Relazione Big Data Secondo Progetto

Progetto Pantheon

Mattia Scaccia, Ilaria Stocchi

01 Settembre 2021

Contents

Repository github del progetto

Per maggiori dettagli sull'implementazione si riporta il link alla repository:

`https://github.com/Progetto-bigData/second_project_pantheon_precision_farming`

Per le istruzioni su come lanciare l'intero progetto e verificarne il funzionamento consultare il README della repository.

Chapter 1

Introduzione al progetto Pantheon

Il progetto Pantheon coinvolge l'Università Roma Tre, ed in particolare il dipartimento di Ingegneria, in una collaborazione al livello europeo con altre tre università: l'Università libera di Bruxelles, l'Università della Tuscia, l'Università di Treviri, e due partner industriali: Ferrero Trading Lux S.A. e Sigma Consulting.
Il progetto ha come obiettivo quello di realizzare e perfezionare uno SCADA (Supervisory Control And Data Acquisition), cioè un sistema di controllo supervisionato e di acquisizione dati per l'agricoltura di precisione di frutteti.
Le esigenze di Ferrero hanno spinto i responsabili del progetto a concentrare la loro attenzione sui campi di nocciole (quelli realmente monitorati attraverso Pantheon si trovano in Italia, nella zona della Tuscia), ma l'obiettivo più grande del progetto è quello di realizzare un prototipo che possa poi operare in frutteti reali.

Il sistema (fig.1.1) è composto da una serie di robot completamente autonomi, operanti sia in aria che al suolo, che si muovono all'interno del campo eseguendo operazioni agricole e raccogliendo dati, i quali vengono poi memorizzati in una unità centrale che li integra per generare feedback sullo stato del terreno e regolare successive operazioni e decisioni degli agronomi.

