Consiglio Nazionale delle Ricerche
Istituto di Matematica Applicata
e Tecnologie Informatiche
“Enrico Magenes”

REPORT SERIES

A. Quarati, A. Clematis, V. Levati

Soluzioni Open Source per la scalabilità di servizi di raccolta ed utilizzo di dati ambientali da sensore

17-11

IMATI REPORT Series

Nr. 17-11
May 2017

Managing Editor
Paola Pietra

Editorial Office
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via Ferrata, 5/a
27100 PAVIA (Italy)
Email: reports@imati.cnr.it
http://www.imati.cnr.it

IMATI Report Series Nr. 17-11
May 12, 2017

Soluzioni Open Source per la scalabilità di servizi di raccolta ed utilizzo di dati ambientali da sensore

Alfonso Quarati[1], Andrea Clematis[1], Vittorio Levati[2]

Corresponding author:
[1] Alfonso Quarati
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*",
Via de Marini, 6 (Torre di Francia)
16149 Genova - Italy
E-mail address: quarati@ge.imati.cnr.it

[1] Andrea Clematis
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*",
Via de Marini, 6 (Torre di Francia)
16149 Genova – Italy

[2] Vittorio Levati
SVL s.a.s.
C.so Tardy e Benech 12/1
17100 SAVONA

Abstract

Il documento presenta uno studio sperimentale atto alla valutazione di aspetti di scalabilità di servizi Web sviluppati nell'ambito del progetto TCUBE, mirato alla gestione di grandi quantità di dati eterogenei, dinamici e distribuiti, mediante una piattaforma integrata di servizi per la mobilità nei settori "Trasporti, Territorio e Turismo". Lo scopo dello studio è stato quello di derivare informazioni qualitative utili alla valutazione del potenziale impatto di alcune delle soluzioni/metodologie adottate nel progetto ed in particolare a quelle relative ad un servizio per la raccolta, trasformazione e sfruttamento di dati provenienti da sensori. Si è effettuata un'indagine sia tecnica che applicativa, finalizzata a supportare lo sviluppo di ulteriori attività progettuali che possano avere come obiettivo la realizzazione di servizi basati su dati metereologici provenienti da reti di sensori e prodotti da o dedicati ad utenti finali.
Il presente rapporto propone una sintesi dei risultati di tale attività di analisi, e rappresenta una guida operativa all'applicazione delle diverse tecnologie Open Source adottate, al fine di indirizzarne l'applicazione in contesti territoriali affini e di più ampia scala, quali aree urbane o territori ad alto impatto turistico.

Keywords: *Scalable solutions, Web services, Sensor data, Open Source.*

[page left intentionally blank]

SOLUZIONI OPEN SOURCE PER LA SCALABILITÀ DI SERVIZI DI RACCOLTA ED UTILIZZO DI DATI AMBIENTALI DA SENSORE

Alfonso Quarati[1], Andrea Clematis[1], Vittorio Levati[2]

(1) IMATI-CNR , Genova, Italy

(2) SVL s.a.s, Savona, italy

12/05/2017

Abstract

Il documento presenta uno studio sperimentale atto alla valutazione di aspetti di scalabilità di servizi Web sviluppati nell'ambito del progetto TCUBE, mirato alla gestione di grandi quantità di dati eterogenei, dinamici e distribuiti, mediante una piattaforma integrata di servizi per la mobilità nei settori "Trasporti, Territorio e Turismo".

Lo scopo dello studio è stato quello di derivare informazioni qualitative utili alla valutazione del potenziale impatto di alcune delle soluzioni/metodologie adottate nel progetto ed in particolare a quelle relative ad un servizio per la raccolta, trasformazione e sfruttamento di dati provenienti da sensori.

Si è effettuata un'indagine sia tecnica che applicativa, finalizzata a supportare lo sviluppo di ulteriori attività progettuali che possano avere come obiettivo la realizzazione di servizi basati su dati metereologici provenienti da reti di sensori e prodotti da o dedicati ad utenti finali.

Il presente rapporto propone una sintesi dei risultati di tale attività di analisi, e rappresenta una guida operativa all'applicazione delle diverse tecnologie Open Source adottate, al fine di indirizzarne l'applicazione in contesti territoriali affini e di più ampia scala, quali aree urbane o territori ad alto impatto turistico.

1 Introduzione

L'attività sperimentale presentata nel seguito del rapporto, si è esplicata nell'ambito del progetto regionale di ricerca "TCUBE: Trasporti Territorio e Turismo"[1], relativo alla "Realizzazione di una piattaforma di servizi per la mobilità basata su tecnologie di tipo Open Data", nell'ambito del "Programma triennale per la ricerca e l'innovazione: progetti integrati ad alta tecnologia" della Regione Liguria. Il progetto ha inteso affrontare quest'ambito applicativo e tecnologico di primaria importanza per lo sviluppo di soluzioni e prodotti a supporto 'della 'Smart Mobility', con particolare riferimento all'area metropolitana di Genova e agli scenari Smart City assunti dall'Amministrazione quale contesto strategico di sviluppo ed evoluzione dei servizi per i cittadini.

Obiettivo del progetto è stato lo studio, la realizzazione e la validazione di tecnologie e soluzioni per l'implementazione, il deployment e la gestione di servizi informativi avanzati per la mobilità basate su tecnologie e metodologie abilitanti tra le quali: Open Data; crowdsourcing e social sharing; infrastrutture e servizi basati su spatial data. Dal punto di vista della ricerca sperimentale la realizzazione, a livello prototipale, di tale piattaforma di servizi è stata applicata al contesto territoriale del Comune di Genova, con particolare riferimento alla mobilità urbana connessa ai flussi turistici, quale peculiare componente della mobilità cittadina.

A tal fine il progetto ha visto, in collaborazione con gli utenti locali e istituzionali, la realizzazione di un caso di studio specifico in una delle aree più significative della città di Genova: l'area urbana e il contesto di trasporto connesso al waterfront della città e all'area del Porto antico e al bacino di mobilità della rete urbana afferente all'area. Quest'area rappresenta un caso d'uso molto significativo in quanto rappresenta uno tra i luoghi più visitati di Genova. Ogni anno l'area di 130.000 mq, waterfront della città, accoglie ogni anno circa 4.000.000 di visitatori. Il bisogno di migliorare la gestione dei flussi turistici e implementare una piattaforma di servizi a essi dedicati risulta quindi strategico per la città.

Il caso di studio affrontato è particolarmente interessante poiché caratterizzato da elementi che rendono ripetibile l'esperienza sviluppata nell'area della sperimentazione, permettendo l'applicazione delle tecnologie adottate e della struttura di progetto in altre aree simili, quali centri storici e zone espositive caratterizzate da forte flusso turistico, assenza o limitazione del traffico veicolare e presenza di elementi di attrazione geo-localizzati con capacità ricettiva limitata.

IMATI è stato Responsabile scientifico del progetto, contribuendo in particolare all'analisi e selezione delle tecnologie abilitanti, con particolare riguardo agli aspetti relativi all'interoperabilità basata su standards ed ai processi di validazione della piattaforma a servizi realizzata. Inoltre data la sua pregressa esperienza nello sviluppo di servizi di mashup di dati provenienti da reti di sensori [1][2], IMATI ha progettato e sviluppato uno specifico servizio per l'integrazione di dati provenienti da due reti di sensori [3]. Il servizio *Personal Weather Stations*, integrato nella architettura di TCUBE, fornisce la situazione meteo per l'area limitrofa al Porto Antico di Genova, sito dimostratore del progetto.

[1] http://www.research.softeco.it/tcube.aspx

Obiettivo dell'attività di valutazione di seguito riportata, è stato quello di analizzare la scalabilità di alcune delle soluzioni/metodologie prodotte dalla piattaforma TCUBE, focalizzandosi nell'esaminare proposte, e fornire risposte qualitative e metodologiche alle domande tecnologiche proprie dei temi di ricerca di cui sopra. Lo studio ha quindi preso in esame una tipologia di servizi rilasciati dalla piattaforma TCUBE: specificatamente quella relativa alla raccolta, trasformazione e sfruttamento di dati provenienti da sensori. In particolare, si è effettuata un'indagine sia tecnica che applicativa, propedeutica a possibili successivi progetti che abbiano come obiettivo la realizzazione e l'utilizzo operativo di servizi basati su dati metereologici o provenienti da reti di sensori dedicati agli utenti finali.

Le reti di sensori meteo, sia per il loro impiego nell'applicazione TCUBE, sia per la loro diffusione si prestano a tale tipo di analisi, che può essere naturalmente estesa ad altri tipi di sensori e/o di dati provenienti da fonti di crowdsourcing o traffico urbano.

In questo studio si è inoltre voluto dare riposta alla seguente domanda:

> *Q1: "Considerando come scenario applicativo, al momento dati metereologici in futuro, dati da sensori in contesti IoT o Smart Cities - quali strumenti fra quelli Open Source possono costituire una filiera, di limitata complessità ma, utile alla raccolta ed elaborazione di grandi quantità di dati (Big Data) sulla quale costruire applicazioni?"*

Il lavoro si è svolto attraverso le seguenti fasi:

- Analisi degli obiettivi, dei dati disponibili e degli strumenti utilizzabili nell'ambito dell'elaborazione dei Big Data;
- Valutazione dei vincoli presenti e delle risorse, ragionevolmente disponibili, per la formulazione di proposte e soluzioni;
- Disegno di uno scenario di applicabilità operativa, entro i cui termini valutare i risultati potenziali ottenibili e le possibili performance in termini di scalabilità e sostenibilità nel tempo, alla luce delle possibili evoluzioni tecniche e di contesto applicativo.

Nel corso dello studio, si è sperimentata una filiera di raccolta e lavorazione per grandi quantitativi di dati provenienti da sensore con strumenti Big Data di tipo Open Source e per la fruizione da parte di utenti a diversi livelli di preparazione: utenti finali o sviluppatori SQL. È stato inoltre sperimentato l'utilizzo di quanto selezionato in ambiente "on premise" e in Cloud. Oltre le problematiche di raccolta ed elaborazione dei dati è stato testato anche un ambiente di machine learning idoneo ad un'analisi sulla gestione e lo sfruttamento di Big Data.

Nel presentare le sperimentazioni effettuate, in particolare per quanto riguarda gli aspetti più squisitamente tecnico-pratico affrontati nelle sezioni 4 e 5, si è adottato un approccio didattico/operativo in cui gli strumenti, ed i metodi applicativi per l'utilizzo degli stessi, sono stati introdotti facendo largo uso di schermate di esempio, e di brevi descrizioni delle stesse. Tale soluzione è stata pensata, oltre che a fini meramente esemplificativi, anche per fornire una sorta di handbook di sostegno all'implementazione di soluzioni tecnologiche con analoghi requisiti. A questo scopo le Appendici racchiudono tre listati di esempio di alcuni dei codici implementati nel corso dell'attività.

2 Il contesto Big Data

Il termine Big Data ha ben presto travalicato i confini dell'ambito informatico per essere citato in articoli, pubblicazioni e discussioni, assai spesso per gli effetti sulla vita quotidiana di ciascuno. In verità sono molti gli aspetti che ci consentono di mettere a fuoco i contorni del concetto Big Data.

