

Architectures for Big Data

Federico Bruzzone

7 ottobre 2022

Indice

1	Course presentation	3
1.1	You are going to learn	3
1.2	Topics Overview	3
1.3	Technologies Overview	4
1.4	Workshops Overview	4
2	Architecture 101	4
2.1	Multiple Views	4
2.1.1	Building Architecture	4
2.1.2	Different Stakeholders	5
2.1.3	Building Software Architecture	5
2.1.4	Zachman Framework for Building	6
2.1.5	Zachman Framework for Information System	6
2.1.6	Different point of views	6
2.2	Architectural Styles	7
2.2.1	Building Architecture	7
2.2.2	Building Software Architecture	7
2.2.3	Elements	8
2.2.4	Enterprise Architecture Styles	8
2.3	Style and Material	9
2.3.1	Building Architecture	9
2.3.2	Building Software Architecture	9
2.4	When an Architecture is designed	9
2.5	Architecture as a framework for abstractions	10
2.5.1	Example of Abstraction	10
2.5.2	System-subsystem Abstraction	11
2.6	BOC-App	12
2.6.1	Bully Operation Center	12
2.6.2	Problem Abstraction	12
2.6.3	SubSystem Perspective	13
2.6.4	Network Perspective	13
2.6.5	SQL through firewall needs care	14
2.6.6	Data Scientist Perspective	14
2.6.7	CFO Perspective	15

1 Course presentation

The course aims at describing **big data processing frameworks**, both in terms of **methodologies** and **technologies**.

Part of the lesson will focus on **Apache spark** and **distributed patterns**.

"May I ask..." a brave student voice break the presentation.

It is not a spurious correlation

- What an Architecture is?
- Why so I need to know this stuff?
- What is this "Hadoop"? Do I really need to know what a Name Node is?
- I would like to put a jBoss inside a Docker to allow Kubernetes load balancing it! (No! This is too much even for a joke)

1.1 You are going to learn

- How to **distribute computation** over clusters using Map Reduce model
- How to write **Apache Spark** code
- How **Hadoop works** and why it works that way
- What a **software architecture** is
- How to design batch architectures to manage **data workflows**
- Several **design patterns** that could be used in a **distributed** environment
- The **limit of traditional SQL** with Big Data

1.2 Topics Overview

1. Enterprise Architectures
2. Design Patterns
3. Hadoop
4. Distributed Algorithms
5. Big Data and SQL
6. Big Data Document
7. Containers

1.3 Technologies Overview

1. Python
2. Apache Spark - Resilient Distributed Dataset
3. ELK Stack: Elastic Search, Logstash, Kibana
4. Docker

1.4 Workshops Overview

1. Workshop 1 - R. Tommasi (Marelli)
2. Workshop 2 - F. Palladino (artea.com)
3. Workshop 3 - D. Malchiodi (Unimi)
4. Workshop 4 - D. Malagodi (Google)

2 Architecture 101

Architectures:

- The art or practice of **designing** and **building** structure and especially habitable ones.
- A unifying or coherent **form** or **structure**

Foundation for the study of Software Architecture / L. Wolf, 1992

Software architecture principles can be **inherited** by appealing to several well-established architectural disciplines.

While the subject matter for the two is quite different, there are a number of interesting **architectural points** in building architecture that are suggestive for software architecture

- multiple **views**
- architectural **styles** item style and **materials** +

2.1 Multiple Views

2.1.1 Building Architecture

Building Architecture uses MULTIPLE VIEWS

A building architect works with the customer by means of a number of different views in which some **particular aspect of the building** is emphasized.

For example, there are elevations and floor plans that give the **exterior views** and "**top-down**" views, respectively.

The elevation views may be supplemented by **contextual drawings** or even scale models to provide the customer with the look of the building in its context.

2.1.2 Different Stakeholders

Each perspective is not just a matter of different level or detail.

It is linked with **different natures** and **accountability**.