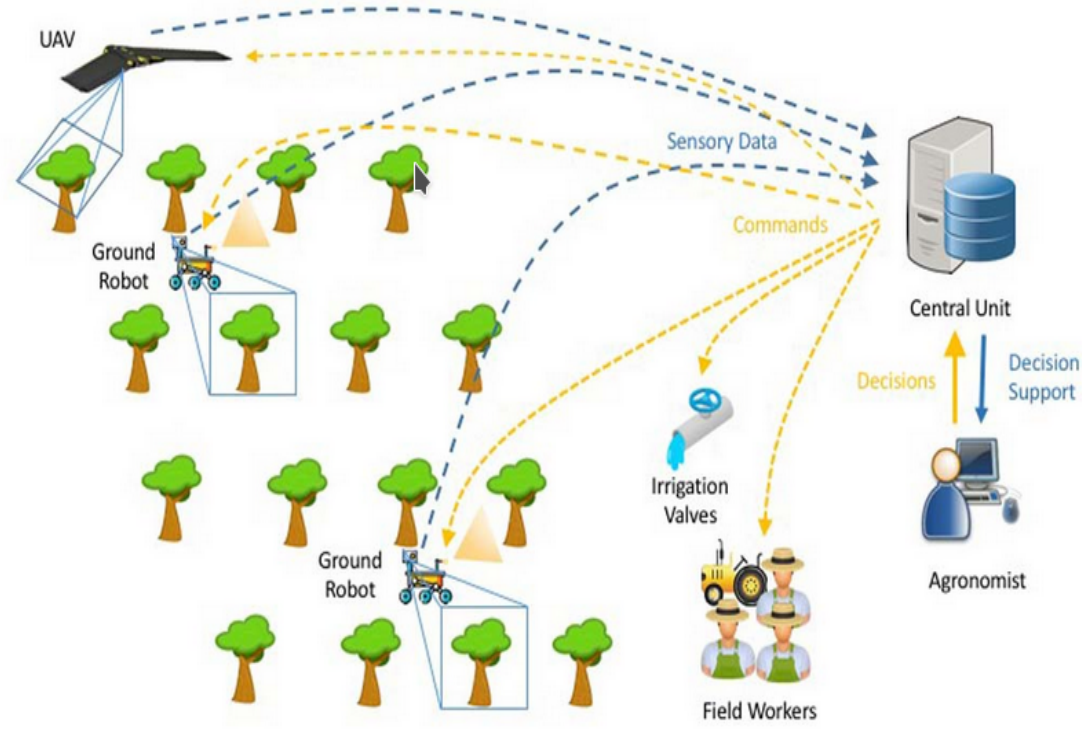

Figure 1.1: Pantheon SCADA System

I robot acquisiscono i dati a livello della singola pianta, ciò permette di individuare più dettagliatamente eventuali fattori che possono limitare la salute della pianta stessa, e di conseguenza avviare interventi più capillari per migliorare lo stato generale di tutto il terreno. L'obiettivo ultimo è aumentare la capacità produttiva da un lato, e dall'altro favorire l'ecosostenibilità, dato che l'acqua necessaria all'irrigazione, per esempio, viene impiegata nella quantità necessaria perché regolata dai sensori che continuamente collezionano dati, oppure l'uso di pesticidi per contrastare eventuali aggressioni.

Chapter 2

Sorgenti dei dati

Ci è stata fornita un'API per il download dei dati già precedentemente memorizzati nell'unità centrale.
Seguendo l'url `http://89.97.5.192:3000/api/v1/measurement/download/:format/:meas_type/:query/:start/:end`
abbiamo ottenuto i dati relativi alla seconda metà dell'anno 2020, nel periodo compreso tra i mesi di Giugno e Dicembre.
I parametri che sono stati modificati per estrarre i dati sono i seguenti:

- **format**: rappresenta il formato dei dati (CSV per formato .csv, oppure JSON per formato .json).
- **meas_type**: rappresenta se è stata fatta o meno una elaborazione preliminare dei dati prima del loro salvataggio (RAW per i dati semplici, ottenuti direttamente dai sensori, oppure ELABORATED per i dati salvati nel formato tabellare *timestamp*, *id_sensore*, *tipo_di_misurazione*, *unità_di_misura*, *valore_misurato*.
- **query**: rappresenta la tipologia di dato che si vuole estrarre (alcuni esempi: TEMPERATURE_AVERAGE, oppure WIND_SPEED_MIN etc..)
- **start**: rappresenta la data completa (yyyy-mm-dd) di inizio dell'intervallo di tempo in cui si vogliono estrarre i dati.
- **end**: rappresenta la data completa (yyyy-mm-dd) di fine dell'intervallo di tempo in cui si vogliono estrarre i dati.

Ai fini del progetto relativo al corso di Big Data ci siamo serviti dei dati elaborati, salvati nel formato csv.

I dati provengono da due fonti diverse: la stazione meteo e i sensori al suolo. La stazione meteo è dotata di diversi strumenti che registrano i principali fattori atmosferici, l'igrometro per l'umidità, il barometro per la pressione, il pluviometro per le piogge, il termometro per le temperature, l'anemometro per velocità e direzione del vento, il solarimetro per le radiazioni solari, mentre i sensori al suolo registrano acqua e temperatura del terreno.
I sensori a terra sono diciotto, perché il frutteto è stato diviso in nove aree, ciascuna con il proprio strumento di rilevazione che opera a due profondità diverse del terreno (15 cm e 40 cm).

Chapter 3

Architettura del sistema

L'architettura realizzata (fig. 3.1) ha come obiettivo quello di simulare lo streaming real time dei dati dai sensori all'unità centrale di memorizzazione.