Le 3V (volume, varietà e velocità) riferite al paradigma Big Data, originariamente proposte da Doug Laney nel suo lavoro seminale del 2001 [4], si sono evolute costantemente negli ultimi anni per includere più V, dalle sette (includendo cioè gli aspetti di variabilità, veridicità, visualizzazione e valore) esaminate da Eileen McNulty[2], fino ad a arrivare alle dieci V suggerite da Kirk Borne[3] che ha aggiunto alle precedenti altre tre V (venue (sede), vocabolario e vaghezza). Questa caratterizzazione V-based, del concetto di Big Data è utile per evidenziarne le sue principali sfide: l'acquisizione, la pulizia, la curatela, la convalida, l'integrazione, la conservazione, l'elaborazione, la ricerca, e l'analisi di grandi volumi di dati in continua ed inarrestabile crescita.

[2] http://dataconomy.com/seven-vs-big-data/
[3] https://www.mapr.com/blog/top-10-big-data-challenges-serious-look-10-big-data-vs

I molteplici fattori che entrano in gioco in tale contesto, rendono non più adatte le tecnologie convenzionali comunemente diffuse nelle aziende e nei laboratori, basate sui personal computer, su server di medie dimensioni e programmi sviluppati nella logica strutturata dei database SQL.

Queste tecnologie presentano presto limiti di scalabilità poiché al crescere dei volumi crescono esponenzialmente anche la complessità dei sistemi e i costi di gestione.

Tali problematiche sono emerse evidentissime già da diversi anni presso le società che hanno fortemente basato il loro sviluppo su Internet, come Google, Yahoo, Amazon, che hanno sviluppato il loro core business in rete su scala globale raccogliendo e generando dati caratterizzati dalle citate caratteristiche di volume, velocità e varietà.

Il successo commerciale conseguito da queste multinazionali Web, ha permesso loro di finanziare attività di ricerca e sviluppo al fine di dotarsi di tecnologie adeguate a risolvere i nuovi problemi che si manifestavano nella gestione di moli crescenti di Big Data.

Contemporaneamente si è verificata la progressiva discesa dei costi e l'incremento delle capacità di elaborazione e trasporto in rete dei dati e dall'evoluzione delle CPU, delle memorie, delle capacità di archiviazione, degli apparati di networking e in genere di tutte le componenti del mondo ICT.

Dall'incontro fra il nuovo potenziale tecnologico hardware, le sfide poste dai Big Data e i capitali provenienti dai nuovi business che avevano come mercato la rete sono emerse:

- nuove architetture basate sull'integrazione parallela di molti server standard, prodotti in larga scala, in un approccio antitetico a quello dei super-computer;
- il consolidamento in grandi centri di elaborazione dati di tantissime risorse hardware pronte ad offrire diffusamente servizi Cloud, senza richiedere forti investimenti iniziali per gli utenti finali;
- l'affermazione del modello di sviluppo software Open Source, capace di aggregare una base di contributi creativi assai più ampia del modello proprietario, coinvolgendo simbioticamente aziende, università e singoli ricercatori. Questo modello è stato capace di processi interni di autoselezione delle migliori idee dando origine in pochi anni ad un numero considerevole di prodotti innovativi.

2.1 L'evoluzione Open Source

Il modello Open Source [5] rivoluziona il concetto di licenza software vicino all'idea di diritto d'autore, di opera d'ingegno e di brevetto, schemi attraverso i quali viene remunerato in maniera diretta il lavoro di chi produce il software.

Esso invece si basa sul lavoro di chi già riceve una remunerazione, da un'azienda, da un ente pubblico o che volontariamente offre il suo contributo. Tutti i contributori si sentono comunque beneficiati o gratificati da questo lavoro in forme indirette, indipendentemente dall'imporre un prezzo per l'uso dei risultati ottenuti, peraltro difficilmente quantificabile in una tale contesto di lavoro condiviso.

Questo modello di sviluppo Open Source è comunque relazionato al mercato commerciale perché molte società, quasi sempre esse stesse contributrici allo sviluppo dei programmi, sono nate o sono entrate nel mercato per offrire servizi a pagamento a corredo e integrazione dei prodotti Open Source. Cloudera, Hortonworks, MapR sono solo le più famose fra le nuove nate, ma gli stessi tradizionali big dell'informatica hanno in portafoglio servizi dedicati ai prodotti Big Data Open Source o li offrono come servizi sulle loro piattaforme cloud. Servizi quali training, consulenza, fornitura di strumenti di supporto, sono interessanti per aziende ed organizzazioni che intendono beneficiare rapidamente di queste tecnologie limitando l'impegno diretto e lo sforzo di apprendimento.

Osservando il modello di sviluppo Open Source si riscontra che le aziende e gli enti che ne sono principali contributori operano sul mercato in una relazione di "coopetition" [6] cioè di collaborazione nelle fasi di sviluppo, i cui costi vengono di fatto condivisi riducendo i rischi d'impresa, e di competizione nella vendita dei servizi di supporto.

Al contrario il modello tradizionale di sviluppo proprietario, imponeva forti investimenti iniziali, un forte rischio d'impresa associato all'incertezza di realizzare un prodotto di successo, un lungo tempo per conoscere il gradimento del mercato in base al quale sviluppare nuove evoluzioni, ma soprattutto lunghi tempi di ritorno degli investimenti.

È pur vero che sul mercato oggi coesistono entrambi i modelli, ma mentre quello tradizionale è maggiormente presente sui prodotti maturi, che hanno già una loro storia di successo, quello Open Source prevale fra le tecnologie più complesse, innovative e recenti.

2.2 Aspetti applicativi

Tra i principali ambiti applicativi da cui scaturiscono problematiche ed istanze tecnologiche relative ai Big Data, ricordiamo:

- I comportamenti degli utenti Internet, sui siti di e-commerce, sui social network, la loro profilazione, le preferenze dei clienti e degli iscritti ai piani di fidelizzazione, i sistemi CRM (Customer Relationship Management) delle grandi aziende di telecomunicazioni e degli utilities che hanno vastissime basi di clienti e clienti potenziali, la sentiment analysis di quanto viene scritto sui social, il monitoraggio e la protezione dei marchi.
- Il trattamento dei dati di processo dei sistemi industriali, prodotti in grande quantità con frequenze temporali molto rapide, non più utilizzati solo per il controllo real-time, ma anche per un'analisi più capillare di quanto fatto in precedenza, in tema di qualità e manutenzione predittiva.
- Nel campo dell'IoT (Internet of Things) la proliferazione di dati dai sensori che ci circondano in casa, negli ambienti di lavoro e nelle smart-cities, generano volumi di dati che rientrano appieno nell'ambito Big Data. Anche la raccolta dei dati meteo di cui parleremo in seguito, assimilabili in senso più vasto alla categoria dei dati ambientali, rientra in quest'ultima categoria.

Il panorama tecnologico appena sintetizzato ha portato, tra l'altro, all'affermarsi del paradigma di calcolo Map-Reduce [7] attraverso il quale è possibile ripensare molti algoritmi di elaborazione di grandi quantità di dati, visti come una somma di più semplici operazioni di Map e di Reduce eseguibili grazie al programma di gestione Hadoop e al file system distribuito HDFS ripartendo i carichi di lavoro su cluster di server industriali.

Le idee dalle quali è nato Hadoop si sono poi evolute nel framework YARN o nell'alternativo SPARK, con la possibilità di eseguire altri tipi di calcolo parallelo (Tez). Intorno ad Hadoop si è sviluppato un ecosistema di strati applicativi specializzati, che include, tra l'altro, molti database non SQL (Hbase, Cassandra ecc.), nuovi linguaggi (Pig), motori SQL (Hive, Impala, Drill) per interpretare query su dati non nativamente strutturati per SQL, motori di ricerca (Nutch), strumenti di gestione (HCatalog, Hue, Ambari, Oozie, Avro, Zookeeper), librerie di machine learning (Mahout).

Discuteremo nelle Sezioni 4 e 5 l'utilizzo di Hadoop, e di alcune di queste soluzioni tecnologiche in un contesto di laboratorio sperimentale scalabile.

3 Il contesto meteorologico

Negli ultimi anni si è posta sempre più attenzione all'analisi dei dati territoriali legati alle attività umane e ai loro riflessi economici. Queste attività, siano esse legate alle attività produttive ed agricole, che alla logistica degli spostamenti e dei trasporti o alle attività turistiche che in Liguria, come del resto in quasi tutte le regioni d'Italia, rappresentano una realtà di rilievo, sono in qualche modo influenzate dalle condizioni metereologiche.

3.1 Reti di sensori meteorologici

La gestione dei dati metereologici può essere affrontata, da due punti di vista. Uno è quello dello studio scientifico di questi fenomeni e delle loro previsioni, l'altro è legato alla diffusione delle tecnologie di osservazione in ampi settori non strettamente legati allo studio scientifico e professionale della meteorologia. È su questa seconda realtà che poniamo il nostro interesse.

La diffusione di punti di osservazione amatoriali o semiprofessionali sul territorio, e lo sviluppo di comunità legate da un interesse comune come quello meteorologico, ha favorito il diffondersi di servizi che consentono di cogliere informazioni al di là di quelle provenienti dalle istituzioni o dalla comunità scientifica. Questi servizi sono forniti da organizzazioni pubbliche e/o private, anche dette Reti di sensori, che collezionano i dati provenienti dalle singole stazioni, e li restituiscono a terzi secondo diverse modalità di accesso e offerta di servizi.

A titolo di esempio il sito Weather Underground (http://www.wunderground.com/) è il risultato più eclatante di questo approccio di cui riportiamo un'autodefinizione: *"An Alternative Outlook: Weather Underground has challenged the conventions around how weather information is shared with the public since 1993. We're immensely proud of the unique products that our community and meteorologists have created to improve people's access to meaningful weather data from around the globe. As the Internet's 1st weather service, we consider ourselves pioneers within our field and we're constantly seeking new data sets and the next technologies that will help us share more data with more people."*

Nel seguito useremo questo portale come fonte delle informazioni meteorologiche, che ci serviranno da esempio nel presentare le diverse soluzioni tecnologiche proposte.

I dati storici raccolti da Weather Underground sono provenienti sia da osservatori ufficiali che da stazioni amatoriali. Inoltre sono fruibili senza accordi particolari, e gratuitamente entro determinati limiti giornalieri. Tutte le stazioni della rete, private e non, concentrano sul portale Weather Underground i dati di ciascuna osservazione costituita dai principali parametri metereologici circa ogni 10-15 minuti. Sulla base di questi dati Weather Underground aggiorna il suo archivio, che poi è pubblicamente disponibile, ed elabora assieme ad altri tipi di osservazioni (es: da satelliti) le proprie previsioni metereologiche.

Le stazioni meteo private che sono la fonte più numerosa dei dati raccolti. Sono importanti perché rappresentano una novità rispetto alla maggior parte dei sistemi che si occupano di dati metereologici, integrando le informazioni fornite da stazioni istituzionali e supportando attività di prevenzione ambientale [8]. Inoltre possiedono alcune delle caratteristiche di velocità, varietà e volume tipiche dei Big Data nel cui contesto desideriamo muoverci nel seguito.

