Technological Infrastructures

Part I - Prof. Ciavotta

1 Components of NIST

NIST develops reference standards for
public. Software Architecture is a global organization of
software systems, and consists
in:

- Division of software components into subsystems;
- Definition of the policies with which these systems interact;
- Definition of interfaces between the various components.

A reference architecture is essentially
a template (empty box with predefined elements), provides
only the vocabulary commonly used to discuss implementations
of a given software.

A reference architecture for the software
It is nothing more than software architecture where the structures
and the various elements and relationships are provided by the
template.
The reference architecture, provided by NIST,
for Big Data:

- Provides a common language for the stakeholders;
- Encourages adherence to common standards;
- Allows you to implement architectures with a certain consistency;
- Illustrates and improves understanding of Big Data components, processes and systems;

1.1 The 5 main roles for Big Data

The conceptual architecture of Big Data is a cross-architecture
with two axes: Information Value (IV) and Information
Technology (IT) (Fig 1).
The 5 main roles on the 2 axes of Big Data are:

1.1.1 System Orchestrator

The system orchestrator often also involves the In-formation
Value chain, since it is concerned with
implement and monitor business processes
at enterprise levels and the various data policies:
Make data accessible for a limited time
or provide data at different speed (passing data in memory
to disk). Can assign/provide physical or virtual framework
components
to the system, this assignment can very often be elastic and
independent. It can provide
GUI support and connect various applications to
a high level, and through the factory management monitor
the loads and the system to ensure/specify the required
quality of service
for the various loads. It is very often centralized.
Ex. Ambari/Cloudera

1.1.2 Data Provider

It can be either a software (e.g. in a larger pi-peline) or a
person. If it is a
person will enter his data and use the tools
of collection and curation/preparation for uploading
data on the systems and improve their quality, If a

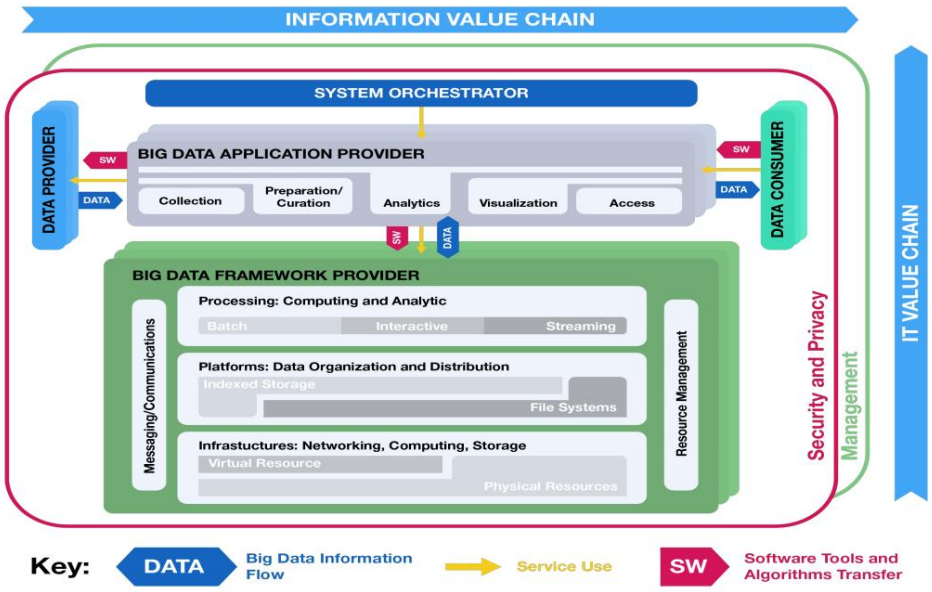

Figure 1: NBDRA Conceptual Model

software will make its data available through interfaces such as Apache Scoop. The
data provider can be internal or external to the platform, must provide access rights to the data, and is obliged to follow the privacy policies and security fabbric. The data can be entered in pull or push. e.g. Flume to load data from MySQL to HDFS.

1.1.3 Consumer Data

It receives the outputs of BigData systems, it can also he can pull and push data, can use the information for data reporting, retrieval/search and visualization. There must be authentication and authorization from the privacy and security fabbric for communication between the architecture and the Data Consumer.

1.1.4 Big Data Application Providers

Corresponds to the typical activities of a Data Scientist, which also exist in traditional systems but have transformations in the implementation with Big Data. These activities
I am:

- Collection: It manages the interface provided by the Data Provider, saves/manages this data in a certain area so that it is not lost, it also implements data extraction functionality from the data providers;
- Preparation: Perform Data Validation, outlier removal, standardization, formatting and enrichment. Try to promote high-quality data;
- Analytics: Extracting knowledge from data, leveraging the underlying software of the Big Data Framework Provider;

- Visualization: Presentation of data in ways visual ra;
- Access: It is the opposite of the collection, it is is concerned about exposing data to the outside world.

1.1.5 Big Data Framework Providers

Provides the infrastructure to support the Big Data Application Provider. He deals with detail of:

- Data processing - has a duality: from one part is a framework that provides programming interfaces to do certain things (e.g. MapReduce) and on the other hand defines how the implementation is done. Frameworks can vary between batch processing and streaming.
- Platforms for data organization and storage - can contain meta-data together with the semantic descriptions of the data- you. It can be either distributed relational or non-relational.
- Infrastructures for the physical execution of our software, is the set of physical or virtual computational resources on which the Our Big Data system runs, it can be made up of large or small servers size. These components provide:
  - Networking - They can be defined through software and can be networks physical which can in turn be partitioned into virtual networks. They can be purely virtualized networks, i.e. everything (firewalls, routers, load balancing) is implemented virtually (e.g. VM inside our machines);
  - Computing - Hardware, Software, OS, memory for computing;
  - Storage - disks for storing (locally), RAID, network, etc.;
  - and other services such as Cooling, the electrical system and safety

  It can be deployed on both physical and virtualized environments (native, hosted or containers).

It also has 2 roles spread across the 3 components above:

- Communication and messaging between the components;
- Resource management for integration of components.

1.2 The 2 widespread roles for Big Data

These 2 roles are called Fabric, the term Fabric is used because these two roles are cross-cutting, that is, they are present a bit everywhere in architecture.

1.2.1 Factory management

The two main associated activities are:

- System management to provide resources, software and package management and finally the management of configurations and performance of the various pipelines;
- Big Data Life Cycle, BDLM (Big Data Life Cycle Management), contains Policy enforcement (e.g. encoding/decoding), meta data management (data governance), data accessibility, data recovery and their preservation.

1.2.2 Security and Privacy Fabric

It deals with the three typical characteristics of security:

- Authentication: Indicates all the activities that validate the user;

- Authorization: Once authenticated the user checks his permissions, e.g. some data may not be accessible to certain users;
- Auditing: concerns the recording of events that occur in the system, can trigger an alarm in case of an abnormal event or a posteriori analyze the sequence of events (with log files).

2 Virtualization

For computers they have been defined with the 5 classic components:

1. Input Devices: Keyboard etc.
2. Output Devices: Display etc.
3. Storage Devices: Volatile(RAM), Permanent(HD, SSD)
4. Processor:
    - Datapath
    - Control
5. Network

Virtualization enables the execution of multiple operating systems simultaneously on the machine in a totally isolated manner. It can be seen as an emulation of a software or hardware on which other software can run, this emulated environment is called a virtual machine.

The concept of VM has been developed over the years 60 from IBM on mainframes. It is abandoned with the birth of modern PCs and resumed with the recent growth of cloud. The virtual machine is obtained through a Virtual Machine Monitor (VMM) also called hypervisor.
Hypervisor is a software that lies under virtualized OS to provide the resource sharing features available in

so that the program or OS running see these resources as if they were dedicated to him.

Resources are CPU, memory, storage and network.
There are several benefits of virtualization:

- Unified view of resources e.g. I see many discs as a single disc;
- Consolidation of virtualized resources, in in order to have optimal use of resources;
- Ease of implementing redundancy to copy virtualized environments;
- It facilitates system migration to another, furthermore if I do not change hypervisor the (virtual) machine will work identically to Before;
- Centralized management of hardware and software.