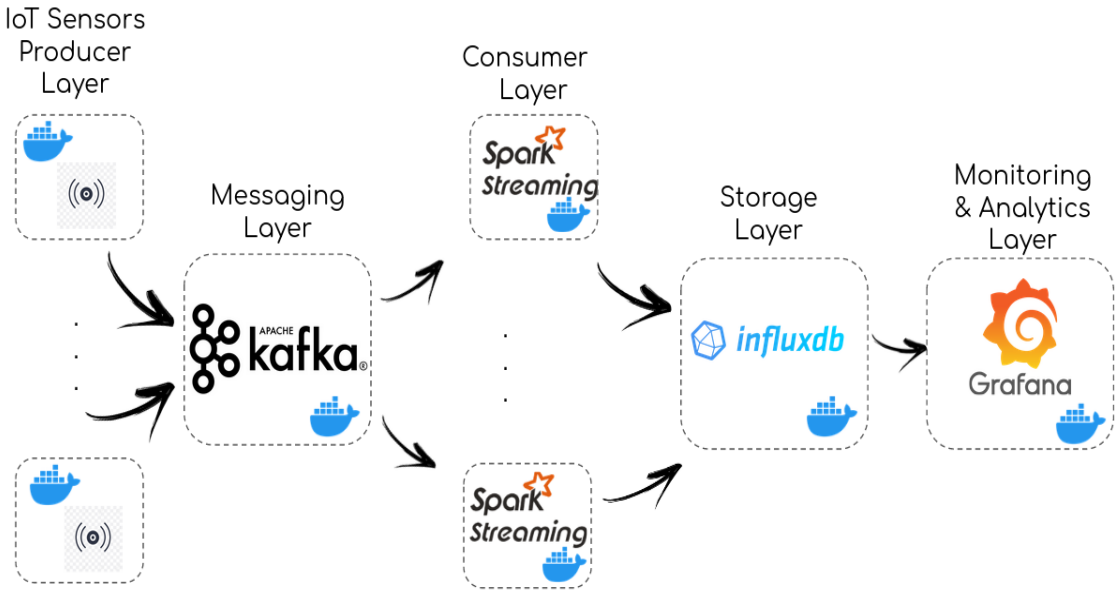

Figure 3.1: Overall system architecture

Per simulare l'invio dei dati è stato predisposto un contenitore Docker per ciascun sensore, questo legge i dati una riga alla volta, dal file csv, ad intervalli di 0.1 secondi. Le informazioni raccolte transitano, sotto forma di

messaggi, all'interno di un canale Kafka, anch'esso specifico per quel particolare sensore. Sono stati predisposti altri cinque contenitori che si comportano da consumatori. Questi sono iscritti a tutti i canali, prelevano i messaggi, controllano che i dati non siano corrotti, e alla fine inviano una richiesta di trascrizione ad influxDB, un database predisposto per dati temporali.
Dal momento che la latenza delle operazioni di scrittura su database è molto più alta di quella dell'invio dei messaggi sul canale, i consumatori sono stati opportunamente replicati e i canali Kafka, invece, replicati in modo che possano lavorare in parallelo.

3.1 Producers layer

Poiché avevamo a disposizione i dati batch, già collezionati, l'idea del progetto è stata quella di simulare la realtà operativa dei sensori, e quindi lo streaming dei dati e la loro successiva memorizzazione. Allo scopo, per ogni tipologia di sensore è stato creato un contenitore separato, e per simulare lo streaming abbiamo impostato la lettura di una riga, ogni 0.1 secondi, dal file csv contenente i dati relativi allo specifico sensore. L'invio da parte dei sensori reali era impostato ad un dato ogni 5 minuti ma la nostra scelta è stata quella di ridurlo a 0.1 secondi per velocizzare l'operazione di raccolta, visto che l'intervallo dei dati raccolti copriva i sei mesi da Giugno a Dicembre.
Abbiamo deciso di usare Kafka (tecnologia in grado di garantire affidabilità, scalabilità e disponibilità per la gestione dello streaming di dati) con Zookeeper e ciascun contenitore (sensore) è stato iscritto ad un topic diverso, nel quale far transitare i rispettivi dati.

3.2 Messaging layer con Kafka

Kafka è una piattaforma open source per il processamento di eventi in streaming che consente ai produttori (producers/publishers) di inviare messaggi (dati) attraverso un canale (topic), e ai consumatori (consumers/subscribers) di prelevare questi messaggi (e processarli successivamente), tramite iscrizione al canale, in maniera del tutto disaccoppiata gli uni dagli altri.
Ciò vuol dire che i producers, per pubblicare messaggi, non devono attendere la presenza/conferma di nessun consumatore e i consumatori, da parte loro, dopo essersi iscritti al canale, possono prelevare i messaggi presenti

all'interno di esso quando vogliono, e processarli in maniera personale, indipendentemente da quello che fanno altri consumatori iscritti allo stesso canale che prelevano gli stessi dati.

