Introduzione generale: Cloud, applicazioni e i servizi di livello PaaS

Davide Salomoni (davide@infn.it)

Corso INFN “Istanziazione e utilizzo di batch system on-demand su infrastrutture Cloud. L’esempio pilota dell’esperimento AMS”

Perugia 25-28/11/2019

Agenda
(dai titoli di corso e introduzione)

- **Corso**: “Istanziazione e utilizzo di batch system on-demand su infrastrutture Cloud. L’esempio pilota dell’esperimento AMS”.
- **Introduzione**: “Cloud, applicazioni e i servizi di livello PaaS”.

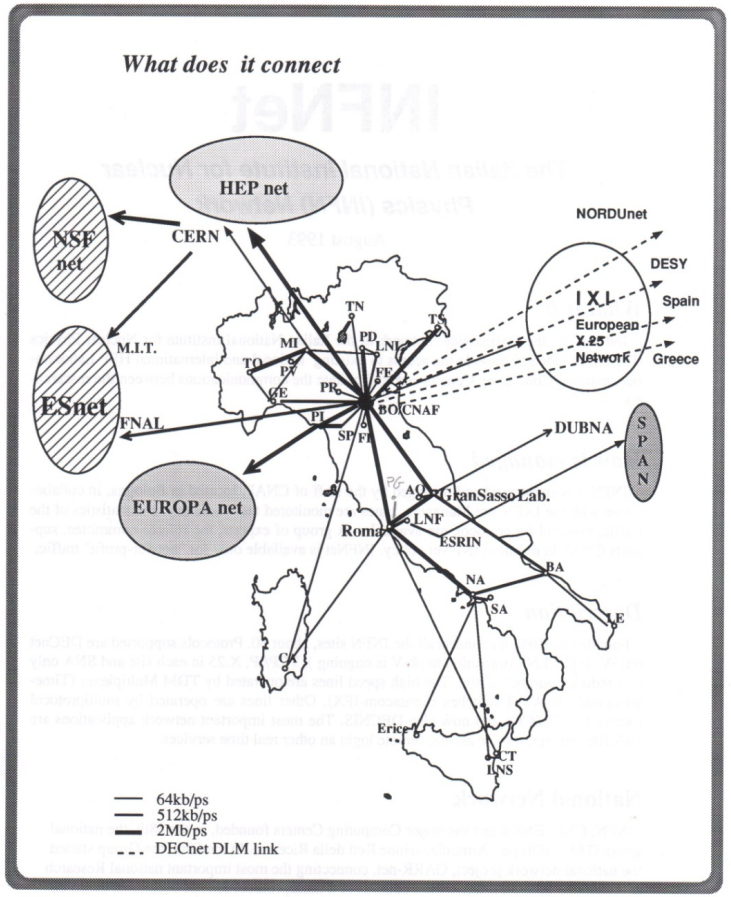
- Da questi titoli, **i punti in agenda oggi** in questo talk sono:
  1. *Infrastrutture*
  2. *Cloud*
  3. *PaaS*
  4. *Applicazioni*

