Web 2.0
Lecture 9: Cloud Architecture

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

Overview

- Introduction
- Cloud Architecture
- Docker Containers
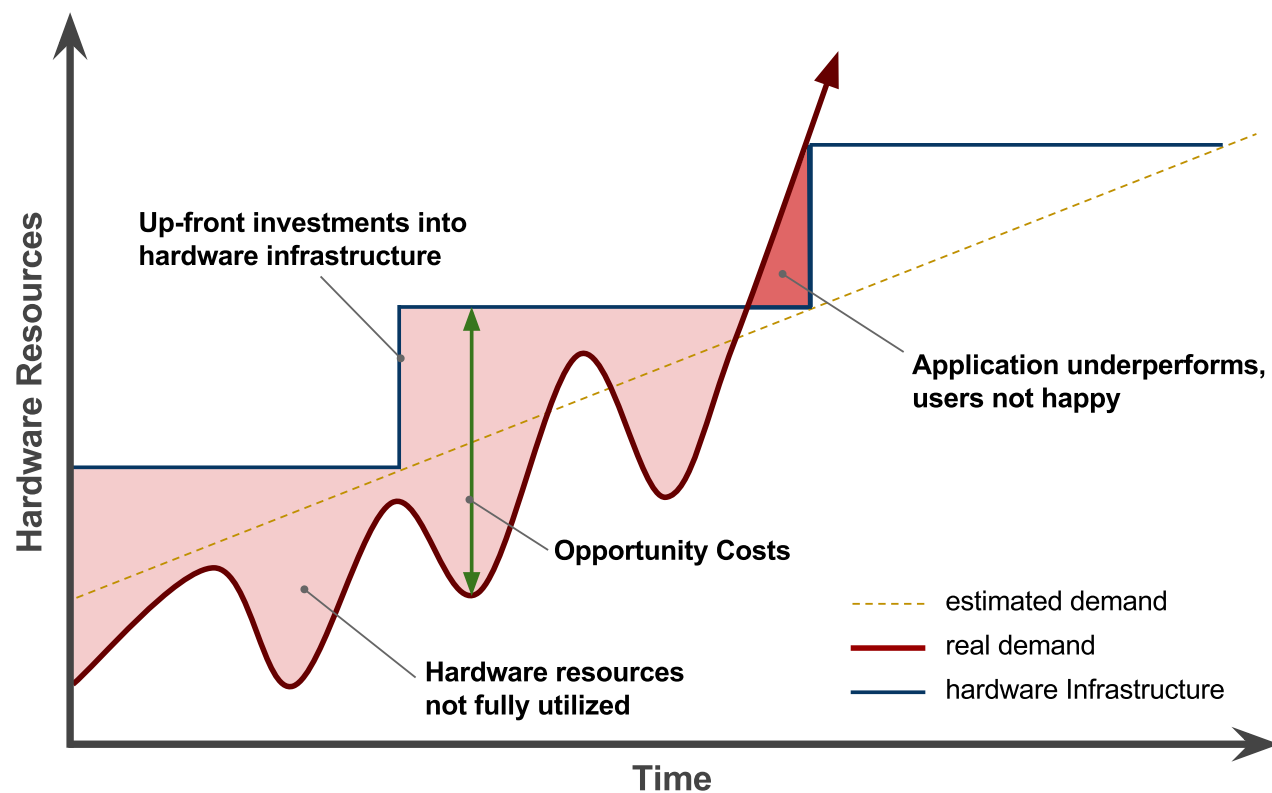
What is a Cloud?

- A different way of thinking
  - *Got your grand mum's savings under your pillow?*
    → probably not, you better have them in your bank
  - *Data is your major asset*
  - *you better have them in a "bank" too*
  - *Someone can abuse your data?*
  - *banks bankrupt too, sometimes – it is a risk you take*
  - *there is a market and a competition*

- Outsourcing of application infrastructure
  - *Reliability and availability*
  - *Low costs – pay-per-use*
  - *Elasticity – can dynamically grow with your apps*

What is a Cloud?

- Any app you access over the web?

- A datacenter?
  - *Offers virtualization*
  - *Any company having a datacenter wants to move to*