Ai fini del progetto abbiamo deciso di separare il flusso delle varie misurazioni sensoristiche facendolo transitare in topic diversi (ne è stato creato uno per tipologia di sensore). Kafka permette il partizionamento dei topic, ovvero la suddivisione in sequenze ordinate di messaggi, in modo che ciascun messaggio che transita nel topic venga appeso ad una sola delle partizioni e che un solo consumatore per ogni gruppo possa leggere una partizione, come mostrato in figura 3.2, così da evitare che un singolo messaggio venga letto più volte. Ai fini del progetto abbiamo deciso di suddividere i topic creati in 5 partizioni. Nella sezione che segue presentiamo il motivo della nostra scelta.

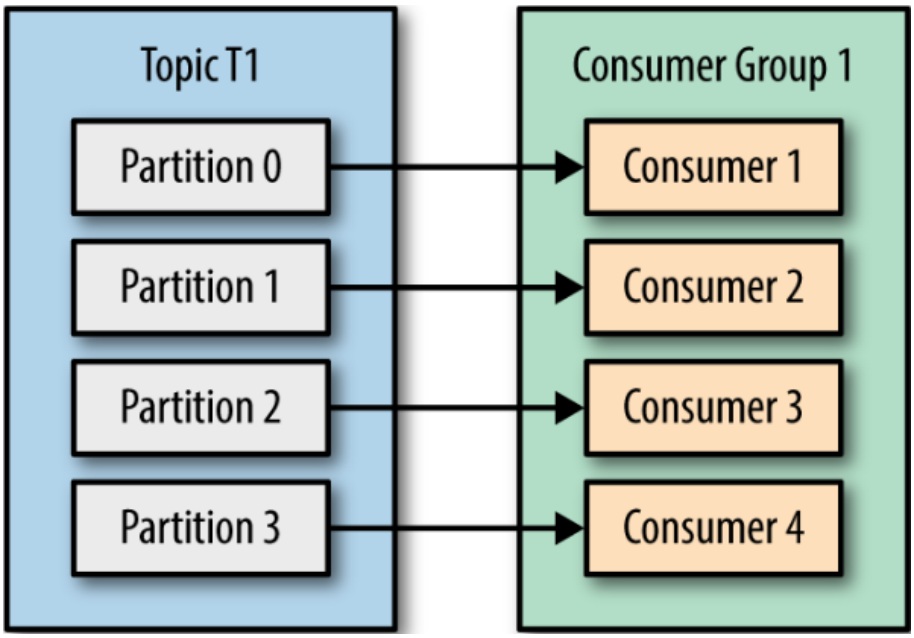

Figure 3.2: Numero di partizioni uguale al numero di consumatori in un gruppo

3.3 Consumers layer con Spark Streaming

I topic sono stati divisi in 5 partizioni in funzione del fatto che è stato creato un solo consumer group, per evitare che i messaggi venissero consumati più di una volta da consumatori diversi in diversi consumer groups; ricordiamo infatti che la "consumazione" dei dati per noi significa la notifica e l'avvio della loro scrittura su influxDB, quindi non vogliamo che lo stesso dato venga scritto più di una volta nel database.
Il motivo delle 5 partizioni dei topic risiede quindi nel fatto che, siccome l'operazione di scrittura su db è più lenta della consumazione dei messaggi, possiamo affidare le partizioni a consumatori diversi nell'ambito dell'unico consumer group creato e parallelizzare così le operazioni di scrittura. Si noti come ai fini del progetto ci siamo concentrati sul miglioramento della scalabilità dell'architettura (garantendo che non si perdano messaggi nei topic, che invece si accumulerebbero per troppo tempo senza essere consumati, nell'attesa di venire elaborati dai consumatori e quindi nell'attesa del completamento di scritture relative a messaggi precedenti).
E' stato scelto il numero 5 perchè era il massimo numero di contenitori che la macchina in locale era in grado di gestire oltre ai contenitori per i sensori e tutti gli altri componenti. Si è provato a spostare l'intero stack di contenitori su AWS per risolvere questo problema, consultare il capitolo 4 per ulteriori informazioni.

