DT0222: Software Architecture

a.y. 2017-2018

https://app.schoology.com/course/1179370010/

Henry Muccini
University of L'Aquila, Italy

Masaccio – Monitoring for urbAn SAfety with the IoT

Deliverable D1 Template (v1.0)

Date	13/11/2017
Team ID	VAS

Team Members		
Name and Surname	Matriculation number	E-mail address
Valentina Cecchini	-	*valentina.cecchini@student.univaq.it*
Stefano Valentini	*254825*	*stefano.valentini2@student.univaq.it*
Andrea Perelli	*254758*	*andrea.perelli@student.univaq.it*

Table of Contents

Challenges/Risk Analysis

Risk	Date the risk is identified	Date the risk is resolved	Explanation on how the risk has been managed
Critical messages delivery times (in case of disaster)	13/11/2017	20/11/2017	We plan on deploying redundant servers (located in a different geographic zone, which host the application) that will go online in case of active server's failures.
Big data storage	13/11/2017	17/11/2017	We plan on using a dedicated NoSQL database to handle that amount of data.
Sensors failures	13/11/2017	24/11/2017	We plan on using redundancy of sensors, which will start working in case of failure (of the active sensors).

List of Assumptions

Assumption	Description
Network availability	We assume that we have network coverage across the monitored areas.
Privacy	We assume that we can acquire images of the users in monitored areas.
Microcontrollers	We assume that we can use a microcontrollers to manage each sensor.

The following architecture is designed to be adaptable to various situations. We are planning to instantiate the problem to the monitoring of the UnivAQ's **existing** buildings.
The following are the main services we want to provide:

- **Access control** (only certain users are allowed to enter certain areas)
- **Security monitoring**
  - Air quality control (with respect to, e.g. chemical labs)
  - Smoke detection
  - Firefighting measures
- **Alarm dispatching** (sirens, loudspeakers) in case of emergencies
  - First responders' communication service
- **Smart Information service:** provides situational aware information about the monitored area.

So, in our instance:

Governance → UnivAQ's rector
Areas → Buildings, floors, rooms
Citizens → Students

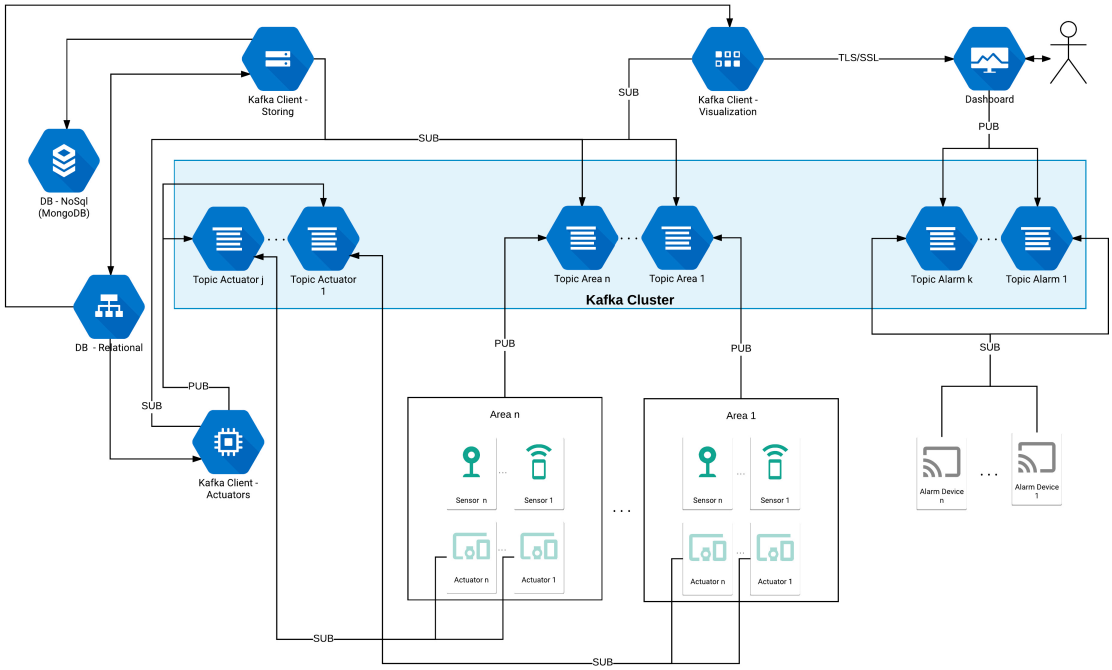
State of the art

In the field of building monitoring and access control (our instance) the main provided services are [1] [2]:

