Large Scale Data Management

Ricardo Vilaça

rmvilaca@di.uminho.pt

https://rmpvilaca.github.io/

Cloud Computing

Cloud Computing

- Machines operated by a third party in large data centers
  - sysadmin, electricity, backup, maintenance externalized
- Rent access by the hour
  - Renting machines (Linux boxes): Infrastructure as a Service
  - Renting systems (Redshift SQL): Platform-as-a-service
  - Renting an software solution (Salesforce): Software-as-a-service
- {Cloud,Cluster} are independent concepts, but they are often combined!
  - Hadoop on Google Cloud Platform

Economics of Cloud Computing

- A major argument for Cloud Computing is pricing:
  - We could own our machines
    - and pay for electricity, cooling, operators
    - and allocate enough capacity to deal with peak demand
  - Since machines rarely operate at more than 30% capacity, we are paying for wasted resources
- Pay-as-you-go rental model
  - Rent machine instances by the hour
  - Pay for storage by space/month
  - Pay for bandwidth by space/hour
- No other costs
- This makes computing a commodity
  - Just like other commodity services (sewage, electricity etc.)

Cloud Services

- Cloud services divided into 3 main levels of abstraction:
  - Infrastructure-as-a-Service (IaaS)
  - Platform-as-a-Service (PaaS)
  - Software-as-a-Service (SaaS)

Cloud infrastructure

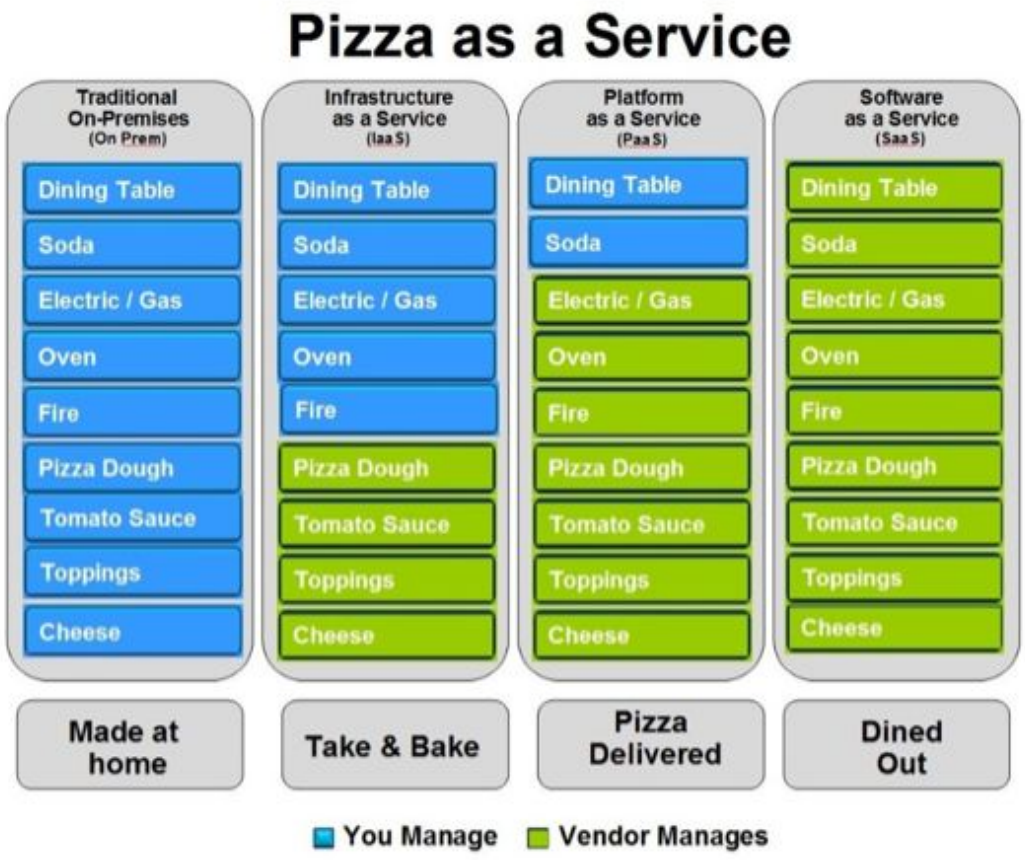
- Infrastructure-as-a-Service (IaaS):
  - provides virtualized hardware resources such as computing, storage and networking
  - resources are allocated on demand and in a pay-per-use fashion
  - an example of IaaS is Amazon EC2 (for computing) and Amazon S3 (for storage)

