

# **Hazelnutella: un sistema di monitoraggio dei dati provenienti da sensori IoT**

Jerin George Mathew, Luca Pasquini <sup>1</sup>

Luglio 2019

<sup>1</sup>Gruppo Bug Data

## Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Progetto PANTHEON . . . . .	2
1.1.1	Architettura software del progetto PANTHEON . . . . .	2
<b>2</b>	<b>Dati a disposizione</b>	<b>5</b>
<b>3</b>	<b>Architettura del sistema e tecnologie usate</b>	<b>8</b>
3.1	Dati prodotti dai sensori IoT . . . . .	9
3.2	Messaging layer . . . . .	9
3.3	Processing layer . . . . .	10
3.4	Storage layer . . . . .	10
3.5	Visualization layer . . . . .	17
<b>4</b>	<b>Conclusioni e sviluppi futuri</b>	<b>19</b>
<b>5</b>	<b>Riferimenti</b>	<b>20</b>

## 1 Introduzione

Il presente progetto, per il corso di *Big Data*, propone un sistema di monitoraggio dei dati provenienti da sensori installati in un nocciolo nel contesto del progetto *PANTHEON*.

### 1.1 Progetto PANTHEON

PANTHEON è un progetto finanziato dall'Unione Europea nel contesto del programma H2020 che si pone come obiettivo quello di realizzare l'equivalente di un sistema industriale di *Supervisory Control And Data Acquisition (SCADA)* nel contesto dell'*agricoltura di precisione*.

In particolare l'obiettivo di PANTHEON è quello di progettare un sistema per la gestione dei noccioli composto da una serie di elementi robotici (sia terrestri che aerei) che si muovono all'interno della coltura di interesse e una rete di sensori IoT, alimentati ad energia solare, per monitorare costantemente le condizioni ambientali della coltura in cui sono installati.

Le informazioni raccolte dai robot e dai sensori vengono poi collezionate in una unità operativa centrale che analizza i dati per poter poi eseguire automaticamente opportune azioni (come ad esempio avviare il sistema di irrigazione) oltre a supportare le decisioni degli agronomi relativamente alla gestione del nocciolo.

In particolare il sistema proposto in PANTHEON prevede di raccogliere dati relativi alle singole piante al fine di monitorare meglio le condizioni complessive della coltura oltre a migliorare l'efficacia e l'efficienza delle operazioni agricole tipicamente effettuate.

Ad esempio, nel contesto dell'irrigazione si può pensare di irrigare un particolare gruppo di piante che hanno particolare necessità di acqua piuttosto che l'intera coltivazione, portando a dei risparmi in termini di quantità d'acqua impiegata per l'irrigazione. Analogi discorsi possono essere estesi all'applicazione dei pesticidi.

#### 1.1.1 Architettura software del progetto PANTHEON

Al fine di gestire l'enorme mole di dati in tempo reale provenienti dai sensori e dai robot, gli autori del progetto PANTHEON hanno ideato una architettura software composta da tecnologie pensate per gestire e manipolare *big data*, come l'ecosistema *Hadoop* e i sistemi *noSQL*.

In particolare l'architettura si compone fondamentalmente di tre blocchi (si vedano anche le figure 2 e 3):

- *Data Collection e Pre-processing layer (DCP layer)*, che è uno strato che viene replicato per ogni nocciolo e si occupa della collezione dei dati provenienti dai sensori e dai robot presenti nell'apezzamento di terreno in questione.

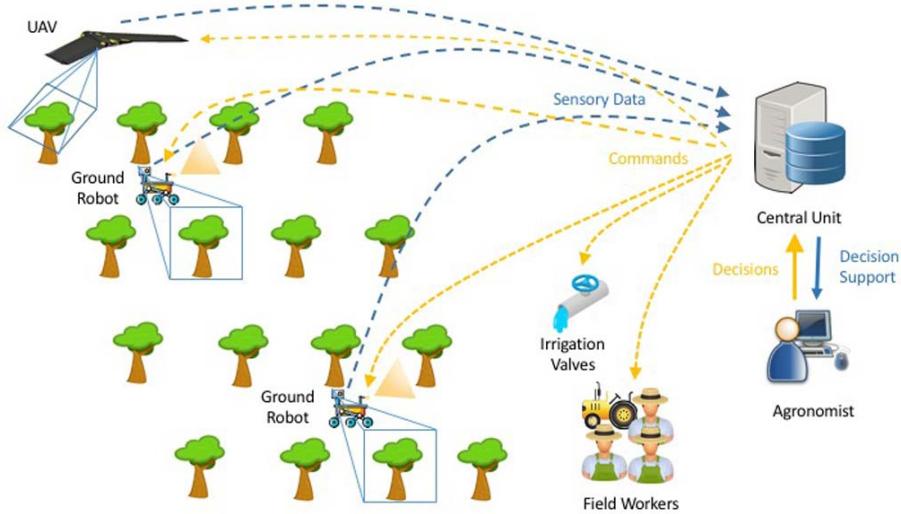


Figura 1: Panoramica del progetto PANTHEON

- *Data Transfer layer (DT layer)*, che è uno strato di middleware che si occupa di consentire il trasferimento di dati tra il DCP layer e il DSP layer (descritto di seguito) in entrambe le direzioni anche in assenza di connessione di rete.
- *Data Storage and Processing layer (DSP layer)*, che consiste in una unità centrale in cui confluiscono i dati provenienti dai vari layer DCP delle varie colture. Tali dati vengono memorizzati e analizzati secondo un approccio *data lake*, dove viene impiegato una grande *repository* centrale per poter memorizzare e processare dati (anche eterogenei) provenienti dalle varie sorgenti.