Il numero di queste stazioni è infatti molto più grande degli osservatori ufficiali, quindi fornisce un grande volume di dati, non li presenta in maniera omogenea né talvolta garantisce la disponibilità del dato stesso da cui la caratteristica di varietà. Il vantaggio che quindi portano le stazioni private rispetto a una rete di sole stazioni ufficiali è una maggior capillarità territoriale delle osservazioni seppur a rischio potenziale di una minore affidabilità delle misurazioni.

3.2 Il servizio Personal Weather Stations

A titolo di esempio di applicazioni che si giovano delle informazioni rilasciate da reti di sensori meteorologici, presentiamo in questa sezione l'attività svolta nel progetto TCUBE, in cui abbiamo progettato e sviluppato il servizio Personal Weather Stations.

Il servizio colleziona dati da due reti di sensori, WeatherUnderground e MeteoNetwork, fornendo attraverso oltre 50 stazioni meteo personali presenti sul territorio genovese una buona copertura dell'area circostante il sito del dimostratore. WeatherUnderground permette solo un numero limitato di query giornaliere gratuite, richiedendo costi diversi per numeri maggior di query[4]. Tuttavia, poiché il servizio Personal Weather Stations è ancora in versione prototipale, abbiamo deciso di sfruttare, per la fase di sviluppo, il libero accesso, che concede 500 chiamate (via API) al giorno, considerato sufficiente per questa fase. La cooperazione esistente tra IMATI e la rete MeteoNetwork[5] (stabilito in nel recente progetto europeo DRIHM[6]), e la copertura della Liguria garantita da questa rete ci ha incoraggiato a utilizzare le sue API. Inoltre, entrambi le reti restituiscono i loro dati in formato JSON, facilitando la codifica dell'applicazione. Inoltre JSON è frequentemente utilizzato come formato per l'interscambio ed il riuso di dati Open Data [9]. Quest'ultima caratteristica è stata particolarmente apprezzata, considerando che, come detto nell'Introduzione, Open Data è stata una delle tecnologie abilitanti alla base di TCUBE.

Una volta recuperati i dati, originariamente in formati (i.e. strutture dati) eterogenei, sono stati integrati, secondo uno schema comune (vedere Listato 1), e rilasciati alla piattaforma TCUBE, attraverso l'interfaccia providePWSData[7], per ciascuna richiesta effettuata dagli utenti. Quest'ultima si prende cura della loro visualizzazione su dispositivo mobile degli utenti. Il servizio Personal Weather Stations è stato strutturato in due sotto-componenti in base allo schema UML di Figura 1.

[4] https://www.wunderground.com/weather/api/d/pricing.html
[5] http://www.meteonetwork.it/supporto/meteonetwork-api-eng
[6] http://www.drihm.eu/
[7] http://150.145.8.157:9000/providePWSData

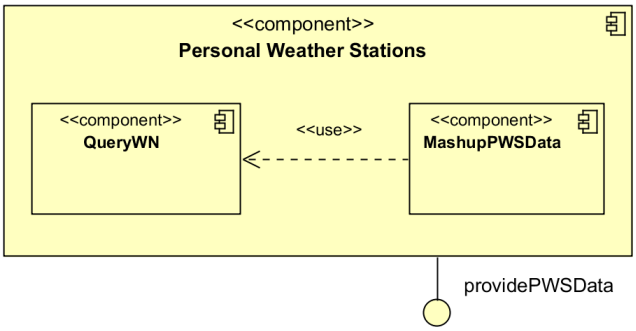


Figura 1. Sotto-componenti del servizio Personal Weather Stations e interfaccia esposta verso la piattaforma TCUBE

```
{
 ''responseHeader'': {
       ''status'': Integer,
       ''error_message'': String
 },
 ''response'': {
       ''numFound'': Integer,
       ''stations'': [
             {
                   ''network'': String,
                   ''id'': String,
                   ''latitudine'': String,
                   ''longitudine'': String,
                   ''temperatura'': String,
                   ''wind_speed'': String,
                   ''rain_rate'': String,
                   ''dew_point'': String,
                   ''smlp'': String,
                   ''daily_rain'': String,
                   ''rh'': String,
                   ''date'': String,
             }
       ]
 }
}
```

Listato 1. Schema dati JSON della risposta restituita dal servizio attraverso l'interfaccia providePWSData.

Componente QueryWN

- Recupero dei dati ogni 30 minuti – dalle due reti di sensori
- Salvataggio su database (NoSQL)

Nel diagramma del componente QueryWN di Figura 2, si evidenzia anche il ruolo dei vari *adapters*, che gestiscono le specificità (e.g. il formato, gli schemi strutture dati, ...) di ogni rete che fornisce i dati. Il loro uso consente il disaccoppiamento tra le principali funzionalità del componente (cioè recuperare e memorizzare dati) e le peculiarità di ogni rete. Questa scelta progettuale consente facilmente di incorporare dati da nuove reti che si rendessero disponibili, così da estendere (od integrare) l'area di copertura del servizio.

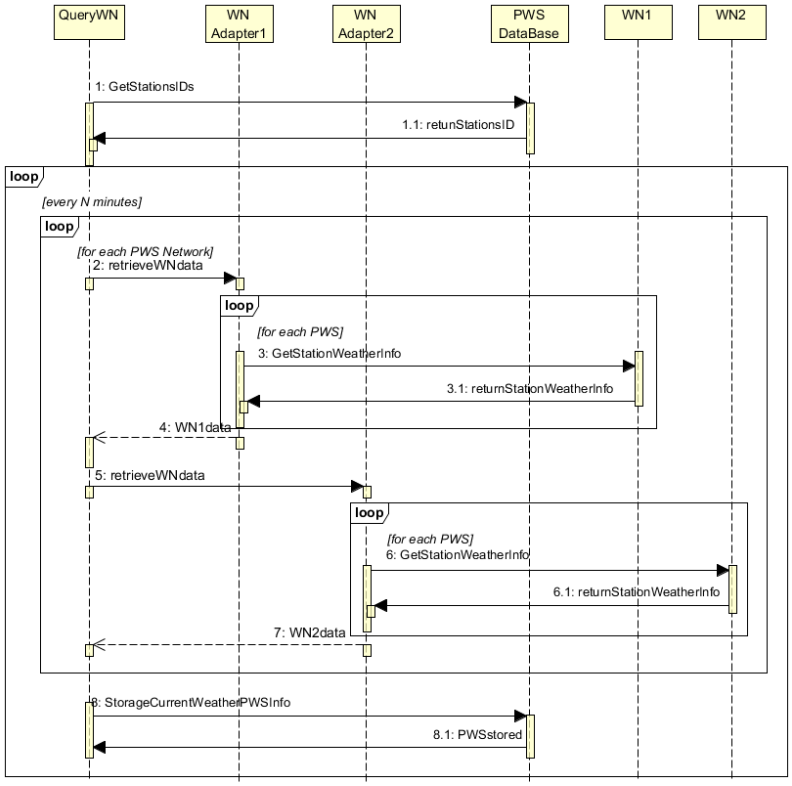


Figura 2. Diagramma sequenziale UML del componente QueryWN. Esso agisce come un demone, scaricando continuamente i dati da due reti di sensori ed archiviandole nella memoria dati locali

Componente MashupPWSData

- interroga il database locale (in seguito a richiesta proveniente da App utenti, attraverso la piattaforma TCUBE)
- restituisce i dati integrati delle due reti di sensori alla piattaforma. via l'interfaccia providePWSData

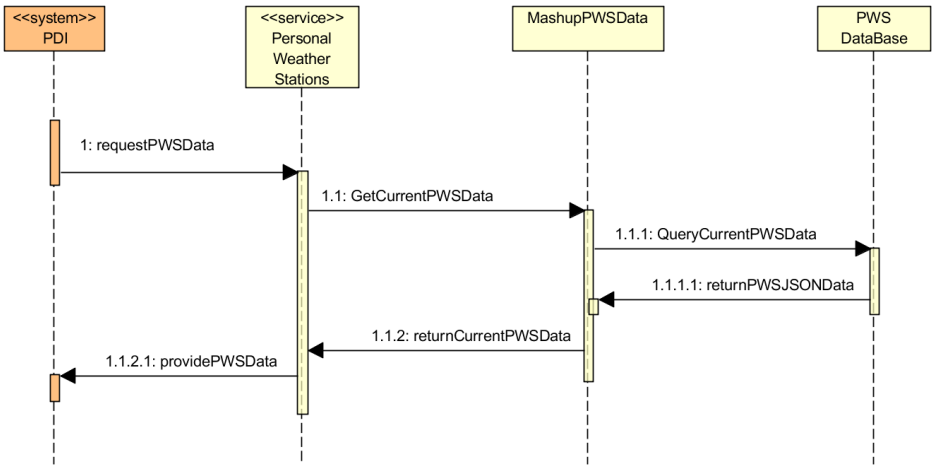


Figura 3. Diagramma sequenziale UML del componente MashupPWSData. Per ogni chiamata dalla piattaforma di integrazione (attivata da utenti TCUBE App), integra e ritorna i dati meteo estratti dalla memoria interna.

La Figura 4 mostra come appare all'utente TCUBE, visitatore area Porto Antico, l'interfaccia al servizio PWS proposta dall'App Mobile.

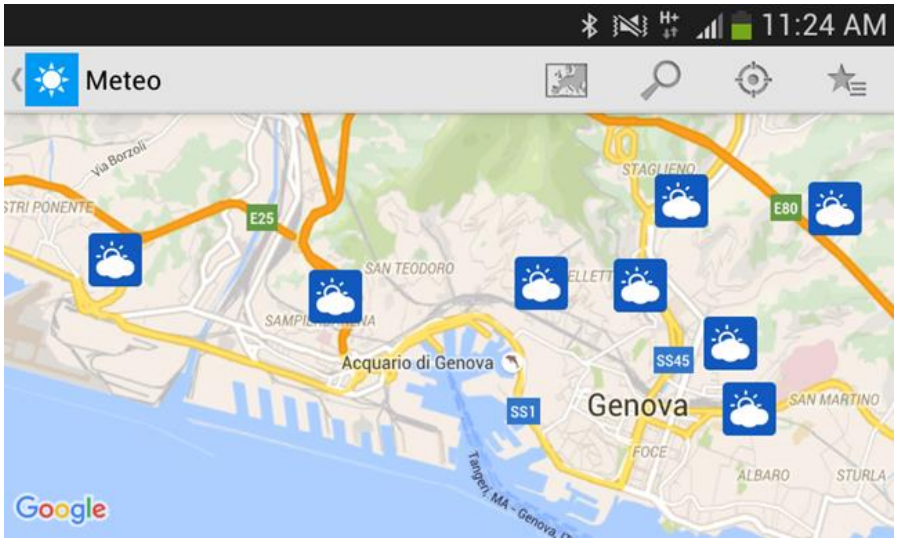


Figura 4. Visualizzazione dell'App TCUBE, delle stazioni meteo presenti sull'area limitrofa al Porto Antico di Genova – sito dimostratore del progetto TCUBE

3.3 Valutazione serie temporali dati meteo

Per lo sviluppo dell'attività di valutazione oggetto di questo documento si è scelto di implementare una soluzione alternativa al servizio Personal Weather Station, presentato in Sezione 3.2. Tale scelta è scaturita da due ragioni. Innanzitutto, dalle diverse fasi (i.e. Workpackages) di gestione del progetto TCUBE, che hanno richiesto che l'attività qui documentata fosse messa in essere prima del rilascio del servizio Personal Weather Station e della sua integrazione con la piattaforma di integrazione. Inoltre Personal Weather Station è stato progettato per raccogliere, integrare e fornire dati in tempo reali, mentre il presente lavoro, vuole analizzare il comportamento di scalabilità di fronte a dati via via crescenti al fine di simulare, almeno, le richieste hardware e software necessarie per soddisfare tali quantità in un'ottica Big Data.