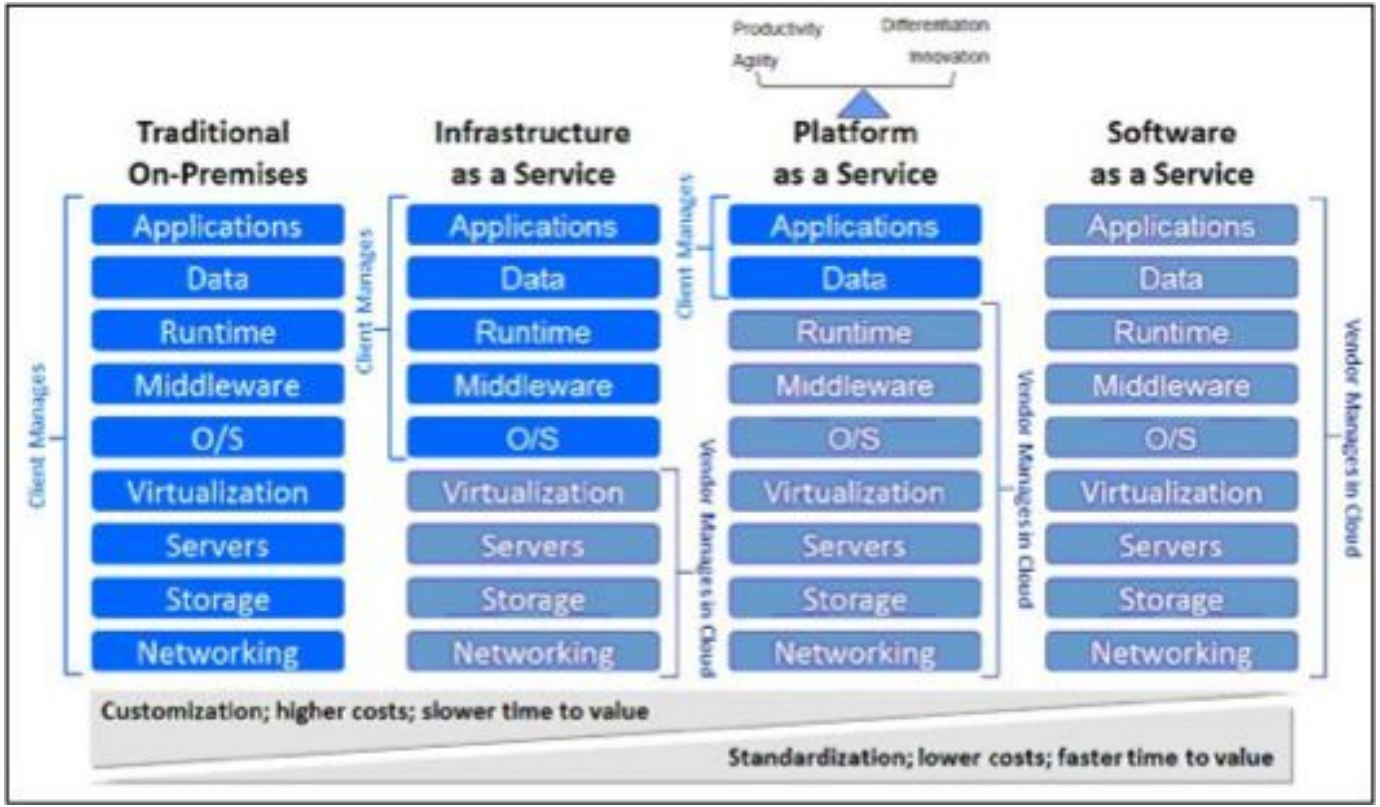
Cloud infrastructure

- Platform-as-a-Service (PaaS):
  - offers an encapsulation of a development environment abstraction that can be used to develop, deploy and run applications
  - examples include the Google App Engine

Cloud infrastructure

- Software-as-a-Service (SaaS):
  - features full applications or generic software like databases (also called DBaaS)
  - offered as a service and accessible as a web service or through a web browser
  - SalesForce.com and the Google Apps like Gmail are some well known instances of this type


Pizza as a Service
Traditional On-Premises (On Prem)
Infrastructure as a Service (IaaS)
Platform as a Service (PaaS)
Software as a Service (SaaS)
Dining Table
Soda
Electric / Gas
Oven
Fire
Pizza Dough
Tomato Sauce
Toppings
Cheese
Made at home
Take & Bake
Pizza Delivered
Dined Out
You Manage
Vendor Manages



https://www.ibm.com/blogs/cloud-computing/2011/10/paas-comes-of-age-why-platform-as-a-service-needs-to-be-part-of-your-cloud-strategy-2/

Infrastructure-as-a-Service

- Datacenters scattered around the world (Asia, Europe and, North and South America)
- Each with around 80 000 servers
- Top main players: Google, Amazon and Microsoft

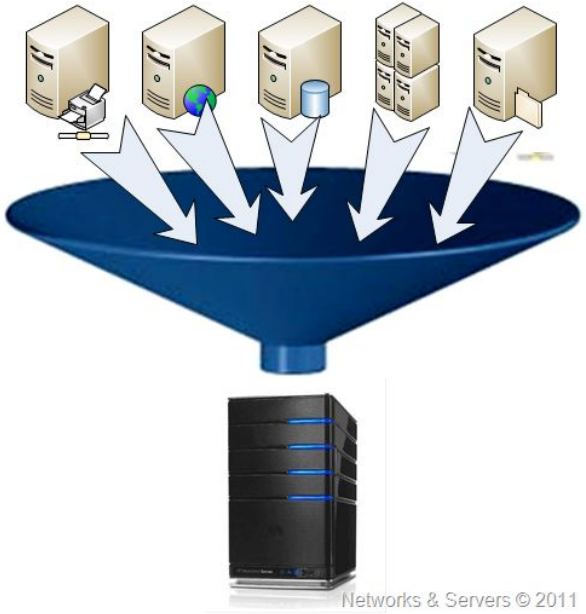
Infrastructure-as-a-Service

- How can we manage many datacenters each with thousands of servers?
  - Virtualization
    - Technique that allows creating a software-based virtual device or resource that, in practice, is an abstraction provided on top of existing hardware or software resources.

Virtualization

Server Virtualization

- Virtual Machines (VMs)