## 1 INTRODUZIONE

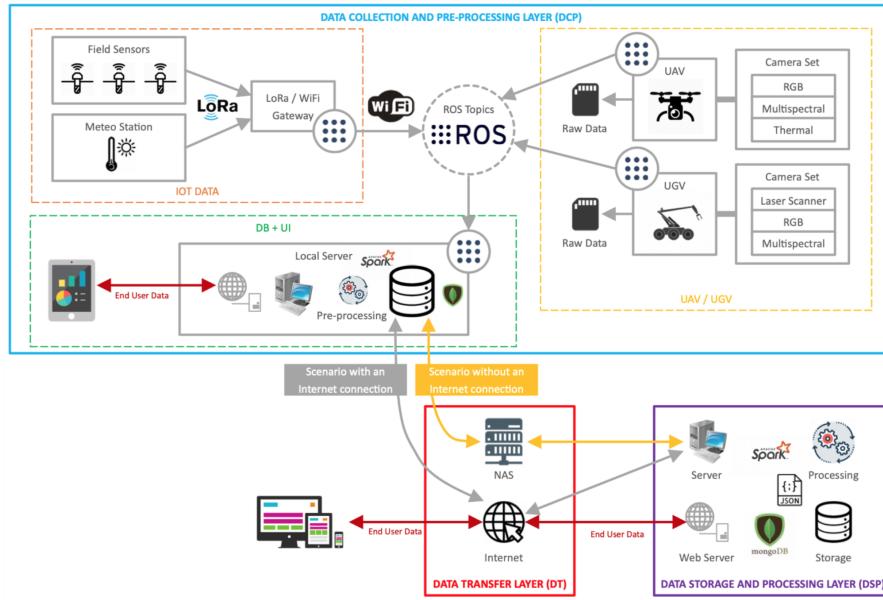


Figura 2: Architettura software del progetto PANTHEON

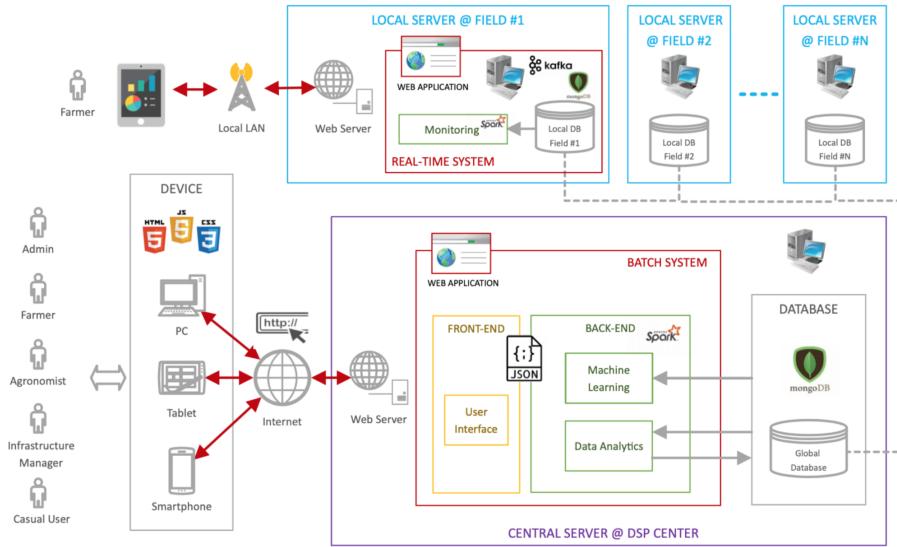


Figura 3: Architettura software del progetto PANTHEON dal punto di vista dell'utente

## 2 Dati a disposizione

Per validare l'architettura proposta dal progetto PANTHEON gli autori dello stesso hanno preso in considerazione un vero noccioleto su cui effettuare le sperimentazioni. Il noccioleto in questione appartiene all'*Azienda Agricola Vignola*, una impresa agricola situata nel comune di Caprarola, in provincia di Viterbo.

In particolare, ai fini della sperimentazione del progetto PANTHEON, sono stati presi in considerazione 3 appezzamenti di terreno di proprietà dell'Azienda Agricola Vignola. Dei 3 appezzamenti, solo 2 sono stati poi effettivamente impiegati per effettuare i test (field 16 e 18 nella figura 4).