Other benefits/properties of virtualization are-
No:

Workload Isolation: through virtualization it is possible to completely isolate programs, which also has improvements in security, it also increases reliability since the failure of one program does not lead to the failure of the programs since they are isolated, furthermore it

They also solve problems regarding conflicts of libraries in this way. Finally you get a performance control as execution of one VM does not affect the performance of the other;

Workload Migration: This helps in:

- Hardware Maintenance;
- Load Balancing;
- Fault Tolerance;
- Disaster Recovery.

Since we can move the entire virtualized environment to a new machine quite transparently, to do this the machine

should be suspended, totally serialized to be sent into the network, migrated to a new machine and restarted immediately or just saved without running.

Consolidation: Leveraging Workload Mi-gration you can consolidate separate machines

on a single platform reducing costs (used
very often in data centers during off-peak hours).

The different types of HyperVisors are (Fig 2):

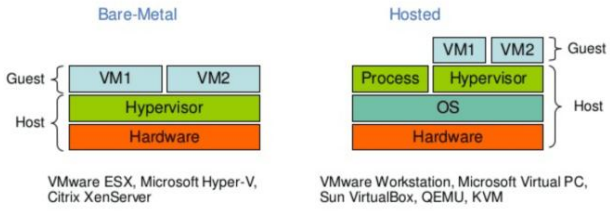

Figure 2: Hypervisor types

- **Hosted:** in this case the hypervisor is a
  process that runs above the operating system and
  allows the execution of multiple Guest machines. In
  this case, therefore, it is necessary
  first install an OS on which VMM (hypervisor) will be
  installed and at this point the host will be able to run
  the applications inside
  of its window. The advantage here is the ease of
  installation and configuration, furthermore the HostOS
  and GuestOS remain unchanged
  modified and do not depend on the particular
  hardware, but the disadvantages are performance
  degradation and lack of
  real time OS support as there are various
  entity/software in between;
- **Bare-Metal:** in this case the hypervisor
  works directly on top of the hardware. In this case the
  hypervisor is an OS
  very light and communicates directly with
  hardware instead of depending on another
  OS. The advantages are improvement in I/O
  and real time support, while the disadvantages
  are the difficulty of installation and configuration and
  the dependence on the type of hardware
  specific.

There are mainly 2 virtualization techniques: **Software Virtualization** (which we have talked about so far) and **Hardware Assisted Virtualization.**

In **Total Virtualization** VMM takes care of completely emulating everything
the hardware, so we will have a processor, memory, disk and virtual network. In this case
The guest OS is not aware of the existence of the virtual environment and each machine is completely independent. At the CPU level, binary translation takes place: This happens in several loops of
safety, in particular there are 4 rings, where
Ring 0 is the most privileged one and allows the code to be executed directly on the hardware
and here the OS kernel is executed. User applications are executed on the latest
ring (ring 3).
The hypervisor runs on ring 0 while the GuestOS runs on ring 1, so they have more permissions than normal applications. VMM has access on the ring
0 to have direct access to the CPU rather
than virtualize it.

Para **-Virtualization** has a different approach than total virtualization (it is a
(a middle ground between total virtualization and bare-metal),
in this case the guests are aware
of the VMM and uses special calls in some cases
to be executed directly on the hardware and
This results in improved performance but makes it less flexible.

OS **-level Virtualization (Containeriza-tion)** does not use VMM, virtualization is provided directly by the HostOS which performs all the
functions of a fully virtualized hypervisor, so it has a purely virtual partition of resources, and this involves an assignment

flexible allocation of resources to various applications. Ex. Docker.
There are 3 service models:

1. **Server Virtualization:** Suppose
   have different servers on different machines, in
   in case of a problem (node crash) or you
   it's a need for upgrade, in case of machines
   physical I will have to take the same model or the
   same vendor, the solution is to exploit the same
   machine with a virtual
   lizer and put the different servers together.
   This way I have a consolidation, shared
   resources, centralized management,
   ease of migration, greater ROI and less space
   occupied. The Disaster Recovery
   and scalability is facilitated, different models
   (hardware) to choose from (as long as it is the same
   the hypervisor) and I have more availability.
2. **Desktop Virtualization:** It is the technology
   that separates the desktop environment and software
   applications from the physical customer who uses it.
   Our desktop becomes a thin client that
   use a PC without the RAM and processing power
   needed to run a
   real machine, the machine runs on a pool
   of VMs on a server in a data center.
   The benefits are multiple: Easy software and OS
   upgrade, high availability, fault
   tolerance, accessible from LAN, WAN, Internet,
   there is also the possibility of having a small part
   of the computation performed by the
   local device.
3. **Application Virtualization:** Very similar to
   Desktop Virtualization, except that
   Instead of the entire desktop, it is the applications
   that are virtualized, so a
   application that runs on the desktop, does the
   computation on the cloud, a small part of the
   computation can also be done in
   local. The advantages are very similar to those
   of Desktop Virtualization, an advantage
   additionally it may be that I only pay that
   which I use, reducing licensing costs.

3 Cloud

"True" definition of cloud to remember: it is a type of
distributed service of interconnected systems
and virtualized computers dynamically provisioned
and presented as a single computational resource
based on service-level agreements.
Basically the cloud is the ability to have
access to computational resources through the
on-demand network and is composed of 5 key points,
3 service models and 4 deployment models.

3.1 The 5 Properties of Cloud

The 5 properties revolve around a central idea of
cloud: Utility Computing, SOA (Service Oriented
Architecture) + SLA (Service Level
Agreement).
Utility Computing in the sense that the service
provider provides the computational resources and
infrastructure that the customer needs and charges for them.
usage-based rather than a fixed cost, like on-demand
services, to maximize
efficient use, minimising costs in the least transparent
way possible (without showing the
customer everything that is done).
SOA (Service Oriented Architecture): is a
set of services that communicate with each other
since a single service is usually not enough to
implement a complete web service for a user.
SLA (Service Level Agreement): is a contract
between the service provider and the customer that
specifies the level of service that the service provider
must provide in terms of QoS. The QoS (Quality
of Service) is a set of technologies for managing the
network traffic in order to improve user experience
experience, is now associated with a more
generic regarding technology assessments
non-functional, that is, I am not only interested in the
functionality (the result of my request) but also

that its efficiency.

The most common SLA metrics are up-time and down-time, Response time. If the guaranteed metrics are not respected there are penalties on the service providers.

The technology by which cloud works I am:

- Hardware Virtualization;
- Distributed and parallel computing;
- Service-oriented Computing;
- Autonomic Computing (reaction to system changes).

The 5 **properties/characteristics** that identify cano the cloud are:

1. **Scalability, Elasticity:**
   - Scalability is a property of the system grow gradually as the requests, can be horizontal (Scale In & Scale Out) or vertical (Scale Up & Scale Down).
   - Elasticity is the ability to adapt automatically to initialize scalability.

   This can be achieved through provisioning dynamic, which refers to an environment complex that allows on-demand allocation and de-allocation of instances/resources through an application or an administrative console. Usually rules are set to automate provisioning, resulting in thus a reduction in costs and performance improved.
2. **Availability, Reliability:**
   - Availability is the ratio between up-time and total execution time;
   - Reliability is the ability to function in special situations (breaking of a knot, hacker attack) for a certain time.

   These points can be obtained through systems of:
   - Fault Tolerance is the system's ability to continue to function even after a failure of one of its components, often the service is degraded proportionally to the severity of the failure but the service continues to function.

     The main features are that it does not has a single point of failure, the failed component is identified and isolated to prevent propagation of the failure;
   - Resilience is the ability to offer and maintain an acceptable level of service even after a failure, essentially going back to operating state after a system failure (e.g. power failure) electricity). So the systems must have recovery policies and procedures, e.g. Backup (of off-site data or of the entire system) or preparation against fault like an uninterruptible current (UPS) or power/voltage surges.
   - Security, which concerns the use of policies and technologies and control systems to protect applications and infrastructure from malicious access, so we have its use in:
     - data protection, keeping access to data confidential only on the basis of privileges;
     - Identity management to ensure access to resources based on their privileges (data protection);
     - Application security is about the ability to shield our applications from malicious access;
     - Privacy to hide data according to privacy laws and managed only by users competent.
3. **Manageability, Interoperability:** Manageability is about the management of these systems

through a single access point while interoperability is a property of a product or system for working with others products/systems. This is achieved through **System control automation** and **System State Monitoring** which is a process that monitors hardware status, system metrics performance, system logs, the pattern of network access etc. It is also linked to the Billing system, as users pay for what that they use and the cloud provider must register the resources/service used by each user.