- Cloud provider should also offer services, such as:
  - *scalability, storage*
  - *Possible to configure programmatically*
    - → *integration to enterprise administration processes*
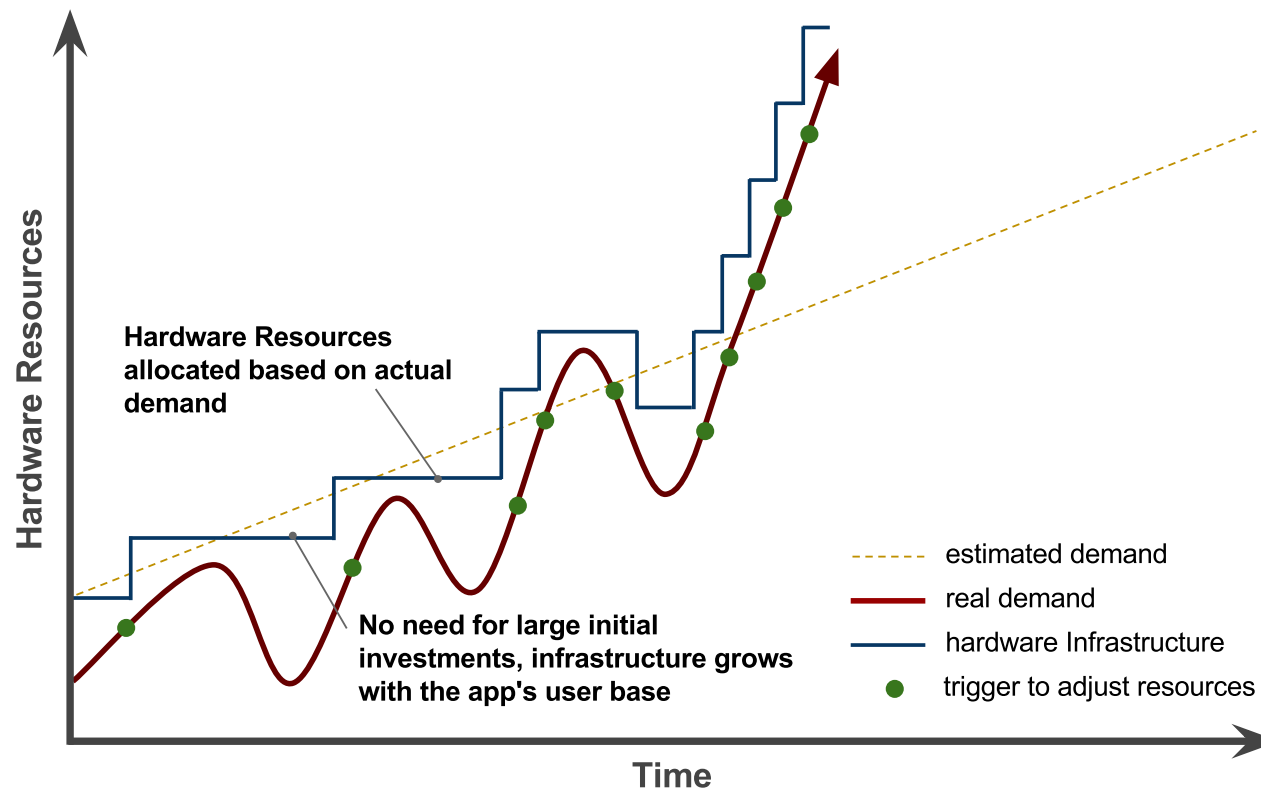   - → *usually REST interface*

Traditional Solution to Infrastructure

- Traditional hardware model
  - *Up-front hardware investments*
  - *Hardware not optimally utilized*

Up-front investments into hardware infrastructure

Application underperforms, users not happy

Opportunity Costs

Hardware resources not fully utilized

Hardware Resources

Time

- - - estimated demand
- real demand
- hardware Infrastructure

Good Performance – Cloud Solution

- Cloud Computing model
  - *No up-front hardware investments*
  - *Hardware optimally utilized*

Hardware Resources allocated based on actual demand

No need for large initial investments, infrastructure grows with the app's user base

- - - estimated demand
— real demand
— hardware Infrastructure
● trigger to adjust resources

Cloud Computing Concepts

- **Resource Pooling**
  - *Resources reused by multiple tenants (multitenancy)*
  - *Resources: CPU, memory, storage, network*

- **On-demand and Self-service**
  - *Resources are provisioned as they are requested and when they are required*
  - *No human interaction, automatic*

- **Scalability and Elasticity**
  - *Infrastructure may grow and shrink according to needs*
  - *Automatic or manual*

- **Pay-per-use**
  - *Consumers only pay for resources when they use them*

Cloud Computing Concepts (Cont.)

- Service Models (aka Cloud Layers)
  - *IaaS – Infrastructure as a Service*
  - *PaaS – Platform as a Service*
    - $\rightarrow$ *MWaaS, DBaaS, ...*
  - *SaaS – Software as a Service*