- The **Owner** needs the building for a specific purpose. He/she does not know how, but he/she knows perfectly **why**
- The **Architect** needs to project and formalize something that fit completely with owner's needs, to plan the **what**
- The **Builder** needs to design **how** the what will be built matching with natural laws and technological constraints

2.1.3 Building Software Architecture

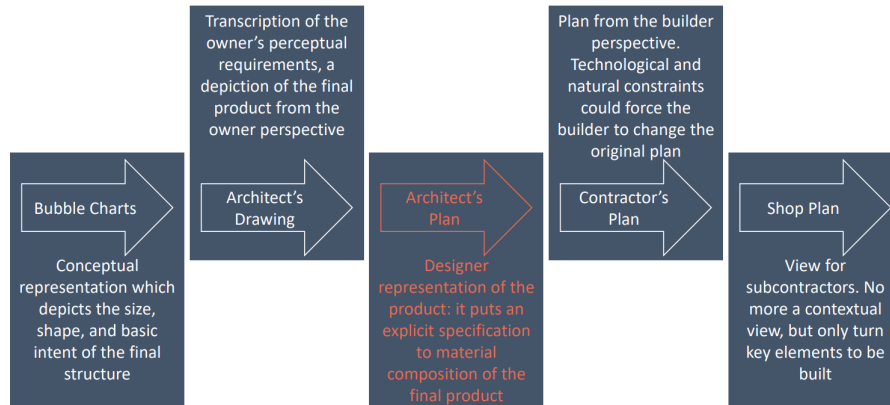
Building Software Architecture uses MULTIPLE VIEWS

Different **type of users** will use Software Architecture: each of them will need a specific point of view.

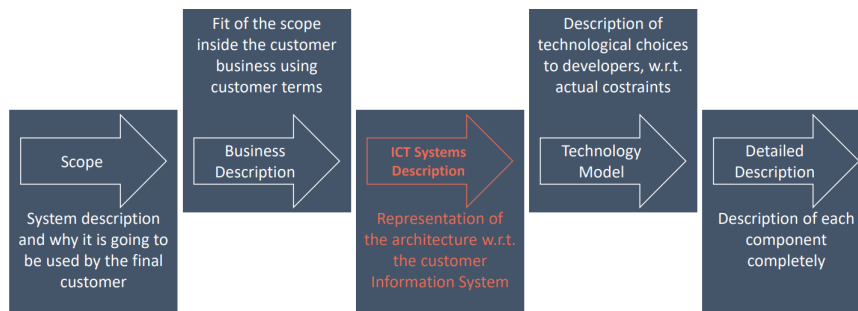
A **Full Stack** developer needs to know how to write code inside the Architecture while a **Data Scientist** where are data they need.

*Since the technology permits **deistributing** large amounts of computing facilities in small packages to **remote location**, some kind of structure (or architecture) is imperative because **decentralization without structure is chaos**.*

2.1.4 Zachman Framework for Building



2.1.5 Zachman Framework for Information System



2.1.6 Different point of views

Each perspective is not just a matter of different level of detail.

It is linked with **different natures** and **accountability**.

- **Input-Process-Output**

Product description in detail w.r.t. intended capabilities, appearance, and interactions with users

- **Entity-Relationship-Entity**

«Stuff things is made of», description of data in each building blocks

- **Node-Line-Node**

Flows between each component

2.2 Architectural Styles

Software Architecture A software architecture is a set of **architectural elements** that have a particular form.

[...]

The architectural form consists of weighted **properties** and **relationship**.

[...]

An underlying, but integral, part of an architecture is the rationale for the various choice made in defining an architecture.

2.2.1 Building Architecture

Building Architecture exploits different ARCHITECTURAL STYLES

Descriptively, architectural style defines a particular codification of **design elements** and formal arrangements.

Prescriptively, style limits the kinds of design elements and their **formal arrangements**.

That is, an architectural style constrains both the **design elements** and the **formal relationship** among the design elements.

2.2.2 Building Software Architecture

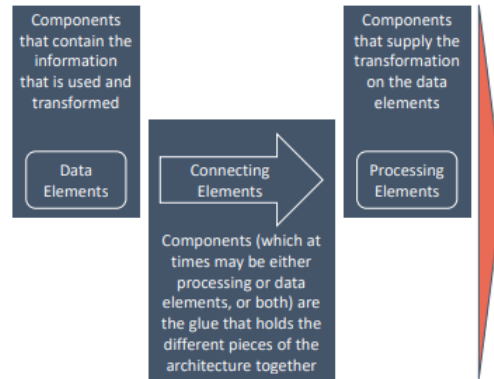
Building Software Architecture exploits different ARCHITECTURAL STYLES

Architectural Style **encapsulates** important decision about elements and emphasizes important constraints on them and their relationships.

We can use Architecture Style both to **constrain** the architecture and to **coordinate** cooperating architects.