4. **Accessibility, Portability**
5. **Optimization, Performance** The **Load Balancing** is a technique for distribution of the workload on a multi-computer system, CPU, HD etc. to achieve optimal usage, thus improving: system usage, performance and energy efficiency by reducing overload.

The **Job Scheduler** is an application that has the task of running background processes (also called batch processes). In the cloud, computationally intensive tasks or tasks that grow dynamically and must be planned with Job Scheduler.

From a user's point of view, he doesn't want know how and what is done nor who manages the service, but only wants a functional service.

3.2 The 3 Service Model of cloud

Let's start with the three models of cloud service I am:

1. **IaaS (Infrastructure as a Service):** e.g. virtual machines, in this case the provider/vendor manages the virtualization, the customer obtains virtual resources, usually from a catalog which contains the hardware specifications (virtualization) pre-selected by the provider. In most In some cases, the OS are also pre-set for performance/stability and compatibility issues with hypervisor used by them. Usually managed from a system administrator.
2. **PaaS (Platform as a Service):** we are provided with a little box where we can write and run/test the applications, the provider will be the ensuring that the system is sufficiently responsive and has sufficient runtime. It is less flexible than IaaS but at the same time the developer loses less time in the configuration of the machines. Managed by usually from developers.
3. **SaaS (Software as a Service):** e.g. GMail so user can't do anything except use the software. Used by individuals or business analyst.

In reality the real cloud is a bit more obfuscated than as said above, it is often not easy to place a company's cloud model exactly in one of 3 models.

A very important aspect to consider is the economic one, that is, thanks to the pay-as-you-go cloud model the capital cost (CAPEX) is transformed into operational cost (OPEX), furthermore the the risk of error in choosing the machines disappears, if I need more power I ask for one stronger machine and if I need less power I lower the power and save. At the same time cloud service providers have different benefits: profit by exploiting economies of scale (buying a servant costs X, but buying N server does not cost N*X), they can capitalize on their investments (Amazon selling the residual power) or they can use it to promote their product (e.g. to make their customers use .NET) developers).

3.2.1 IaaS

IaaS automatically provides the processing, storage, networking and other resources essential to the computing, and the user can install the software which are necessary for him.
IaaS supports these 3 features of cloud:

- Manageability and Interoperability: Through one interface can manage all the VMs that he wants, he can pay them as he wants;
- Availability and Reliability: managed by the cloud providers across availability zones;
- Scalability and elasticity: Properties of the system but it needs to be implemented by the user.

The architecture consists of the hardware followed by a virtualization layer and the user has access to two interfaces:

- Interface for resource management, these resources are:
  - VM: creation, deletion, management, VM suspension
  - Virtual Storage: Allocate space, reduce/increase DB, choose services with read/write speed different (by price);
  - Virtual Network: management of IPs associated with VMs, registrations to the IP domain, choosing the bandwidth band.
- System monitoring interface: is made available by VIM: The orchestration of machines/resources, in a fast and dynamic way, on a server is managed by the Virtual Infrastructure Manager (VIM), different metrics are used for monitoring eg:
  - VM: CPU usage, memory usage;
  - Virtual Storage: Disk usage, data duplication level and speed disk access;
  - Virtual Network: The use of bandwidth, the connectivity status and balance on the net.

3.2.2 PaaS

PaaS provides the user with languages programming and tools (such as IDE) supported by the provider, so the control over the application deployment is in the hands of the consumer but the underlying infrastructure is managed by the cloud providers.
At the base there is an architecture like that of IaaS, Runtime Environments are added and above these are the Programming IDEs and the APIs/tools supported by the environment, often have in common functions such as: computation and storage.
The Runtime Environment refers to a collection of software, implemented with a collection of libraries, the properties provided by the Runtime Environment are:

- Manageability and Interoperability
- Performance and Optimization
- Availability and Reliability
- Scalability and Elasticity

The interface made available for control
The system provides a set of actions based on:

- Policy based control: actions follow rules to make decisions therefore actions followed by if <> then <>
- Workflow control: description of the installation and configuration flow of the resources or daemons.

3.2.3 SaaS

SaaS: Applications are provided to the customer
cloud provider, the user cannot manage or develop the
application or manage the architecture in
cloud but can interact with it through interfaces such as
portals/web applications, which with the introduction of Web
2.0 has enhanced iterability
with cloud application, eg: Facebook(Mes-senger), Office,
Skype, DropBox, CRM system,
national medical system, transportation system
public.
The key point of this type of cloud is
Accessibility and portability.
Oss: The difference between a web portal and a web page
is that the portal allows for integration
of different pages through logins.

3.3 The 4 Cloud Deployment Models

The 4 primary models that differentiate a cloud,
based on access possibilities, the size
and the ownership of the services, are:

- **Public Cloud** (or multi-tenant or external
  cloud): The infrastructure is made available to the
  general public or at least to large organizations through
  payment for its use, the features

  I am:

  – Homogeneous infrastructure to guarantee the
    same performance to two users even if they are
    in different places
    they use the same service;
  – Common policies;
  – Shared resources;
  – Infrastructure is rented;
  – Economy of Scale.
- **Private Cloud:** (or on-premise cloud or internal cloud) It
  is operated by a single organization and can be
  managed by the same organization
  or another organization, i.e. the category
  of users is limited. Unlike the public
  cloud features are:
  – Heterogeneous infrastructure due to the
    cost and the fact that a private company does
    not throw away hardware just for
    have them homogeneous;
  – Customized ad-hoc policies for the
    users;
  – Dedicated resources;
  – Infrastructure managed by house;
  – End-to-end control over processes.
- **Community Cloud:** it is a structure managed by different
  entities, in practice several entities put together their
  private clouds to
  generate a super cloud, for example some American
  universities can join their
  cloud for some specific research etc.
- **Hybrid Cloud:** it is the co-position of more
  types of cloud, such as a private company
  creates your own private cloud for managing
  sensitive information and use the public cloud
  for the rest, or use it for scalability
  in case of increased traffic.

4 Amazon ECOSYSTEM-IaaS

They are designed to work independently but
can interact with each other, sharing the same naming
conventions.
and authentication and minimizing connections
internal.
Amazon makes the concept available
Region and Availability Zone:

- Each **Availability Zone** is an independent data center
with its own power grid and network connection, the zones
within
a region are connected to each other through
low latency connection. The name comes from the

the fact that they are high availability zones, so
if a zone fails its effect is not noticed-
from other areas thus creating stability and
high fault tolerance.

• **Region** is a cluster of Availability Zones located in a geographical area. When you create
an Amazon instance gives us the ability to choose an
availability zone (or by default we let Amazon choose it)
and within the
same region it is possible to distribute your own
instance across multiple availability zones, so if one
zone fails the other (most likely) continues to work. Amazon
does not charge for data transfers within the same region,

but communications are made between regions
pay.
There is AWS GovCloud which is an AWS Region
isolated and allows you to manage extremely large data
sensitive and can only be accessed by citizens
Americans verified by the US government and on
American soil.

One of the core services of AWS is **Simple Sto-rage Service (S3),** an object storage that is a
flat storage system, i.e. it is not possible to create folders
inside it, the data is
saved in binary format and are distributed
automatically. Access can be done by web calls (REST,
Soap, BitTorrent)
or SQL queries and objects can also be
very large (up to 5TB each).
S3 is the basis of all the infrastructure that
is the basis of Amazon e-commerce. In addition to

S3 Amazon also provides an **EBS (Elastic Block Store),** this disk can be formatted, mounted and
used as a local hard disk, it is
a volume that persists regardless of the
rest. While VMs can die, volumes
always remain, and the guarantee of durability
is provided by Amazon.

EBS is categorized into SSD and HDD, depending
the type of machines/discs is paid for in a manner
different.