Agenda

1. **Infrastrutture**
2. Cloud
3. PaaS
4. Applicazioni

A sample infrastructure…

… which in 1993 was this…

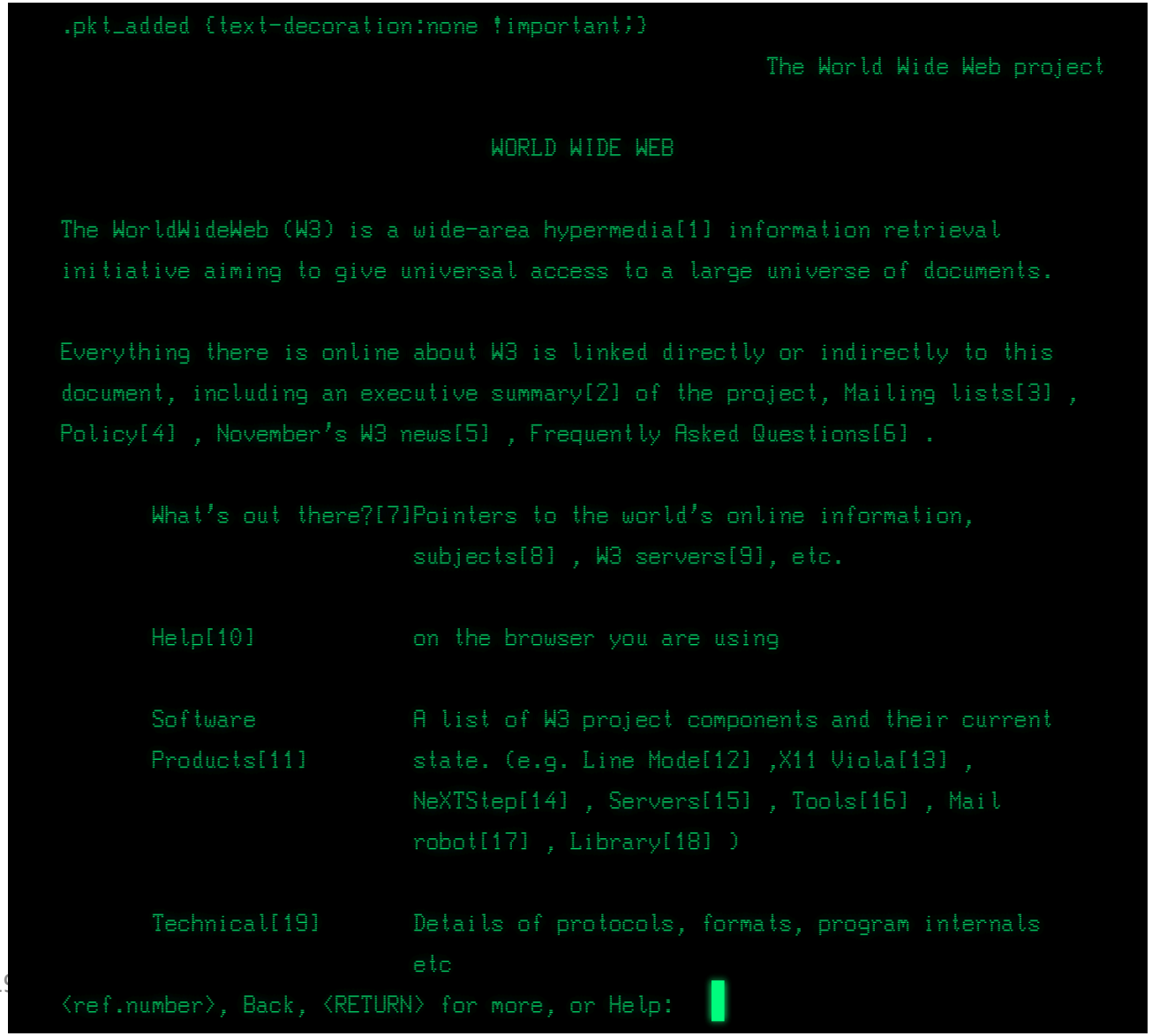


INFNet/GARR 1993

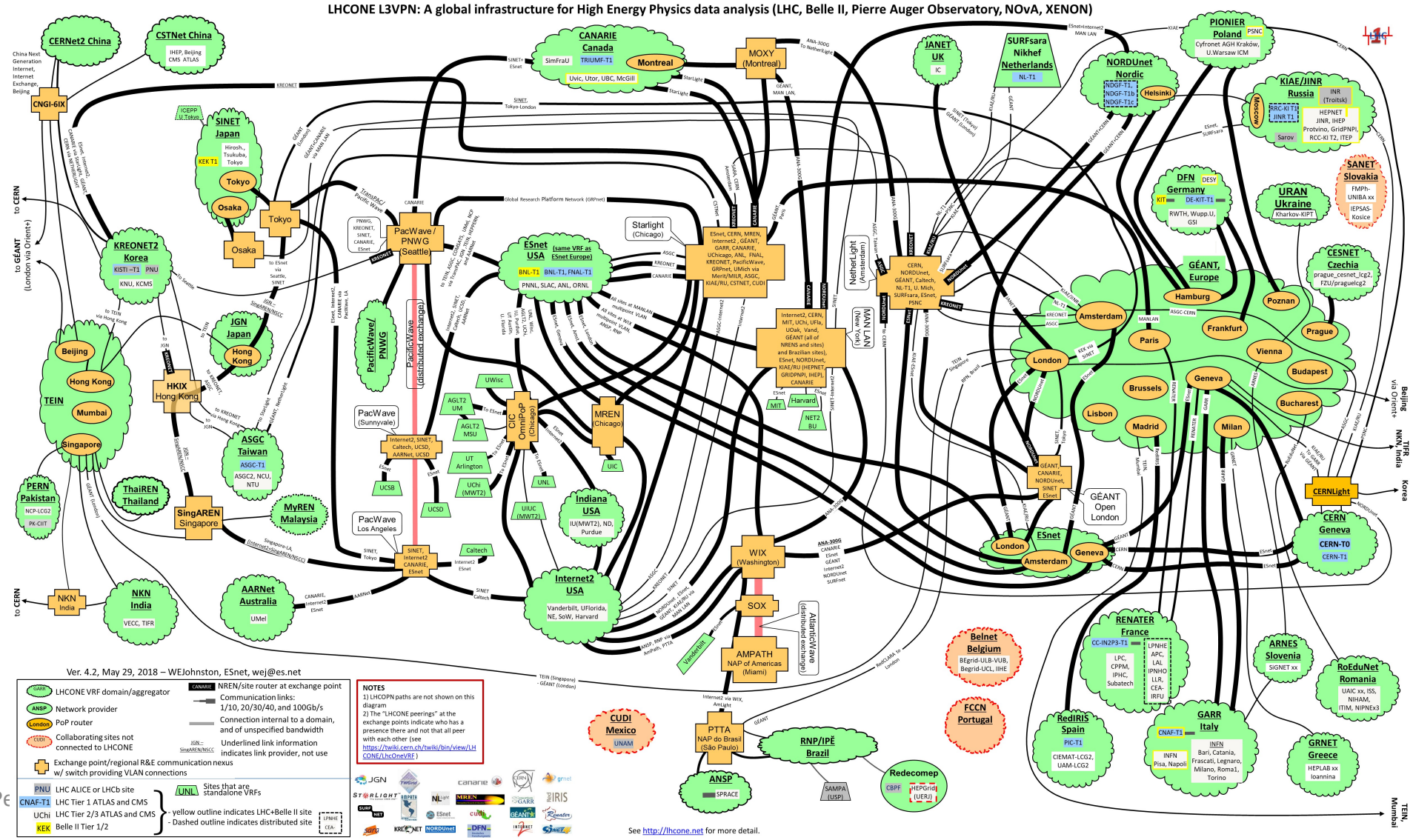
- IXI (International X.25 Infrastructure) from the COSINE project, since 1990-94 (Cooperation for Open System Interconnection Networking in Europe)
- Since 83 HEPnet, the european HEP network (X.25, DECnet, TCP/IP, SNA)
- 93-97 EUROPAnet, the European Research Network before TEN-34
- Esnet (Energy Science network), , DECnet and TCP/IP, USA
- NSFnet (National Science Foundation network), TCP/IP,BITnet USA
- SPAN (Space Physics Analysis Network) NASA, ESA, Astronet in IT (DECnet)
- DUBNA, Joint Institute for Nuclear Research

18

… at the time accessed with this:

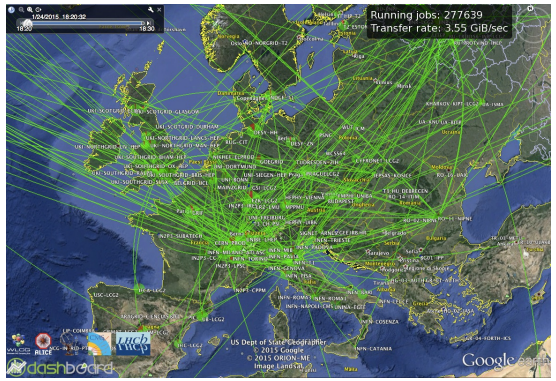


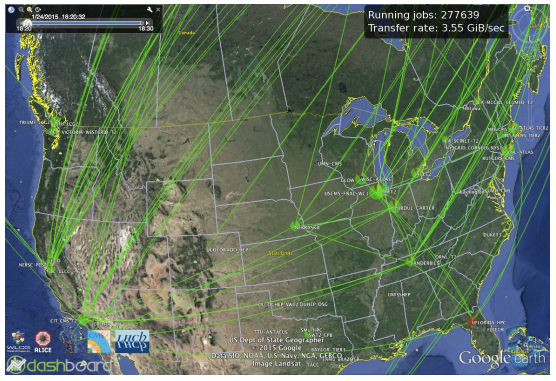
Today things are more like this (this is just *a part* of a network layer):



Or like this:


1/24/2015 18:20:32
18:20
18:30
Running jobs: 277639
Transfer rate: 3.55 GiB/sec
US Dept of State Geographer
© 2015 Google
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat
Google earth
dashboard


Running jobs: 277639
Transfer rate: 3.55 GiB/sec
dashboard


Running jobs: 277639
Transfer rate: 3.55 GiB/sec
dashboard

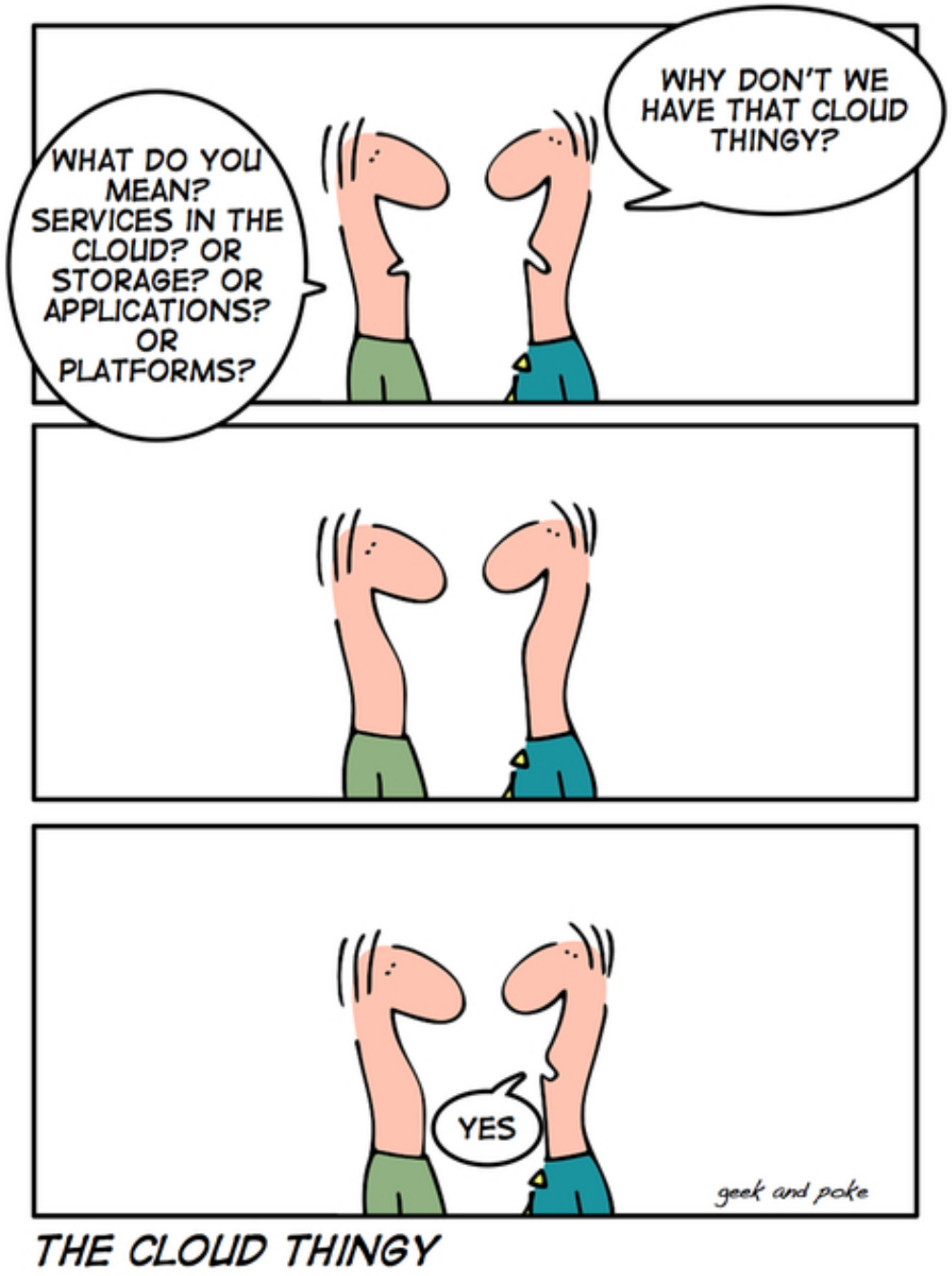
What are *[e-]infrastructures* then?