Moreover, style **embodies** those decision that suffer **erosion and drift**: an emphasis on it as a constraint on the architecture provides a visibility to certain aspects of the architecture so that violations of those aspects and insensitivity of them will be more obvious.

2.2.3 Elements



Properties are used to **constrain the choice of architectural elements**. They define the minimum desired constraints unless otherwise stated: by default on "what is not constrained by the architect may take any form desired by the designer or implementer"

Relationship are used to **constrain the "placement" of architectural element** - how the different elements may interact and how they are organized with respect to each other in the architecture

Rationale is an underlying, but integral, part of an architecture for the various choices made in defining an architecture. **It captures the motivation for the choice of architectural style, the choice of elements, and the form to satisfy the system constraints**

2.2.4 Enterprise Architecture Styles

1. **1990 - Common Object Request Broker Architecture - COBRA** "Framework to allow objects hosted in different systems to make remote procedures call via a computer network using an Object Request Broker which marshals and serializes these requests"
2. **2003 - Service Oriented Architecture - SOA** "Framework for integrating business processes and supporting IT infrastructure as secure, standardized components - services - that can be reused and combined to address changing business priorities" Bieberstein, Bose et al. 2005
 1. 2012 - Microservices
3. **2004 - Message Oriented Architecture - MOM** "Framework to allow objects hosted in different systems to send messages via a computer network using Message Broker to distribute Application modules over heterogeneous platform"
4. ...

2.3 Style and Material

2.3.1 Building Architecture

Classical Architecture combines STYLE and MATERIALS

The materials have **certain properties** that are exploited in providing a particular style. One may combine structural with aesthetic uses of materials, such as that found in the post and beam construction of tudor-style houses.

However, **one does not build a skyscraper with wooden posts** and beams.

The **material aspects** of the design elements provide both aesthetic and engineering bases for an architecture.

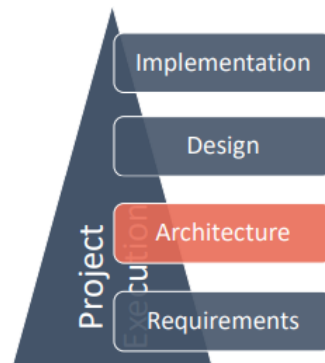
2.3.2 Building Software Architecture

Building Software Architecture combines STYLE and MATERIALS

The same function can be obtained using **different subsystems**.

To train a **Neural Network Python** could be the best fit, while to put the trained Network in production using **FPGA** to physically build the network could be a better solution.

2.4 When an Architecture is designed



- Implementation: Representations of the algorithms and data types that satisfy the **design, architecture** and **requirements**
- Design: Modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the **architecture** and to satisfy the requirements.

- Architecture: Selection of **architectural elements**, their **interactions**, and the **constraints** to provide a framework in which to satisfy the **requirements** and serve as a basis for the **design**
- Requirements: Determination of the information, processing, and the characteristics of that information and processing needed by the user of the system

There new problems involve the system-level design of software, in which the important decisions are concerned with the kinds of modules and subsystems to use and the way these modules and subsystems are organized.

This level of organization, the software architecture level, requires new kinds of abstractions that capture essential properties of major subsystems and the ways they interact.

2.5 Architecture as a framework for abstractions

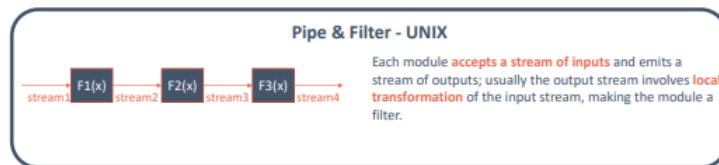
- The essence of **abstraction** is recognizing a pattern, naming and defining it, analyzing it, finding ways to specify it, and providing some way to invoke the pattern by its name without error-prone manual intervention
- This process **suppresses the detail of the pattern's implementation**, reduces the opportunity for clerical error, and simplifies understanding of the result
- In other words, good abstraction is **ignoring the right detail** at the right times

*"The development of **individual abstractions** often follows a common pattern:*

