=8080
     ```

  4. *Expose the app as a load balancer service.*

     ```
     kubectl expose deployment hello-node --type=LoadBalancer
     ```

  5. *Explore the app in minikube dashboard.*

     ```
     minikube dashboard
     ```

  6. *Fire requests at the service and count them [see test.sh]*

     `./test.sh`.

  7. *Change the number of replicas by using the dashboard or* `kubectl`.