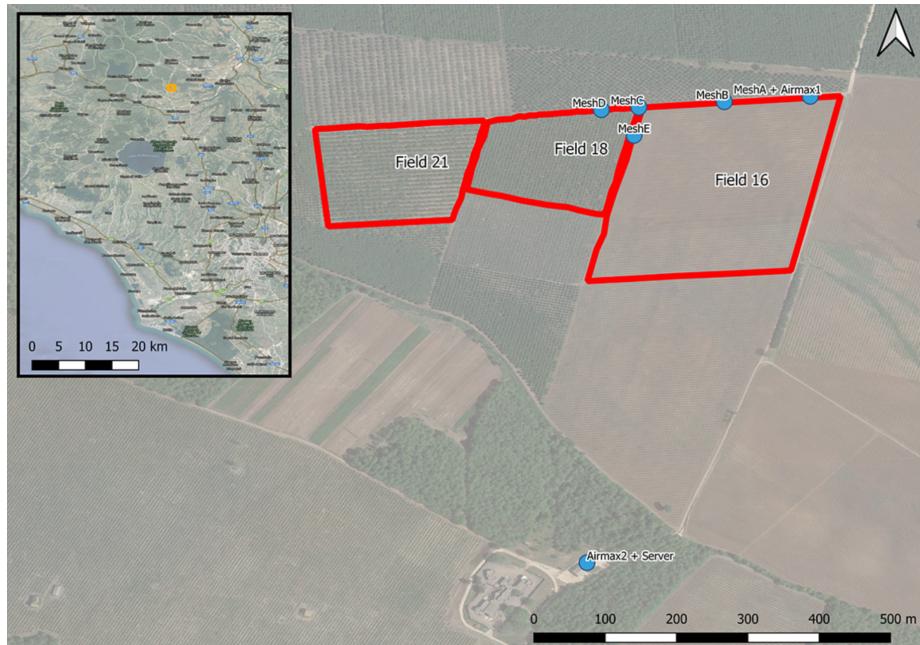


Figura 4: Noccioleti selezionati per le sperimentazioni del progetto PANTHEON

I dati a disposizione ai fini del progetto provengono dai due noccioleti in questione. I dati in particolare consistono in due file in formato *csv*, *pantheon20190612-stazione.csv* e *pantheon20190612-nodi.csv*.

Il primo file in particolare contiene dati climatici raccolti dalla stazione meteo presente nel noccioleto della Azienda Agricola Vignola, mentre il secondo file contiene dati provenienti da 9 sensori installati nel terreno, che raccolgono informazioni relativi alla temperatura e alla quantità di acqua presente nel terreno.

Di seguito vengono riportati i campi che compongono il file *pantheon20190612-stazione.csv*:

- **id\_dato**, che contiene un numero progressivo che identifica univocamente ciascuna riga del file;
- **data\_ora**, contenente l'ora solare in cui sono stati ricevuti i dati della riga in questione;
- **temp1\_media**, valore medio di temperatura registrata (gradi Celsius);
- **temp1\_min**, valore minimo di temperatura registrata (gradi Celsius);
- **temp1\_max**, valore massimo di temperatura registrata (gradi Celsius);
- **temp1\_ur1\_n letture**, numero di campioni usati per calcolare la temperatura media, minima e massima;
- **ur1\_media**, valore medio di umidità registrata;
- **ur1\_min**, valore minimo di umidità registrata;
- **ur1\_max**, valore massimo di umidità registrata;
- **pioggia\_mm**, mm di pioggia caduta;
- **rad W/mq**, radiazione solare (espressa in W/mq);
- **rad\_n letture**, numero di campioni usati per calcolare la radiazione solare (rad W/mq);
- **wind\_dir**, direzione del vento;
- **wind\_dir\_n letture**, numero di campioni considerati per computare la direzione del vento;
- **wind\_speed\_media**, velocità media del vento;
- **wind\_speed\_max**, velocità massima del vento;
- **wind\_speed\_n letture**, numero di campioni considerati per calcolare la velocità massima e media del vento;
- **pressione\_mbar**, pressione atmosferica (in mbar);
- **pressione\_standard\_mbar**, pressione atmosferica corretta al livello del mare (in mbar);
- **pressione\_n letture**, numero di campioni impiegati per calcolare la pressione atmosferica (in mbar);
- **rad W/mq array**, elenco dei valori di radiazione al minuto acquisiti durante l'intervallo di acquisizione.

Si riportano ora i campi che compongono il file  
`pantheon20190612-nodi.csv`:

## 2 DATI A DISPOSIZIONE

- **id\_dato**, che contiene un numero progressivo che identifica univocamente ciascuna riga del file;
- **data\_ora**, contenente l'ora solare in cui sono stati ricevuti i dati della riga in questione;
- **soil\_0\_water\_media**, quantità d'acqua presente nel terreno a 15 cm di profondità (espressa in termini percentuali);
- **soil\_0\_water\_n letture**, numero di campioni usati per calcolare quantità di acqua nel terreno a 15 cm di profondità;
- **soil\_0\_temp\_media**, valore medio di temperatura del terreno a 15 cm di profondità (gradi celsius);
- **soil\_0\_temp\_n letture**, numero di campioni usati per calcolare la temperatura del suolo a 15 cm di profondità;
- **soil\_1\_water\_media**, quantità d'acqua presente nel terreno a 40 cm di profondità (espressa in termini percentuali);
- **soil\_1\_water\_n letture**, numero di campioni usati per calcolare quantità di acqua nel terreno a 40 cm di profondità;
- **soil\_1\_temp\_media**, valore medio di temperatura del terreno a 40 cm di profondità (gradi celsius);
- **soil\_1\_temp\_n letture**, numero di campioni usati per calcolare la temperatura del suolo a 40 cm di profondità;

In particolare i dati registrati dalla stazione meteo vanno dal 12-10-2018 al 12-06-2019.

I dati registrati per i vari nodi/sensori invece variano a seconda del nodo:

- per il nodo 1 e 2 vanno dal 29-01-2019 al 12-06-2019;
- per il nodo 3 vanno dal 18-01-2019 al 03-06-2019;
- per il nodo 4, 5, 7, 8, 9 e 10 vanno dal 07-05-2019 al 12-06-2019 (notare come non siano presenti dati per il nodo 6)

### 3 Architettura del sistema e tecnologie usate

Nel contesto del progetto PANTHEON la presente relazione descrive una possibile implementazione di un sistema di monitoraggio nel contesto del *Data Collection and Pre-processing layer* (si veda il paragrafo 1.1.1).