Si è sviluppato pertanto un servizio di raccolta dati basato sulle stazioni messe a disposizione da un'unica rete di stazioni meteo, non considerando ai nostri fini gli aspetti di integrazione tra reti diverse, ma focalizzandosi piuttosto sulla raccolta di dati storici (i.e. serie temporali) su cui effettuare operazioni di elaborazione (e.g. Map-Reduce) ed analisi (e.g. machine learning), come spiegato nel seguito.

Sul piano operativo Weather Underground rende disponibili i propri dati attraverso API secondo diversi piani di servizio. La nostra scelta è stata di ususfruire dei servizi gratuiti del piano "Stratus" per sviluppatori, che prevede 500 richieste al giorno con un massimo di 10 al minuto. Di queste se ne effettuano un massimo di 464, pari a 58 stazioni per 8 giorni di letture, ad una velocità di 9 al minuto. La scelta di 464 interrogazioni inferiore alle 500 consentite riserva 36 richieste per test e controlli mentre l'autolimitazione a 9 per minuto garantisce di rientrare nel limite massimo consentito.

Il numero di 58 stazioni monitorate è stato scelto per coprire capillarmente l'area di interesse ma al contempo consentire lo scaricamento di 8 giornate di dati, per stazione, per ogni sessione giornaliera di interrogazione (8x58=464), in tal modo si ha la possibilità di recuperare i dati del giorno stesso e altri 7 giorni antecedenti.

Queste 7 letture aggiuntive consentono di recuperare eventuali sessioni di interrogazione saltate nei giorni immediatamente precedenti, fino ad un massimo di 7, o altrimenti di recuperare per le query ancora disponibili, i giorni nel passato precedenti al più vecchio giorno di lettura recuperato. Questa scelta consente una certa flessibilità nel lancio manuale della sessione di lettura da un normale PC, senza dover necessariamente disporre di un server sempre acceso programmato per effettuare automaticamente le letture ogni giorno. Inoltre, leggendo le osservazioni all'indietro nel passato si estende progressivamente il database storico che viene così accresciuto gradualmente.

Le ricerche su Weather Underground avvengono mediante chiamate “http GET” al relativo sito con il passaggio di alcuni parametri tra cui la password necessaria all’autenticazione, la funzione richiesta e i parametri per l’esecuzione della funzione stessa. Tra le API messe a disposizione per gli sviluppatori sono state utilizzate le seguenti funzioni: “geolookup” e “history”.

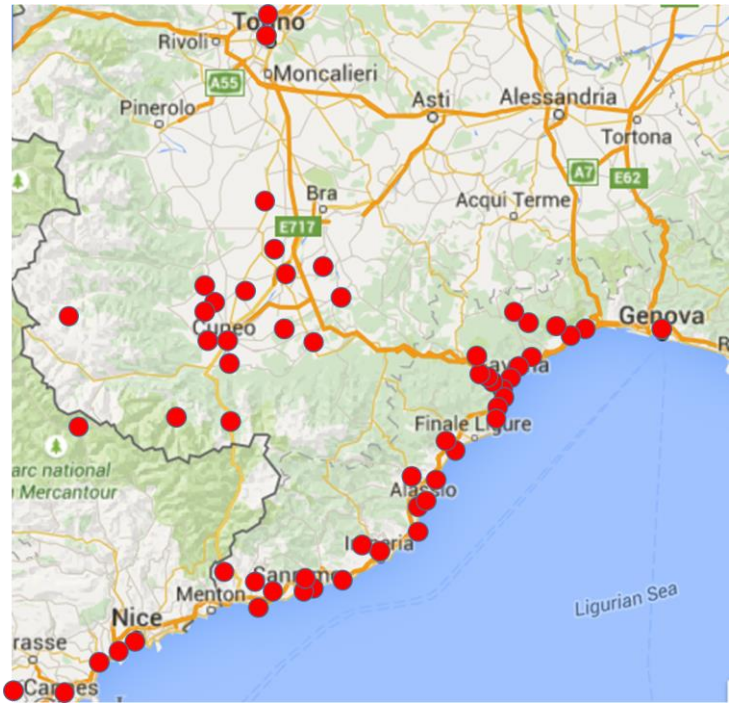


Figura 5. La copertura geografica delle stazioni WU prese in esame nello studio.

La funzione “geolookup” ritorna le 50 stazioni meteo più vicine ad una stazione indicata nel raggio di 40 km. Abbiamo usato “geolookup” per scegliere le stazioni di interesse, sviluppando un programma java “WScircle.java” che ha letto in input un file di testo contenente le città da interrogare in modo che, in base ai risultati, sia possibili ottenere la copertura territoriale desiderata. Il numero delle stazioni finale è quindi risultato essere un punto di equilibrio fra la copertura del territorio e i limiti sul numero di interrogazioni descritto nel precedente paragrafo.

Questa la sintassi di “geolookup”

```
http://api.wunderground.com/api/”Keystring”/geolookup/q/Italy/Albenga.xml
```

dove “keystring” va sostituito con la password assegnata al piano sottoscritto

Queste sono le informazioni disponibili per ogni stazione:

- nome stazione: una codifica Weather Underground
- sigla nazione/nazione
- tipo stazione: “aeroport” o “pws” per personal weather stations
- località/indirizzo
- città/paese
- longitudine
- latitudine

In Appendice sono presentati due programmi Java: WScircle.java per la ricerca ed estrazione dati delle 50 stazioni prescelte e Observations.java che consente una semplice rielaborazione dei dati storici letti ed archiviati dal precedente programma.

4 La piattaforma tecnologica utilizzata

Per rendere effettiva e riproducibile in altri contesti, l'analisi qui presentata, è stato necessario innanzitutto progettare uno schema di soluzione architetturale adeguata – rispetto a flessibilità degli strumenti software e hardware adottati e alla scalabilità delle soluzioni tecnologiche proposte in modo da premettere la loro adozione in analoghi, ma reali scenari, di produzione. In Figura 6 è schematizzato l'approccio di laboratorio che si è seguito, individuandone i tratti tecnologici caratteristici e la loro possibile evoluzione da ambiente di test, in sito locale sino al suo deployment in Cloud.

Diamo nel seguito una breve descrizione degli aspetti caratterizzanti il laboratorio sperimentale.

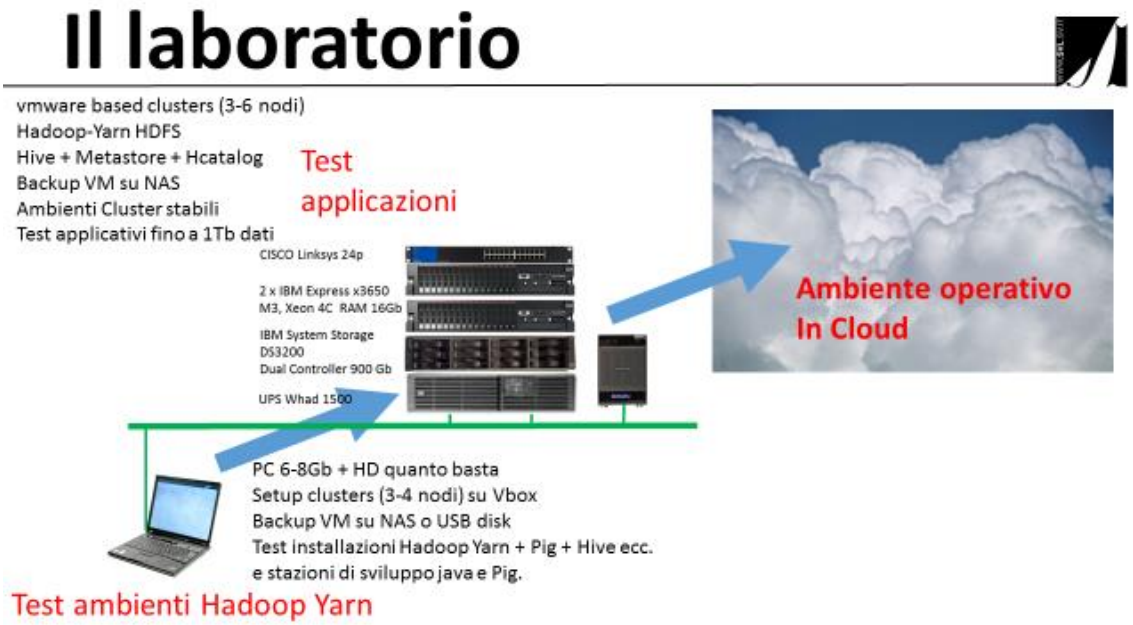


Figura 6. Schematizzazione dei diversi step di progettazione e realizzazione del laboratorio hardware, avente caratteristiche di scalabilità e replicabilità, alla base dell'attività sperimentale.

4.1 Hardware e Software di sistema

Per lo sviluppo del progetto è stata utilizzata prevalentemente una piattaforma hardware costituita da un piccolo cluster composto da due server IBM con uno storage condiviso e con le caratteristiche indicate nello schema di Figura 6.

Inoltre si sono usati due semplici Laptop con Microsoft Windows 7 con 6 e 8Gb di RAM e capacità e hard disk di circa 1Tb.

Un NAS Con funzioni di repository, condiviso fra i PC e le macchine virtuali sia sugli stessi PC che sui server.

Sui due server si è direttamente installato sull'hardware VMware vSphere in modo da poter simulare facilmente ambiente prototipali secondo le necessità.

Sui PC oltre a Microsoft Windows 7 è anche stato installato il sistema di virtualizzazione Oracle VirtualBox.

Su tutte le macchine virtuali che sono state utilizzate, sia sui PC in VirtualBox che sui server con VMware è stato installato Oracle Linux 7.

I PC sono stati utilizzati in maniera diretta per lo sviluppo, sotto Windows con l'IDE Eclipse, per i programmi JAVA di estrazione dati da Weather Underground. Sullo stesso PC viene eseguito, il programma di estrazione ed accumulo.

Sui PC è stato anche installato e utilizzato l'ambiente VirtualBox allo scopo di testare le installazioni dei vari strumenti Hadoop, Pig e Hive prima di effettuare la configurazione più stabile e definitiva, almeno per la sperimentazione applicativa che si è realizzata.

Macchine VirtualBox sono state utilizzate come stazioni di sviluppo per l'ambiente Hadoop.