Security Groups define the set of
possible connections for a certain instance, and these sets
can be protocols, ports, ranges
of the IPs.

4.1 EC2 - Elastic Cloud Computing

EC2(Elastic Cloud Computing) is one of the
services offered for the creation/management of
on-demand virtual machines. EC2 is based on the

programmable data center concept, the possibility of
creating your own infrastructure (machines in multiple
availability zones or regions) via
Programming languages. The key concepts
which constitute EC2:

- **Amazon Machine Image** (AMI): is a file containing a description of a VM,
  i.e. any software and configurations,
  can be pre-built, created, modified
  and sold. Each AMI has a unique ID;
- **Instance:** Represents a copy of AMI in execution, multiple copies of a single AMI
  can be executed;
- **Elastic IP address:** allocation of a
  Static IP and connect them to your own instance,
  each instance can have at most one IP
  static.

The cores made available by the server (of which not
we know practically nothing) are virtualis-zed, first dividing into
cores and then further dividing the core-time. The unit of
measurement is Elastic Compute units which corresponds to

Intel Xeon

(or AMD Opteron) 1.0-1.2 GHz from 2007.
Resources can be:

- Persistent: even in case of failure of the

Amazon hardware guarantees its persistence through
redundancy, automatic recovery

tico and automated failover. The components
persistent are:

- Elastic IP Address
- EBS
- Elastic Load Balancer
- Security Groups
- AMI saved in S3 or EBS

• Ephemeral: the user must guarantee its redundancy and maintenance through other EC2 services, in case of failure the saved data is generally lost. The instances are

ephemeral.

The price models are:

• **On demand:** Payment is made for
the skills (time) used with no obligation to
long term.

• **Reserved:** A sort of subscription is made
with a payment of a sum first and then the
machines can be used for the duration of the subscription at
a discounted price, it is available
in 3 types Light, Medium and Heavy and can be
for 1 or 3 years and can save up to 71%
compared to On-Demand. Please note that we do
not reserve resources for the duration of the subscription
but only the price.

• **Spot:** The unused computational power of Amazon EC2
is auctioned off, which is
which cost less, because I am not guaranteed the durability
of the machine over time, in
if the offered price is higher than the cost necessary to run
that machine Amazon will give
his car, but since the price of electricity is variable and if it
were to exceed the offered price Amazon can turn off that car
without
advise.

Another cost index is **Data Transfer,** the
Data transfer in or out of the EC2 instance is paid (based on
the quantity)
if the transfer does not take place outside the
region. The transfer within the region is
completely free.

4.2 AutoScaling

One very important service of Amazon is **Au-toScaling (Fig** 3). To make AutoScaling work, the **Elastic Load Balancer is also important,** thanks to which the end user sees only

a connection and Elastic Load Balancer takes care of distributing the requests to our machines.

EC2 instances can be categorized into
auto scaling groups, usually with a range between
a minimum and a maximum, if a new machine starts running
or is deallocated
The Load Balancer is notified of the change
so that it can work well. The Cars
Scaling groups are EC2 instances that share
similar characteristics and must be made for
scale, and automatically adds a new
node based on user-defined policies, scheduling or health checks and workloads and

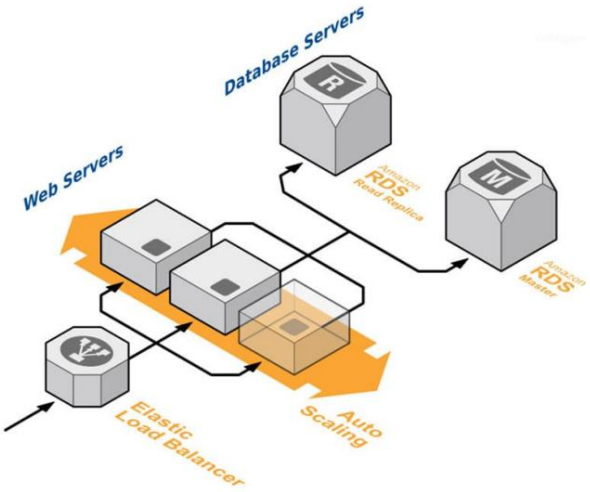

Figure 3: Auto Scaling for AWS example

the new instance can also come from different availability zones of the same region.
Auto Scaling group contains only a launch configuration that describes the instances that should be started (it contains the AMI parameters), if you update the launch configuration this will only affect new instances, old instances remain old but they are terminated earlier during a scale in.

Please note that a machine can either be powered off or terminated, if it is terminated it is completely deleted and AutoScaling group terminates a machine.

The Auto Scaling ecosystem consists of:

- Cloud Watch: To monitor instan-
  ze EC2, the most used metrics for the CloudWatch are:
  - CPU usage
  - Latency
  - Number of Requests
  - The number of Hosts (machines) in good condition Health
  - The number of cars not in good condition health •
- Elastic Load Balancer: To distribute work across any number of instances
  EC2 and perform periodic checks on the health of the node (based on the response time) and in case they are not in good health
  the load balancer stops sending it requests. •
- Auto Scaling: Use data collected by CloudWatch to build systems that can scale (in or out) across the entire
  range

A **Trigger** is a mechanism for activating a policy, in this case to increase or decrease the number of nodes. The trigger can be activated by a cloud watch alarm (configured to watch a cloud watch metric) or
re an auto scaling policy that describes what to do in case of an alarm. When the trigger fires it launches a process called Scaling activity that executes the auto scaling policy. Remark Auto Scaling supports but does not require Elastic Load Balancer. In general at least two triggers are needed, one for Scale up and one for Scale down, to maintain a desired balance. An alarm is a metric and checks whether this metric exceeds a set limit for a certain time.

An alarm is an entity that can continuously observe a certain metric and tells us if ta-
the metric has exceeded a certain preset value for a certain time, to create an alarm you need to specify:

- Metric to watch • The threshold
  of the metric • The evaluation period
  number

The states in which the alarm can be found are:

- Ok, all good! Metric is below threshold. • Alarm: Metric is above threshold, action required.
- Insufficient data, metric not available or not enough data.

If the alarm changes state and remains in that state for a certain evaluation period, an action is invoked according to the policy.

The automatic auto scaling scenarios are:

- Fleet Management: ensures optimal machine performance by checking the health of the instance with Health Check: if an instance should terminate, it is identified and another machine is started.
- Scheduled scaling: Actions are executed on a scheduled basis, so it is a kind of time-based event for recursive or scheduled events.
- Dynamic Scaling: Based on the alarm on a metric there is a policy that responds

to the alarm. The action can depend step-wise on the severity of the alarm or by doing tracking a certain metric and adjust the system so as to calm the alarm.

Machine termination can also be done for machine rebalancing

In availability zones, shutdown precedence is given to balance between the areas followed by the desire to preserve the machines with the most recent launch configuration followed by the machine next to launch of the hour (for the price you pay per hour). Au-to Scaling initializes a new instance before turn one off so as not to compromise the availability and performance of the application.
After the Auto Scaling phase has started there is a cooldown period, during which time no other scaling activity can start.
Candidates for autoscaling are:

- Web Tier - systems that provide services web
- Application Tier
- Load Balancing Tier - systems that manage the load
- Stateless Tier - systems are stateless e.g. pure functions, they are memoryless and so from a functional point of view it doesn't It changes whether I autoscale or not.

Candidates who should not autoscale:

- Relational Databases
- Non-relational databases
- Distributed caching system
- ElasticSearch
- Stateful systems – e.g. Kafka, not it would make sense to do automatic autoscaling.

5 PaaS with Azure

Essentially PaaS provides a programming interface, a language of programming (runtime) and a whole sort of necessary services. According to the traditional approach, an application is developed locally and then it is deployed on the cloud, but there may be some problems like “it works on my machine”: one should exactly replicate your working conditions on the cloud. While with PaaS the development environment becomes the deployment environment and such management directly depends on from the provider.

Top 5 PaaS Services • Web App -  I am:

PaaS Services for the Development of web applications;

- Mobile App - Development Services of mobile applications (content management, users);
- Function App - Microscopic actions (small functions) that can be deployed on the cloud;
- API App - Develop APIs or use APIs from third parties;
- Logic App - Used to write logic application integrations.