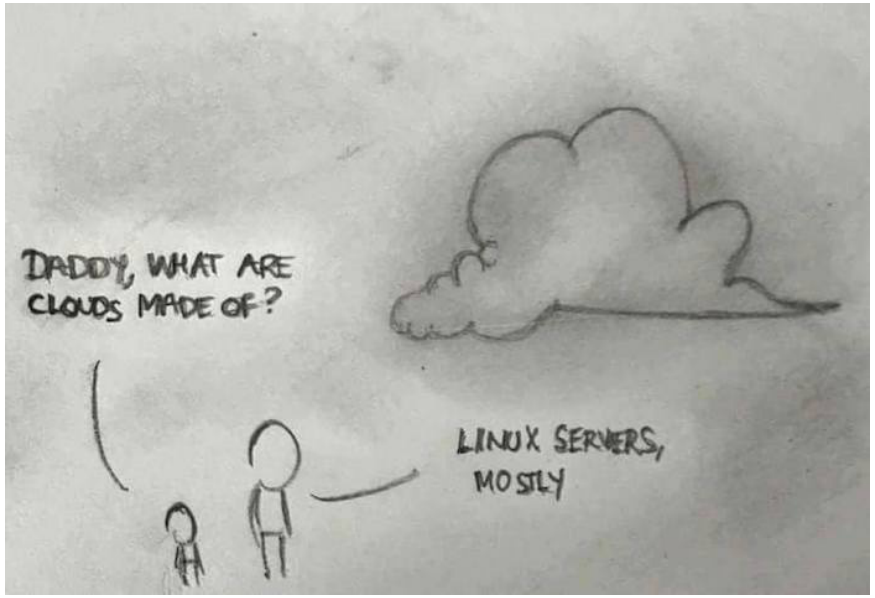
- They are **virtual places** where a broad spectrum of resources for advanced data-driven research can be *found*, *accessed* and *used* by those who need them.
- In modern terms, we often think of infrastructures as part of **the Cloud**, or of *Cloud Computing*.
- But *what is the Cloud*, really?

Agenda

1. Infrastrutture
2. **Cloud**
3. PaaS
4. Applicazioni

The «Cloud»


WHY DON'T WE HAVE THAT CLOUD THINGY?
WHAT DO YOU MEAN? SERVICES IN THE CLOUD? OR STORAGE? OR APPLICATIONS? OR PLATFORMS?
YES
geek and poke
THE CLOUD THINGY


DADDY, WHAT ARE CLOUDS MADE OF?
LINUX SERVERS, MOSTLY


There is no cloud
it's just someone else's computer

Cloud Computing, defined

From Wikipedia:

- "Cloud computing is an information technology (IT) paradigm that enables:
  - **ubiquitous access**
  - **to shared pools**
  - **of configurable system resources**
  - **and higher-level services** that can be
  - **rapidly provisioned with minimal management effort**, often
  - **over the Internet.**
- Cloud computing relies on the sharing of resources to achieve **coherence and economies of scale, similar to a public utility.**"

What is Cloud Computing?

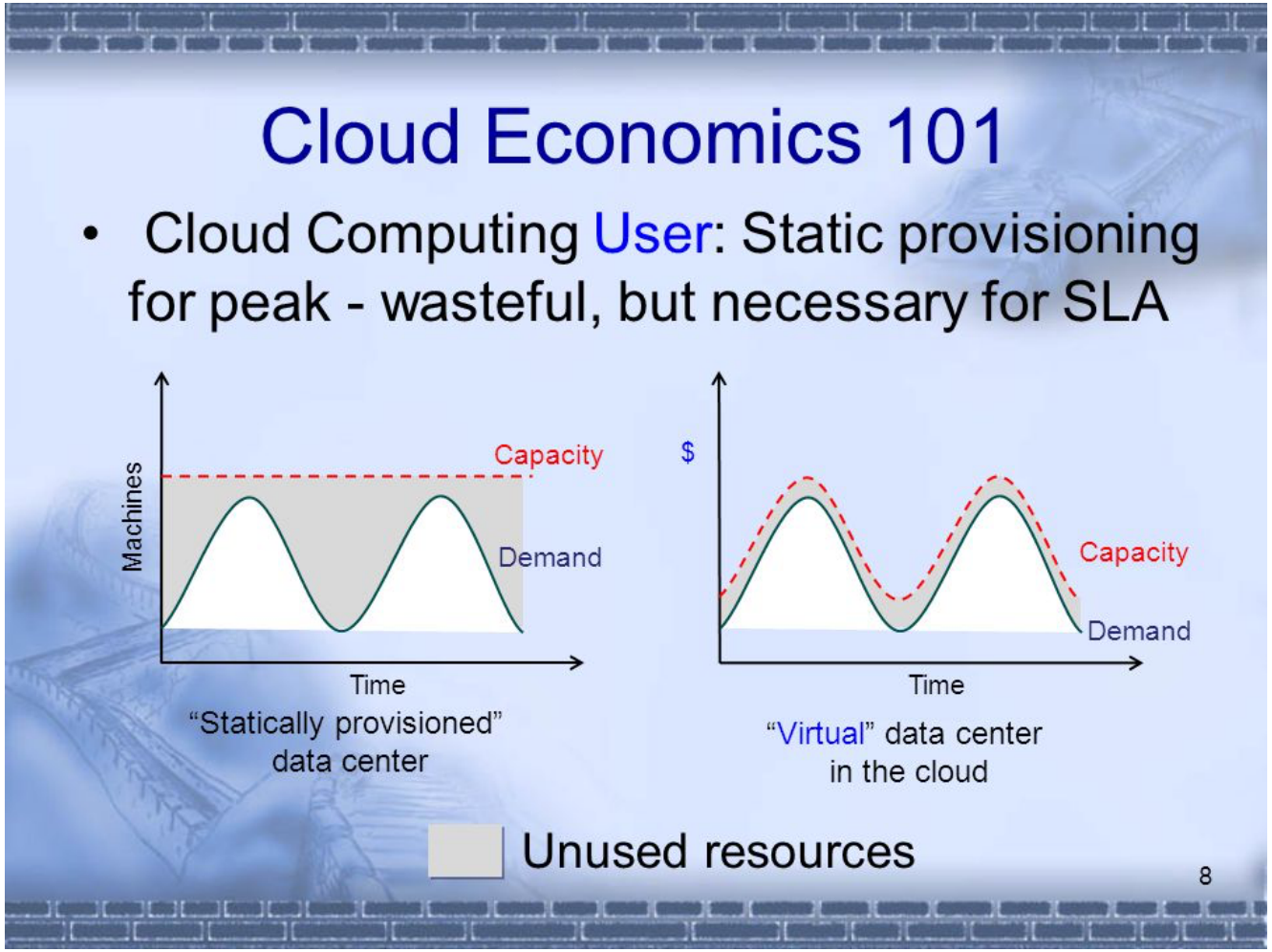
An analogy: think of electricity services…

You simply plug into a vast electrical grid managed by experts to get a low cost, reliable power supply – available to you with much greater efficiency than you could generate on your own.