Tutto il progetto è stato pensato per realizzare una sperimentazione a tre livelli:

- Primi test d'installazione e client di sviluppo: macchine VirtualBox su PC
- Cluster per test applicativi locali: cluster di macchine VMmware su server locali
- Test di ambienti su scala operativa: cloud

4.2 I cluster Hadoop

Durante la sperimentazione si è fatto ampiamente uso degli ambienti di virtualizzazione sia per i server componenti il cluster, sia per la worksation operatore che li ha governati. Questa scelta è stata estremamente utile per la flessibilità di gestione e per la possibilità di salvare intere macchine virtuali a fronte delle molte installazioni e prove e anche per la portabilità fra le macchine fisiche disponibili attualmente o in futuro.

Gli ambienti Hadoop utilizzati nello studio sono evoluti secondo i passi seguenti.

- Come suggerito dai suoi tutorial, il cluster Hadoop può essere simulato su una singola macchina al fine di provare ed eseguire semplici programmi Map-Reduce. Questa modalità è stata installata sulla workstation di lavoro ed utilizzata per le prove più semplici.
- Sullo stesso PC si è poi installato, su 4 macchina Virtual Box limitate a 768Mb di RAM, un vero e proprio cluster Hadoop con tutte le sue problematiche di configurazione.
- Per poter fare prove applicative più avanzate si è poi passati al cluster dei due server fisici sui quali è installato direttamente VMware vSphere. In questo ambiente sono state preparate quattro macchine virtuali per un totale di 4 nodi Hadoop.
- Consapevoli che anche questo cluster non ha una dimensione tale da essere considerato un 'vero' ambiente Big Data, ci siamo spostati su un ambiente cloud che, nello scenario immaginato rappresenta il passo verso ambienti operativi per applicazioni reali. A tal scopo si è scelta la piattaforma Amazon Web Services.

4.3 L'ambiente iniziale Hadoop in VirtualBox e VMWare

L'ambiente più semplice che è stato configurato richiede un solo PC. L'immagine che segue mostra la console di VirtulBox e le macchine virtuali utilizzate: "Hadoopmaster00", "Hadoopslave01,02,03". Esse costituiscono il cluster Hadoop, mentre "Development Pig/Hive/Oracle metastore" (il nome non riflette esattamente quanto poi installato) è la workstation di sviluppo attraverso la quale si sono governati e si sono acceduti tutti i cluster usati nel progetto.

Questo ambiente di ridottissime risorse hardware è stato prevalentemente utilizzato per effettuare le varie prove di installazione del cluster e degli altri programmi adottati, sfruttando comunque la duttilità e l'autonomia di avere tutto l'ambiente di lavoro su un solo PC.

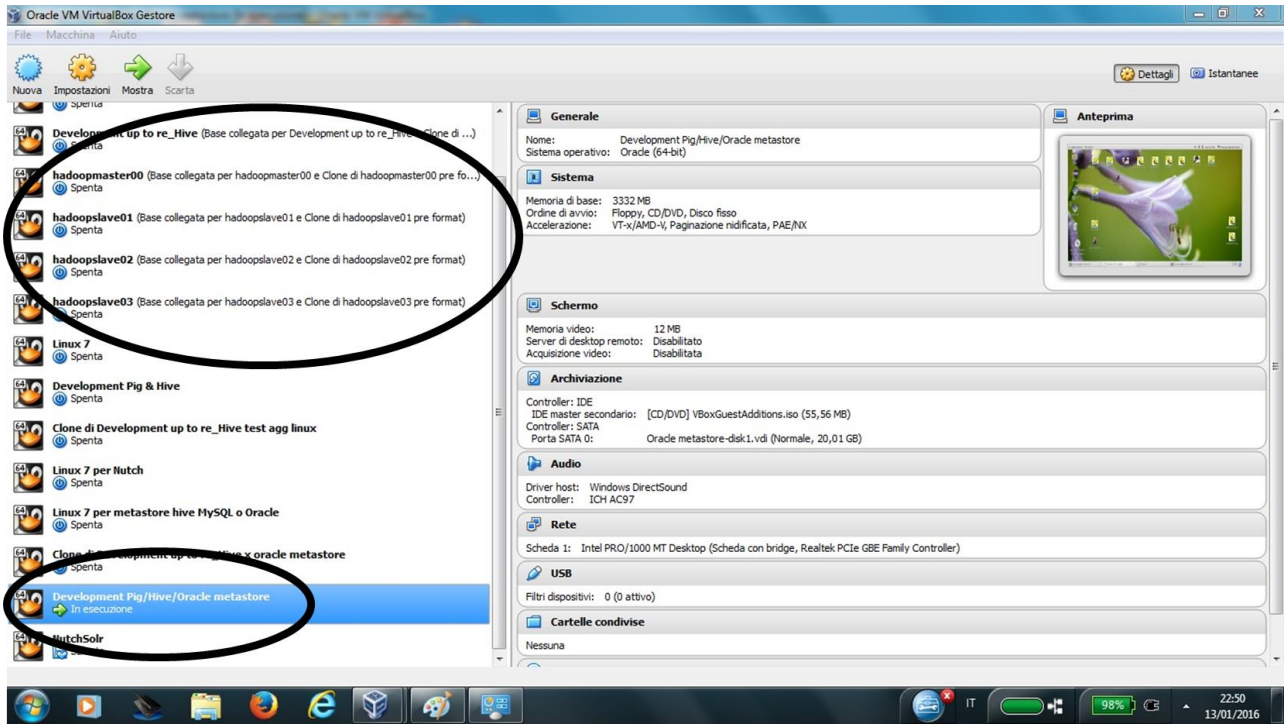


Figura 7. La console di VirtulBox

Testata sul cluster VirtualBox la configurazione desiderata, questa è stata replicata sul cluster VMware sui server, dove sono state eseguite la maggior parte delle prove applicative. Di seguito l'immagine delle console VMware sui due server fisici con la configurazione dei nodi: "HAD0", "HAD1", "HAD2", "HAD3".

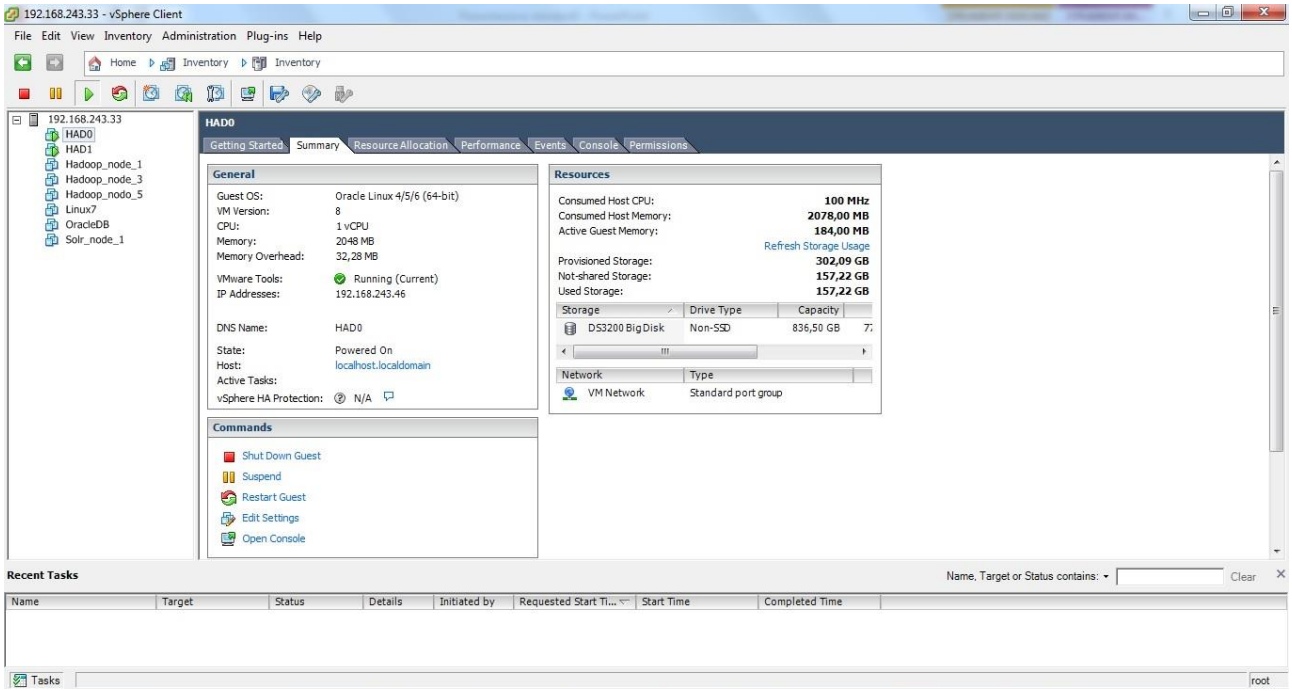


Figura 8. La console di VMware e le VM dei nodi Hadoop

Si noti anche la presenza attiva (triangolino verde in funzione) della macchina virtuale OracleDB dedicata ad ospitare il Metastore di Hive (vd. Sezione 5.4.1).

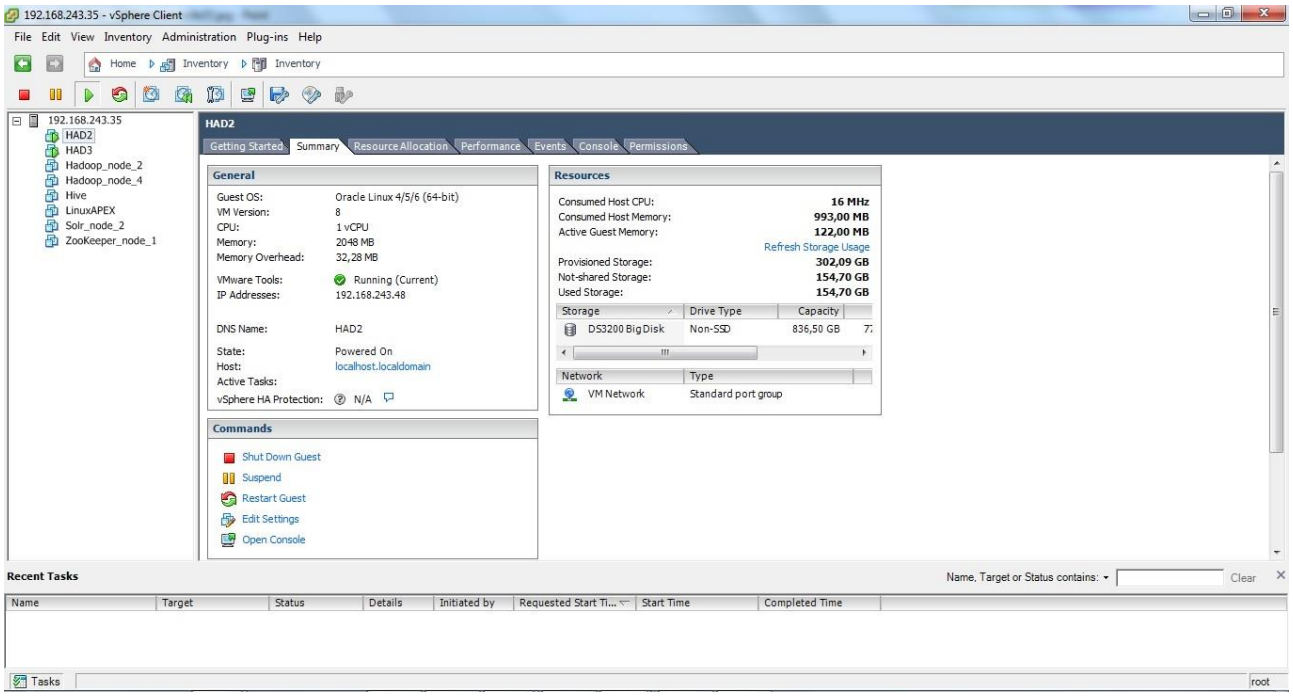


Figura 9. La console VMware e la VM dell'Hive metastore

Il cluster Hadoop su VMware è stato sempre aittivato o spento dalla console del nodo master con gli script associati alle due icone SVLHDPstart e SVLHDPstop.