Il codice del progetto realizzato è open-source ed è disponibile presso il seguente link: [https://github.com/bug-data/Big\\_Data\\_Second\\_Project](https://github.com/bug-data/Big_Data_Second_Project).

Nello specifico è stato realizzato un sistema che acquisisce dati dai “sensori” (in realtà dai file csv a disposizione), li processa, li salva su un database e infine visualizza i dati memorizzati su browser in tempo reale.

Al fine di realizzare questo sistema è stato utilizzato *docker compose* (<https://docs.docker.com/compose>), uno strumento per la *composizione* di contenitori, ovvero uno strumento che consente di realizzare applicazioni composte da più contenitori che comunicano tra loro.

In particolare, grazie a docker compose, i vari servizi che compongono il sistema realizzato (come il servizio di processamento dei dati e il servizio di memorizzazione dei dati) sono stati eseguiti come dei *container Docker* e collegati tramite una rete virtuale che consente ai container di poter interagire tra loro.

Di seguito si riporta schematicamente l’architettura del sistema realizzato.

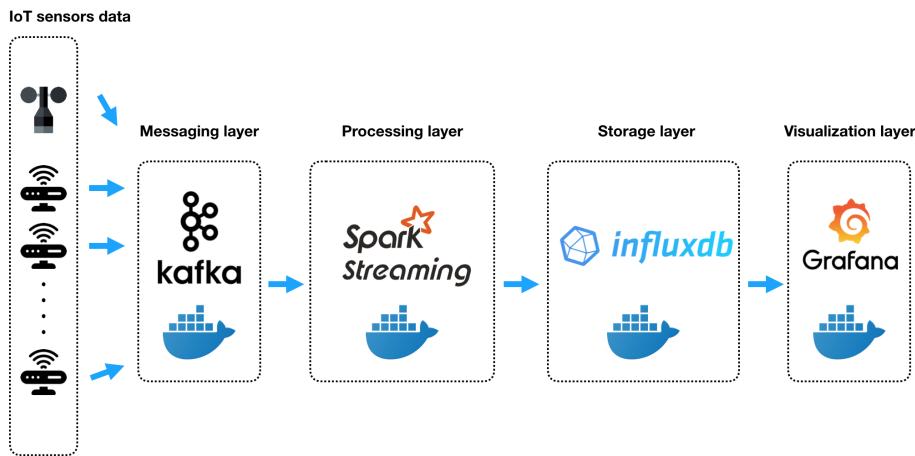


Figura 5: Rappresentazione schematica del sistema di monitoraggio realizzato

Dunque il sistema realizzato è composto fondamentalmente da 5 blocchi:

- *IoT sensors data*, che sono i sensori installati nel nocciolo, i quali rilevano dati che vengono poi inviati al blocco successivo. Come si è detto in precedenza abbiamo simulato la presenza di questi sensori trasmettendo le righe dei file a disposizione al blocco successivo dell’architettura;

- *Messaging layer*, che si occupa di memorizzare i dati/messaggi provenienti dai sensori (leggasi file csv) per poi fornirli al blocco che si occupa del processamento dei dati;
- *Processing layer*, che si occupa di processare in tempo reale i dati provenienti dal messaging layer;
- *Storage layer*, che si occupa di memorizzare i dati processati dal blocco precedente;
- *Visualization layer*, che si occupa di visualizzare i dati memorizzati in tempo reale.

Nelle sezioni a seguire verranno descritti più in dettaglio i blocchi che compongono l'architettura in figura 5.

#### 3.1 **Dati prodotti dai sensori IoT**

Come detto in precedenza nel paragrafo 2, sono stati messi a disposizione, ai fini del progetto, dati provenienti sia dalla stazione meteo che dai sensori installati a terra, che misurano l'umidità e la temperatura del terreno.

Al fine di simulare l'invio di dati dai sensori al messaging layer (realizzato usando Kafka), è stato realizzato uno script, scritto in Python, che legge ciascuna riga del file in input, la converte in un JSON (usando come chiavi del JSON gli stessi campi riportati nel file csv in input) e lo trasmette al container su cui è in esecuzione Kafka (raggiungibile all'endpoint `kafka:9092`) ad intervalli di 1 secondo. Lo script in questione è stato poi eseguito fornendo in input i file `pantheon20190612-stazione.csv` e `pantheon20190612-nodi.csv`.

Per consentire allo script Python di poter interagire con Kafka è stato utilizzato il package `kafka-python` (<https://github.com/dpkp/kafka-python>) che consiste in un client Kafka per il linguaggio Python.

#### 3.2 **Messaging layer**

I messaggi trasmessi attraverso lo script Python descritto in precedenza giungono quindi al messaging layer, realizzato con Kafka. Il vantaggio di usare un blocco intermedio tra i produttori di dati (sensori IoT) e i consumatori (Spark Streaming) sta nella possibilità di disaccoppiare gli uni dagli altri. Ad esempio, grazie a questo strato intermedio, sarebbe possibile utilizzare una diversa tecnologia per processare gli stream di dati prodotti dai sensori (ad esempio Apache Storm) senza che questo abbia alcun impatto sui produttori di dati. In assenza di questa indirezione offerta da Kafka sarebbe stato necessario configurare i sensori affinché inviassero dati al cluster su cui è in esecuzione Apache Storm.