Power is a utility service - available to you on-demand and you pay only for what you use.

Source: https://www.slideshare.net/AmazonWebServices/introduction-to-amazon-web-services-7708257

Why is Cloud computing useful in general?



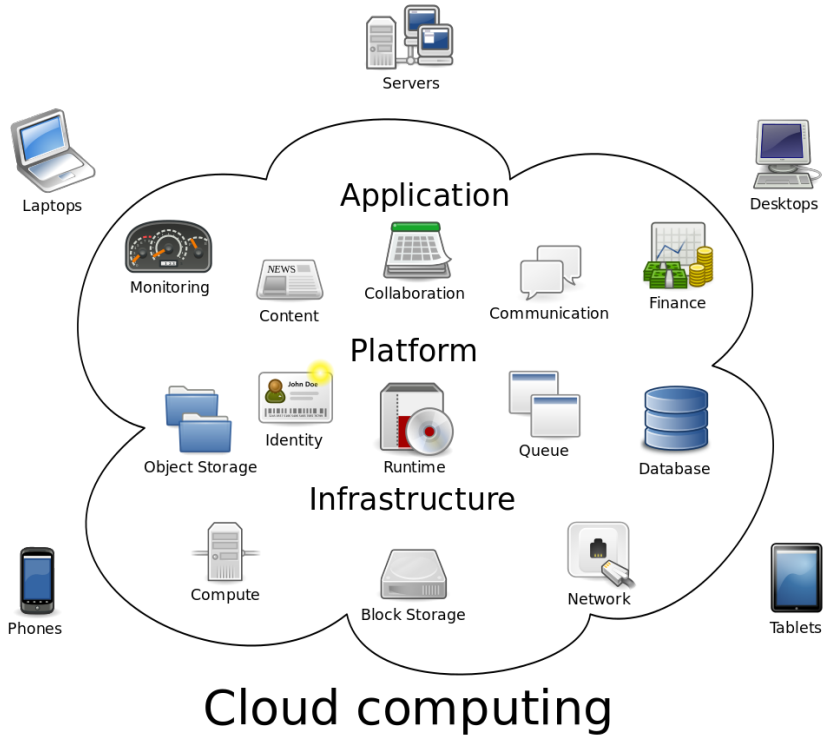
Source: http://slideplayer.com/slide/6085063/

Cloud Computing, defined again

- The **canonical definition** comes from the US National Institute of Standards and Technology (NIST), http://goo.gl/eBGBk.

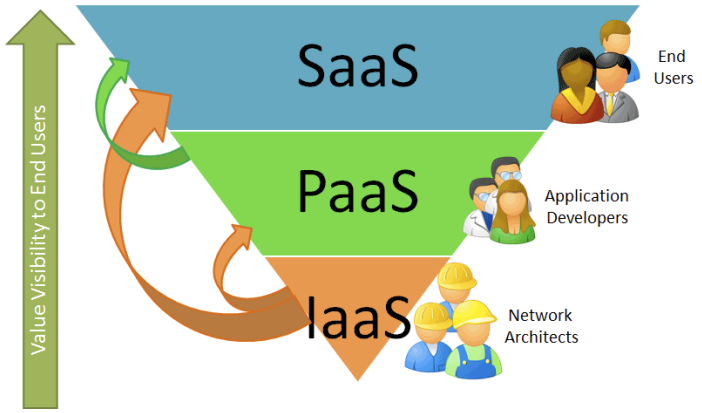
- In a nutshell, Cloud Computing deals with:

1. Supplying
2. information and communication technologies
3. as a service

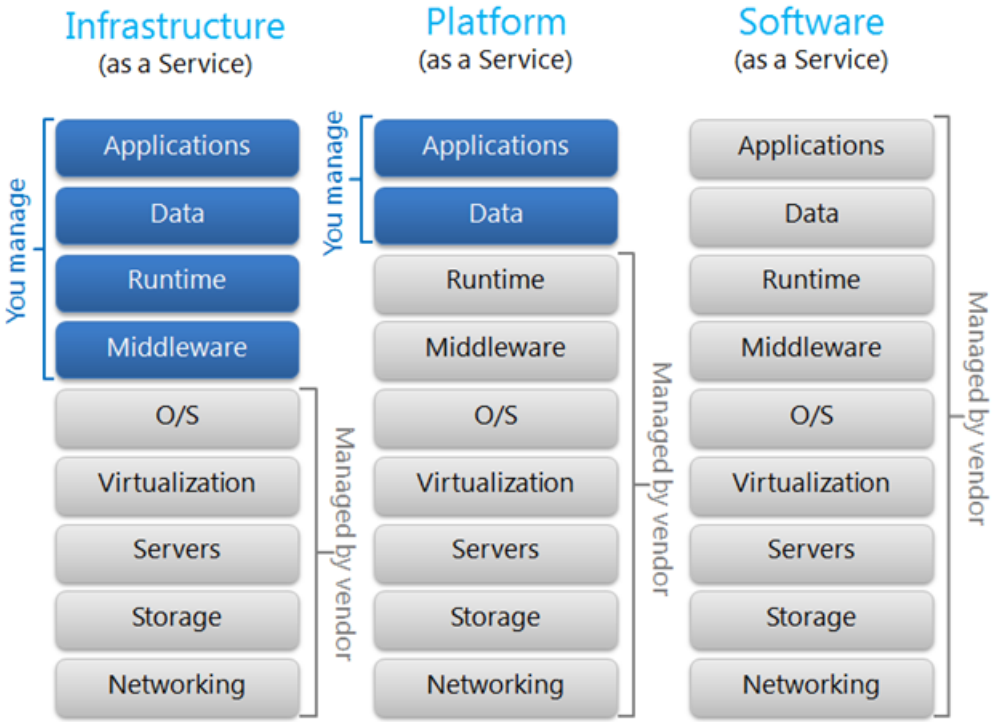


The 5 Cloud postulates

1. Self-service, on-demand provisioning
2. Network-based access
3. Resource sharing
4. Elasticity (*with infinite resources*)
5. Pay-per-use



What matters at the end
are the applications.