The web application can have 2 roles:

- Web Role: The execution of a web app, has the task of presentation.
- Worker Role: Supports bac- execution kend.

Note: A new role is not a VM, it is similar to a sandbox (the application sees itself as the only running app), so you can have multiple roles on the same machine, also in
In the case of a WebApp we do not know the memory or CPU information, you only pay requests served (or processor cycles used),

autoscaling and auto loading are also guaranteed balancing. Communication between Web Role and Worker Role occurs through a message-coding system, and a load balancer distributes the requests between the various Web Roles.

For data persistence Azure provides disposition:

- **BLOB (Binary Large Object) Storage:** It is an object store very similar to the S3 model. Amazon and can save any type of file even large ones, it is a distributed filesystem but the difference compared to S3 is that BLOB storage is two-layered, the first layer takes the Container name and the second level is BLOB.

The interface with which you interact with BLOBS is REST API, so PUT, GET, (UN)DELETE, COPY, SNAPSHOT, LEASE.

- **Azure Cloud SQL:** SQL Server with no administrative overhead, you can choose if you want this DB on shared or reserved hardware, the model that follows is Pay-as-you-grow. It is with high availability (99.99% monthly).
- **Tables:** A slightly structured storage, it is a key-value database so NoSQL, each entity can have 255 properties and be maximum 1 MB in size, like other key-value models We have two keys: one key per partition and one for the row. The key uniquely identifies an entity and uses Timestamp. Observation is strongly Consistent, and is highly Available, so according to the CAP theorem it is not particularly tolerant to Partitions.
- **Queues:** A system of message queues similar to Apache Kafka.
- **CosmoDB:** is a service equivalent to MongoDB offered by Azure, is a polyglot and It is a key-value, graph, column based and document type DB, it supports an automatic index in based on high-demand queries just like Mongo. It also provides very low latency low worldwide.

Azure offers its own Auto Scaling service, but Among the various possibilities offered there is the possibility to create your own rules-based auto scaling. The rules can be:

- Constraint Rules: minimum and maximum nu- of instances running for a certain role and let Azure decide how and when to scale;
- Reactive Rules: followed in response to certain events, these rules are more complex than what Amazon provides us, in particular, we can increase the number of Roles running but Azure also allows you to change the behavior of our application (Throttling) dynamically:
  - Request rejection: Reject requests from certain users based on certain rules e.g. reject 10% of requests from China or from users who have already done so n requests in a given period to lower the load;
  - Disable or degrade certain functionality until the situation is resolved stabilizes;
  - Delay operations performed for low priority applications

How Scaling can be defined in Amazon Groups to group multiple Roles and define the Scaling Rules for all Roles in the group.

6 Containerization/Docker

It is an open source project that aims to automate the application deployment inside a software container, providing a higher level of abstraction than the

of the operating system. The basic idea is to have a uniform packaging system that can run anywhere (x86(_64) and x86(_64) architectures) Linux kernel). The reason why docker is useful is that it works on the Linux kernel and is so independent of the particular distro and I can install it libraries that would generally conflict with the system libraries.

Docker adheres to OCI policies and is not the only system to do it, all the systems that do it adhere in theory should allow interoperability. One of the main objectives is isolate the applications, that is, I want them not to see the system processes (for security reasons). They are also lightweight since they share the OS kernel and there is no virtualiz- tions in between.

The idea is to eliminate hypervisor and eliminate GuestOS, allowing the execution of applications directly on the OS. The technology container has the duality images - container, im-age is a file/template that contains the meta data and configuration and this file is used for create containers, the difference from VMs is that the container is not a copy of the image (see AMI) but rather a performance of it, another difference is the execution time compared to VMs which take times in the order of minutes since docker is a process it starts in orders of seconds.

Images are usually saved on Hub or local re-gistry. The docker **filesystem** is only reading and is layered (UnionFS), creating the image layered allows for maximum reusability of the images, you can start creating an image again starting from an existing one, for example if I have a debian image with emacs on it I can start from debian and put nano on top and to do this I don't have to redownload debian I can reuse already the debian image i have. It can crea- re this layered model following a Copy-on-Write, that is, creates a copy of the modified file. All technologies except the filesystem layers are due using the linux kernel, one of these technologies is the **namespaces:** it allows the isolation of processes, identifying things which can and cannot see the process. the other main technology that allows the execution of the docker is **cgroups:** which is meant to limit and monitor access to resources (CPU, Memory, I/O, network).

Docker is a linux daemon that displays an inter- REST face that is usable via a CLI (command line interface) provided by docker. The

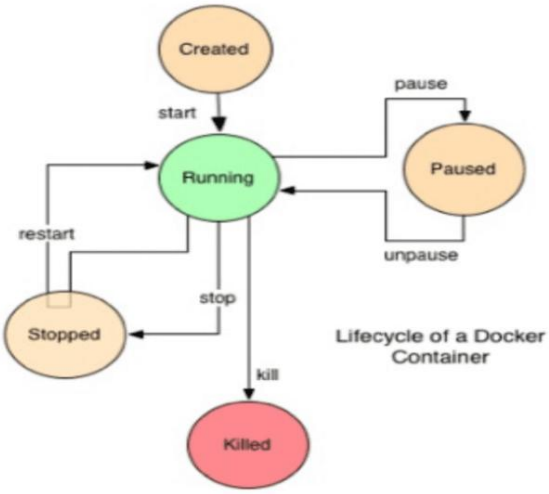

Figure 4: Docker lifecycle

life cycle of a container is as indicated in Figure 4 the main commands are:

- docker create: creates but does not run docker
- docker run: builds and runs docker
- dockerstop
- dockerstart
- dockerrestart

- dockerrm
- dockerkill

Since a process like the other default docker can see all the resources that the computer has available, it is possible to limit with soft limits (lower limit of resources) and hard limits (upper limit) the access to these resources.

When you delete the running image you lose all data, hence the concept of volume which are folders shared with the system (so as not to lose data). There are three types of volumes:

- Bind Mount: is a mounted folder that is bound (linked) to a folder on the host system; • Volume Mount: these are also folders

  shared but managed by docker, avoiding breech problems;
  • tmpfs Mount: folders that

are not on the machine's disk but on memory.

This whole volume model when we are in the cloud does not work, there may be various errors (a VM fails) and the volume is explicitly on the Host, to make the volume persist even in the cloud docker was written in a way to be able to use plugins, the plugin to work locally (the one we have seen) is called local persist, there are various plugins already created by the various cloud providers to work on their system.

The docker building process is the output of a dockerfile that contains the instructions to start and the context that is a series of other necessary information. The instructions are sequential and are executed by the docker daemon. Among the existing commands the most important are:

- **ENTRYPOINT:** defines a process that must be executed when the container is launched, it can have two forms:

ENTRYPOINT["executable", "param1",\ "param2"] (exec form)

ENTRYPOINT executable param1\ (shell
form) param2

The difference is that the shell form is controlled by the shell so for example you can terminate a process with ctrl+c. • **CMD** has the same meaning as

ENTRY-POINT, but has a difference, if an ENTRYPOINT already exists, ENTRYPOINT is executed and CMD passes its parameters to it and if there are multiple CMDs, the last CMD is executed.

Ex:

```
CMD["--port 27017"]
ENTRYPOINT /usr/bin/mongod
# ENTRYPOINT receives parameter from CMD
```

Communication between containers is done through docker networking, which is a network virtualization technology, connected dockers believe they have their own connection, this connection is a docker plugin, this connection is also distributed and decentralized so each docker has a piece of information on the network. This idea is formalized in the CNM (Container Network Model) which consists of 4 main elements:
1. Sandbox: is a structure that contains the network configurations at container level, therefore it manages the interfaces, the routing table and a series of DNS

information. A sandbox can have more than one end-point and connected to multiple networks but each container can only be connected to one Sandbox; 2. Endpoint: is the implementation of the network interface (e.g. veth - Virtual Ethernet) An endpoint can be associated with only one network and only one sandbox;

3. Network: is the set of many endpoints that can communicate with each other;

4. Cluster: The set of dockers connected to a net.

When we create a network this can be of the following type: • NULL:

The container has no access to the network; • Host: The container and the host have the same network interface; • Bridge: It is a service to

which various endpoints are connected; • Overlay: A network that is created

between multiple Docker hosts; • Remote: It is an interface defined in a

such as to be implemented by third parties.