- Access control: installation of card readers at every critical entrance point, which are connected to a central access control panel. You can assign cards or key fobs that grant access via a user interface or software.
- Web-based Dashboard: browser-based systems are easy-to-use and allow you to access your system at any time from any location with internet connectivity, including your mobile device. This means that you don't have to be tied to your security room to terminate access, lock down doors, or change a schedule.
- Notifications: you'll receive a textual notification for events like unauthorized entry and doors forced or left open.
- Video recording: linking your access control system with video surveillance allows you to see exactly who is entering and exiting your building.
- People counters: all buildings are required by law to have an emergency evacuation procedure complete with muster point locations. Access control systems can be designed to provide you with an accurate count of how many people are currently in the building. This allows you to quickly login and check the names and photos of everyone that was in the building before the evacuation.
- Air quality monitoring: air sensors provide novel ways to assess and characterize environments qualitatively and quantitatively in terms of pollution, and human exposure. More specifically, air sensors offer a rare opportunity to assess air quality of indoor environments in real-time. Most IAQ (Indoor Air Quality) sensors, with installed communication protocols, are able to detect and transmit data in real-time to digital platforms, e.g., to a server, PC or smartphone, which in turn broadcast the data to a designated web portal for real-time analysis and visualization.

[1] - http://www.spottersecurity.com/services-access-control-systems/
[2] - http://www.sciencedirect.com/science/article/pii/S0048969716307124

Informal Description of your system and its Software/System Architecture



The system is composed by a **Sensor Network**, an **Actuators Network**, a **Kafka Cluster**, two **DB** (NoSQL and Relational), an **Administration Dashboard** and three **Kafka Clients** (Storing, Visualize and Actuator).

The **Sensor Network** represent the whole sensing subsystem: it senses the environment and communicates the gathered data to the rest of the system publishing on **Kafka Topics**. There will be a Topic for each area (an area may be a building, square, quarter, etc.).

The **Kafka Cluster** is an aggregation of **Kafka Brokers**. A broker receives messages from producers, assigns offsets to them, and commits the messages to storage on disk. It also services consumers (clients), responding to fetch requests for partitions and responding with the messages that have been committed to disk.

The **Kafka Client - Storing** deals with the asynchronous storage of all the raw data collect by the system, by subscribing to the various Area Topics.

The **Kafka Client - Visualize** is responsible to provide refined data to the dashboard and to highlight sensors readings that are out of certain security bounds (those bounds are retrieved from the relational db).

The **Kafka Client - Actuators** reads the data from the sensor network by subscribing to Area Topics and publishes messages on Actuators Topic when certain sensor readings are received, activating the respective actuator(s).

The system stores the raw data on a **NoSQL DB** (MongoDB), this allows for a high flow of data, every reading from a sensor is saved as a *json* file containing the identifier of the sensor, the identifier of the area the sensor belong and the actual reading.

The **Relational DB** is used to store the structure and organization/disposition of the various devices (sensors and actuators) across the areas. It also stores the data regarding the user's authorizations to access certain areas and the various services provided by the system.

The **Administration Dashboard** receives data from the **Kafka Client - Visualize** through a secure connection, then it displays the received data to the user. It also shows a warning when there are sensor's readings that are not within certain "safe" bounds. In those cases, the systems ask the user for a check before emitting an alarm message that, in case of confirmation is published on Alarm Topics that will trigger the respective Alarm Devices.

The **Actuator Network** represent the whole actuation subsystem: the actuator waits for actuation messages to be published on its Actuation Topic, when such message is published the actuator activates and publishes on the Area Topic that it has performed a certain action.