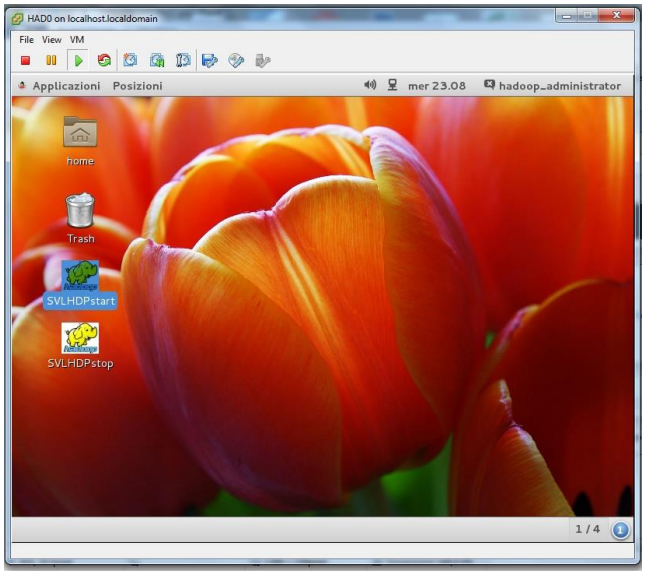


Figura 10. Il desktop del master node su VMware

4.4 La workstation di sviluppo per Hadoop

In questo paragrafo descriviamo sommariamente la principale postazione di lavoro di tutto il progetto. Si tratta di una macchina VirtualBox sul Windows 7 di un PC Lenovo Intel Core i5 2.53Ghz con 6Gb di RAM.

Questa macchina virtuale ospita Oracle Linux Server 7.1 ed è stata utilizzata per:

- eseguire i programmi PIG (vd. Sezione 5.3) localmente;
- per simulare Hadoop su una singola macchina;

- per gestire Il cluster Hadoop VirtualBox locale a questo PC (4 macchine VirtulBox per i primi test di installazione);
- come stazione di sviluppo PIG e Hive con destinazione:
    - il cluster locale VirtualBox
    - il cluster Hadoop basato sui due server VMware sulla LAN locale
    - il cluster Hadoop remoto su Amazon Web Services

Poiché questa workstation è stata pensata per lavorare alternativamente

- sulla simulazione del cluster Hadoop fatto su singola macchina
- sul cluster Hadoop locale basato su Virtual Box
- sul cluster Hadoop dei server VMware
- sul cluster Hadoop AWS

si è optato per alcuni script atti a predisporre la configurazione della macchina in maniera tale da lavorare con l'uno o l'altro cluster.

Una volta certi che la macchina sia stata configurata per operare sul cluster prescelto si possono utilizzare gli altri script che faranno riferimento così al cluster appropriato.

A questo punto è necessario fare una premessa. Quando si è approcciata l'installazione dell'SQL engine Hive (vd. Sezione 5.4.2) il PC con VirtualBox non disponeva di RAM sufficiente all'installazione della componente Metastore (DB dei metadati) di Hive in maniera condivisa fra più utenti quindi è stata fatta la seguente scelta:

- la configurazione locale al PC di Hive e del suo Metastore è stata fatta sul simulatore per singola macchina di Hadoop e nella configurazione non condivisa, che prevede l'installazione del Metastore sul più semplice database Derby;
- la configurazione sul 'vero' Cluster Hadoop di Hive e del suo Metastore condiviso, è stata fatta sul cluster VMware aggiungendo una macchina con un Oracle database dedicato al Metastore.

In tal modo si è data la possibilità di fare piccole e semplici prove del funzionamento Hive sul singolo PC e per le prove in cluster si à passati direttamente al cluster VMware sui server fisici.

4.5 Soluzione Cloud

Considerati i limiti degli ambienti hardware generalmente disponibili o comunque il costo e le difficoltà di gestione che si devono affrontare per mantenere tali cluster, il lavoro è stato progettato nella prospettiva di realizzare in Cloud gli ambienti operativi che dovessero derivare da successivi progetti.

I test di portabilità degli sviluppi locali in Cloud sono stati successivamente eseguiti su Amazon Web Services.

Per quanto l'intento sia stato di orientarsi sull'utilizzo di strumenti stabili e diffusi su tutte le piattaforme Cloud (come PIG e Hive), senza sposarne una in particolare, la scelta di AWS è stata fatta per la maturità dei servizi offerti, fra i primi nati per una diffusione su larga scala, per la gratuità iniziale dei servizi base, per il costo irrisorio (senza costi fissi iniziali) dei servizi a pagamento usati, per la nuova disponibilità di servizi di Machine Learning semplici e quindi di potenziale facile diffusione.

5 Esempio di gestione dei dati da sensore nel contesto Big Data

Le attività e i programmi finora descritti hanno riguardato la raccolta dei dati ambientali da una fonte web, delegando a Weather Underground l'interfacciamento con i sensori sul territorio. Come anticipato, la continua raccolta di questi dati nel tempo e la possibilità di aggiungerne altri da fonti e di contenuti diversi, può far crescere il loro volume a valori tali da rientrare nel contesto Big Data. Inoltre, proprio l'interesse verso queste tecnologie e la possibilità che offrono di indagare nuovi aspetti applicativi, ci hanno indotto a selezionare alcuni strumenti che, per diffusione e generalità, bene si prestano agli scopi proposti. Inoltre la scelta è ricaduta per quanto possibile su soluzioni di tipo Open Source, in accordo con la motivazione di ricerca Q1 ed i benefici di questo approccio brevemente discussi in Sezione 2.1.

5.1 Hadoop

5.1.1 Attivazione del cluster Hadoop

Le seguenti spiegazioni si riferiscono al cluster Hadoop dei server VMware ma in maniera analoga si può opera sul cluster (vd. Sezione 4.4)

Dalla console del master server eseguire lo script che lancia il comando /opt/hadoop.2.6.0/sbin/start-all.sh, il quale attiva tutti i server del cluster.

A questo punto possiamo monitorare il cluster hadoop da qualunque browser della rete con la URL:

```
http://"ip-address-del-nodo-master":50070/status.html
```

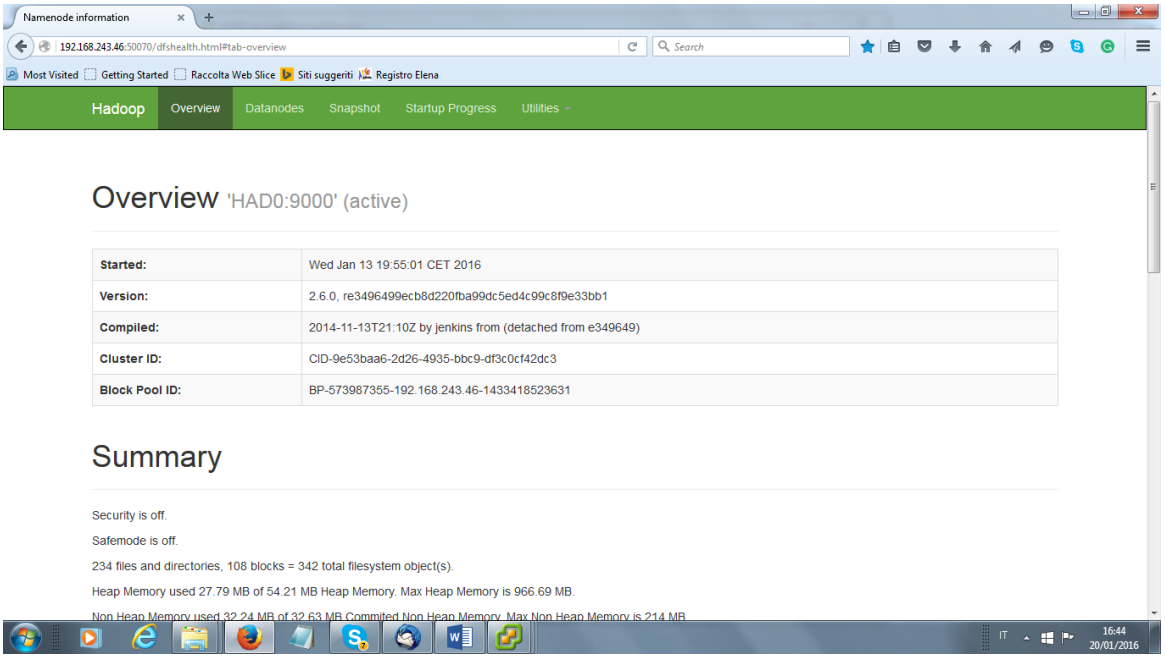


Figura 11. Monitoraggio del cluster Hadoop

e, la URL `http://"ip-address-del-nodo-master":50070`

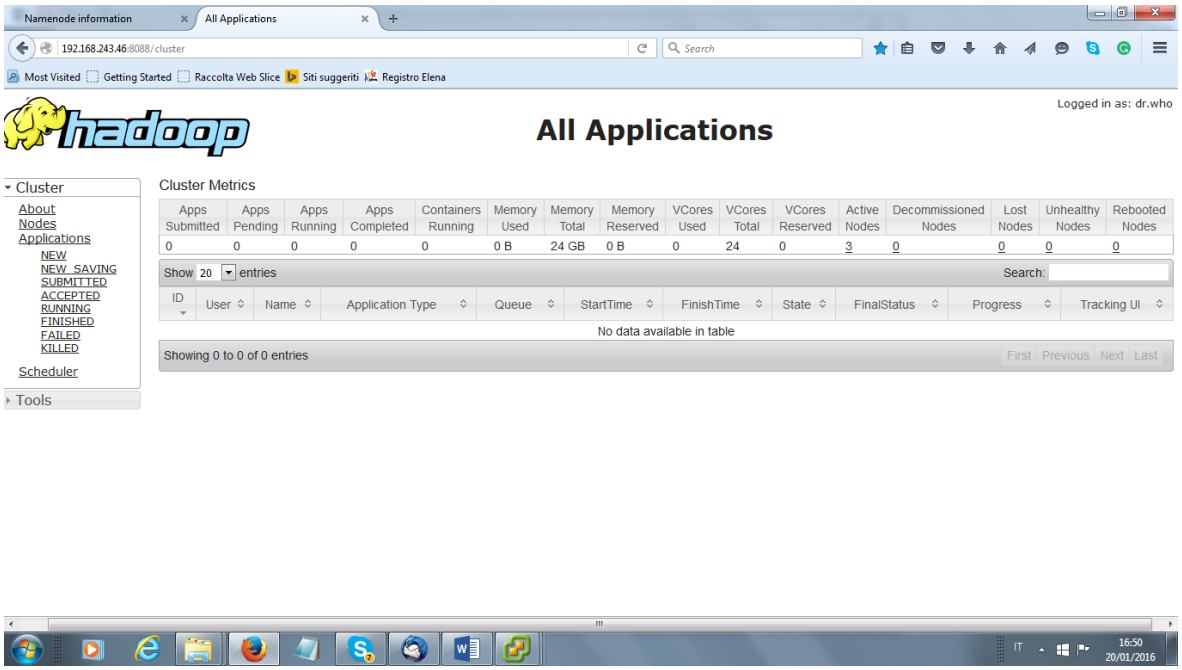


Figura 12. Monitoraggio del cluster Hadoop

5.1.2 Esecuzione di comandi HDFS

Da una shell Linux sulla console del nodo master possiamo eseguire i comandi Hadoop. In maniera analoga potremmo farlo da una shell della workstation di sviluppo sul cluster VirtualBox. Ad esempio per visualizzare il file system HDFS. Nota: Non si descrivono qui i comandi Hadoop né HDFS rimandando alla documentazione dei rispettivi sistemi.