- Deployment Models
  - *Public Cloud*
  - *Private Cloud*
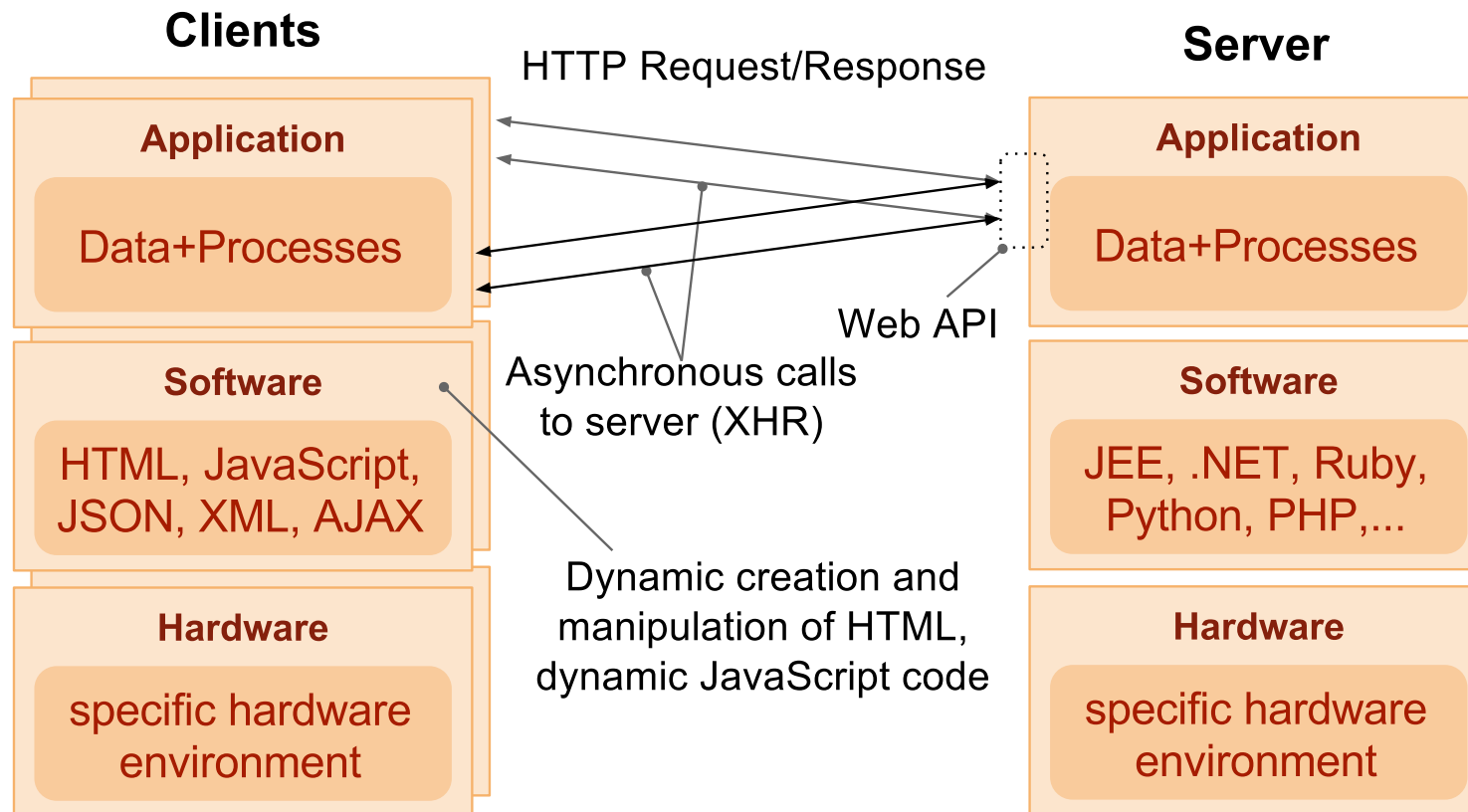 - *Hybrid Cloud*

Cloud Provider Example – Amazon AWS
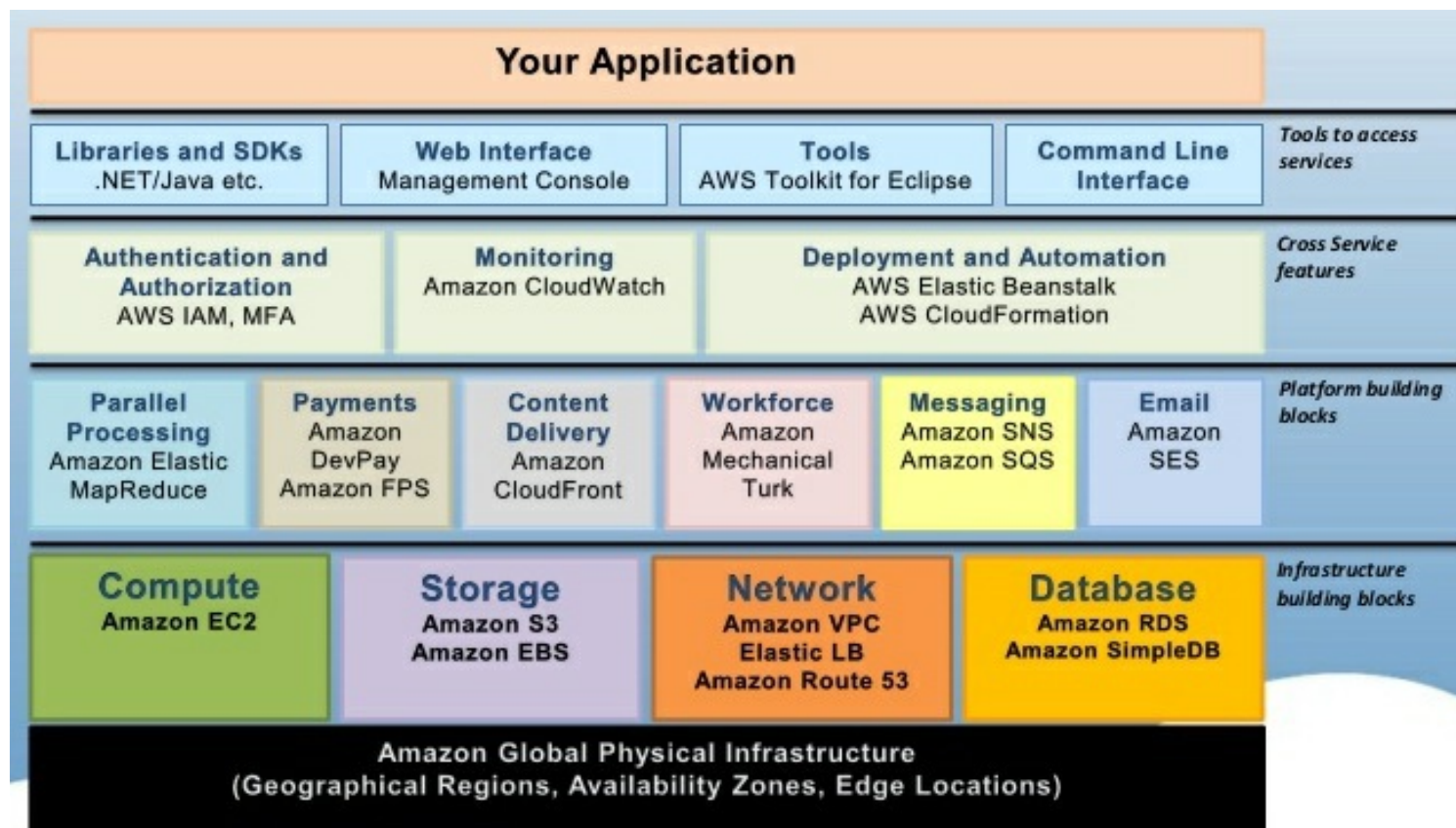
Overview