Architectural Pattern

The main architectural pattern used in the system is the Publish/Subscribe pattern. Publish/subscribe messaging is a pattern that is characterized by the sender (publisher) of a piece of data (message) not specifically directing it to a receiver. Instead, the publisher classifies the message somehow, and that receiver (subscriber) subscribes to receive certain classes of messages. Pub/Sub systems often have a broker, a central point where messages are published, to facilitate this.

Publish–subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. This pattern provides greater network scalability and a more dynamic network topology.

To implement this pattern we chose to use **Apache Kafka**. Apache Kafka is often described as a "distributed commit log" or more recently as a "distributing streaming platform." A filesystem or database commit log is designed to provide a durable record of all transactions so that they can be replayed to consistently build the state of a system. Similarly, data within Kafka is stored durably, in order, and can be read deterministically. In addition, the data can be distributed within the system to provide additional protections against failures, as well as significant opportunities for scaling performance.

Sub-systems

- **Monitoring** sub-system: is composed by the Kafka Cluster and Kafka Client - Storing, it is responsible of collecting, storing, organizing and analysing the data coming from the sensor network.
- **Dashboard** sub-system: is composed by the Kafka Client - Visualize and the Dashboard component, it is responsible of displaying the sensors data and highlighting the critical situations.
- **Actuating** sub-system: is composed by the Kafka Client - Actuator and the Actuators Network, the Kafka Client watches for messages published by the various sensors and publishes messages on Actuator Topics that will activate the respective actuator(s).

User Stories

Functional requirements

FR1: the system has to **monitor** several areas.

FR1.1: the system exploits a **sensor** network in order to read data.

FR1.2: the system has to **collect** raw data from sensors.

FR1.3: the system has to **store** the sensed data.

FR1.4: the system has to **organize** the collected data and has to provide a way of checking users accesses on restricted areas.

FR1.5: the system has to **analyse** and produce statistics about the collected data.

FR2: the system has to provide a set of **visualization** tools in order to show the current state of the monitored areas and highlight crucial situations.

FR3: the system has to provide a set of **actuators** that support the human user actions in case of emergencies.

Extra-Functional requirements

EF1: dependability

EF1.1: **availability** - the system has to correctly and continuously operate under the regular hardware state/condition.

EF1.2: safety - the system has to avoid false alarms and other unwanted behaviors that may be dangerous for user's safety.

EF1.3: **security** - the system must ensure that the data and the services are accessed only by the authorized users.

EF1.4: **fault-tolerance** - the system has to be operative even in case of disasters (with an eventual degraded mode) and has to guarantee that no critical messages are lost and delivered in at most 5 seconds.

EF2: **performance** - the system has to handle 40.000 messages per hour coming from (up to) 2.000 sensors, the sensed data are collected within different time ranges (from a few seconds to few minutes, with an average of one message every 180 seconds).

EF3: **scalability** - the system should be easily upgradable (in terms of new areas and sensors added to the network).

EF4: **usability** - the system has to be easy to use for the end user.

User stories

- Citizen

1. *<citizen>, <FR2, FR3, EF1.2, EF1.1, EF4>*
   As a citizen, I want to feel safe in my city: in case there's an emergency I would like to receive a notification of how to behave and possibly which areas of the city it is best to avoid.
2. *<citizen>, <FR3, EF1.3, EF1.1, EF4>*
   As an authorized user, I want to be able to access the areas for which I have permission.
3. *<citizen>, <FR2, EF1.2, EF4>*
   As a citizen, I want to be informed about the crowding of the events that are held in the city: when there are different events I would like to avoid crowded areas because I hate doing long and boring queues to enter in the ones I'm interested in.

- Emergency Operator

1. *<emergency operator>, <FR2, FR3, EF1.1, EF1.4, EF2, EF4>*
   As a safety operator, when there's an emergency, I need to be alerted by a notification that indicates me where the emergency is, so I can go to help people as soon as possible.
2. *<emergency operator>, <FR2, EF1.1, EF1.4, EF4>*
   As a safety operator, I want to be informed about the current situation of the place where I am, to monitor it and avoid critical situation.