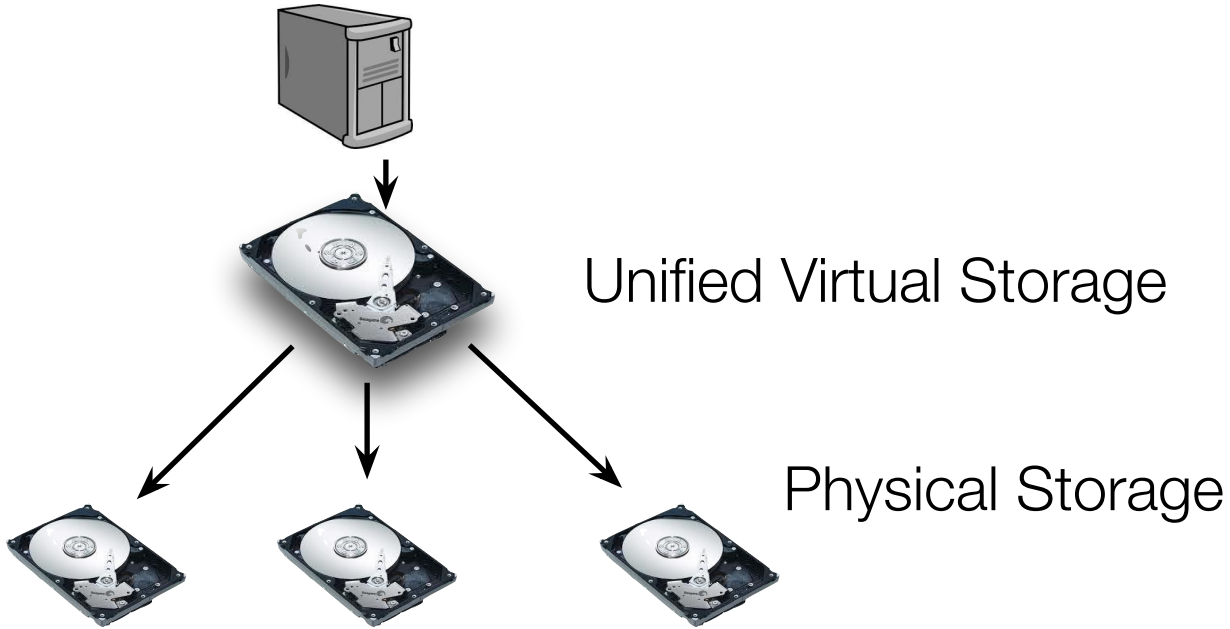


Storage Virtualization

- Disk partitioning

Storage Virtualization

- Distributed storage



Other Examples

- Network devices

  - Random Access Memory
  - Containers
  - …

Virtualization in Practice Virtual Desktop Infrastructures (VDIs)

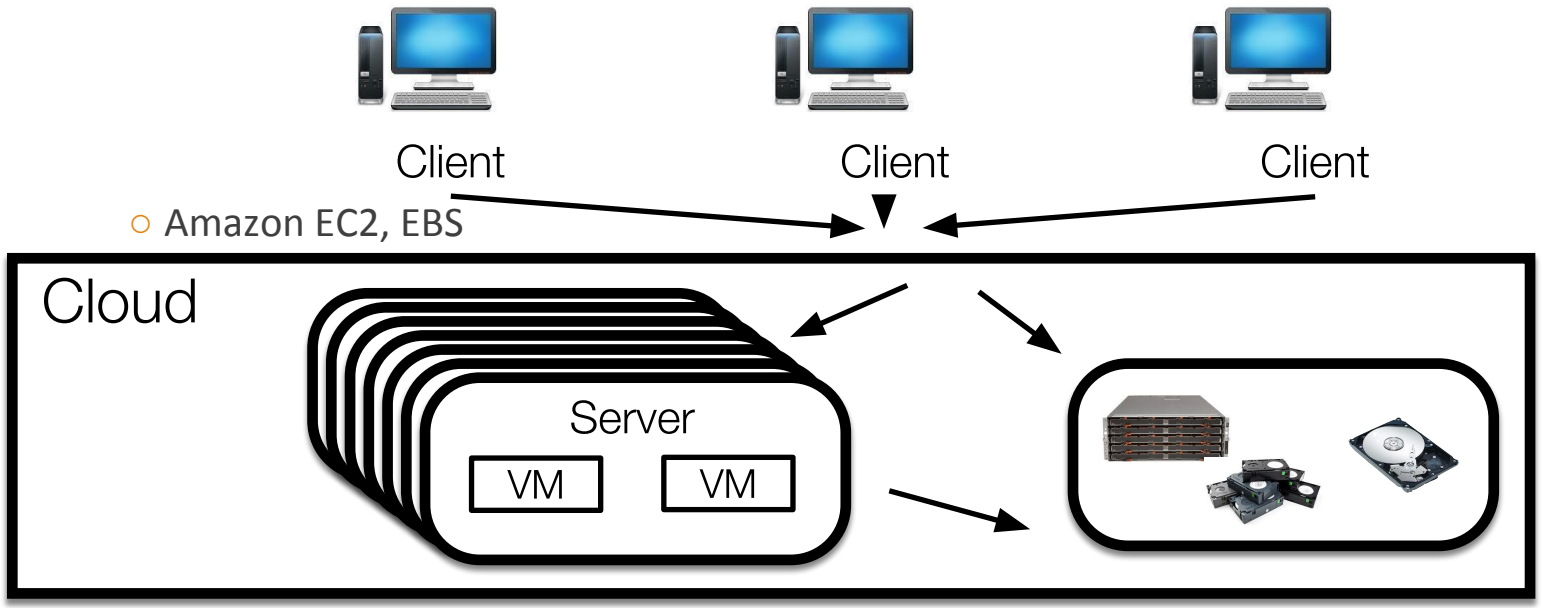
- VMWare Horizon 7
- Amazon WorkSpaces

Virtualization in Practice Storage Virtualization (Software-Defined Storage)

- Amazon S3, Glacier
- EMC ViPR

Logical Devices

Virtualization in Practice
Simplified Cloud deployment



- Amazon EC2, EBS

Advantages

Heterogeneity

- Virtual resources can be provided on top of different/multiple physical (hardware) resources
- A virtual resource can support different applications/operating systems while resorting to the same physical hardware (e.g., VMs)

Advantages

Transparency

- User interaction with virtual resources is similar to the interaction with a physical one

Advantages

Isolation

- Virtual resources are fully isolated from the underlying hardware/software in terms of:
  - Security
  - Performance
  - Failures (including OS/data corruption)

Advantages

Resource Optimization

- Physical resources can be leveraged to support more clients/applications
  - Server Consolidation
  - Lower Costs

Advantages

Simplified Management

- Managing a virtual resource is simpler than managing bare-metal resources.
  - High Availability
  - Dynamic Load Balancing
    - E.g., VMs Live Migration/Snapshots

Disadvantages

Performance

- The abstraction of resources often includes a performance penalty
  - CPU, Network, I/O

Disadvantages

Overprovisioning

- Deploying more virtualized resources than needed may lead to performance degradation

Disadvantages

Security/Privacy

- If isolation is not properly addressed or, a malicious user/sysadmin has access to the physical resources (e.g., server), security may be compromised

Disadvantages

Dependability

- The failure of physical resource may result in the failure of multiple virtual ones.