3.4 Storage layer con InfluxDB

Per lo storage layer la scelta è ricaduta su InfluxDB, un Time Series Database open source altamente performante, basato su un linguaggio di interrogazione simile a SQL chiamato InfluxQL. È stato progettato per sostenere elevate capacità di scrittura e carichi di interrogazioni in maniera scalabile e distribuita ed è parte del TICK Stack, un set di progetti open source per la gestione di enormi volumi di dati time-stamped.
InfluxDB è particolarmente indicato per salvare dati derivanti da IoT, sensoristica e log di applicazioni, ed è stato scelto quindi per questo e anche per la compatibilità con l'utilizzo di grafana (per la realizzazione di un monitoring layer descritto nella sezione 3.5), e infine per la solidità e affidabilità della tecnologia come dimostrano i ranking dei time series databases, si veda figura 3.3, e in generale la popolarità che i time series DB hanno acquisito

negli ultimi anni, si veda figura 3.4.

☐ include secondary database models — 36 systems in ranking, August 2021

Rank Aug 2021	Rank Jul 2021	Rank Aug 2020	DBMS	Database Model	Score Aug 2021	Score Jul 2021	Score Aug 2020
1.	1.	1.	InfluxDB	Time Series, Multi-model	29.56	+0.32	+6.69
2.	2.	2.	Kdb+	Time Series, Multi-model	7.99	-0.11	+1.19
3.	3.	3.	Prometheus	Time Series	6.20	+0.18	+0.84
4.	4.	4.	Graphite	Time Series	4.86	+0.01	+0.75
5.	5.	↑ 6.	TimescaleDB	Time Series, Multi-model	3.41	+0.27	+0.87
6.	6.	↑ 7.	Apache Druid	Multi-model	3.00	0.00	+0.83
7.	7.	↓ 5.	RRDtool	Time Series	2.43	-0.14	-0.59
8.	8.	8.	OpenTSDB	Time Series	1.90	+0.00	-0.23
9.	9.	9.	Fauna	Multi-model	1.53	-0.21	-0.17
10.	10.	↑ 11.	GridDB	Time Series, Multi-model	1.37	+0.10	+0.71
11.	11.	↑ 12.	DolphinDB	Time Series	1.07	+0.06	+0.47
12.	12.	↓ 10.	KairosDB	Time Series	0.82	-0.03	+0.14
13.	13.	13.	eXtremeDB	Multi-model	0.73	+0.01	+0.28
14.	14.	↑ 15.	Amazon Timestream	Time Series	0.71	+0.03	+0.34
15.	15.	↑ 20.	QuestDB	Time Series, Multi-model	0.64	+0.03	+0.39

Figure 3.3: Time series DB ranking secondo `https://db-engines.com/en/ranking/time+series+dbms`

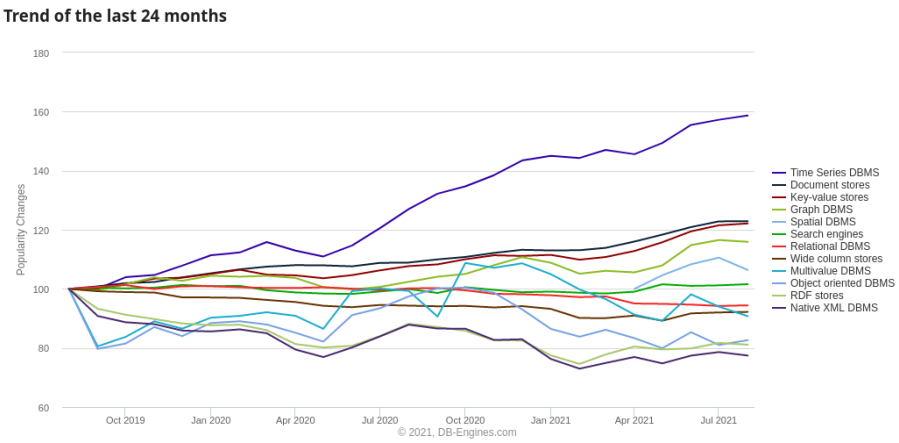

Figure 3.4: Popolarità tipi di DB secondo `https://db-engines.com/en/ranking_categories`