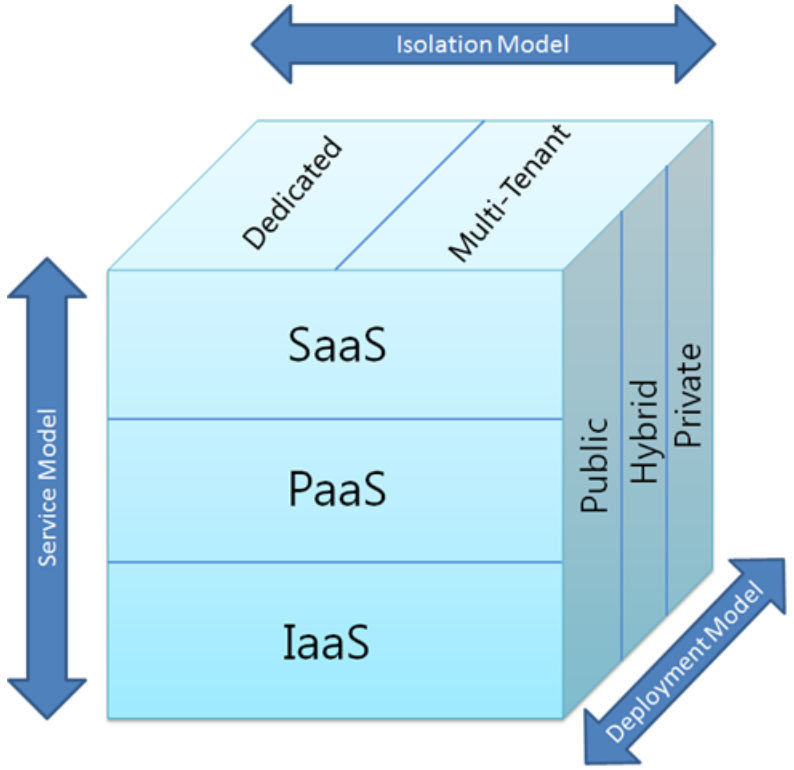
IaaS vs. PaaS vs. SaaS

	IaaS	PaaS	SaaS
What you get	You get the infrastructure. Freedom to use or install any OS or software	You get what you demand: software, hardware, OS, environment.	You don't have to worry about anything. A pre-installed, pre-configured package as per your requirement is given.
Deals with	Virtual Machines, Storage (Hard Disks), Servers, Network, Load Balancers etc.	Runtimes (like java runtimes), Databases (like mysql, Oracle), Web Servers (Tomcat etc.)	Applications like email (Gmail, Yahoo mail etc.), Social Networking sites (Facebook etc.)
Popularity	Highly skilled developers, researchers who require custom configuration as per their requirement or field of research.	Most popular among developers as they can directly focus on the development of their possibly complex apps or scripts.	Most popular among normal consumers or companies which rely on software such as email, file sharing, social networking as they don't have to worry about the technicalities.

See https://goo.gl/ZwZtMQ

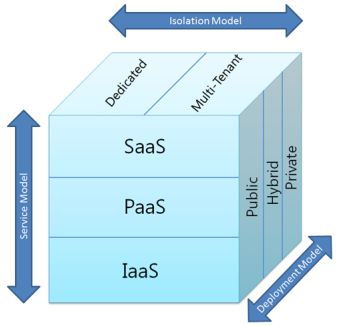
Let’s add some dimensions

- Beyond the *service models* (IaaS, PaaS, SaaS), in order to better understand the Cloud we need to consider other dimensions related to:
  - **Deployment** (where I distribute services).
  - **Isolation** (how I isolate services).



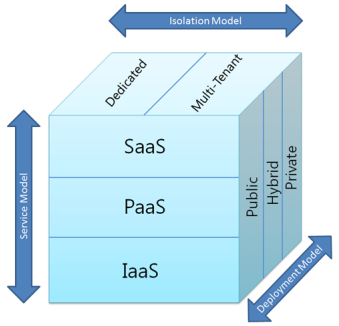
Source: http://goo.gl/1jmkR

Deployment: the «cloud types»



- **Private Cloud:**
  - The resources are **procured for exclusive *use*** by a single organization. Management, operation, ownership, location of the private cloud, however, can be independent by the organization using it.
- **Community Cloud:**
  - The infrastructure is **available to a community** of organizations sharing a common goal (for instance: mission, security requirements, adherence to common regulatory rules, etc.)
- **Public Cloud:**
  - The infrastructure is **available to the public** at large. Management can be either public or private. The location is at some service supplier premises.
- **Hybrid Cloud:**
  - The infrastructure is a **combination of two or more Cloud infrastructures** (private, public, community Cloud), connected so that there is some form of portability and interoperability of e.g. data or applications.

Isolation: how alone am I?



- Cloud **isolation models** are important and often somewhat ignored. We could have:
  - Dedicated infrastructures.
  - Multi-tenant infrastructures (i.e., with several [types of] customers).
- The isolation type is essential in many regards, such as:
  - Resource segmentation
  - Data protection
  - Application security
  - Auditing
  - Disaster recovery

Agenda

1. Infrastrutture
2. Cloud
3. **PaaS**
4. Applicazioni

PaaS: let's make it more concrete

- IaaS = **Infrastructure** as a Service. That's quite straightforward: computers (or *Virtual Machines* – VMs), disks, etc. provisioned over a distributed infrastructure – the Cloud.
- SaaS = **Software** as a Service. That's also straightforward: Cloud-based solutions (applications, services) that are ready to use, such as email, collaborative office suites, etc.
- PaaS = **Platform** as a Service. But what does *this* mean, in practice?

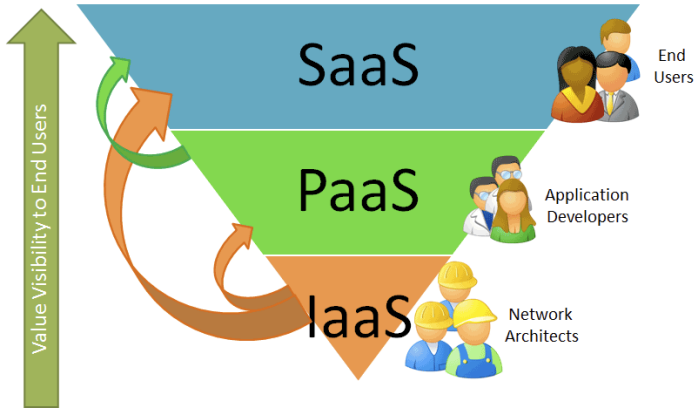
The **PaaS** layer is a way to **program** a Cloud infrastructure, i.e. it provides a platform and an environment allowing developers to build applications and services.

PaaS for developers

- The PaaS layer allows users to ***create and orchestrate*** *(we'll see later what this means)* ***software applications*** *using tools supplied by a Cloud provider*.
  - PaaS services can include ***preconfigured features*** that customers can subscribe to; they can choose to include those features that meet their requirements.
- The infrastructure and the PaaS services are ***managed*** for customers and ***support*** is normally available.
- Services are ***constantly updated***, with existing features upgraded and additional features added.
- PaaS providers can ***assist developers*** from the conception of their original ideas to the creation of applications, down to testing and deployment.

Really?

- *"PaaS providers can **assist developers** from the conception of their original ideas to the creation of applications, down to testing and deployment."*
- There are indeed a few PaaS services that we'll cover in this course. They focus on:
  1. Authentication and authorization
  2. Requirement abstraction
  3. Provider independence
  4. Resource orchestration
- They are useful, if they help us to reach more easily the top of the inverted triangle, i.e. to **develop modern, robust, distributed applications**.



1. Authentication and authorization

- Infrastructures normally have some way of ***authenticating*** *those who want to access their resources*… (i.e.: tell me who you are, and I'll tell you if you are welcome here or not)
- … and some way of ***authorizing*** *some actions on the resources* (i.e. are you allowed to perform some operations on this or that resource?)
  - The simplest form of authentication to a resource (which could be e.g. an application, or a system) is to have local usernames and passwords.
- However, in distributed systems this is usually not very scalable (and it does not say anything *per se* about authorization). We need something more sophisticated and flexible.
  - **On Thursday**, we will explore in some detail how a PaaS-level service called **INDIGO-IAM** can help us in implementing authentication and authorization in our (possibly batch-system based) distributed applications. However, we will see very soon how to use INDIGO-IAM as a "black box system".

2. Requirement abstraction

- Different applications (or even different versions or runs of a certain application) naturally have different requirements.
  - For example, they might need a certain library, a certain operating system, some specific configuration of a service or of the network, or some storage system with some characteristics, etc.
- Our job of developing robust applications is greatly simplified, if we can **avoid dealing with too many details**, and if we can avoid reinventing the wheel every time we need to write an application, and **implement proper automation procedures**.
- The PaaS can help in providing us with tools useful to abstract our requirements, so that these can be easily specified and reused.
  - **On Tuesday**, we will explore two software provisioning, configuration management, application-deployment and resource abstraction tools, **Ansible and TOSCA**, and we will learn how they can be combined in Cloud infrastructures. The PaaS components we'll use directly support them.