Disadvantages

Learning Curve

- Administrators need to understand and manage a new paradigm
- In some virtualization techniques, applications must be rewritten or need additional configuration to be deployable

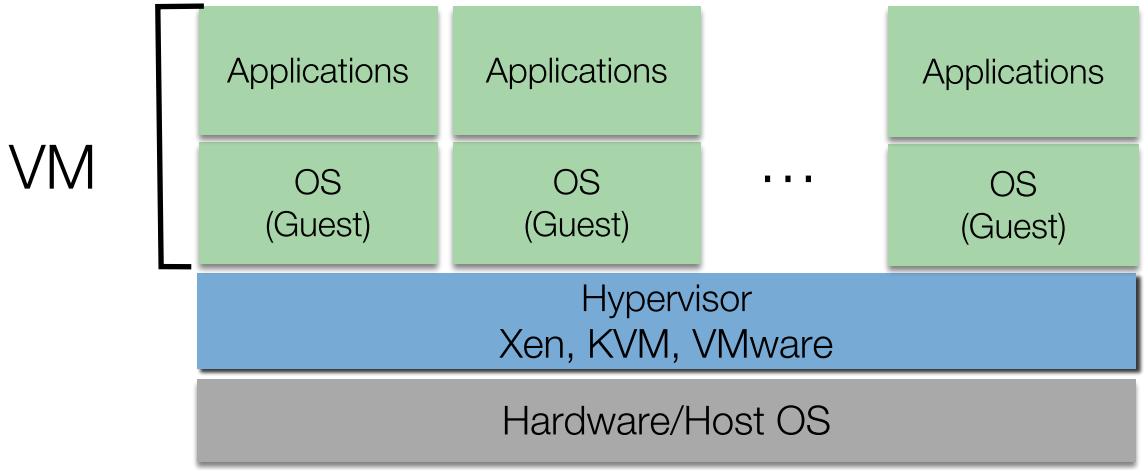
Virtual Machines

- Changing an application to run on different Operating Systems is a costly and hard task
- VMs allow running different operating systems on top of the same physical server

Virtual Machines (context)

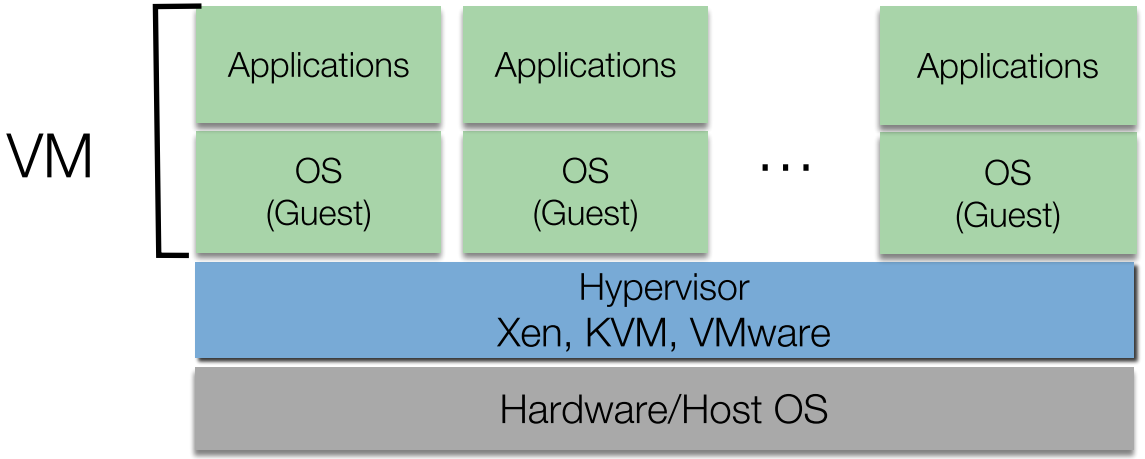
- IBM mainframe systems (from about 45 years ago) allowed applications to use isolated portions of a system's resources
- Virtualization became mainstream in the early 2000's with the X86 server architecture due to:
  - Under-utilised resources
  - Infrastructure costs

Virtual Machines Architecture



Virtual Machines Architecture

- Operations are intercepted by the Hypervisor and executed on the physical hardware
  - RAM
  - Disk
  - CPU
  - Network
  - ...



Virtual Machines
Hypervisor

- Controls the low-level interaction between VMs and the underlying host OS/hardware
  - access to shared disk, network, CPU, and RAM resources

Virtual Machines
Storage

- Each VM allocates a specific chunk of the host's storage capacity
- The storage system must handle multiple writers/readers efficiently
- Thin-provisioning (e.g., sparse allocation, copy-on-write)

Virtual Machines
CPU

- Time slicing - processing requests are sliced up and shared across the VMs
- Similar to running multiple processes in a physical host
- Overcommitting vCPUs may lead to poor performance

Virtual Machines Network

- VMs share network bandwidth (similarly to storage)
- Over provisioning is usually not a problem as all VMs in a host do not use the full bandwidth
- Each VM can be configured with a different network setup

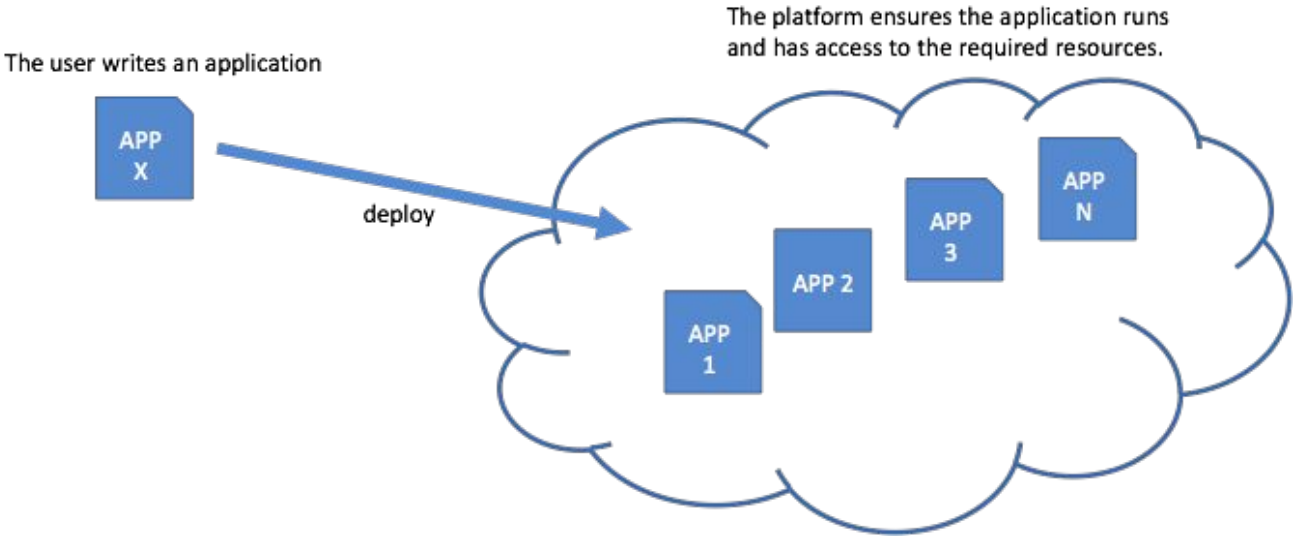
Virtual Machines
RAM

- VMs share RAM (similarly to storage)
- It is possible to perform memory reclamation across VMs deployed in the same host