Tornando alla descrizione dell'architettura del sistema realizzato, come detto in precedenza, i messaggi prodotti dallo script Python vengono poi trasmessi e memorizzati in un container su cui è in esecuzione Kafka.

In particolare i messaggi provenienti dalla stazione meteo e dai nodi vengono memorizzati in *topic* differenti. Più precisamente, all'avvio del contenitore su cui è in esecuzione Kafka, vengono configurati 10 topic, di cui 1 per la stazione (**weather**) e 9 per ciascun nodo o sensore (**nodeN**, dove N è l'identificativo del nodo, e.g. **node1**, **node2** ecc.).

I topic sono stati configurati a mezzo della variabile d'ambiente **KAFKA\_CREATE\_TOPICS** e ciascun topic ha un fattore di replicazione pari ad 1 (solo per fini di test) ed un numero di partizioni pari ad 1 (per preservare l'ordine temporale in cui vengono ricevuti i dati quando quest'ultimi vengono passati al processing layer).

#### 3.3 Processing layer

I dati memorizzati su Kafka vengono poi inviati ad un cluster di container Spark composti da un singolo *master* e da un singolo *worker*. Il processamento dei dati avviene tramite Spark Streaming. Gli script per processare i dati sono stati realizzati in Python.

Al fine di poter leggere lo stream di dati memorizzato in Kafka si è fatto ricorso alla libreria Java *spark-streaming-kafka*.

Per quanto riguarda il processamento dei dati, durante l'esecuzione dello script Spark vengono letti da Kafka i messaggi provenienti dai 10 topic configurati in precedenza. Tali dati confluiscono poi in input a Spark Streaming come uno stream di dati. La durata dei *minibatch* di dati su cui opera Spark Streaming è stata configurata a 10 secondi.

Successivamente i dati vengono aggregati in base al topic, per poi procedere effettivamente con l'elaborazione vera e propria dei dati. In particolare per ciascun minibatch viene calcolato il valore medio di ciascun campo di ciascun topic (al fine di ridurre la mole di dati da salvare sullo storage layer) e vengono calcolati ulteriori parametri come ad esempio l'escursione termica.

Infine, i dati così processati vengono salvati sullo storage layer, che consiste in un container su cui è installato InfluxDB. Il salvataggio dei dati prodotti da Spark Streaming su InfluxDB avviene a mezzo del package Python *influxdb* (<https://pypi.org/project/influxdb/>).

#### 3.4 Storage layer

InfluxDB è un base di dati che appartiene alla categoria dei *Time Series Database (TSDB)*, ovvero una tipologia di database ottimizzati per *time series data*. I time series data sono dati o eventi che vengono registrati, aggregati e monitorati nel tempo, come ad esempio la quantità di CPU e RAM usata in un server, i dati provenienti da sensori, o ancora le transazioni effettuate in borsa e così via.

Tipicamente queste tipologie di dati sono accomunate da analoghe elaborazioni. Difatti i dati relativi ad eventi che sono avvenuti in tempi non molto recenti vengono spesso aggregati al fine di ridurre lo spazio su disco o addirittura

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

cancellati perché non più necessari. Allo stesso modo, queste tipologie di dati vengono letti specificando spesso un intervallo di tempo.

Un time series database è una particolare tipologia di basi di dati per la memorizzazione di misure o eventi che sono associati ad un timestamp ed è ottimizzato per misurare i cambiamenti nel tempo di una data metrica (come ad esempio la percentuale di CPU utilizzata nell'esempio precedente).

Notare come nell'ultimo anno i TSDB siano stati la categoria di database a maggiore crescita in termini di popolarità secondo DB-Engines ([https://db-engines.com/en/ranking\\_categories](https://db-engines.com/en/ranking_categories))

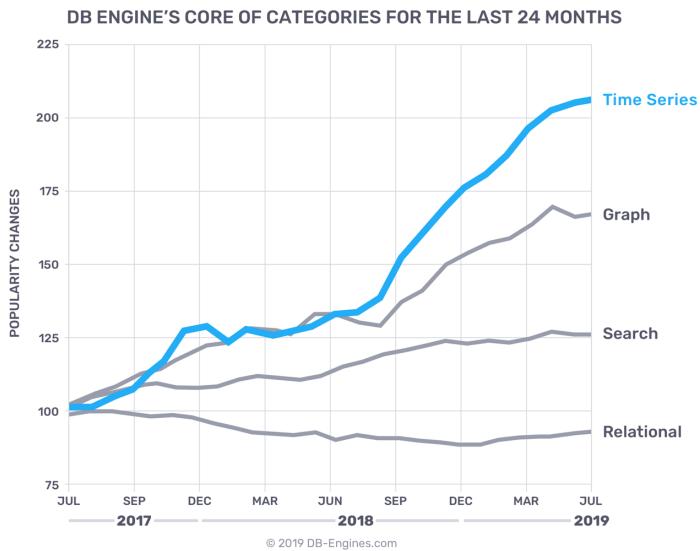


Figura 6: Trend dei diversi modelli di database nell'ultimo anno in termini di popolarità

Dunque i TSBD sembrano prestarsi bene al contesto in cui si è svolto il progetto descritto nella presente relazione e questo ci ha portato a selezionare un TSDB per memorizzare i dati prodotti da Spark Streaming.