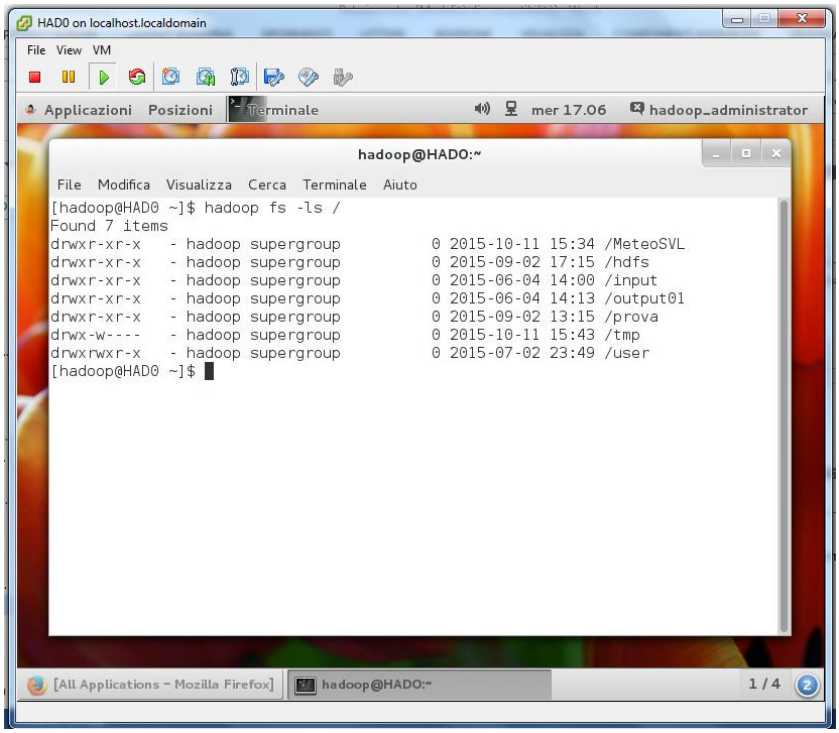


Figura 13. Contenuto del file system HDFS

5.2 JAVA e Map-Reduce

Il linguaggio Java è certamente un efficacie e potente strumento per realizzare applicazioni Map-Reduce in Hadoop. Programmando in Java, dal quale chiamare le funzioni di Map e Reduce, è possibile fare le prime esperienze con Hadoop [1]. Tuttavia dal momento che si desidera affrontare efficacemente progetti applicativi più ampi, è lecito chiedersi se sia meglio utilizzare uno strumento in grado di risolvere qualsiasi problema usando Map-Reduce in maniera nativa ma molto tecnica, o se sia più produttivo spostarsi su strumenti di più alto livello, che possano più facilmente ottenere i risultati desiderati pur rinunciando ad un potenziale non sempre necessario ai propri scopi. Per questi motivi si è guardato alle possibili alternative, esaminate nel seguito.

5.3 PIG

PIG è una piattaforma per lo sviluppo ad alto livello secondo il paradigma Map-Reduce su Hadoop. Il nome deriva dal fatto che il maiale è un animale che mangia di tutto, caratteristica che i progettisti di PIG hanno cercato di dargli nella capacità di trattare facilmente ogni tipo di dato.

PIG Latin è il linguaggio di programmazione di questa piattaforma che opera in una logica data-flow. In verità sono moltissimi gli strumenti di alto livello creati per agevolare l'uso di Hadoop, ma PIG è fra i primi e i più diffusi.

Nonostante il forte sviluppo che hanno avuto i motori SQL per interfacciare applicazioni Hadoop, soprattutto per andare incontro alla grande diffusione e conoscenza di questo linguaggio di interrogazione di database da parte di moltissimi programmatori, gli estimatori di PIG lo considerano duttile e adatto a risolvere qualunque problematica si debba affrontare, senza dover ricorrere ad altro. Un'altra sua caratteristica, che lo rende particolarmente flessibile, è l'estendibilità che si ottiene scrivendo funzione in altri linguaggi quali Java, Python, JavaScript, Ruby or Groovy.

Così come in tantissimi casi in cui viene utilizzato, noi lo abbiamo adottato per incamerare i dati grezzi, omogeneizzarne il formato, pulirli in una sorta di processo ETL (Extract, Transform and Load) pensando di creare poi una sorta di datawarehouse accessibile nel tempo da molti altri fruitori.

In verità siamo andati oltre le operazioni di lettura e pulitura dei dati grezzi, effettuando anche semplici elaborazioni che già possono dare risultati interessanti.

Ipotizzando infatti che già si conoscano le estrazioni e le elaborazioni da effettuarsi periodicamente, potrebbe essere inutile passare dalle interrogazioni in SQL su un datawarehouse, al fine di fornire a utenti finali piccoli file leggibili in uno spreadsheet. O, in altri casi, un utente capace di scrivere semplici programmi PIG potrebbe in ogni momento fare estrazioni ed elaborazioni a partire da grandissimi file di testo.

Un'interessante caratteristica del linguaggio, è quella di essere interattivo come tutti i linguaggi interpretati. È infatti possibile scrivere un'istruzione alla volta per la quale PIG esegue una valutazione sintattica e sui metadati fino a quel punto accumulati (tipi, relazioni ecc.), identificando immediatamente gli eventuali errori, consentendo un efficace metodo di debugging. Alla prima istruzione che prevede la produzione di un output, il programma viene compilato e mandato in esecuzione in Map-Reduce sul cluster Hadoop.

Per rendere più agevole la fase di sviluppo è anche possibile, estrarre dei piccoli sottoinsiemi casuali dei dati contenuti nelle tabelle su cui si intende lavorare (istruzione *sample*), ciò consente di eseguire rapide esecuzioni e facilitare la programmazione e il debugging, funzionalità che si abbina al meglio con l'ulteriore possibilità di eseguire PIG localmente sul PC e non in cluster.

Una volta terminato lo sviluppo è possibile eseguire il programma PIG completo e non interpretandolo un'istruzione alla volta.

Non ci addentriamo nella descrizione del linguaggio per il quale si rimanda al sito dell'Apache Foundation[8], o alla numerosissima documentazione rintracciabile in rete [10].

5.3.1 Cleaning

Questa operazione è generalmente effettuata sui dati per ottenere un data-set elaborabile al meglio, ripulendoli dai campi inutili o non omogenei nel loro formato.

Analizzeremo ora le istruzioni PIG che eseguono questo compito.

[8] https://pig.apache.org/

- La prima consiste nella lettura dei dati dal file di testo.
- Due trattini "–" segnano l'inizio di una riga di commento
- Alla sinistra dell'uguale la "relazione" nel nostro caso **rawdata**, termine con cui si indica un insieme di righe di dati come fosse un'assegnazione ad una variabile, dopo aver eseguito le operazioni esplicitate a destra dell'uguale.
- **Load** è l'istruzione PIG che legge il file di input dal percorso indicato
- **Using Pigstorage(' ')** fa parte dell'istruzione e indica che lo spazio è il separatore fra i campi
- **AS** introduce l'elenco dei nomi a cui si farà riferimento ai campi via via letti
- Per alcuni di essi (es. location:chararray) si indica il tipo di dato. Agli altri PIG attribuisce il tipo secondo lui più affine. In questo caso si associa il tipo chararray anche ai campi che abitualmente contengono numeri perché si è constatato che in taluni casi le stazioni scrivono la stringa "null" in assenza del dato numerico che ci saremmo aspettati.

```
-- letti tutti come chararray per pulire prima i dati numerici che hanno qualche
valore null scritto come stringa
rawdata = load '/home/hadoop/MeteoSVL/input' using PigStorage(' ') AS (time, apm,
cest, on, month, day, year, code, country, sttype, location:chararray, city,
region:chararray, lat, lon, tempm:chararray, dewptm:chararray, hum:chararray,
wspdm:chararray, wgustm:chararray, wdird:chararray, wdire:chararray,
vism:chararray, pressurem:chararray, windchillm:chararray, heatindexm:chararray,
precipm:chararray, conds:chararray, icon:chararray, fog:chararray,
rain:chararray, snow:chararray, hail:chararray, thunder:chararray,
tornado:chararray);


-- ricavare un sample se è necessario operare su pochi dati
rawdatasample = sample rawdata 0.01;
```

Nella riga successiva inizia una fase di adattamento dei dati, ad esempio si trasformano i nomi dei mesi nel corrispondente numero, ma il punto essenziale da cogliere è che l'istruzione **foreach** analizza e modifica i campi indicati di ogni riga della relazione **rawdata,** precedentemente letta dal file, e genera una nuova relazione **cleaned0.**