From IaaS to PaaS

- From managed allocation and provision of resources to managed infrastructure
- Actual resources become transparent
- Focus on the application, which is the deployment item
- The interface is a programming environment, with APIs to IaaS/SaaS services
- The user can focus on the functionality to deploy instead on what are the resource requirements to support it
- Fast prototyping, easy deployment, managed elasticity and load balancing

PaaS


The user writes an application
APP X
deploy
The platform ensures the application runs and has access to the required resources.
APP 1
APP 2
APP 3
APP N

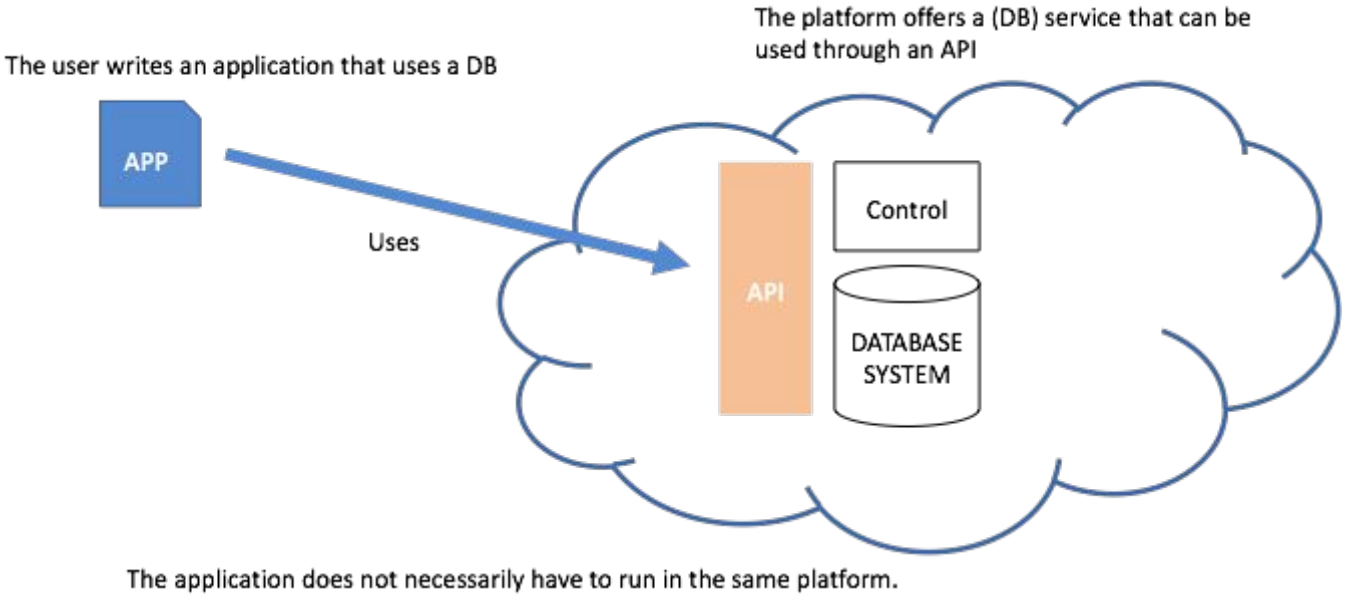
PaaS

- Example: Google App Engine
  - based on container instances
  - supports multiple languages (Java, Python, PHP, …)
  - and tools (Cloud SDK, IntelliJ IDEA, …)
  - and APIs (Google Cloud Storage, …)
  - versioning, testing, monitoring, logging, and security features
  - automatic elasticity and fault-tolerance

From PaaS to SaaS

- Specific services are provided
- Managed software components that export their traditional APIs but limited configuration interfaces
- Examples are database management systems
- There is no deployment item – the DB is exposed through a client and used as a traditional DB but with minimal configuration needed and with remote access
- Transparent management, elasticity and fault-tolerance

SaaS



Example: Amazon DynamoDB

IaaS, PaaS and SaaS: complex distributed systems

- Virtualization
- Provisioning
- Monitoring
- Reporting
- Billing
- Interoperability between all of the above

Advantages
IaaS, PaaS and SaaS

Convenience

- From IaaS:
  - avoid upfront costs on infrastructure management and hardware
  - “easily” deploy legacy applications
- to PaaS
  - focus on the application development itself and its requirements
  - powerful development, deployment, debugging, and benchmarking tools already in place
  - transparent elasticity and fault-tolerance
- to SaaS
  - leverage existing components (databases, web/application servers)
  - transparent elasticity and fault-tolerance

Fast

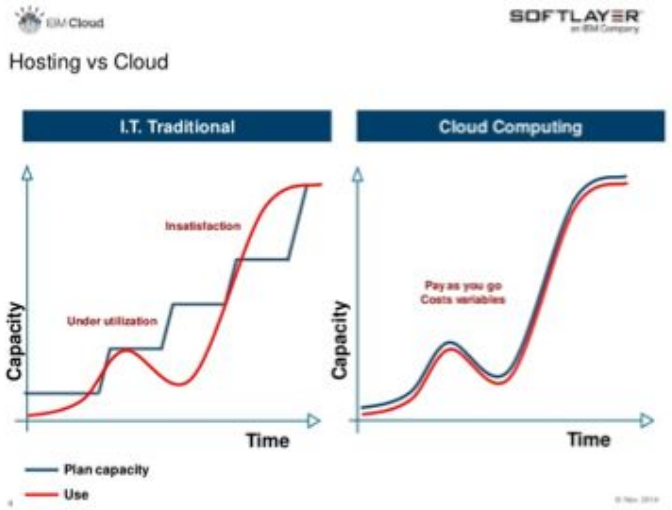
- From IaaS:
  - infrastructure is already installed and configured
- to PaaS
  - a development framework is already installed and configured
  - High degree of automation for deployment allows quick enhancement and bug fixing deploy
- to SaaS
  - Quick integration of different cloud software solutions