Nel panorama dei TSDB la nostra scelta è poi ricaduta su InfluxDB. Questa scelta è motivata dalle seguenti ragioni:

- Popolarità: tra i time-series databases, InfluxDB si classifica al primo posto secondo DB-Engines (figura 7), cosa che può suggerire il fatto che si tratta di una tecnologia matura
- Compatibilità con *Grafana* (che verrà descritto nella sezione successiva)

Si riportano di seguito i concetti di base di InfluxDB:

- *Measurement*, che corrisponde al concetto di *tabella* in un RDBMS;

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

RANK JUL 2019	DBMS	SCORE		
		JUL 2019	24 MOS ▲	12 MOS ▲
1	InfluxDB	18	+9.89	+6.30
2	Kdb+	5.88	+4.30	+2.70
3	Prometheus	3.46	+2.90	+2.06
4	Graphite	3.44	+1.40	+0.89
5	RRDtool	2.78	-0.25	+0.13
6	OpenTSDB	2.3	+0.50	+0.56
7	Druid	1.85	+0.83	+0.67
8	TimescaleDB	1.27	+1.27	+1.11
9	KairosDB	0.53	-0.03	+0.08
10	eXtremeDB	0.43	+0.07	+0.10

Source: DB-Engines

23 Systems in Ranking, July 2019

Figura 7: Ranking dei TSDMB secondo DB-Engines

- *Tag*, che può essere visto come una colonna indicizzata in un RDBMS (ad esempio un tag potrebbe corrispondere all'identificativo di ciascuno dei 9 nodi/sensori);
- *Field*, che corrisponde al concetto di colonna (non indicizzata) in un RDBMS;
- *Point*, che corrisponde al concetto di riga in un RDBMS.

Si riporta di seguito una sequenza di dati di esempio rappresentati sia in un RDBMS che in InfluxDB:

---

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

park_id	planet	time	#_foodships
1	Earth	14291856000000000000	0
1	Earth	14291856010000000000	3
1	Earth	14291856020000000000	15
1	Earth	14291856030000000000	15
2	Saturn	14291856000000000000	5
2	Saturn	14291856010000000000	9
2	Saturn	14291856020000000000	10
2	Saturn	14291856030000000000	14
3	Jupiter	14291856000000000000	20
3	Jupiter	14291856010000000000	21
3	Jupiter	14291856020000000000	21
3	Jupiter	14291856030000000000	20
4	Saturn	14291856000000000000	5
4	Saturn	14291856010000000000	5
4	Saturn	14291856020000000000	6
4	Saturn	14291856030000000000	5

Figura 8: Esempio di dati memorizzati in un RDBMS

---

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

```
name: foodships
tags: park_id=1, planet=Earth
time          #_foodships
-----
2015-04-16T12:00:00Z 0
2015-04-16T12:00:01Z 3
2015-04-16T12:00:02Z 15
2015-04-16T12:00:03Z 15

name: foodships
tags: park_id=2, planet=Saturn
time          #_foodships
-----
2015-04-16T12:00:00Z 5
2015-04-16T12:00:01Z 9
2015-04-16T12:00:02Z 10
2015-04-16T12:00:03Z 14

name: foodships
tags: park_id=3, planet=Jupiter
time          #_foodships
-----
2015-04-16T12:00:00Z 20
2015-04-16T12:00:01Z 21
2015-04-16T12:00:02Z 21
2015-04-16T12:00:03Z 20

name: foodships
tags: park_id=4, planet=Saturn
time          #_foodships
-----
2015-04-16T12:00:00Z 5
2015-04-16T12:00:01Z 5
2015-04-16T12:00:02Z 6
2015-04-16T12:00:03Z 5
```

Figura 9: Esempio di dati memorizzati in InfluxDB

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

Con riferimento ad InfluxDB in questo esempio:

- `foodship` corrisponde alla measurement
- `park_id` e `planet` sono i tag
- `#_foodships` corrisponde ad un field

Notare come in InfluxDB sia presente per ciascuna riga una colonna aggiuntiva, di nome `time` che rappresenta un timestamp associato alla riga in questione.

Per quanto riguarda i dati processati da Spark Streaming, questi sono stati salvati in due measurement distinte:

- **Stazioni**, che contiene i dati processati da Spark Streaming relativi alla stazione meteo. In particolare sono stati creati tanti field quanti sono i campi processati da Spark relativamente alla stazione meteo (ad esempio `temp1_min`, `temp1_max` ecc.). Inoltre è stato utilizzato l'identificatore di riga (campo `id_dato` dei file csv) come tag.
- **Nodi**, che contiene i dati processati da Spark Streaming relativi ai sensori/nodi. In particolare i dati dei vari sensori vengono distinti a mezzo di tag della forma `nodeN` (e.g. `node1` per il primo nodo, `node2` per il secondo e così via). I field della measurement corrispondono ai campi che sono stati processati con Spark Streaming (ad esempio `soil_0_water_content`, `soil_1_water_content` ecc.)