In influxDB ogni dato è salvato sotto forma di una serie di coppie chiave valore:

- *Measurement*: il cui valore è una stringa che rappresenta l'identificatore dell'insieme di dati, simile al concetto di tabella in un RDBMS.
- *Time*: il cui valore è un timestamp che, insieme all'insieme di *tags*, rappresenta l'identificatore univoco di un dato di un *measurement*, simile al concetto di chiave in un RDBMS. I dati vengono indicizzati secondo questo valore.
- *Tags*: un insieme di coppie chiave valore opzionale che, insieme al campo *time*, rappresenta l'identificatore univoco di un dato in un *measurement*, simile al concetto di chiave in un RDBMS. I dati vengono indicizzati secondo queste coppie.
- *Fields*: un insieme di coppie chiave valore opzionali e di quantità arbitraria che rappresentano i dati effettivi associati al *time* specificato per quel dato *measurement*, simile al concetto di colonne in un RDBMS.

Un dato relativo ad un sensore è stato quindi salvato seguendo questa struttura:

- *Measurement*: *id_sensore*, ad esempio "anemometro";
- *Time*: *timestamp*, ad esempio "2020-06-05T13:35:28.000Z";
- *Tags*: *tipo_di_misurazione*, ad esempio "direzione del vento";
- *Field*: *unità_di_misura* : valore, ad esempio *unità_di_misura* : "gradi"
- *Field*: *valore_misurato* : valore, ad esempio *valore_misurato* : "310".

3.5 Monitoring and data analytics layer con Grafana

Grafana è un'applicazione web per la visualizzazione e l'analisi interattiva di dati, anch'esso software open source, multi-platform. E' il layer messo a disposizione per gli agronomi, i quali possono interfacciarsi con esso per monitorare in tempo reale lo stato dei terreni ed intervenire tempestivamente se si dovessero riscontrare eventuali anomalie. Grafana infatti permette di visualizzare infografiche, unificando le varie sorgenti dei dati. I dati possono essere visualizzati in dashboards costruite attraverso query eseguite direttamente su influxDB. Grafana offre anche la possibilità di inviare degli alert nel caso in cui i parametri fissati escano fuori dai limiti (nel nostro caso, per esempio, l'eccessiva o insufficiente acqua del suolo) in modo da poter intervenire in maniera automatica.
Sono stati creati due insiemi di dashboards, il primo relativo ai dati raccolti dalla stazione meteo e il secondo relativo ai dati raccolti dai sensori sui terreni. Oltre alla semplice visualizzazione dei dati raccolti sono state anche prodotte e visualizzate nuove caratteristiche e informazioni utili a partire dai dati a disposizione come, ad esempio, l'escursione termica giornaliera.
Di seguito ne riportiamo alcuni esempi.