Elasticity

- From IaaS
  - illusion of virtually infinite resources
  - increase and decrease computational power, storage space and other resources according to demand (manual or resorting to third-party tools)
- to PaaS and SaaS
  - no need to manually manage elasticity

Flexible pricing model

- IaaS, PaaS and SaaS
  - no upfront costs
  - pay as you go allows to quickly correct possible over and under estimations with respect to system sizing
  - costs of infrastructure management and maintenance are included in the service



Disadvantages
IaaS, PaaS and SaaS

Loss of control

- Some provider hosts our data
  - But we can only access it using proprietary (non-standard) APIs
  - Lock-in makes customers vulnerable to price increases and dependent upon the provider
- Providers may control our data in unexpected ways:
  - July 2009: Amazon remotely remove books from Kindles
  - Twitter prevents exporting tweets more than 3200 posts back
  - Paying customers forced off Picasa towards Google Plus

Loss of control

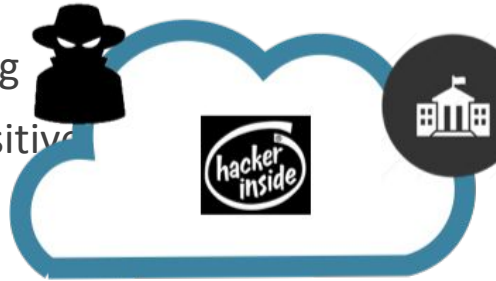
- From IaaS
  - no control over specific hardware and virtualization software
  - no possibility of fine tuning and optimizing the infrastructure
- to PaaS
  - no control over specific hardware and the PaaS platform software
  - possible performance bottlenecks out of the control of the application owner
  - management and monitoring is reduced to the tools provided by the platform
- to SaaS
  - third-party black-box cloud applications

Security

- IaaS, PaaS and SaaS
  - as secure as the provider – any vulnerability of the provider is a vulnerability of the application
  - fixes to vulnerabilities must be done by the provider
  - if (even if unlikely) the provider fails, the application fails and recovery is out of the control of the application owner
  - Breaches
    - Numerous examples of Web mail accounts hacked
    - Many many cases of (UK) governmental data loss
    - TJX Companies Inc. (2007): 45 million credit and debit card numbers stolen
    - Every day there seems to be another instance of private data being leaked to the public

Privacy

- IaaS, PaaS and SaaS
- Data is uploaded to third-party infrastructures
  - Who can access it?
  - Governments? Other people?
  - Snowden is the Chernobyl of Big Data
- Privacy guarantees needs to be clearly stated and kept-to
- Local laws (e.g. privacy) might prohibit externalizing data processing
- Unauthorised government or third-party access to private and sensitiv
- Data is no longer fully controlled by its rightful owner



Containers

Containers

- Lightweight virtual environment that groups and isolates a set of processes and resources (memory, CPU, disk, ...), from the host and any other containers
- Containers are an evolution of the chroot tool presented in 1982
- Aplications (group of processes) are encapsulated into containers that:
  - Provide an isolated execution environment

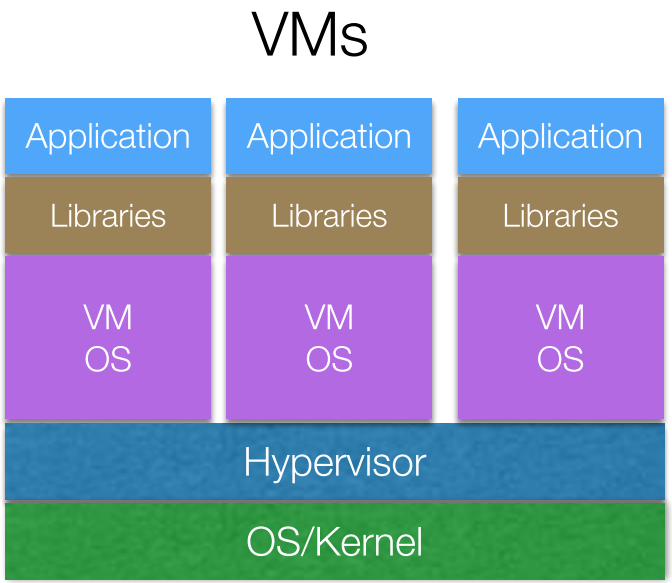
Why containers?

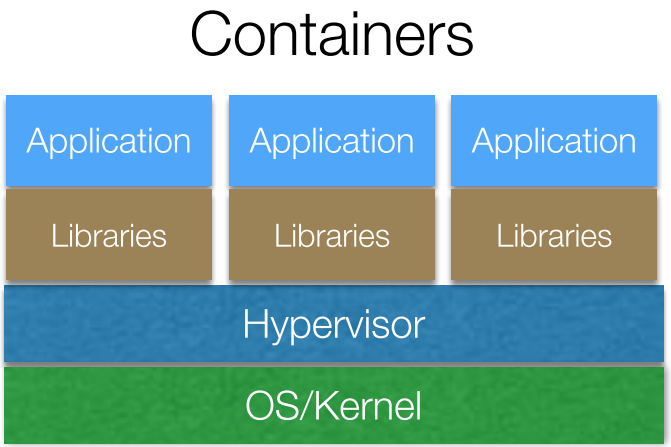
- Running different isolated versions of the same software/application (e.g., database) in a shared OS/Kernel environment
- Portability/migration across servers
- Easy packaging of applications and their dependencies

Linux containers

- Can be seen as a lightweight VM where it is not needed the simulation of all hardware resources of a real computer
- The host system is compartmentalised for each container in terms ofCPU and I/O (memory, disk, network)
- Each container shares the hardware and kernel/OS with the host system
- Each container is isolated from others