- Security Manager

1. *<security manager>, <FR3, EF1.1, EF1.3, EF1.4>*
   As a security manager, I must ensure that only the authorized people access the restricted areas.
2. *<security manager>, <EF1.3>*
   As a security manager, I have to make sure that the collected data is not accessed by unauthorized people.
3. *<security manager>, <EF1.3>*
   As a security manager, I have to guarantee the privacy for the people and keep their sensible data safe.

- Sensor Network Administrator

1. *<sensor network administrator>, <FR1.1, FR1.2, EF1.1, EF1.2, EF1.4, EF2>*
   As sensor network administrator, I have to ensure that all the sensors operate correctly.
2. *<sensor network administrator>, <EF3>*
   As sensor network administrator, I want to be sure that if the number of sensors increases, the system still works as intended.

- Database Administrator

    1. *<database administrator>, <FR1.3, FR1.4, EF1.1, EF1.4, EF2>*
       As a database administrator, I have to ensure that the databases are correctly working at any time.
    2. *<database administrator>, <FR1.3, FR1.4, EF1.3>*
       As a database administrator, I have to be sure that the data is not accessed by unauthorized people.

- Governance

    1. *<governance>, <FR1.5, EF1.1, EF1.2, EF1.3>*
       As the government, I'm concerned about citizens safety, security and information retrieved by data analysis.

- System Administrator

    1. *<system administrator>, <FR1, FR2, FR3, EF1, EF2, EF3, EF4>*
       As a system administrator, I'm concerned about every aspect of the system.

Views and Viewpoints

Stakeholders:

- Citizen
- Emergency Operator: is the user's class that is responsive into the safety of the citizen.
- Security Manager: is the user responsible of the cyber-security of the system.
- Sensor Network Administrator: is the user responsible of the proper functioning of the sensors and the network where the MOSACCIO system is deployed.
- Database Administrator
- Governance: is the customer who requested and paid for the deployment of the system.
- System Administrator: is the user responsible of the whole system.

	Citizen	Emergency Operator	Security Manager	Sensor Network Administrator	Database Administrator	Governance	System Administrator
Security	X		X		X	X	X
Privacy	X		X		X	X	X
Sensing				X			X
Emergency Response		X					X
Energy Consumption				X			X
Networking & Communication				X			X
Usability	X	X					X
Dependability	X	X		X			X
Performance	X	X		X			X
Costs				X		X	X
Citizen Engagement						X	X
Data Analysis					X	X	X

Design Decisions

- **Storing**

The storage is achieved by utilizing two kinds of databases. The NoSQL database (we are planning to use MongoDB) will be used to store raw data, or more in general all the data coming from the various sensors and actuators. The reading of the sensor is stored as a *.json* file containing the id of the sensor, the id of the area the sensor is operating in and the sensor's reading. We chose a NoSQL db because we want to be able to store huge amount of data with the smallest amount of latency we can achieve. MongoDB as a documental db allow us to organize the data in a way that is particularly suited for our use. Allowing the definition of time to live for the stored data, decentralization, great scalability and replication (data redundancy and automatic failover).
The NoSQL db is coupled with a relational db that is used to store the data regarding the authorizations, sensor-area organization, and other general information about the system that needs to have a strict structure.
A dedicated Kafka Client is appointed to the storing of the incoming data. It is subscribed to all the area topics and it is only responsible of inserting all the incoming readings into the NoSQL db, it uses the relational one to verify sensors dispositions etc.
As the client is dedicated we are removing load and overhead from the other component of the system allowing for even more performance.
As requested, a REST interface is used to wrap the system for external (by external we mean "outside" of the system) querying.

- **Sensors organization**

We chose to organize the sensors in areas, each area describes a geographical zone around the city and contains different sensors and actuators. This allow the system to have **great scalability** as if we need to add more sensors/actuators we just need to install them and let them publish their messages to the dedicated area topic.

- **Data acquisition**

We chose to follow the PubSub architectural pattern and to implement it using Apache Kafka. Kafka allows to implement secure real-time streaming data pipelines that reliably get data between systems or applications and secure real-time streaming applications that transform or react to the streams of data [3].
Kafka can be used to stream data, as a message dispatcher and as a storage system.
Security is achieved by encrypting the data transferred between brokers and clients, between brokers, or between brokers and tools using SSL/TLS.
Kafka also allow for a throughput of millions of records/second [4], replication, redundancy, scalability, decentralization and fault tolerance that are the crucial points of our system.
The sensors are organized in areas, each area has its dedicated topic, so the sensor belonging to the area *x,* publish its reading to the topic of the area *x*, the published message consists of the id of the sensor itself and the actual reading.