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE

---

```

[root@9d56769cd341:/# influx
Connected to http://localhost:8086 version 1.7.7
InfluxDB shell version: 1.7.7
> use pantheon
Using database pantheon
> select * from Nodi where cod_nodo='TN_01' limit 40
name: Nodi
time           brushId cod_nodo soil_0_temp_media soil_0_water_media soil_1_temp_media soil_1_water_media
----           -----
1548839405000000000 702  TN_01  3.2599999999999993 0.2          5.24          0.2
1548843905000000000 717  TN_01  3.1399999999999997 0.2          5.2          0.2
1548844405000000000 732  TN_01  3.04          0.2          5.18          0.2
1548857405000000000 762  TN_01  3.0300000000000002 0.1999999999999998 5.1000000000000005 0.1999999999999998
1548931205000000000 774  TN_01  2.9           0.3          4.4          0.3
1548935785000000000 784  TN_01  2.92          0.3          4.4          0.3
1548940285000000000 794  TN_01  3.1399999999999997 0.3          4.4          0.3
1548944785000000000 804  TN_01  3.6           0.3          4.4          0.3
1548949285000000000 814  TN_01  4.2           0.3          4.4          0.3
1548953785000000000 824  TN_01  4.5600000000000005 0.3          4.4          0.3
1548958285000000000 834  TN_01  4.8           0.3          4.4          0.36
1548962785000000000 844  TN_01  4.92          0.3          4.4          0.4
1548967285000000000 854  TN_01  5           0.3          4.5200000000000005 0.3
1548971785000000000 864  TN_01  5           0.3          4.6          0.3
1548976285000000000 874  TN_01  5.0200000000000005 0.3          4.68          0.36
1548980785000000000 884  TN_01  5.1           0.3          4.7          0.48
1548985285000000000 894  TN_01  5.1           0.3          4.76          0.5
1548989785000000000 904  TN_01  5.119999999999999 0.3          4.8          0.5
1548994285000000000 914  TN_01  5.2           0.3          4.839999999999999 0.5
1548998785000000000 924  TN_01  5.26          0.3          4.9          0.4200000000000004
1549003285000000000 934  TN_01  5.359999999999999 0.3          5          0.4400000000000006
1549014985000000000 943  TN_01  5.5600000000000005 0.3          5.32          0.48
1549015485000000000 953  TN_01  6.2           0.3          5.939999999999995 0.4
1549023985000000000 963  TN_01  6.5400000000000001 0.3          5.880000000000001 0.4199999999999993
1549028485000000000 973  TN_01  6.859999999999999 0.3          5.74          0.3
1549032985000000000 983  TN_01  7.159999999999999 0.3          5.7          0.3
1549037485000000000 993  TN_01  7.42          0.3          5.7          0.3
1549041985000000000 1003  TN_01  7.6           0.3          5.7          0.3
1549046485000000000 1013  TN_01  7.74          0.3          5.8          0.3
1549058985000000000 1023  TN_01  7.82          0.3          5.8          0.3
1549055405000000000 1033  TN_01  7.92          0.3          5.92          0.4400000000000006
1549055985000000000 1043  TN_01  8.02          0.3          6.040000000000001 0.4
1549064405000000000 1053  TN_01  8.1           0.3          6.26          0.5
1549063905000000000 1063  TN_01  8.3           0.32          7.58          0.5
1549073405000000000 1073  TN_01  8.74          0.34          8.91999999999998 0.5
1549077905000000000 1083  TN_01  9.18          0.36          9.32          0.5
1549082405000000000 1093  TN_01  9.639999999999999 0.36          9.580000000000002 0.5
1549086590500000000 1103  TN_01  9.5           0.3          9.440000000000001 0.38
1549091405000000000 1113  TN_01  9.419999999999998 0.3          9.04          0.32
1549095905000000000 1123  TN_01  9.4           0.3          8.74          0.3
> 

```

Figura 10: Prime 40 righe della tabella Nodi contenente i dati processati per il nodo con identificativo “TN\_01”

### 3.5 Visualization layer

Infine per visualizzare i dati che vengono periodicamente memorizzati in InfluxDB è stato impiegato Grafana. Si tratta di uno strumenti di visualizzazione di serie temporali, accessibile tramite browser, che supporta una serie di database da cui leggere i dati quali *Graphite*, *MySQL*, *PostgreSQL*, *ElasticSearch* e molti altri oltre al già citato InfluxDB.

In particolare grazie a Grafana è stata realizzata una *dashboard* in cui sono state riportate le serie temporali associati a ciascuno dei campi processati tramite Spark Streaming e memorizzati in InfluxDB. Si riporta di seguito una porzione della dashboard realizzata.

Si precisa come di default Grafana richieda all'avvio di effettuare il login (lo username e la password di default sono *admin*, *admin*) di specificare la sorgente da cui leggere i dati (nel nostro caso InfluxDB) per poi realizzare la dashboard attraverso l'interfaccia grafica di Grafana. Per automatizzare il processo di creazione della dashboard, si è proceduto nella seguente maniera: è stato dapprima creata la dashboard “a mano” attraverso l'interfaccia grafica di Grafana, per poi salvarlo in formato JSON (cliccando sulla opzione messa a disposizione da Grafana per esportare le dashboard). Successivamente, modificando opportuni file di configurazione è stato possibile disabilitare ad ogni avvio il login (ai soli fini del progetto) e caricare in automatico la dashboard a partire dal file JSON scaricato in precedenza.