6.1 Docker Compose

It is a declarative language expressible through yaml files, which describes how a multicontainer application such as network and volume is built. In practice it uses docker commands behind and makes the developer's life easier. The docker-compose.yml file is composed of three sections: Services, Networks and Volumes.

A **Docker Swarm** is a cluster of machines that use docker and logically join together in a system, for deployment purposes it is as if they became a single machine. The cluster is no longer managed by individual docker instances on the machines but by a single instance called swarm manager, the other machines are called workers (or nodes). The manager can use different strategies to run containers in the various workers, an example is emptiest node where containers are started in the empty node. Only the manager can decide to execute commands or authorize the addition of other workers to the swarm, workers therefore make available only computational resources and receive orders from the manager.

Docker swarm features:

• Cluster management integrated with docker-machine, does not require additional external software; • Declarative service model;

• Automatic scalability management, also

creating container replicas and constantly continues to check the life status of workers;

• Reconciliation of the current state with the state declared by the user; • Management of multi host

networking with overlay networks; • Implementation of the discovery

service to identify nodes in a unique way; • Load Balancing, in case of existence of various instances of the same service,

these will receive the distributed load; • Secure by default since the various nodes communicate through encrypted http protocols using TLS technologies.

You can have more than one manager but only one is a leader, the others are called Reachable and are a sort of backup leader in case the leader fails, obtaining high availability, they can use consensus protocols to make decisions together.

7 Serverless

There are 2 other models for cloud besides IaaS, PaaS and SaaS: **CaaS** (Container as a Service) and **FaaS** (Function as a Service).

Serverless can be understood as BaaS (Backend as a Service) or FaaS. BaaS is the incorporation of autonomous services in PaaS mode of an application, for example the authentication service. In FaaS the developer does not have to develop entire applications, but only write functions that are as simple as possible, these

functions must have an atomic and granular role.
They are computational containers, which
They contain an ephemeral function, which is
executed in case of an external event and these
functions are managed entirely by the cloud provider.
In this case there is no centralized management,
in fact he prefers choreography (coordination)
of the functions with respect to orchestration (there is no
a central decision maker who calls the functions),
thus obtaining:

- Flexibility and Extensibility: other functions can be added without altering the existing ones pre-existing;
- Division of concerns;
- Reduced cost: You don't pay if the function it's not running, so if a feature is rarely used I don't pay for a server constantly access.

Also I am no longer limited to the languages offered
by the cloud provider, any language that compiles on
a UNIX process (so in
docker) is fine. There are some constraints, for
example in AWS a function can last at most
5 minutes, if a function takes more than 5 minutes
maybe it's better to get an IaaS or PaaS, or
you can split functions into multiple functions.
Furthermore, there is no guarantee that states will
persist across multiple subsequent invocations.
since the variables are saved to memory or a
local disk.
Functions in FaaS are typically triggered by events
defined by the cloud provider e.g.
AWS includes updates to S3, scheduled tasks,
receiving messages in a queue service
(like Kafka) or for responding to HTTP requests using
API Gateway.
Another disadvantage is the so called cold start -
create a new instance, start host functions
etc. and the power-on period depends on the
programming language, the libraries to be loaded
care and the function configuration and the amount
of code, even if they are in the order of seconds
It is very slow compared to warm start, which reuses
an existing Lambda function instance from a
previous call.
The benefits are many:

- Reduced Operational, Development and Execution Costs;
- **Greener** and more efficient computing .

There are also disadvantages:

- Vendor Control: Downtime, limitations, cost changes, product updates API;
- Maintenance problems;
- The duration of execution;
- Initialization latency;
- Testing, Debugging and Monitoring difficult and limited.

So serverless is suboptimal for functions that need
states and are longer in general e.g. Deep Learning
Training,
Heavy Streaming, Hadoop Analytics, DB Management,
Video Streaming etc. are very useful for
fast, stateless and event-driven events such as
microservices, IoT, ML Inferences, lightweight streaming
etc.

8 Apache Spark

The basic idea is not to move data
in the various machines but the codes (data locality),
so once you have obtained the parallelism and given
locality and in addition we implement a system that
be able to manage failures at various levels
then we get the framework called Hadoop.
Spark is slowly replacing part of the Hadoop
ecosystem.
To introduce the concept of operators, we need
understand what functional language is: it is a para-

programming digma, which sees computation as a process of evaluating mathematical functions, these functions must be immutable and stateless. The properties of functional programming: • Composition of functions to obtain higher-order functions • Functions are stateless
(they are pure), they take state thanks to an accumulator which is a variable passed as input
to a function

• They cannot change the status of others functions (no side-effect) •
Possibility of Recursion • Lazy-
Evaluation, execution only when it is really requested

• Functions do not modify data structures, but create new ones during the process. Thanks to these properties you get automatic parallelization, improved performance, no bugs and memory management.

The two most interesting operators for Spark are: • **Map:** a higher-order operator, takes a function and an input list as input and applies this input function to all the elements of the input list, returning another list; • **Reduce:** a higher-order operator, takes a function and an input list as input and applies this
function recursively to all the elements, returning a combination of the list.

8.1 MapReduce

A combined application of these is MapReduce, which is used to process large amounts of data with parallelized algorithms distributed across a cluster. The system shuffles data in the phases between Map and Reduce. The nodes of the system share no a distributed filesystem and are exploited in two phases:

• Map Phase : The master node partitions the files into M splits using a key, and assigns the maps to be executed to the slaves.
These slaves after executing the Map task, write their output to the disk partitioned into R regions through a hash function;

• Other slaves are assigned the **Re-duce task,** each slave reads data from the corresponding mapper disk and groups them by key and performs the Reduce function.

This is the architecture used in Hadoop 1.x, one of its disadvantages is the blocking that is created to the various slaves, since when the Mappers are working the Reducers are stopped and when the Reducers are working the Mappers are stopped, therefore the maximum capacity of the cluster is not exploited.
In Hadoop 2.x and Spark they tried to solve this problem.
Another disadvantage is that at each iteration of Map and Reduce the file is written to disk which slows down the whole process.
Also not all jobs can be written in trees of Map and Reduce operations, e.g. loops over the data itself.

8.2 Spark Ecosystem

This is where an open source application comes into play

: it performs calculations in memory without write them back to disk if not needed and allows to generalize the calculations using acyclic graphs getting much more flexibility. It enlarges the set of operations from just Map and Reduce to **transformations** and **actions,** where transformations and actions are a set of different operations that can be

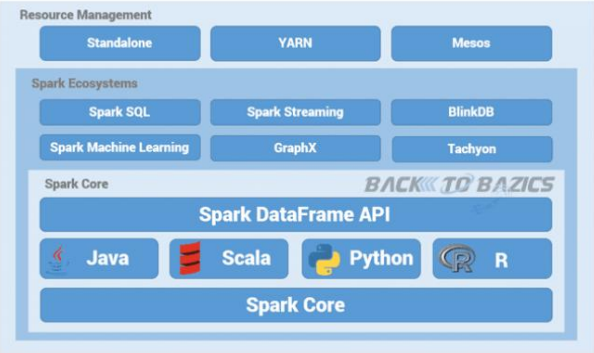

Figure 5: Spark Ecosystem

combined in an arbitrary way and Spark 2 also implemented a query optimizer (Catalyst).

Apache Spark supports: •
Manipulating data through SQL, viewing files as SQL tables; • There is a MLlib library
for Data Science
and distributed machine learning; •
Processing of huge-sized graphs
me;
• Real-time data processing; • Distributed in-memory file system (cache); • Approximate queries on data; • Inherits from Hadoop the
support of Avro, Parquet files and the sources of this data can be HDFS, Cassandra, Hive, Mongo etc.;
• Development in languages such as Python, R, Scala, Java, .Net; At the heart of the entire ecosystem is the Spark Core
which takes care of:

• Distributed computing management, therefore it takes care of the distribution, coordination and scheduling of computational tasks and also takes care of managing failures and minimizes data shuffling between the various nodes; • Provide
APIs for the abstraction of **RDDs,**
which is a collection of immutable, fault-tolerant objects partitioned across a cluster, which can be manipulated in parallel.