- Introduction
- Cloud Architecture
  - *Service Models*
  - *Multitenancy*
- Docker Containers

Web 2.0 Web Architecture

Clients

HTTP Request/Response

Server

Application

Data+Processes

Application

Data+Processes

Web API

Asynchronous calls
to server (XHR)

Software

HTML, JavaScript,
JSON, XML, AJAX

Software

JEE, .NET, Ruby,
Python, PHP,...

Dynamic creation and
manipulation of HTML,
dynamic JavaScript code

Hardware

specific hardware
environment

Hardware

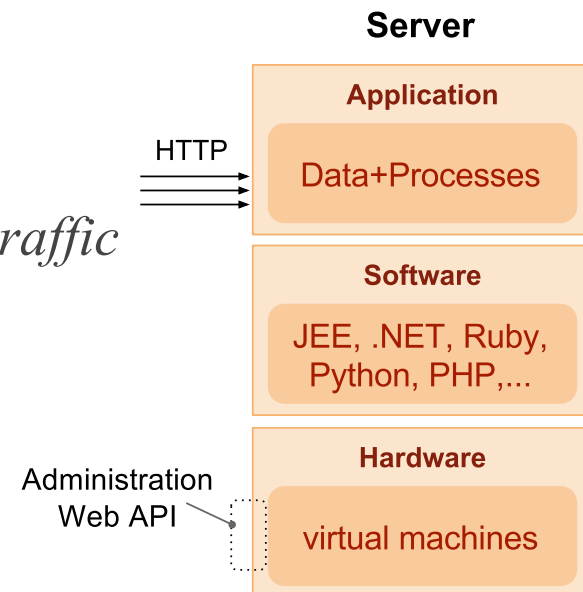specific hardware
environment

IaaS: Infrastructure as a Service

- Provides basic computing resources and services for application providers
  - *Services for application providers*
  - *A consumer is able to deploy and run arbitrary software*
- Infrastructure implications
  - *Exposing of infrastructure resources through abstraction*
  - *Support for infrastructure resources – compute (hardware/OS/VM), storage, network, etc.*
  - *Supports isolation for multitenant environments*

IaaS: Infrastructure as a Service

- Usage
  - *Predefined machine instances (micro, small, large, extra-large)*
    → Linux OS, 613 MB of memory, 30 GB of Storage, Load Balancer, etc.
  - *Pay-per-use – pay for resources you use (time or amount); no up-front costs*

- IaaS Services Examples
  - *Elastic Storage*
  - *Monitoring resources*
    → Amazon CloudWatch)
  - *Auto Scalling of running instances*
  - *Load Balancing – distributing incoming traffic across multiple instances*

- IaaS providers
  - *Amazon EC2, GoGrid, Rackspace, OpenNebula, ...*

Server

Application

HTTP

Data+Processes

Software

JEE, .NET, Ruby, Python, PHP,...