- **Visualization and analysis**

  A dedicated Kafka Client is responsible to read the incoming data from the area topics (just as the client dedicated to the storing) and transmit them with a secure connection to a dashboard.
  The data can be subjected to analysis and refactoring before being sent to the dashboard.
  The data is displayed in form of graphics, having a dedicated client that directly (i.e. without necessarily going through the database(s)) allow us to eliminate the overhead that would have been present otherwise.
  Sensors readings that are out of certain "safe" bounds (retrieved from the relational db) will trigger a warning notification to be displayed in the dashboard.
  An operator that sees the warning can activate an alarm. Triggering an alarm is achieved by publishing on the topic dedicated to that alarm (or alarms) device(s), these devices will be activated by the published messages.

- **Actuation**

  Another dedicated Kafka Client is subscribed to the area topics. When it reads a message that implies the triggering of an actuator, it publishes to the topic related to that actuator (also more actuators could be subscribed to that topic) activating it (or them).
  As for the sensors organization, this pattern allows for a high factor of scalability, as for introducing more actuators is enough to install them, allow them to publish on their topic and insert the record *sensors reading -> actuation* into the relational db.

[3] - https://kafka.apache.org/intro
[4] - https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

The following are the descriptions of the design decisions that we consider crucial for our architecture.

Concern (Identifier: Description)		*Con#1: How can we gather the data from the sensors?*
Ranking criteria (Identifier: Name)		*Cr#1: Reliability - all messages must be received* *Cr#2: Performance - the messages must be received in real-time* *Cr#3: Fault-tolerance - the system must be resilient to failures and critical conditions* *Cr#4: Scalability - the system must be easily scalable (by means of adding sensors and areas)*
Options	***Identifier: Name***	*Con#1 - Opt#1: **PubSub (Apache Kafka)***
	Description	*The data is collected by allowing the sensors to publish messages containing their readings to specific topics, several clients (dedicated to different aspects of the system) can subscribe to these topics in order to retrieve the data.*
	Status	***decided***
	Relationship(s)	-
	Evaluation	*Cr#1: Kafka guarantees that every message that is published is then received by the subscriber (sooner or later).* *Cr#2: Kafka allows to transmit order of millions of messages per second (that is much more than the required 40.000/hour).* *Cr#3: Kafka implements several fault-tolerance techniques such as: in case of a cluster fails another one takes its place, with all the updated data.* *Cr#4: Kafka allows our system to have great scalability: if we need to add more sensors/actuators we just need to install them and let them publish their messages to the dedicated topic.*
	Rationale of decision	*We **chose** this option as it satisfies all the criteria.*
	Identifier: Name	*Con#1 - Opt#2: **PubSub (plain MQTT)***
	Description	*The data is collected by allowing the sensors to publish messages containing their readings to specific topics, several clients (dedicated to different aspects of the system) can subscribe to these topics in order to retrieve the data.*
	Status	***rejected***
	Relationship(s)	-
	Evaluation	*Cr#1: MQTT guarantees that every message that is published is then received by the subscriber (sooner or later).* *Cr#2: MQTT allows to transmit order of millions of messages per second (that is much more than the required 40.000/hour).* *Cr#3: MQTT as a machine to machine protocol lacks of dedicated fault-tolerance techniques (it requires wrapping).* *Cr#4: MQTT allows our system to have a good enough scalability: if we need to add more sensors/actuators we just need to install them and let them publish their messages to the dedicated topic.*
	Rationale of decision	*We **rejected** this option because it requires wrapping to deliver fault-tolerance.*