Linux Containers





Linux Containers - Types

- OS Containers
  - Containers are similar to VMs, running multiple processes
  - Kernel is shared, have independent OS
  - Flexibility/isolation vs performance
  - E.g., LXC
- Application Containers
  - Focused on deploying applications
  - Each application is seen as an independent process
  - Share the Kernel and OS (or some OS functionalities)
  - Flexibility/isolation vs performance
  - E.g., Docker

Linux Containers - Building Blocks

- Namespaces (Isolation)
  - Component of the Linux Kernel that makes a global host resource appear as a dedicated resource for a group of processes running in the same namespace
- Allows sharing of host resources across different containers without conflicts
  - Network, storage (filesystem), ...
- Control Groups (Resource Management)
  - Allows allocating resources (CPU, RAM, Disk I/O, network bandwidth) among groups of processes
  - In other words, this mechanism allows limiting the amount of resources used by each container and prioritising one container over others (e.g., The I/O of a critical database service)
- SELinux (Security)
  - Provides additional security over namespaces so that a container is not able to compromise the host system and other containers
  - Enforces access control and security policies

Comparison to VMs

- Advantages
  - Easier and Faster testing/provisioning/migration
  - Better resource utilisation and lower costs
  - Can be deployed on both physical and virtualized servers
  - Easier management
  - Performance
- Disadvantages
  - Weaker isolation/security (OS/Kernel are shared)
  - Flexibility in running different OSs (e.g., Linux, Windows, BSD, ...)

Docker

- Most widely-known container platform
- Supports Ubuntu, Fedora, RHEL, CentOS, Windows, etc
- https://www.docker.com

Union File system

- The container is created by deploying several file system layers on top of each other
- Each layer can be seen as an intermediate image that supports different services running on the container
- This approach allows reusing layers across different containers, saving disk space and optimising the time to provision new containers

Docker
Configuration File (Dockerfile)