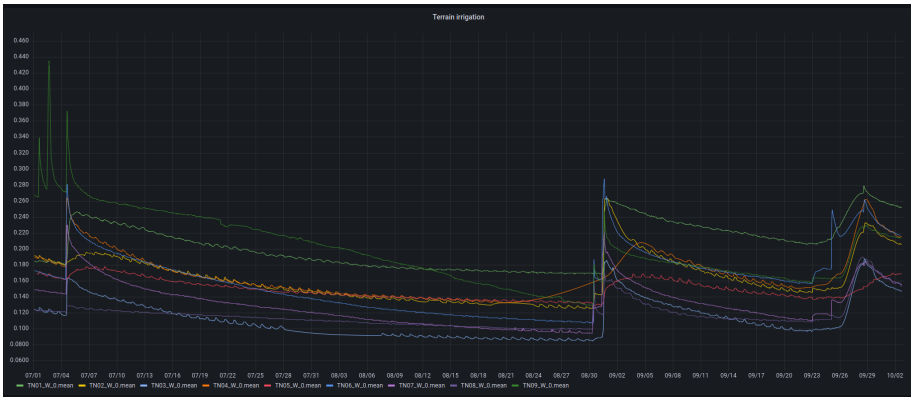

Figure 3.5: Irrigation dashboard

Figure 3.6: A portion of the Weather Station dashboard

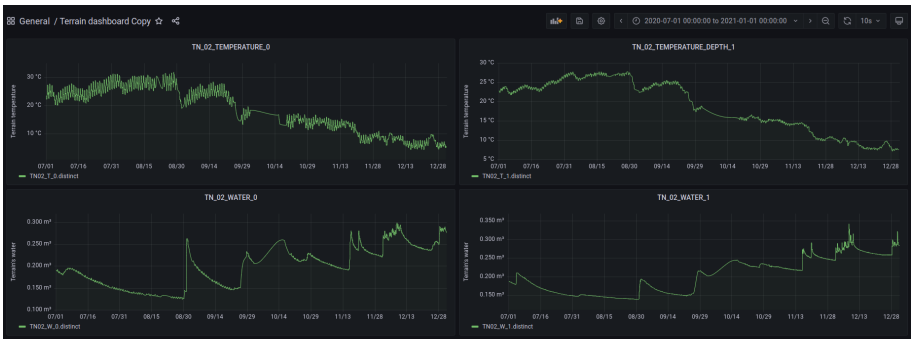

Figure 3.7: A portion of the Terrain dashboard

Chapter 4

Conclusioni e sviluppi futuri

Come ultimo passo del progetto abbiamo provato a portare tutto lo stack dei contenitori docker su AWS in modo da eliminare le limitazioni imposte dall'esecuzione in locale e poter quindi progettare meglio per la scalabilità e la disponibilità, utilizzando le nuove funzionalità offerte da Docker Compose CLI per l'integrazione con Amazon ECS (Elastic Container Service). Purtroppo non siamo riusciti a completare la migrazione su AWS per problemi di permessi che non siamo stati in grado di gestire a causa dell'utenza limitata su AWS Educate che abbiamo disponibile per il corso, ne riportiamo un esempio nella figura 4.1.

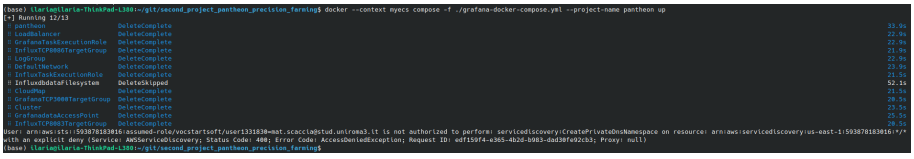

Figure 4.1: AccesDeniedException durante la migrazione su AWS

Il sistema è comunque in teoria già correttamente predisposto per una migrazione su AWS.
Possibili sviluppi futuri potrebbero riguardare un'analisi delle prestazioni di influxDB rispetto all'utilizzo di un document store database, come ad esempio MongoDB che è utilizzato come soluzione di storage in molti dei progetti del team Pantheon.
Altri sviluppi futuri potrebbero riguardare l'introduzione del machine learning per effettuare analisi di tipo predittivo a supporto degli agronomi, con la

precondizione però di acquisire dati etichettati sullo stato dei raccolti, con il supporto di esperti del dominio, in modo che si possano effettuare delle analisi supervisionate.

Bibliography

[1] https://pantheon.inf.uniroma3.it/

[2] https://docs.influxdata.com/influxdb/v2.0/

[3] https://grafana.com/docs/

[4] https://kafka.apache.org/documentation/

[5] https://docs.docker.com/cloud/ecs-integration/

[6] https://spark.apache.org/docs/latest/streaming-programming-guide.html