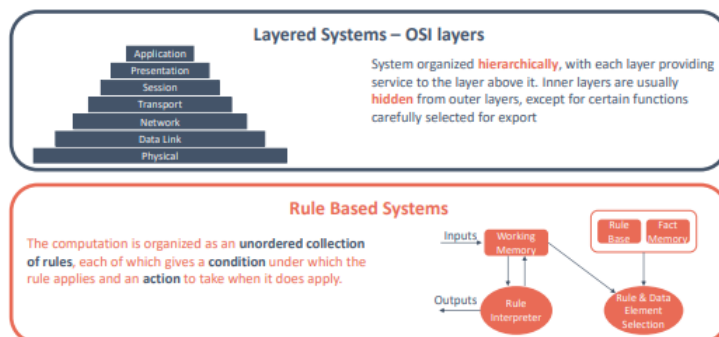
- *First, problems are **solved ad hoc***
- *As experience accumulates, some solutions turn out to work better than others, and **a sort of folklore is passed informally** from person to person*
- *Eventually the useful solutions are understood more systematically, and they are **codified** and analyzed*
- *This in turn enables **a more sophisticated level of practice** and allows us to tackle harder problems"*

2.5.1 Example of Abstraction

First example



Second example



2.5.2 System-subsystem Abstraction

System are constructed by combining **subsystems**:

- independently **compilable modules**, linked by shared data or procedure calls
 - sets of **design decisions** and the **code** that implements them
- Subsystem **have an internal structure**. It is often useful to design that substructure at an architectural level before implementing it

Each subsystem may performs:

- a **specific function** to the system begin implemented
- a **more common function** such as communication or storage

Identifying and **classifying** the system functions that are common to many applications is a significant first step to the development of a software architecture.

2.6 BOC-App

2.6.1 Bully Operation Center

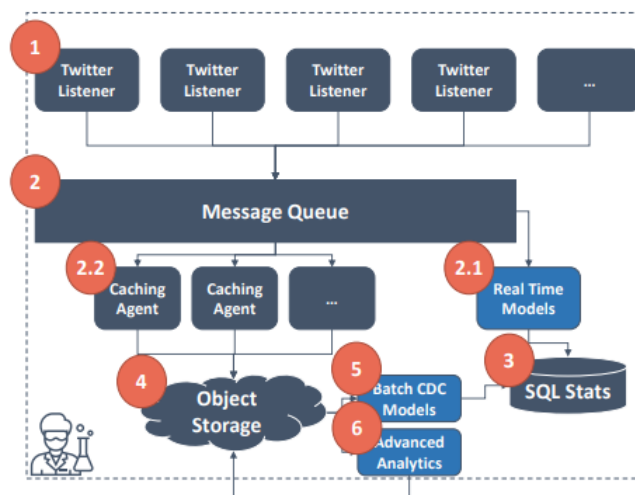
A WebApp supports a **group of Analysts** to spot **aggressive users**.

Natural Language Processing is used to **classify** each tweets.

When the **number of aggressive tweets** done by a given user go beyond a given threshold, this user is surfaced to an Analyst **who can decide to ban** it.

When a user is classified by the Analyst as bully, **the number of bullied users** is computed as the number of users after a bully user comment stop to use tweeter.