Hardware

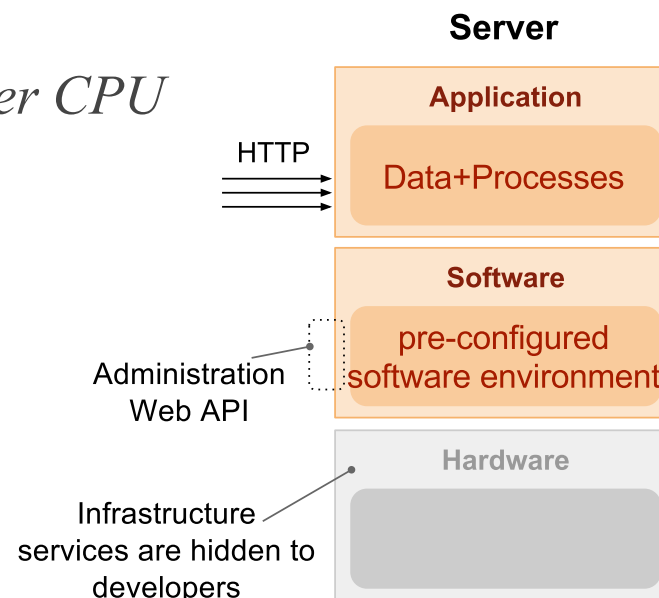Administration Web API

virtual machines

PaaS: Platform as a Service

- Provides scalable platform for applications
  - *Services for application providers*
  - *No costs of buying and managing underlying infrastructure $\rightarrow$ hardware and software*

- Infrastructure implications
  - *Scalable platform, deploy on-demand*
  - *Self service interface to deploy applications and services*
  - *Support for monitoring and measuring platform usage*
  - *Model supporting isolation in multi-tenant environments*

PaaS: Platform as a Service

- Usage
  - *Choose software platform, e.g., JEE, .NET, Python, etc.*
  - *Pay-per-use – pay for the resources you use; no up-front costs*

- PaaS features
  - *Auto Scalling and Load Balancing of applications*
  - *Persistent Storage - usually NoSQL database*
  - *Local development environment*
  - *Backends – for app instances with higher CPU and memory demands*
  - *Administration APIs for its services*

- PaaS providers
  - *Google App Engine, Heroku, Windows Azure, etc.*

- Limitations
  - *HTTP request limit (30 - 60 sec)*
  - *No writes to file system, no thread support*

Server

Application

HTTP

Data+Processes

Software

Administration
Web API

pre-configured
software environment

Hardware

Infrastructure
services are hidden to
developers

SaaS: Software as a Service

- Software delivery model for applications hosted in the cloud
  - *typically software for end-users*
  - *services accessed using a web browser*
  - *provides API for programmatic access*

- SaaS characteristics
  - *Typically build on top of IaaS or PaaS*
  - *Configurable and customizable modern Web applications*
  - *Usually basic version for free, need to pay for pro version*
  - *Global availability - any computer, any device*
  - *Easy management - automatic and fast updates*
  - *Pay-per-use – pay for the time you use*

- SaaS providers
  - *Google Apps, Salesforce, iCloud, Flickr, Picasa, ...*

Overview

- Introduction
- Cloud Architecture
  - *Service Models*
  - *Multitenancy*
- Docker Containers

Multitenancy

- Architectural approach where resources are shared between multiple tenants or consumers

- Implications
  - *Centralization of infrastructure in locations with lower costs*
  - *Peak-load capacity increases*
  - *Utilisation and efficiency improvements for systems that are not well utilised*

- Sharing options
  - *Shared Everything*
  - *Shared Infrastructure*
    - → *Virtual Machines*
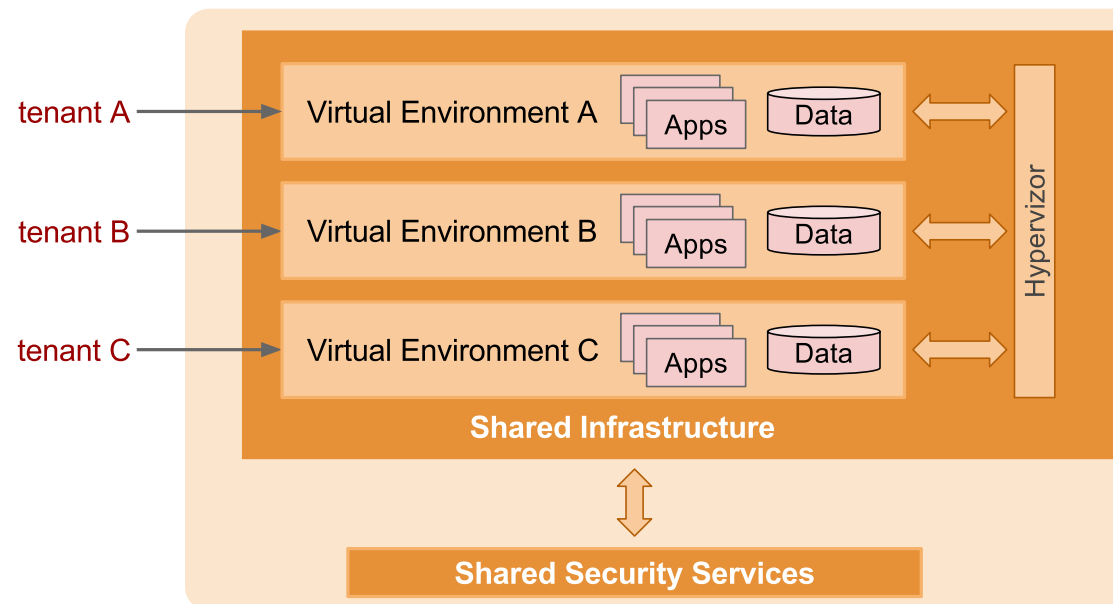   - → *O/S virtualization*