3. Provider independence

- We normally do not want that an application of ours is limited to running only on resources made available by a given provider. It would be much better to be able to exploit any (appropriate) resources, independently by whom the resource provider is – as long as we have the authorization to use those resources, of course. We also call this **vendor neutrality**.
  - **On Tuesday**, we will see how a specific PaaS component, called **Infrastructure Manager (or IM)**, allows us to achieve this.
- For example:
  - Resources available on some internal resources (e.g. one of our data centers).
  - Or resources available through some public Cloud providers (e.g. Amazon Web Services, Microsoft Azure, Google Compute Cloud).
  - Or some opportunistic resources (e.g. resources that become available ”on the spot” for some limited period).

4. Resource orchestration

- “Orchestration” *per se* can mean different things. In this course, we will talk about:
  - **Orchestration of “containers”** across multiple hosts. This is a site-level service.
  - **Orchestration of “resources”** across multiple providers. This is a PaaS-level service.
- **On Tuesday**, we will talk about *orchestration of containers*, covering **Docker Swarm and Kubernetes**.
- Also **on Tuesday**, we will talk about *orchestration of resources*, covering the **INDIGO PaaS Orchestrator**.

Summary of the PaaS components that we will see in this course

Type of PaaS service	Service name
Authentication and authorization	INDIGO-IAM
Requirement abstraction	TOSCA and Ansible support
Provider independence	Infrastructure Manager
Resource orchestration	INDIGO PaaS Orchestrator

- Credit: all the PaaS components that we are going to study and use are production-level and derive from the **INDIGO-DataCloud** project.
- Note that **these components are modular**, i.e. they can be used or not, depending on a given use case.
  - For example, if you do not need or want to orchestrate resources across multiple providers, you could simply avoid using the PaaS orchestrator (but still e.g. use the same TOSCA templates to achieve provider independence for your apps).

Agenda

1. Infrastrutture
2. Cloud
3. PaaS
4. **Applicazioni**

Which applications?

- In this course, we will specifically treat **HTCondor as our main Cloud application**. That is, we want to be able to deploy HTCondor across some Cloud resources.
- When we cover containers (today and tomorrow), we'll see in detail that **applications can be encapsulated in containers**. We'll therefore work on getting some familiarity with containers.
- Why is this useful?
  - **If you are a "final user"**, this will tell you how to self-instantiate an HTCondor cluster, and then use it for your own scientific purposes.
  - **If you are a "system administrator"**, this will tell you how you can simplify your workflow and help users to be more independent.

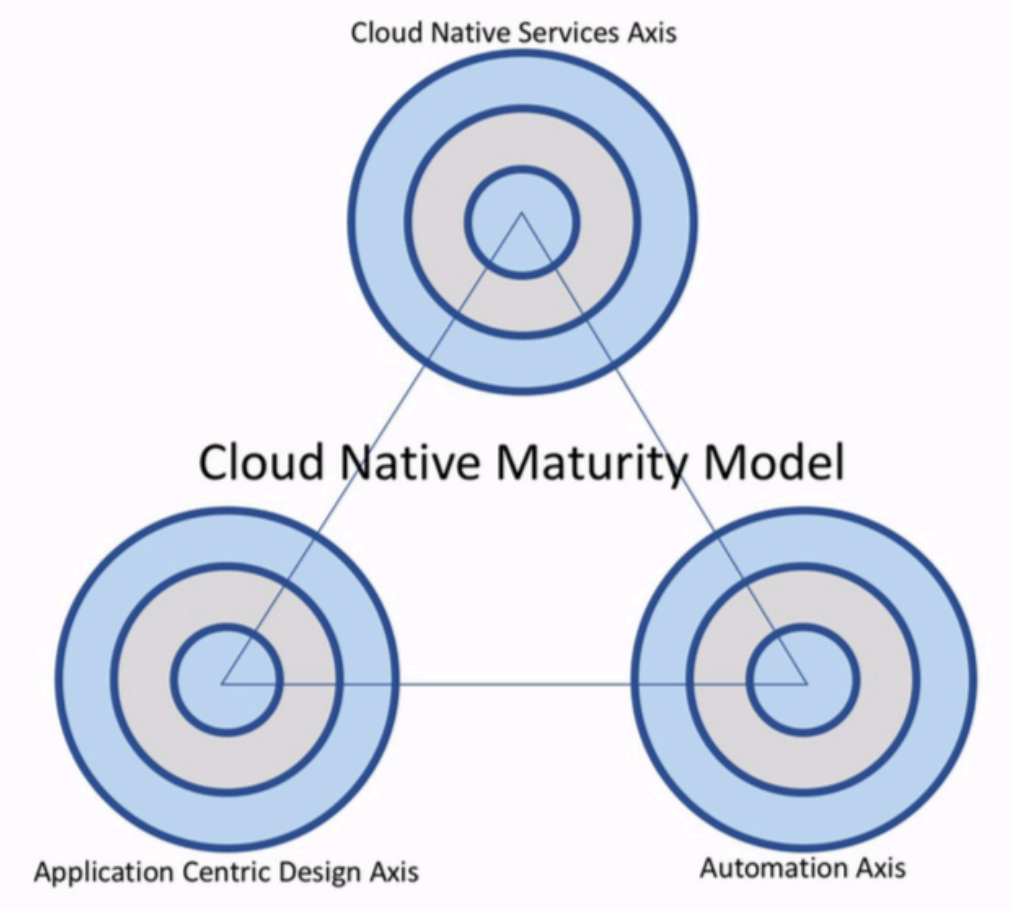
Applications in the Cloud

- Apart from HTCondor itself, you may want to run other applications in the Cloud.
- We will not discuss in this course how to write in detail “Cloud native applications”†, but it is important to define a few general principles about that. You will hopefully be able to verify if your own applications satisfy these principles or not.