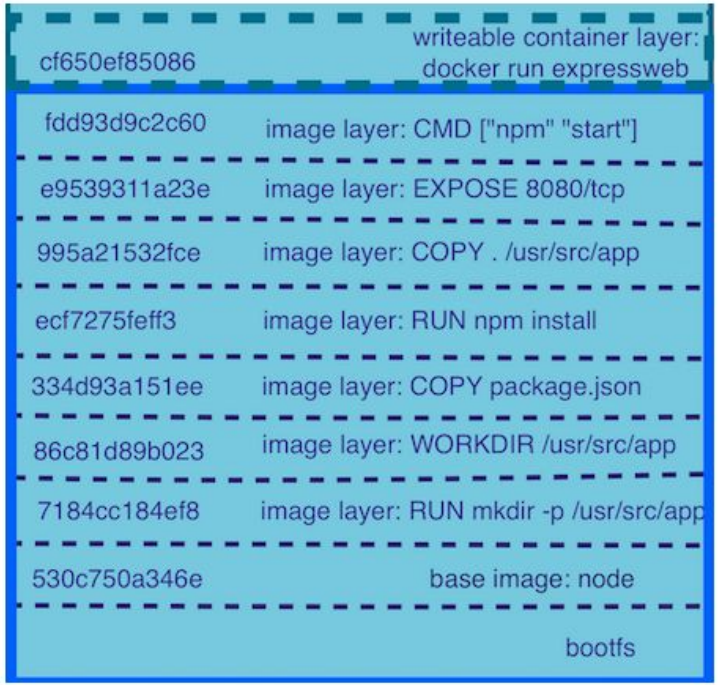
```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

Docker Deployment

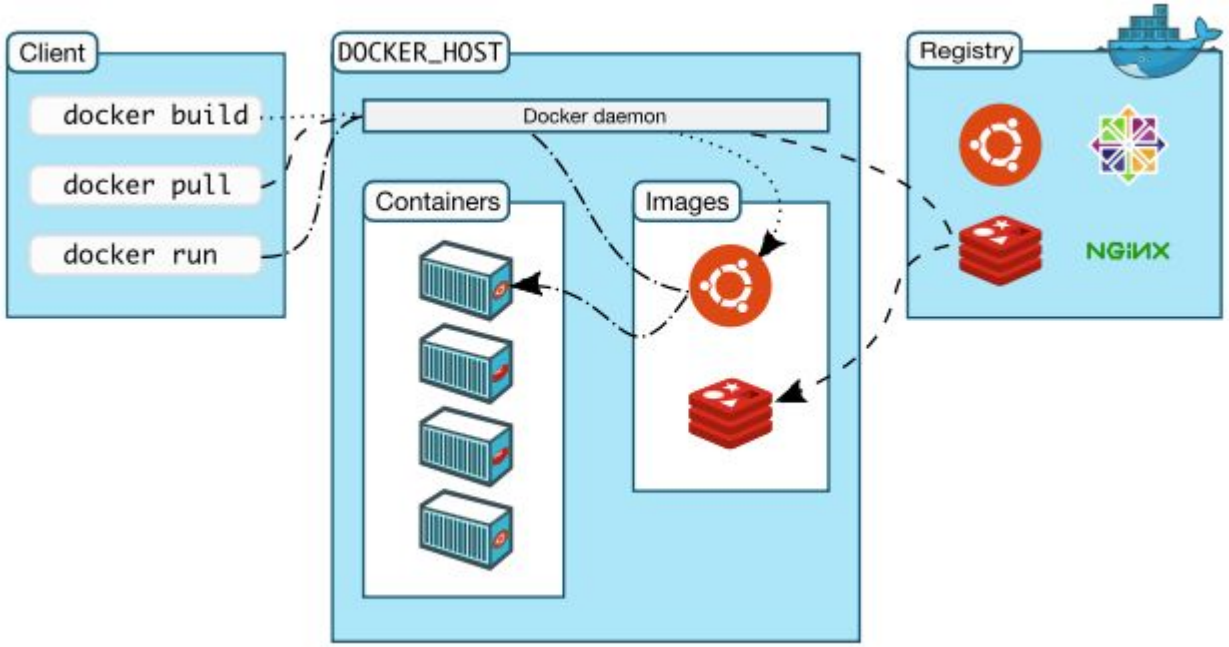
```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
 ---> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
 ---> Running in 5090fde23e44
 ---> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
 ---> Running in 2987746b5fba
 ---> 86c81d89b023
Removing intermediate container 2987746b5fba
Step 4 : COPY package.json /usr/src/app/
 ---> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
 ---> Running in 31ee9721cccb
 ---> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
 ---> 995a21532fce
Removing intermediate container a3b7591bf46d
Step 7 : EXPOSE 8080
 ---> Running in fddb8afb98d7
 ---> e9539311a23e
Removing intermediate container fddb8afb98d7
Step 8 : CMD npm start
 ---> Running in a262fd016da6
 ---> fdd93d9c2c60
Removing intermediate container a262fd016da6
Successfully built fdd93d9c2c60
```

Docker Deployment

- The first layer is writable while the others are read-only



Architecture



Docker engine

- Docker Daemon
  - Builds Images
  - Runs and Manages Containers
  - RESTful API
- Docker CLI
  - https://docs.docker.com/engine/reference/commandline/cli/
  - docker build # Build an image from a Dockerfile
  - docker images # List all images on a Docker host
  - docker run # Run an image
  - docker ps # List all running and stopped instances
  - docker stop # Stop a running instances
  - docker rm # Remove an instance
  - docker rmi # Remove an image

Docker Hub

- https://hub.docker.com/
- A registry of Docker images, where you can find trusted and enterprise ready containers, plugins, and Docker editions.
- Storage for your images
  - free for public images
  - cost for private images
- Automated builds(link github/bitbucket repo; trigger build on commit)
- Docker registry
  - Private Repository of Docker Images

Workflow

1. Find an Image on Docker Hub
2. Pull an Image from Docker Hub
3. Run an Image on Docker Host
4. Stop an Instance
5. Remove an Instance
6. Remove an Image

Volumes

- Can be created by:
  - Attaching a host volume/folder to the container
  - Creating an isolated volume inside the docker container

Docker Cluster

- For performance and scalability purposes, each service may be deployed on a cluster composed by several nodes (hosts)
- Docker Swarm
  - A swarm is a group of machines that are running Docker engine and are members of a cluster
  - The nodes controlling the cluster are swarm managers while the other nodes are workers
  - Swarm Managers are responsible for managing the workers and the tasks (services) assigned to each (Load Balancing)
- Other solutions: Kubernetes

74

Docker Swarm

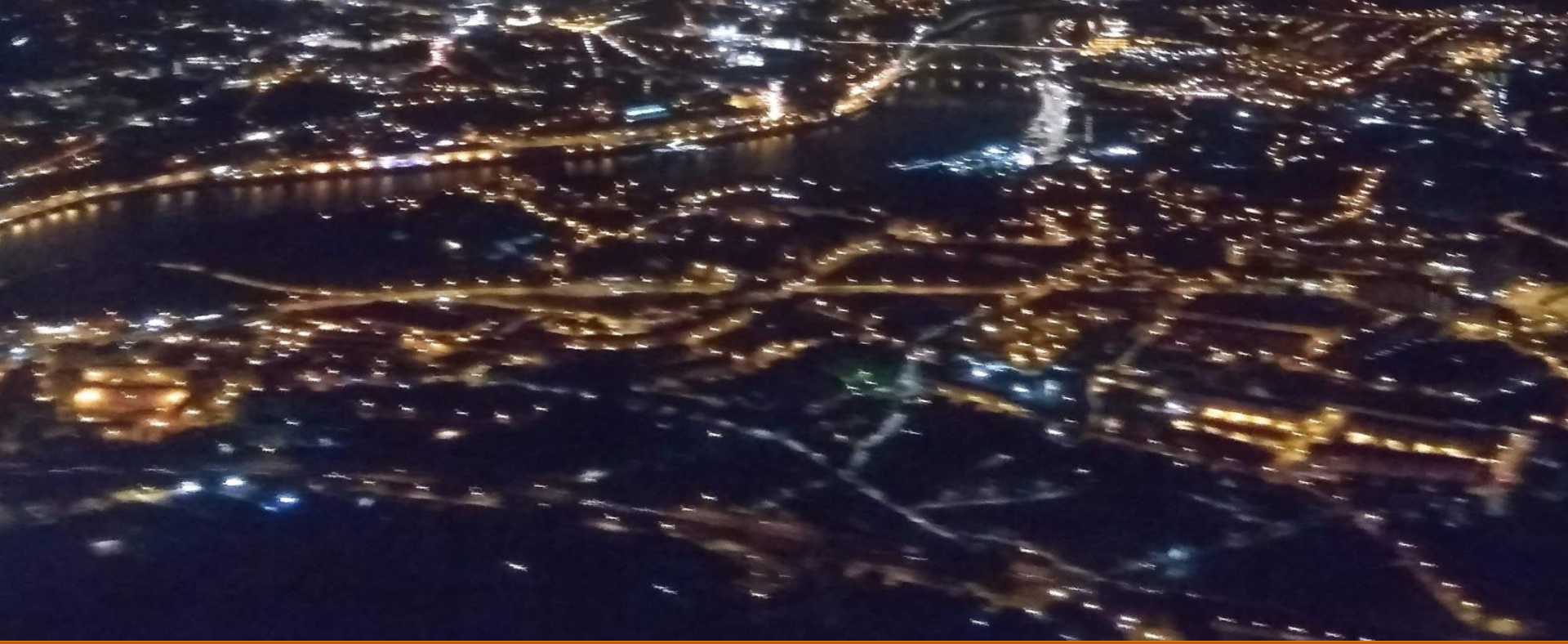
- Able to manage a cluster of Docker Engines called a swarm
- You can use the Docker CLI to:
  - create a swarm
  - deploy application services to a swarm
  - manage swarm behavior
- Features
  - Cluster management integrated with Docker Engine
  - Declarative service model: Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack.
  - Scaling: For each service, you can declare the number of tasks you want to run.
  - Desired state reconciliation: The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state.

Docker Swarm

- Features
  - Multi-host networking: You can specify an overlay network for your services.
  - Service discovery: Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers.
  - Load balancing: You can expose the ports for services to an external load balancer.
  - Secure by default: Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes.
  - Rolling updates: At rollout time you can apply service updates to nodes incrementally.

Docker
Cluster Network

- Cross-Host communication
- Overlay - create a logical network across Docker hosts
  - Ingress
  - Weave
  - Calico
  - Flannel



Large Scale Data Management

rmvilaca@di.uminho.pt