Shared Everything

- Resources are shared between all tenants or consumers
  - *tenant: a service consumer*

- Common for the SaaS model

- The application should provide tenant isolation

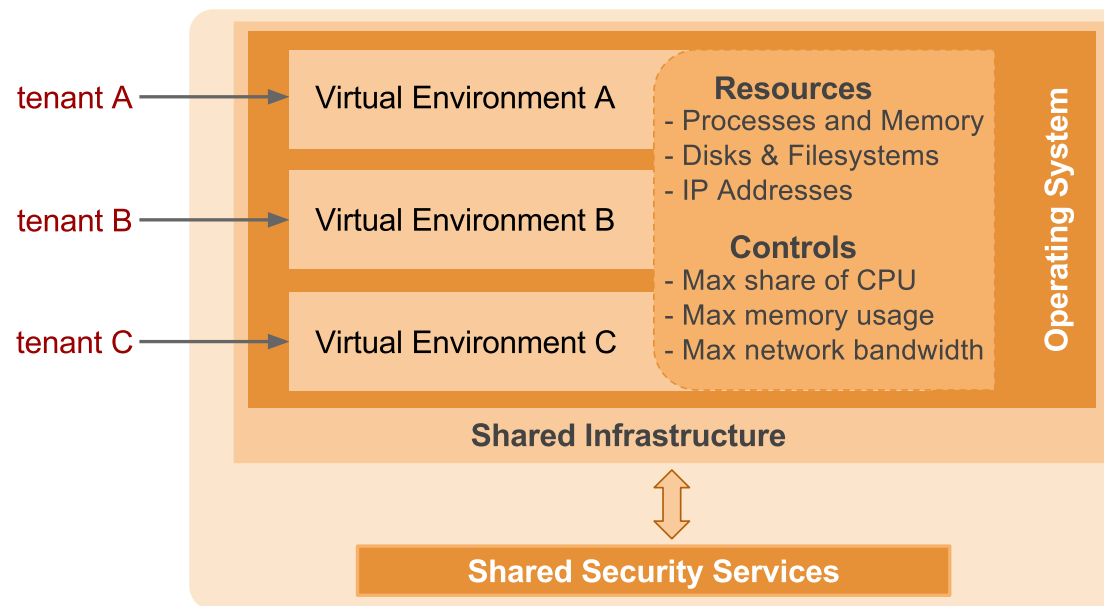- Data for multiple tenants is stored in the same database tables
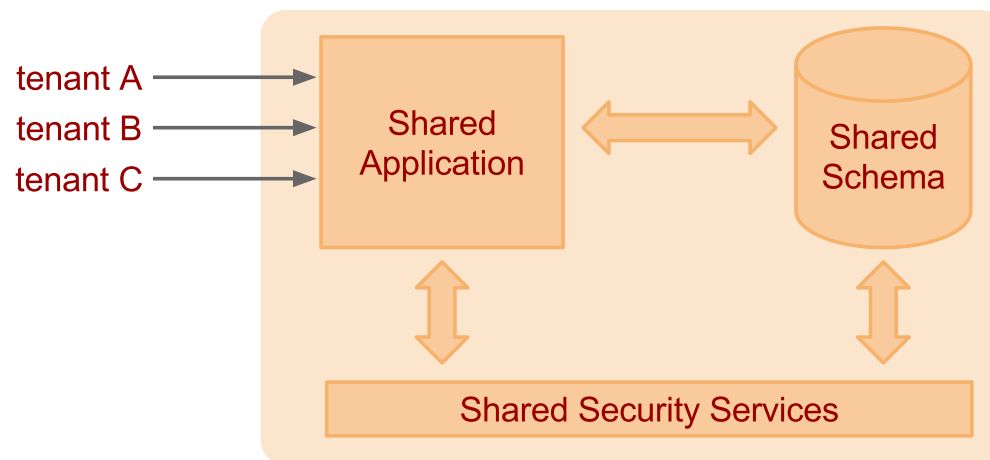
Shared Infrastructure: Virtual Machines

- Infrastructure shared via virtual machines
  - *each tenant has its own virtual environment*
  - *Isolation provided by hypervisor*
    - → *hypervisor: virtual machine manager, runs virtual machines*
  - *Resource contention depends on VM capability and configuration*
  - *Adds an additional layer and processes to run and manage*