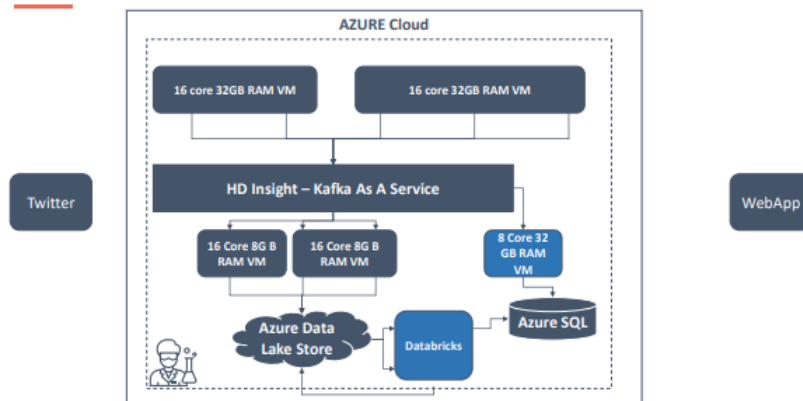
2.6.2 Problem Abstraction



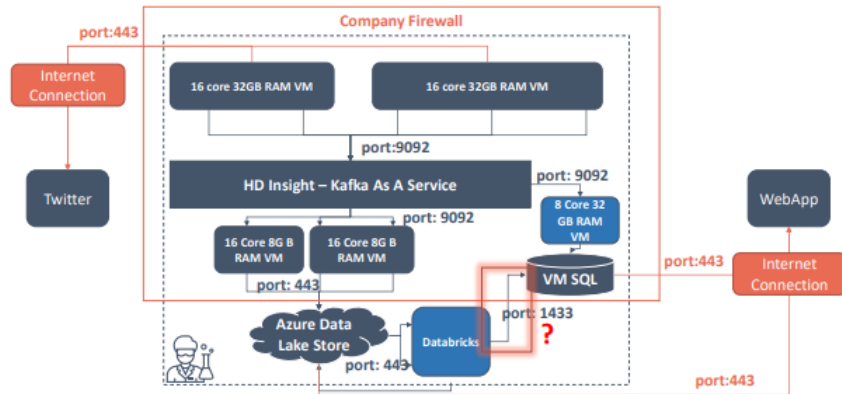
1. Different **Twitter Listeners** ensure:
 - (a) **Scalability**: we can follow as many as hashtag we need
 - (b) **Reliability**: we can re-distribute hashtags to other node if a given Listener fail
2. A **Message Queue** allow serveral concurrent consumers on each hashtag
 - (a) **Real Time Consumer**
 - (b) **Caching Consumer**
3. Data -as-is- is saved using the **Object Storage** model for further analysis
4. Some models **cannot be done in real time** for several reason (e.g., training a classifier)

5. What is I discover a given user is a bully? I will be interested in counting a-posteriori number of users bullied by him

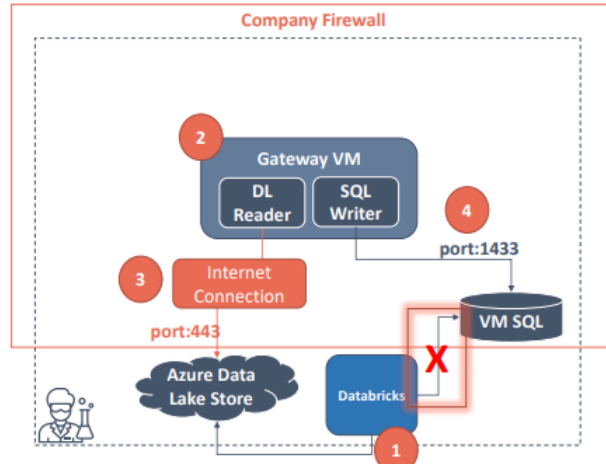
2.6.3 SubSystem Perspective



2.6.4 Network Perspective

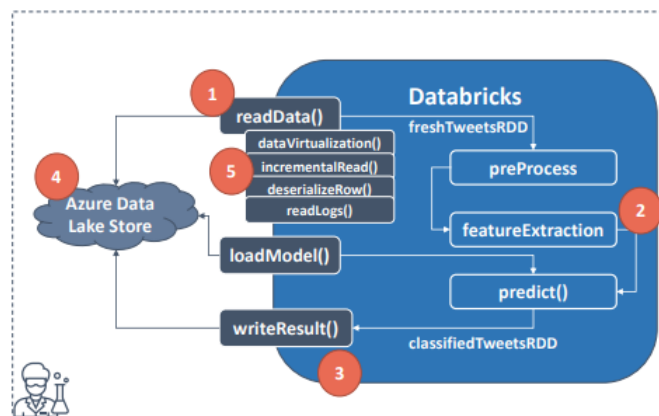


2.6.5 SQL through firewall needs care



1. Databricks writes directly on the **Data Lake**
2. Infrastructure team create a gateway Virtual Machine in the **same network** of the SQL machine
3. A **443 allow rule** for the Gateway is created and a Data Lake reader agent starts to read all fresh data
4. On the same Gateway, a second agent write down data inside the SQL VM

2.6.6 Data Scientist Perspective



1. Data Scientist knows using **BOCArc.readData()** will receive e **fresh tweets**

2. Then he/she can **focus** on the code needed to clean data, extract **features** and **predict**
3. At the end, he/she will call the **BOCArc.writeResult()** method which automagically writes result **somewhere**
4. He/she won't know all tweets are written on the **Data Lake**
5. He/she won't know readData() hides:
 - (a) A **virtualization layer** to decouple physical data with needed one
 - (b) a **CDC strategy implemented** to get only fresh tweets
 - (c) Technical features such as **deserialization** and **log tracing**

2.6.7 CFO Perspective