Spark can be deployed in 3 different modes: 1. As a
library that runs in a program, providing RDD APIs, but you can only use the resources of the machine on which the program runs; 2. It can be run in a distributed manner (on clusters), which
can be standalone (which does not mean just one machine but simply the fact that it does not need an external scheduler) or using external schedulers including YARN, Mesos and K8; 3. In MR mode, in this case access to storage systems through Hadoop APIs to use HBase, S3,
Cassandra
etc.

Spark applications have two fundamental elements plus an optional third: 1. **Spark**
Driver which is the part of the code that is executed non-parallel, this driver creates a SparkContext to manage the jobs, the SparkContext must understand which pieces of our code are parallelizable and must send them to the cluster;

2. In the various clusters there are **Execu-tor** agents that execute the orders from the SparkCon-text and are able to manage a series of tasks in parallel, once the task is completed the result is sent back to the driver; 3. In the middle there can be a **Cluster Manager** to
allow communication between the driver and the various nodes and to allocate resources.

Context was an object that created connections e.g. to connect SQL you created SQLContext, to connect Hive you created Hive-Context, this created confusion and made it difficult for them to collaborate. These different contexts, in Spark 2, are unified into a single object

called SparkSession, getting an entrypoint
unified way to manage data in spark.

8.3 RDD

RDD (Resilient Distributed Dataset), essentially a distributed and resilient list of objects,
distributed in the sense that they are partitioned (based on a key) in memory of various nodes but is seen as a single list thanks to the key of
partitioning each partition contains a
single record that can be operated independently (similar to the Shared Nothing approach),
This allows them to be managed in parallel with the transformations (Map, Filter) and in
in case of failure they are automatically rebuilt.

They are usually stored in memory but
there is the possibility of writing them to disk, they are typed (Integer, double, Objects), they are immutable (so applying a transformation means
create another RDD) and are lazy evaluated. Some possible transformations are: map, filter,
union, intersection, distinct and join (RDD only
of key-value type).
Each RDD keeps track of the transformations
used to build it, this trace is called li-neage, which can be used to reconstruct the data
lost. Actions can take an RDD and
produce another non-RDD object. Transformations are lazy and are not performed until
there is no action on RDD that summons them.
The Driver plans the execution order by converting the transformations and actions into a directed acyclic graph (DAG), these DAGs track the dependencies (Lineage) and the nodes of the graph are RDDs
while the arches represent the transformations.
Actions are the final piece of the graph and trigger the execution of the DAG, an action for example can be reduce, count, collect, save.

Be careful if I have to perform two actions on the
same DAG, DAG transformations are performed 2 times, to avoid this problem
can insert the cache command before executing actions.
Some transformations, such as
sortByKey, join, groupByKey etc., can have a shuffle which can slow down the system, the
transformations can be:

- Narrow dependency: Each RDD is
  consumed by at most one child partition but
  a child can have multiple parents, so the number of
  partitions between them does not change
  one transformation and another, and it doesn't happen
  the shuffle;
- Wide dependency: Each RDD can be
  used by multiple child partitions, so
  change the number of partitions

Spark makes this distinction because it will perform the Operation Narrow all together in parallel,
spark divides the work into stages, based on the presence of shuffle operations, within each
stage operations can be performed in
parallel, also the transformations in the various stages
They are composed to speed up everything.
The advantages of RDD are low-cost control
level, thanks to the API, on what spark does and are
also typesafe: spark allows us to do
operations based on the RDD type and finally there
teach in depth how spark function.
Its disadvantages are that they are difficult to manage (for a data scientist who
must use DataFrames), also changes the
execution speed also based on the language used (using Java/
Scala you get
a performance boost of up to 2x compared to
python) and finally the optimization of the code
happens and therefore programs are less efficient.

8.4 Spark SQL

It is a library that runs on Spark Core and provides the dataframe as an abstraction of the RDD, with all the properties of the dataframes are tables with rows and columns with headers and indexes etc. At this point you can use technology highly developed SQL database, Spark SQL also implemented an optimizer that is used by many SQL engines. This increases also support for the various types of files and sources that Spark can communicate with.
Dataframes being an abstraction of RDDs have their properties: they are immutable, distributed, they represent a collection, they are lazyily evaluated and the actions are eagerly evaluated.
The result of using dataframes is not only the same speed regardless of language chosen but even an improvement over to work directly on RDD, this also because DataFrames are compressed so they use less memory, and the inefficiency of average programmers is also taken into account speaking.

9 Stream Processing

The difference between batch processing and stream processing is:

Batch Processing works on batches of limited data obtained from a data store and the result is another batch of data. Batch has access to all the data, you can work on something large and complex, latency is measured in minutes, usually one is interested in the output (quality) of data that to latency.

Stream Processing works on a continuous (infinite) stream of data, the result is a new data stream and this is due to the fact that the stream process does not end. The function in this case is applied to a datapoint or a small window of recent data, also the computation must be light since the stream it must keep going and the latency must be near real time or at most seconds.
The main elements of a stream application I am:

1. Source: which produces the input data;
2. Transformations: Takes dataflow input and produce some output;
3. Sink: receives/consumes output data of the transformations;
4. Data Pipeline: is a sequence of transformations, usually represented by a graph direct acyclic;
5. Event/Tuple/message: It is the atomic element of the data flow (like a row of a dataframe).

Generally data stream systems are generally distributed, to handle more data in parallel and therefore with greater speed, but sometimes there are sequential operations and it is done struggle to make them distributed. The incoming data can be partitioned and each operator can be executed in a concurrent manner. It obtains the properties of distributed systems including:
Fault tolerance, Load Balancing.
Streaming status can be very useful, for example to recognize some pattern in the events or for data aggregations, and the status of fault tolerance is handled using the two Checkpoint and Replay mechanisms : periodic saving of the operator's state on a secure storage (e.g. HDFS), in case of failure all data after the last checkpoint will have to be re-analyzed (Replay).

In the case of “unlimited” streaming data, it is usually preferable to use windowing to cut

the data giving them a temporal sense. These windows can be:

- Fixed (Tumbling) window e.g. calculation of something every so many time units • Sliding window e.g. calculation of something in a time unit every so many time units (possibly different from that of the calculation) • Session window, the session is defined as the period of activity terminated by a period of inactivity greater than a certain threshold, therefore they are dynamic windows and are totally data driven, indeed they are very compatible with a distributed system and each partition will have its own window.

There are also other approximation methods, besides windowing, to handle unlimited data e.g. sampling with streaming K-means etc., being approximations there are errors. In general they are very complex algorithms since they must also guarantee an upper bound of the errors.

SQL on Streams is about seeing a data stream as a relational table of infinite dimension, at this point we can use SQL to generate an analysis or transformation, the system shows us a (dynamic) table but underneath it has a distributed streaming system that generates an output stream from an incoming stream.

9.1 Apache Storm

It is designed to support streaming applications and supports up to 1 million messages per second, can scale to thousands of nodes per cluster and is fault tolerant and using third-party tools, in particular Trident, it is also able to apply "exactly once" temporal semantics.

The conceptual model is a very simple and includes: • Tuple: is the core unit (the message) and can be seen as a key-value pair; • Stream: is the infinite sequence of tuples; • Spout: is the source of the stream and its task is to emit tuples; • Bolt: receives the tuples, does some computation (it can also read and write data from a store) and optionally emits other tuples.

The topology is a DAG of spouts and bolts, in which each spout/bolt performs only one task, so when defining the topology you need to specify how the spouts and bolts are connected to each other, the partitioning of the stream data between the bolts is called **stream grouping,** the various types of stream grouping are: • Shuffle: the partitioning is random, the spouts randomly send the data to the bolts they are connected to; • Fields: the partitioning is based on a field of the tuple, it can be useful for example in counting tuples with a certain value in a field; • All: each bolt receives an instance of a tuple; • Custom; • Direct: the source decides which bolt to send the tuple to; • Global: All tuples generated by all instances of a source are sent to a single target.