Shared Infrastructure: OS Virtualization

- Infrastructure shared via OS Virtualization
  - *Each tenant has its own processing zone*
  - *Isolation provided by the operating system*
  - *Resource contention depends on zone configuration*
  - *No VMs to run and manage, no abstraction layer between app & OS*

tenant A → Virtual Environment A

tenant B → Virtual Environment B

tenant C → Virtual Environment C

Resources
- Processes and Memory
- Disks & Filesystems
- IP Addresses

Controls
- Max share of CPU
- Max memory usage
- Max network bandwidth

Operating System

Shared Infrastructure

Shared Security Services

Overview

- Introduction

- Cloud Architecture

- Docker Containers
  - *Overview*
  - *Image Layering*
 - *Working with Docker*
  - *Swarm*

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 new Kernel features: control groups (cgroups), kernel namespaces, union-capable 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*

VM vs. Docker Containers
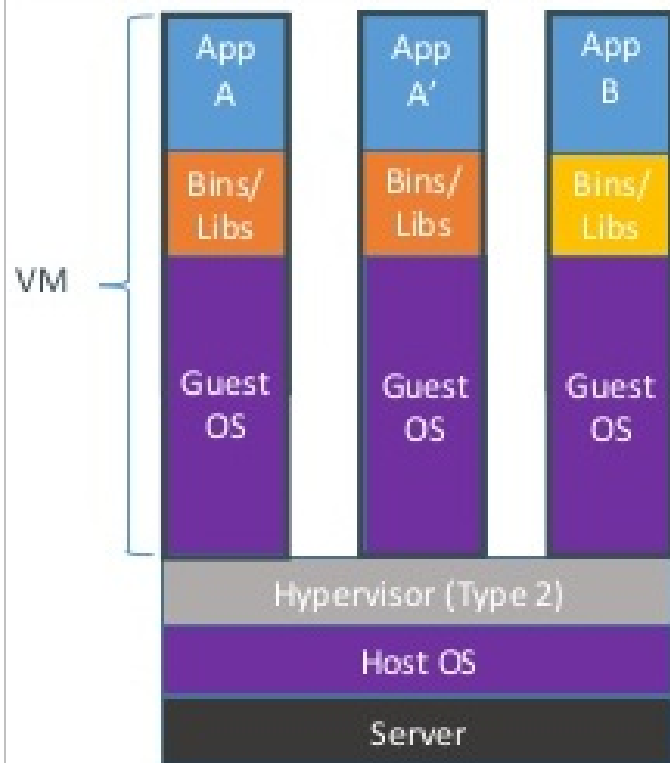
Containers are isolated, but share OS and, where appropriate, bins/libraries
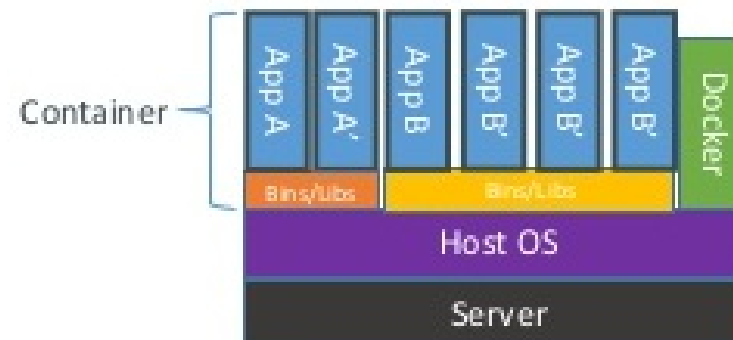
Docker Basic Terms

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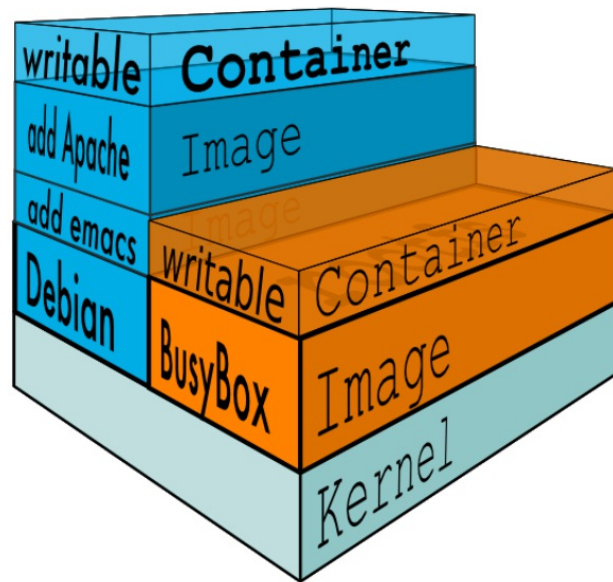
– *Containers are made up of R/O layers via a storage driver (OverlayFS, AUFS, etc.)*
– *Containers are designed to support a single application*
– *Instances are ephemeral, persistent data is stored in bind mounts or data volume containers.*

Overview

- Introduction
- Cloud Architecture
- Docker Containers
  - *Overview*
  - *Image Layering*
  - *Working with Docker*
  - *Swarm*