†: there will be a dedicated INFN course on this topic in 2020

The Cloud Native Maturity Model

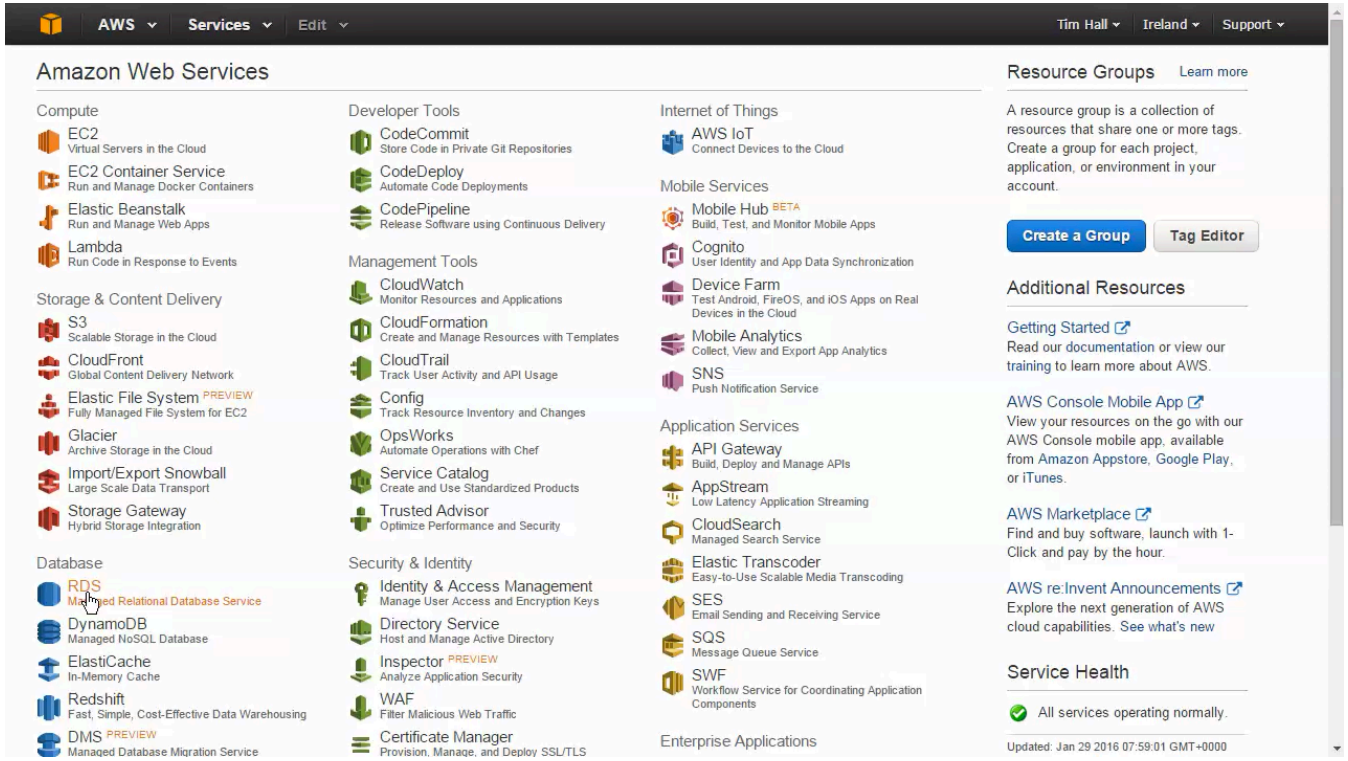
- In order to assess to which extent your application is a “cloud native” one, it is useful to map its architecture to a **Cloud Native Maturity Model** (CNMM), centered across three axes:
  1. Cloud Native Services
  2. Application Centric Design
  3. Automation



Source: Laszewski et al. (2018), Cloud Native Architectures

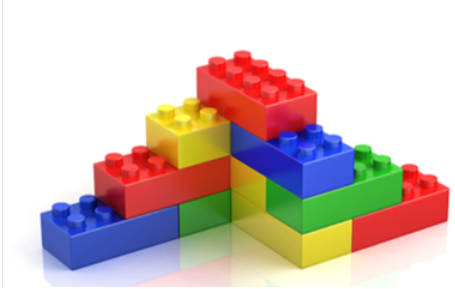
1. Cloud Native Services

- Since we are talking about applications to be deployed in the Cloud, it is important to **know which services are available from your Cloud provider(s)**.
- In this course, we will mostly discuss some already mentioned PaaS-level services. They are all vendor neutral, so you could count on them for applications you may want to deploy in any Cloud.
- If you elect to go with specific vendors, you'll need to get familiar with what they specifically offer.
  - For example, see here a partial list of the Amazon Web Services offering.



Use and compose basic building blocks

- A key point to write successful Cloud applications is always **modularity**. You should get familiar with basic concepts such as virtual servers, containers, block disk storage, object storage, monitoring, etc.
- Normally, all Cloud providers provide these as **basic building blocks**. One should then see how they can be easily combined. We'll cover it in this course, when we talk of templating languages such as TOSCA.



2. Application Centric Design

- A useful reference to design your application architecture is to study and apply the methodology called **12-Factor App** (https://12factor.net/).
- In summary, it states that there are 12 key factors to consider when designing an efficient, cloud-aware application.

I. Codebase
- One codebase tracked in revision control, many deploys

II. Dependencies
- Explicitly declare and isolate dependencies

III. Config
- Store config in the environment

IV. Backing services
- Treat backing services as attached resource

V. Build, release, run
- Strictly separate build and run stages

VI. Processes
- Execute the app as one or more stateless processes

VII. Port binding
- Export services via port binding

VIII. Concurrency
- Scale out via the process model

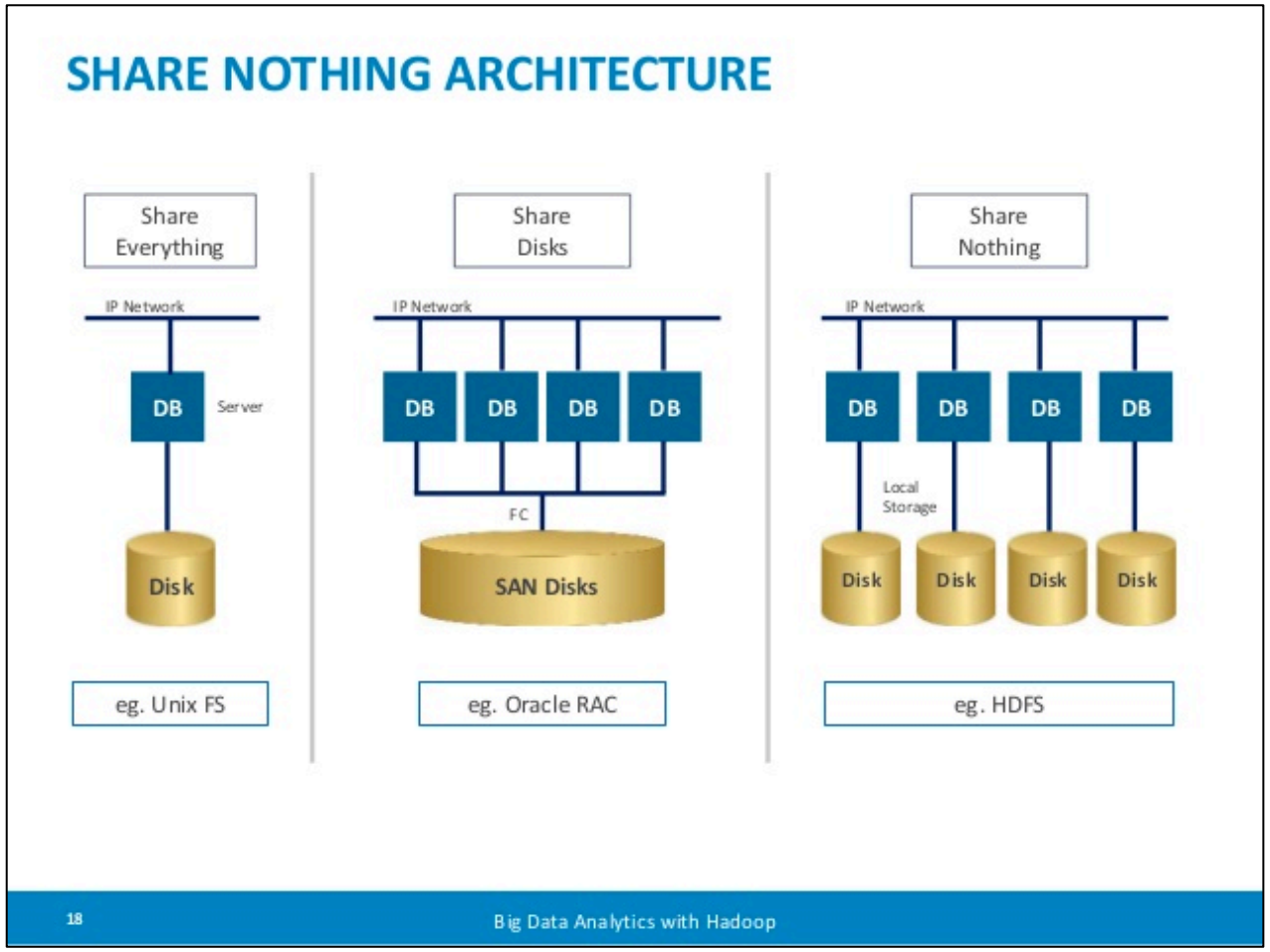
IX. Disposability
- Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
- Keep development, staging, and production as similar as possible