The total cluster is managed by Nimbus which calculates the assignments and sends them to the ZooKeeper and the ZooKeeper sends them to the various supervisors on the various clusters which download the topology from Nimbus, this topology is run through a Java process. The ZooKeeper also takes care of checking the health of the various nodes with

HeartBeat and if a Worker dies, the supervisor
restart it and if it continues to die Nimbus assigns
the worker (which can contain more than one operator) to
another supervisor, while if the entire node dies the work is
assigned to another
node. If Nimbus were to die, the supervisors
continue to work but reassignment becomes impossible, so
it is better to put Nim-bus with high availability (maybe keep
one of
backup), while Zookeeper is fault tolerant for
if.

Processes that ensure reliability (e.g.
that a tuple is necessarily seen) are:

- Ack and Fail: Each generated tuple has
  a single ID and each processed tuple must be ack and
  fail, in case of successful processing an ack is generated

  and fail in case of timeout, the message is traced by a
  task called “acker” in case of failure the spout can restart
  the
  message if it is programmed that way;
- Anchoring: A bolt can emit a new
  tuple anchored to the input tuple, so in the
  failure case spout tuple to root
  of the DAG will have to be re-executed.

There is an external Trident library that allows
to write even very complex architectures at a high level, while
remaining stateful without
let the low-level user define it, allows
to work at the micro batch level, i.e. it does not work
every single message that reaches him but he consolidates
a number of messages, this improves throughput but worsens
latency. It can get through
from at least one of storm to exactly once.

9.2 Spark Streaming

Spark is made to work with the idea of batch,
to implement streaming it uses the so-called mini-batch,
therefore the latency is increased but
also raises the thoughput. Spark streaming therefore brings
with it all the properties already existing in
spark including: distributivity, availability and scalability.
Operation begins with the receivers
that receive and emit batches that in spark become RDDs.

Thanks to spark streaming, the creation of a lambda
architecture takes place not only in the same
language but even in the same program-
but.

At the core of spark are RDDs which can
be distributed and replicated in memory, then
in case of a node problem I don't lose
the data, and thanks to the lineage I can also understand
from which point the computation should be restarted (unlike
storm which has to restart everything). The discretized RDD
minibatches are
they call **Dstreams,** their measurement is made in
so as to have a latency of about 1 second and
have the potential to be combined with the
batch processing. Also with the union function
it was possible to implement the concept of fi-
window.

The master continues to save the Dstream states
in a checkpoint file and in case of failure of the
master this can be reinitialized from
from this checkpoint file, allowing for quick recovery.

Dstreams require a good knowledge of
programming, so it created its own
abstraction under the name of Spark Structured
Streaming, which has higher level APIs
compared to Dstreams. Dstreams show the
streaming as dynamic tables, while
operations are performed incrementally, so the result is
another dataframe with
time-varying data. The analysis of data coming from the
Output mode is performed based on
to a trigger: which can be a function of the
time session.

9.3 Amazon Big Data

Amazon is the leader in cloud computing, and offers a lot of services directly to him often ready even for scalability so there is not even the need to configure or maintain them, and if a service is not directly offered, it is very likely offered in the marketplace.
The Big Data pipeline is built at a high level from:

- Data Ingestion/Data collection;
- Data Storage;
- Process and analyze data;
- Consume or view.

and its goal is to minimize latency and cost and maximize throughput, in the cloud I try to minimize latency for example by grouping nearby machines and also maximize the throughput thanks to storage scalability and all machines read at the same speed so the reading speed grows linearly as the nodes grow and as far as costs are concerned, they are minimized since I only pay what which I use. The problem remains the total transition to cloud, for example because it can be expensive to transmit all the data, or there may be privacy concerns, there are also hybrid approaches where you can use your own cloud for private data and Amazon for normal data.
Amazon stream storage is possible with: Apache Kafka (there is something similar in amazon called Amazon MQ), Amazon Kinesis and Amazon DynamoDB (similar to SQL streaming of spark). The offered storage files can be HDFS, S3 and Amazon Glacier, they are all three supported by Big Data Framework and are highly available, the choice occurs if the data is hot (they are accessed many times) here it is better to use HDFS or if they are cold (few accesses) here Glacier is better, while S3 is a way of

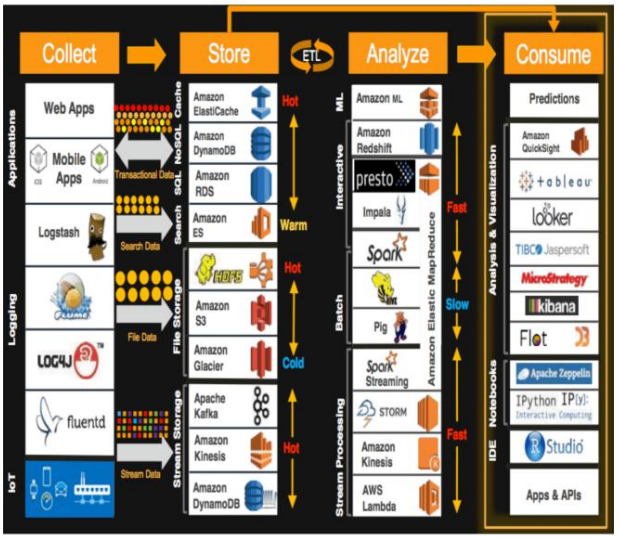

Figure 6: Insight of the Amazon Services

half. Finally for databases and search engines we have: Amazon ElastiCache, DynamoDB, RDS and ES.

Part II - Prof. Melen

10 Introduction

IoT is a global infrastructure for society
(computer science), which creates advanced services
through the interconnection of things based on technologies
interoperable information and communications technologies.
For IoT to work, communication is essential
Machine2Machine is essential.

The main ingredients are therefore: Smart
things, many intelligent objects, autonomous applications
and specialized technologies.

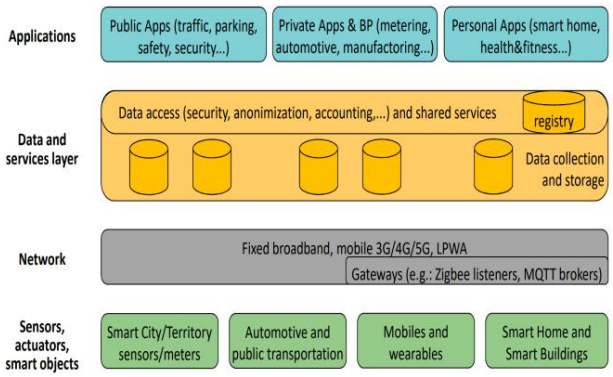

Figure 7: One of the possible layered models

From Figure 7 it is immediately clear that the technologies
IoT enablers are:

- Sensors;
- Connectivity and Protocols;
- Specialized Software and Algorithms for IoT.

The sectors where these technologies have been
developed are: automotive, logistics, manufacturing,
energy and utilities, healthcare, home management and
construction, public sectors such as
lighting etc. and military sector.

10.1 Smart Metering

One of the first things we observe is that a metering
service may need to not
be powered by an electrical network (e.g. gas meter) so
batteries are used, and if it
it runs out you have to go and replace it and this
replacement costs more than a manual check
of the counter.

Enel meters today have 2 chains of
data distribution:

1. Chain 1: goes to the operator who manages the
   network, for official pricing data;
2. Chain 2: goes to the final customers or sellers,
   to allow the management of offers etc.

The idea of smart metering is needed to eventually get
to the smart grid, this is needed because some
power plants have a
fixed production e.g. combustion power plants, but
others have peaks and valleys e.g. wind or solar power
plants. The power plants are very unsuitable
to store energy, one method could
to be offering discounts to the customer for consuming
energy in a certain time slot e.g. if you charge an electric car
in a certain time slot it costs you
less, thus optimising consumption.

11 Sensors

A basic circuit consists of a generator and an impedance
(e.g. resistor, capacitor, inductor), the current passing
through
This impedance generates a potential difference and
therefore a certain voltage level.
Using circuits, physical phenomena can be measured,
using either the voltage on the impedance
(using a voltmeter) or the current flowing
through the circuit (using an ammeter).
The circuit components can change at
depending on a physical phenomenon, and these change