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.*

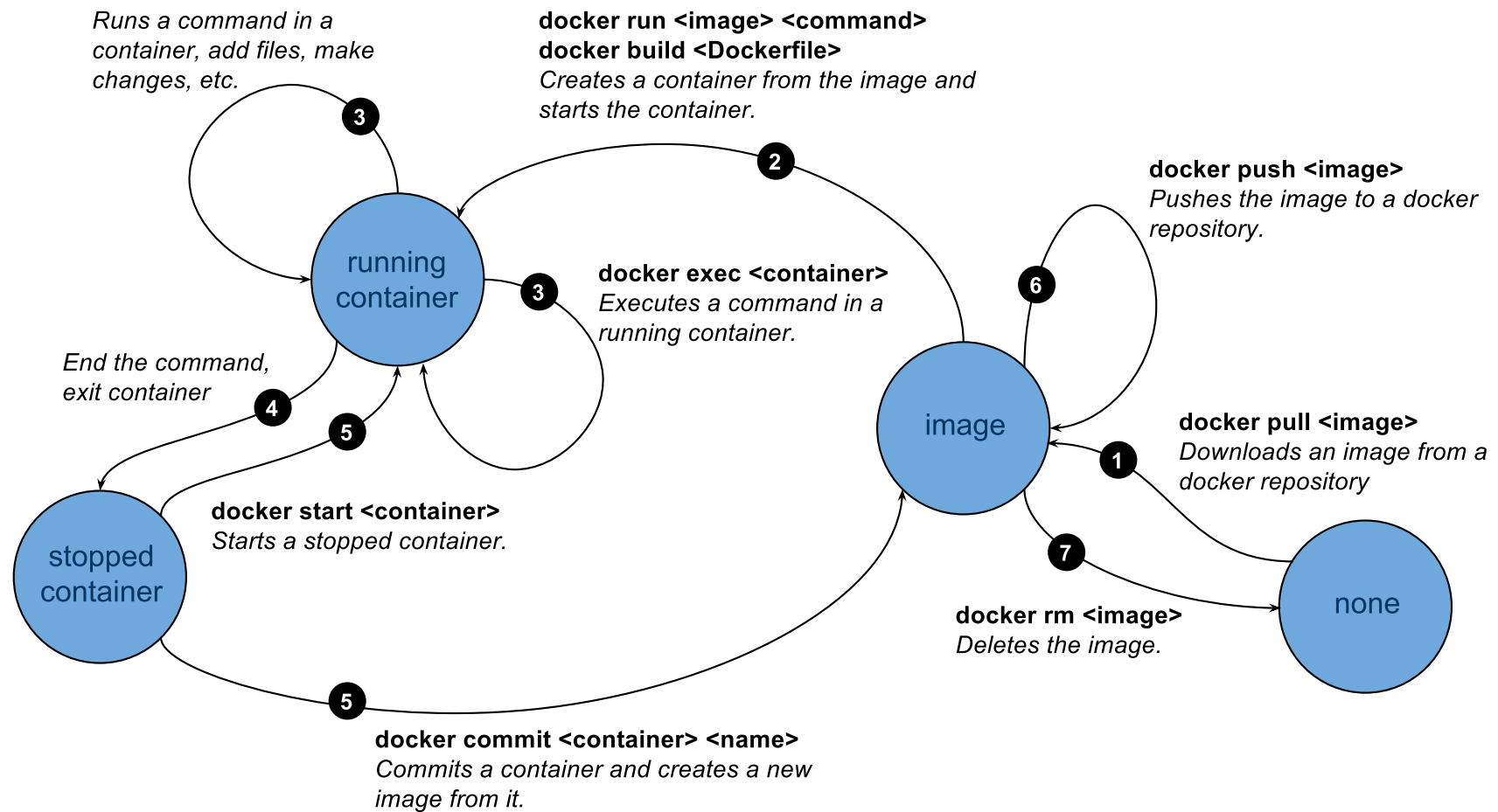- Directories on the file system

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

total 20
drwx------ 3 root root 4096 Jun 20 16:11 38f3ed2eac129654acef11c32670b534670c3a06e483fce313d72e3e0a15baa
drwx------ 3 root root 4096 Jun 20 16:11 55f1e14c361b90570df46371b20ce6d480c434981cbda5fd68c6ff61aa0a535
drwx------ 3 root root 4096 Jun 20 16:11 824c8a961a4f5e8fe4f4243dab57c5be798e7fd195f6d88ab06aea92ba93165
drwx------ 3 root root 4096 Jun 20 16:11 ad0fe55125ebf599da124da175174a4b8c1878afe6907bf7c78570341f30846
drwx------ 3 root root 4096 Jun 20 16:11 edab9b5e5bf73f2997524eebeac1de4cf9c8b904fa8ad3ec43b3504196aa380
```

  – *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

- Introduction
- Cloud Architecture
- Docker Containers
  - *Overview*
  - *Image Layering*
  - *Working with Docker*
  - *Swarm*

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.

running container

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

image

End the command, exit container

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

docker start <container>
Starts a stopped container.

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 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

- Introduction

- Cloud Architecture

- Docker Containers
  – *Overview*
  – *Image Layering*
  – *Working with Docker*
  – *Swarm*

Swarm