Concern (Identifier: Description)		*Con#2: How can we store and organize the data?*
Ranking criteria (Identifier: Name)		*Cr#1: Data organization - the data must be easily organisable* *Cr#2: Scalability - the system must be easily scalable (by means of amount of incoming data)* *Cr#3: Performance - amount of data that can be stored/how fast can we retrieve-insert them*
Options	***Identifier: Name***	*Con#2 - Opt#1:* ***NoSQL DB + relational DB***
	Description	*The sensors readings are stored in the NoSQL DB, the data regarding the organization of the system and the authorizations are stored in the relational DB.*
	Status	***decided***
	Relationship(s)	-
	Evaluation	*Cr#1: The use of a relational DB allows to easily to store the data regarding the system structure.* *Cr#2: The use of a NoSQL DB allows us to store billions of records without performance impacts.* *Cr#3: The NoSQL DB delivers noticeable performance at massive scale: millions of ops/sec, 100s of billions of records, huge amounts of data. Also, splitting the load to two different DBs allows us to avoid further overhead that would derive by using only a DB (we can query the relational DB without impacting on the NoSQL one).*
	Rationale of decision	*We* ***chose*** *this option as it optimally satisfies all the criteria.*
	Identifier: Name	*Con#2 - Opt#2:* ***NoSQL DB***
	Description	*Both sensor's readings and system structure data is stored on the NoSQL DB.*
	Status	***rejected***
	Relationship(s)	-
	Evaluation	*Cr#1: Storing strongly related data in a NoSQL environment is not always easy.* *Cr#2: The use of a NoSQL DB allows us to store billions of records without performance impacts.* *Cr#3: The NoSQL DB delivers noticeable performance at massive scale: millions of ops/sec, 100s of billions of records, huge amounts of data; but all the load is assigned to the NoSQL DB.*
	Rationale of decision	*We* ***rejected*** *this option because it not optimally satisfies the Cr#1.*
	Identifier: Name	*Con#2 - Opt#3:* ***Relational DB***
	Description	*Both sensor's readings and system structure data is stored on the Relational DB.*
	Status	***rejected***
	Relationship(s)	-
	Evaluation	*Cr#1: The use of a Relational DB allows to easily to store the data regarding the system structure.* *Cr#2: The use of the Relational DB does not allow us to easily store huge amount of data, at least not without performance impact.* *Cr#3: The Relational DB usually does not perform well with high amount/volume of data/traffic.*
	Rationale of decision	*We* ***rejected*** *this option because it not satisfies the Cr#2 and Cr#3.*

Concern (Identifier: Description)		*Con#3: Where can we store the data? (in our instance)*
Ranking criteria (Identifier: Name)		*Cr#1: Privacy/Security* *Cr#2: Dependability - the data must be easily organisable* *Cr#3: Costs*
Options	***Identifier: Name***	*Con#3 - Opt#1: **Everything in local storage***
	Description	*All the data are stored in the UnivAQ's servers.*
	Status	***decided***
	Relationship(s)	-
	Evaluation	*Cr#1: The data is stored in a private and closed environment.* *Cr#2: The dependability depends on the dependability of the UnivAQ's infrastructure.* *Cr#3: The costs are near to 0 as we are using an existing infrastructure.*
	Rationale of decision	*We **chose** this option as it optimally satisfies all the criteria: we have low costs, we are certain that the data is only accessed by us and we have a good enough level of dependability.*
	Identifier: Name	*Con#3 - Opt#2: **Everything in the cloud***
	Description	*All the data are stored in the cloud.*
	Status	***rejected***
	Relationship(s)	-
	Evaluation	*Cr#1: Data is located in a not directly controlled environment and it is managed by third parties.* *Cr#2: Maximum level of dependability assured by the cloud provider but it is strongly dependant to the availability of an active internet connection (in case of disasters we could not have an active internet connection).* *Cr#3: The costs could be high with respect to the amount of data we want to store.*
	Rationale of decision	*We **rejected** this option because it does not optimally satisfies Cr#1, Cr#2 and Cr#3.*
	Identifier: Name	*Con#3 - Opt#3: **Part of the data in local storage and part on the cloud***
	Description	*Some of the data is stored in the UnivAQ's infrastructure and some is stored in the cloud.*
	Status	***rejected***
	Relationship(s)	-
	Evaluation	*Inherits the problems from the evaluation of Con#3 - Opt#2*
	Rationale of decision	*We **rejected** this option because it not optimally satisfies Cr#1, Cr#2 and Cr#3.*