Si riporta di seguito una istantanea di una porzione della dashboard realizzata:

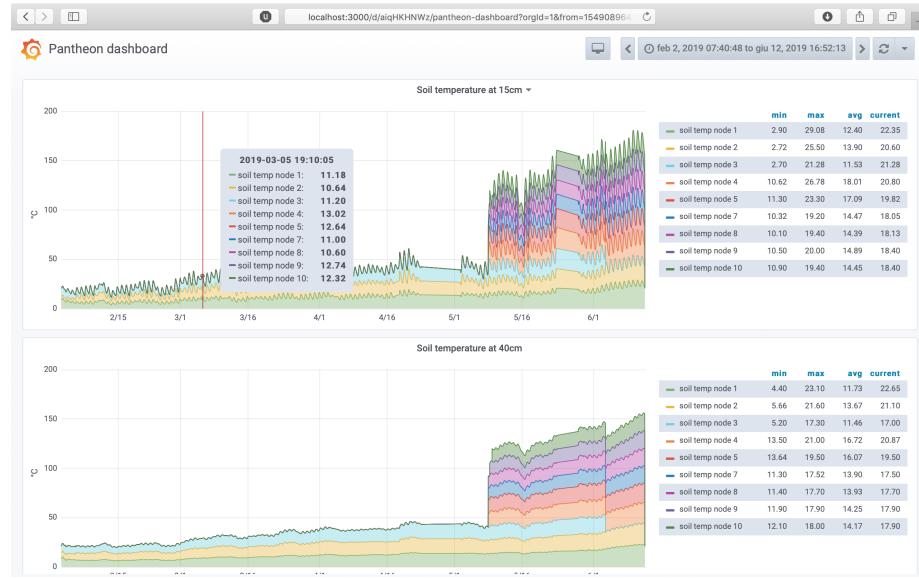


Figura 11: Porzione della dashboard realizzata con Grafana che mostra la temperatura del suolo registrata dai vari sensori

### 3 ARCHITETTURA DEL SISTEMA E TECNOLOGIE USATE



Figura 12: Porzione della dashboard di Grafana che mostra la quantità d'acqua presente nel terreno registrata dai vari sensori

Notare come per ogni serie temporale presente nella dashboard vengano riportati i valori massimi, medi e minimi (i quali vengono aggiornati ad intervalli di 5 secondi - la frequenza di aggiornamento è configurabile a partire dall'interfaccia grafica di Grafana).

## 4 Conclusioni e sviluppi futuri

In conclusione è stata realizzata una pipeline di big data per il monitoraggio in tempo reale di dati provenienti da sensori e da una stazione meteo installati in un nocciolo nel comune di Caprarola (Viterbo) nel contesto del progetto PANTHEON.

L'architettura realizzata consiste in una variante di quella proposta nel progetto PANTHEON per la realizzazione del DSP layer, in cui viene impiegato MongoDB per memorizzare i dati raccolti dai sensori (al posto di InfluxDB).

Per quanto riguarda gli sviluppi futuri, questi possono essere riassunti nei seguenti punti:

- Come detto in precedenza, nel documento relativo a PANTHEON viene proposto MongoDB come database da usare per memorizzare i dati provenienti dai sensori e i dati provenienti dal DCP layer. Un possibile sviluppo futuro potrebbe consistere nel valutare le prestazioni di MongoDB e InfluxDB nel contesto del progetto PANTHEON per stabilire quale possa essere la migliore soluzione. Occorre anche precisare come è possibile che i dati provenienti dal DCP layer possano non essere delle serie temporali, nel qual caso InfluxDB mal si presterebbe come strumento di memorizzazione dei dati nel DSP layer a differenza di MongoDB che offre maggiore flessibilità. In questo caso si potrebbe pensare di usare due database, uno per memorizzare i dati provenienti dai sensori, come InfluxDB, e uno per ricevere i dati processati dal DCP layer, come ad esempio MongoDB.
- Valutare l'uso di Apache Beam per il processamento dei dati nel processing layer del sistema realizzato. Apache Beam offre un modello di processamento dei dati (sia stream che batch) che può essere poi eseguito su diversi runtime, come ad esempio Spark, *Google Cloud Dataflow*, *Apache Flink*, *Apache Samza*, con la possibilità dunque di poter poi (eventualmente) migrare da Spark ad una differente tecnologia a seconda delle esigenze.

## 5 Riferimenti

- Documentazione Docker, <https://docs.docker.com>;
- Documentazione Docker compose, <https://docs.docker.com/compose/>;
- Documentazione kafka-python, <https://kafka-python.readthedocs.io/en/master/index.html>;
- Documentazione Kafka, <https://kafka.apache.org/documentation/>;
- Documentazione Spark Streaming,  
<https://spark.apache.org/docs/latest/streaming-programming-guide.html>;
- Documentazione InfluxDB, <https://docs.influxdata.com/influxdb/v1.7/>;
- Documentazione Grafana, <https://grafana.com/docs/>
- *Laura Giustarini, Sebastian Lamprecht, Rebecca Retzlaff, Thomas Udelho-ven, Nico Bono Rossello', Emanuele Garone, Valerio Cristofori, Mario Contarini, Marco Paolocci, Cristian Silvestri, Stefano Speranza, Ema-nuele Graziani, Romeo Stelliferi, Renzo Fabrizio Carpio, Jacopo Maiolini, Riccardo Torlone, Giovanni Ulivi, and Andrea Gasparri, PANTHEON: SCADA for Precision Agriculture.*
- *Nicolás Bono Rosselló, Andrea Gasparri and Emanuele Garone, Water Management Control*  
(<http://www.project-pantheon.eu/images/Deliverables/D51.pdf>)