XI. Logs
- Treat logs as event streams

XII. Admin processes
- Run admin/management tasks as one-off processes

Example (backing services, concurrency)



Source: https://www.slideshare.net/PhilippeJulio/hadoop-architecture/18-SHARE_NOTHING_ARCHITECTURE_Share_Disks

Monoliths vs. microservices

Monolithic Applications

- Do everything
- Single application
- You have to distribute the entire application
- Single database
- Keep state in each application instance
- Single stack with a single technology

Microservices

- Each has a dedicated task
- Minimal services for each function
- Can be distributed individually
- Each has its own database
- State is external
- Each microservice can adopt its own preferred technology

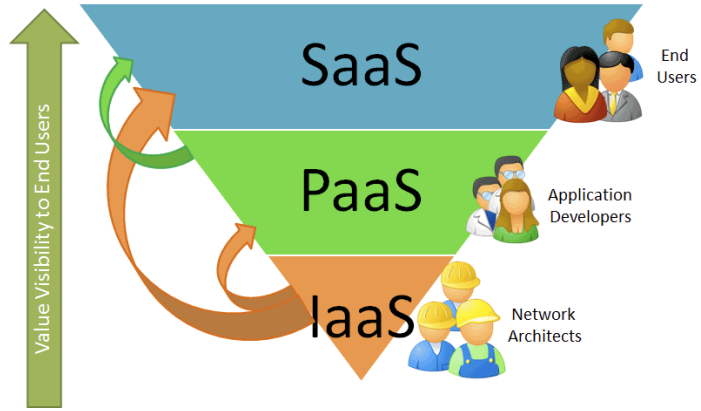
Adapted from AWS

3. Automation

- In writing Cloud-aware applications, we should **automate** as much as possible **all its operational aspects**.
- That is, we want to **abstract as much as we can from hard-wiring details** in the application – not only for what regards its local configuration, but also for what regards its operation and deployment.
- With the idea of **Infrastructure as Code (IaC)**, instead of manually creating the infrastructure we need for our applications (e.g. virtual machines, disk volumes, installations, configurations), we **define what we want** through machine-readable definition files.
  - IaC is based on the realization that "**Complexity kills Productivity**": it therefore aims to simplify how you can realize complex infrastructures and set-ups.

Serverless technologies

- With **serverless technologies**, we perform another step toward automating and facilitating the use of Cloud resources.
  - Remember the inverted triangle: **what eventually matters are the applications, not the infrastructure.**

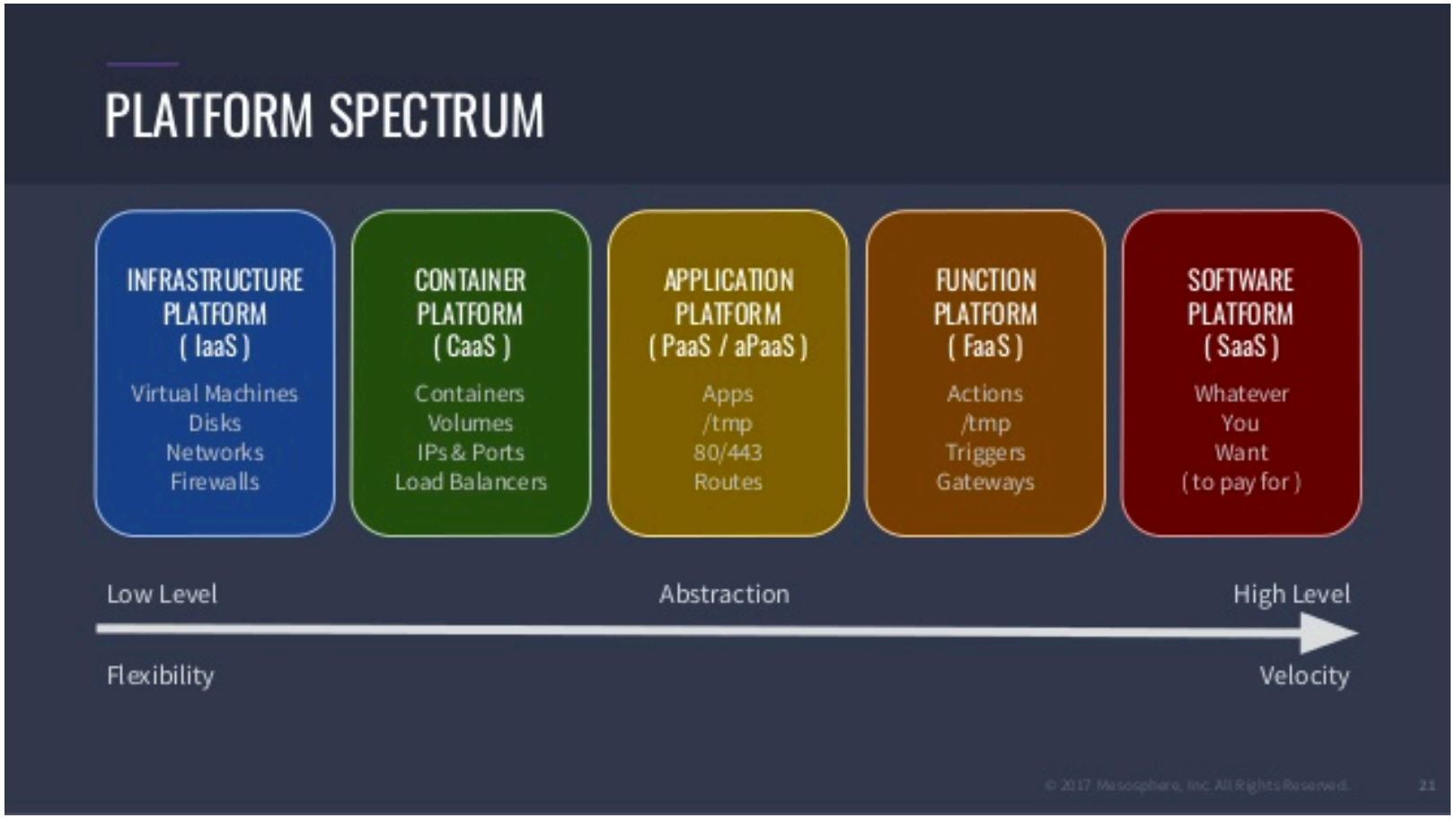


- Recall what happens *with traditional [Cloud] applications*:
  - We need to **provision and manage the resources** (e.g. virtual machines, disks, an object storage bucket, etc.) for our applications.
  - We are **charged if we keep the resources up and running**, even if they are doing nothing.
  - We are responsible to **apply all the updates and security patches** to our servers (virtual or not).

Serverless, or FaaS

- With **serverless**, a Cloud provider is responsible for executing a piece of code (written by you) by **dynamically allocating the resources needed by the code**.
- You are only charged for the resources used to run the code, and only when the code runs.
- This code is typically a function. Thus, serverless computing is also called **Functions as a Services, or FaaS**.
- The running of these functions can be **triggered** depending on some **conditions**, such as for example database events, queueing services, file uploads, scheduled events, various alerts, etc.
- Your applications should therefore be **structured around a set of stateless functions** → this is consistent with the idea of **microservices** we have already seen.

Cloud models revisited, then:



Source: https://techthought.org/cloud-architecture-201-blurring-the-lines-between-iaas-paas-as-the-cloud-enabled-paradigm-shift-accelerates/

In summary

- In this introduction, we have very briefly covered a few concepts that that will be explored more in detail during this course.
- In the course, we will learn **how to create and deploy HTCondor batch systems over Cloud infrastructures**.
- In order to do that, **we will consider 4 main topics**:
  - **Infrastructure**
  - **Cloud**
  - **PaaS**
  - **Applications**
- By working on these concepts, you will learn not only how to effectively instantiate and deal with HTCondor clusters in a Cloud world, but more in general how to exploit Cloud resources for your own applications as much as possible.