Altre modifiche apportate riguardano la riscrittura delle date, usando funzioni di sottostringhe e concatenazioni, in un formato utile alle successive elaborazioni e la sostituzione dei vari modi con cui sono indicate le assenze di dato, spesso rappresentati con un grande numero negativo (-99...) composti da quantità di cifre variabili da caso a caso.

```
-- per pulire i dati prima cerco le uguaglianze tra chararray
cleaned0 = foreach rawdata generate (chararray)year,(month == 'January'?
'01':(month=='February'?'02':(month=='March'?'03':(month=='April'?'04':(month=='M
ay'?'05':(month=='June'?'06':(month=='July'?'07':(month=='August'?'08':(month=='S
eptember'?'09':(month=='October'?'10':(month=='November'?'11':(month=='December'?
'12':''))))))))))) AS monthnum, SUBSTRING(day,0,2) AS
(day:chararray),(SIZE(time)==4?CONCAT('0',time):time) AS time, (chararray)month,
(chararray)code, (chararray)country, (chararray)sttype,(location=='null'or
SUBSTRING(location,0,4)=='-999'?null:(chararray)location) AS location,
(chararray)city, (region=='null'or SUBSTRING(region,0,4)=='-
999'?null:(chararray)region) AS region, (float)lat, (float)lon,
(tempm=='null'?null:(float)tempm) AS tempm, (dewptm=='null'?null:(float)dewptm)
AS dewptm, (hum=='null'?null:(int)hum) AS hum, (wspdm=='-9999' or
wspdm=='null'?null:(float)wspdm)AS wspdm, (wgustm=='-9999' or
wgustm=='null'?null:(float)wgustm)AS wgustm, (wdird=='-9999' or wdird=='null'
?null:(int)wdird)AS wdird, (wdire=='null'or SUBSTRING(wdire,0,4)=='-
999'?null:(chararray)wdire)AS wdire, (vism=='-9999' or vism=='null'
?null:(int)vism) AS vism, (pressurem=='null'?null:(int)pressurem)AS pressurem,
(windchillm=='null'?null:(float)windchillm)AS windchillm,
(heatindexm=='null'?null:(float)heatindexm)AS heatindexm, (precipm=='null'
?null:(float)precipm)AS precipm, (conds == 'null'or SUBSTRING(conds,0,4)=='-
999'or conds =='unknown' or conds == 'Unknown'?null:(chararray)conds)AS conds,
(icon=='null'or SUBSTRING(icon,0,4)=='-999' or icon=='unknown' or
icon=='Unknown'?null:(chararray)icon)AS icon, (fog=='null' ?null:(int)fog)AS fog,
(rain=='null'?null:(int)rain) AS rain, (snow=='null'?null:(int)snow)AS snow,
```

```
(hail=='null'?null:(int)hail)AS    hail,    (thunder=='null'?null:(int)thunder)AS
thunder, (tornado=='null'?null:(int)tornado)AS tornado;
```

Sono state necessarie ancora un paio di istruzioni, visibili nel listato completo riportato in appendice (MeteoSVL.pig), per ottenere una relazione **cleaned2** nel formato desiderato.

A questo punto si è salvato un file CSV con i dati puliti ed utilizzabili utile a successive elaborazioni ed è con l'istruzione **store** che, durante la fase di sviluppo, nella quale il programma è stato elaborato in maniera interattiva (un'istruzione alla volta), questo è stato compilato nella logica Map-Reduce e passato al cluster Hadoop per l'esecuzione.

```
-- creare una tabella pulita su disco
store cleaned2 INTO '/home/hadoop/MeteoSVL/SVIMCNNItable' USING PigStorage (';');
```

5.3.2 La logica del programma

La nostra sperimentazione è proseguita cercando di attuare alcune logiche applicative, ed allo scopo di testare gli strumenti adottati si è ipotizzando a titolo esemplificativo un interesse nel settore agricolo filtrando i dati nei mesi fra la primavera e l'autunno, come riportato dalle linee di codice seguenti.

```
-- filtrare i dati da marzo ad agosto per esempi su ipotetica stagione agricola
season = filter cleaned2 by (monthnum>2 and monthnum<9);
```

Si sono quindi raggruppati i dati per giornata e per località. Ricordiamo che ogni riga del file da cui siamo partiti contiene i dati di una singola osservazione, ma che in una giornata si raccolgono circa 100 osservazioni per ogni stazione meteo.

```
-- raggruppare per giornata e localita'
dateplace              =              group              season              by
(code,sttype,year,monthnum,day,city,location,lat,lon);
```

Poi si è calcola il massimo e il minimo di giornata per ogni variabile metereologica misurata in valori numerici.

```
-- calcolare max min ecc.
maxmindata     =     foreach     dateplace     generate     FLATTEN(group)     as
(code,sttype,year,monthnum,day,city,location,lat,lon),    MAX(season.tempm)    AS
maxtempm,  MIN(season.tempm)  AS  mintempm,  MAX(season.dewptm)  AS  maxdewptm,
MIN(season.dewptm) AS mindewptm, MAX(season.hum) AS maxhum, MIN(season.hum) AS
minhum,   MAX(season.wspdm)   AS   maxwspdm,   MIN(season.wspdm)   AS   minwspdm,
MAX(season.wgustm) AS maxwgustm, MIN(season.wgustm) AS minwgustm,MAX(season.vism)
AS   maxvism,   MIN(season.vism)   AS   minvism,      MAX(season.pressurem)   AS
maxpressurem,MIN(season.pressurem)  AS  minpressurem,  MAX(season.windchillm)  AS
maxwindchillm, MIN(season.windchillm) AS minwindchillm, MAX(season.heatindexm) AS
maxheatindexm,MIN(season.heatindexm)  AS  minheatindexm,  MAX(season.precipm)  AS
maxprecipm, MIN(season.precipm) AS minprecipm;
```

Viene quindi fatta la differenza fra il Massimo e il minimo di giornata per ciascuna variabile

```
-- calcolare l'escursione giornaliera di ciascuna variabile
deltadata          =          foreach          maxmindata          generate
code,sttype,year,monthnum,day,city,location,lat,lon,maxtempm-mintempm          AS
deltatempm,  maxdewptm-mindewptm  AS  deltadewptm,  maxhum-minhum  AS  deltahum,
maxwspdm-minwspdm AS deltawspdm, maxwgustm-minwgustm AS deltawgustm, maxvism-
minvism AS deltavism, maxpressurem-minpressurem AS deltapressurem, maxwindchillm-
minwindchillm AS deltawindchillm, maxheatindexm-minheatindexm AS deltaheatindexm,
maxprecipm-minprecipm AS deltaprecipm;
```

Vengono quindi raggruppate le escursioni giornaliere precedentemente calcolate per ogni località

```
-- raggruppare le escursioni stagionali per ogni località
deltaxplace = group deltadata by (code,sttype,city,location,lat,lon);
```

Si calcolano le medie stagionali delle escursioni giornaliere per ciascuna località

```
-- calcolare le medie stagionali
meandata = foreach deltaxplace generate FLATTEN(group) as
(code,sttype,city,location,lat,lon), AVG(deltadata.deltatempm) AS avgdeltatempm,
AVG(deltadata.deltadewptm) AS avgdeltadewptm, AVG(deltadata.deltahum) AS
avgdeltahum, AVG(deltadata.deltawspdm)AS avgdeltadeltawspdm,
AVG(deltadata.deltawgustm) AS avgdeltawgustm, AVG(deltadata.deltavism) AS
avgdeltavism, AVG(deltadata.deltapressurem) AS avgdeltapressurem,
AVG(deltadata.deltawindchillm) AS avgdeltawindchillm,
AVG(deltadata.deltaheatindexm) AS avgdeltaheatindexm, AVG(deltadata.deltaprecipm)
AS avgdeltaprecipm;
```

Vengono ordinate le località che hanno le maggiori escursioni termiche giornaliere durante la stagione del raccolto agricolo.

```
-- ordinare per la variabile temperatura
srtdeltatempm = ORDER meandata BY avgdeltatempm desc;
```

Infine vien salvata in un file la classifica così ottenuta

```
store srtdeltatempm INTO '/home/hadoop/MeteoSVL/SVIMCNNIescterm' USING PigStorage
(';');
```

Con le seguenti istruzioni si calcolano invece le medie giornaliere delle variabili metereologiche rilevate per vederne l'andamento nel tempo.

```
-- Analisi medie per Savona
-- considerare solo i dati di savona
savona = filter cleaned2 by (code matches 'I90579414');
```

Si accorpano le osservazioni giornaliere.

```
-- raggruppare tutti i dati di ogni giorno
savonaxday = group savona by (year,monthnum,day);
```

Poi si calcolano le medie giornaliere.

```
-- produrre la tabella con le medie giornaliere della stagione agricola per le
variabili principali temperatura, umidità, pressione
avgday = foreach savonaxday generate FLATTEN(group) as (year,monthnum,day),
AVG(savona.tempm) AS avgtempm, AVG(savona.hum) AS avghum, AVG(savona.pressurem)
AS avgpressurem;
```

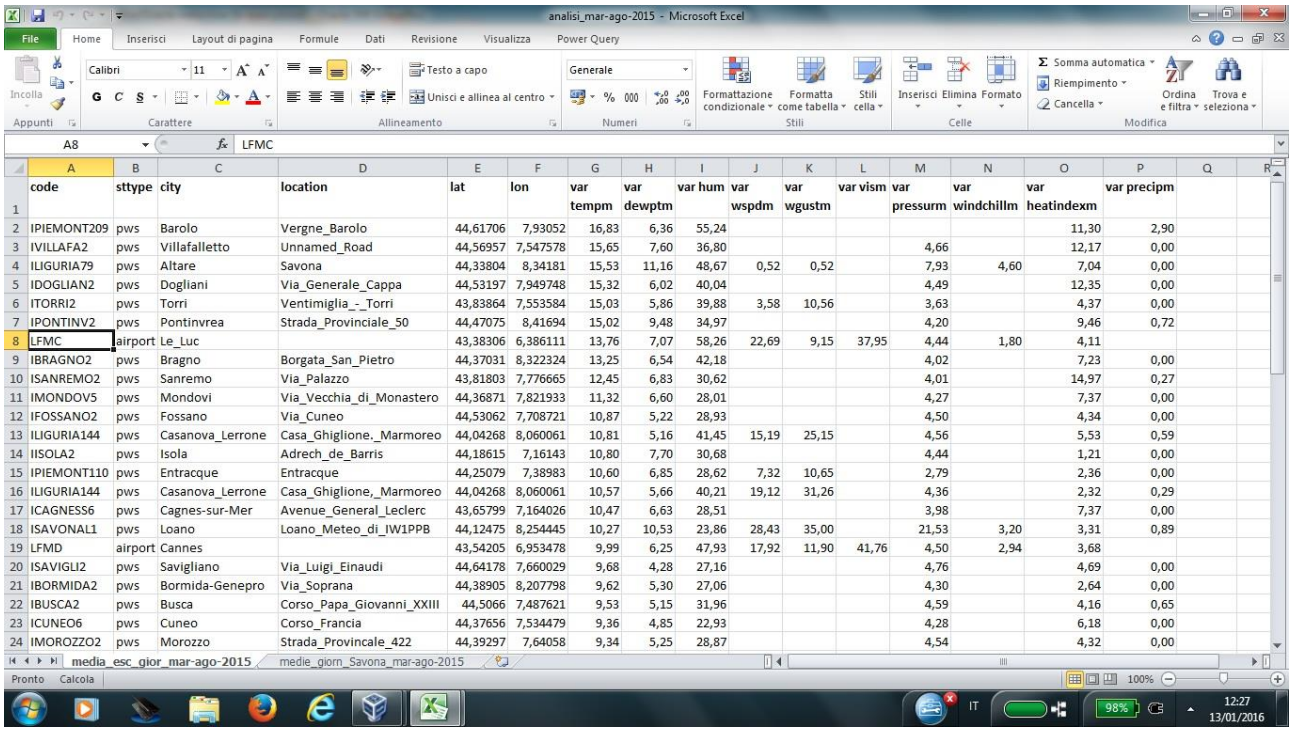
Si ordinano per data

```
-- ordinare alfabeticamente per data
srtavgday = ORDER avgday BY year,monthnum,day;
```

Infine si salva quanto ottenuto dalle linee di codice precedenti in un nuovo file:

```
-- archiviare i dati ottenuti
store srtavgday INTO '/home/hadoop/MeteoSVL/Savona' USING PigStorage (';');
```

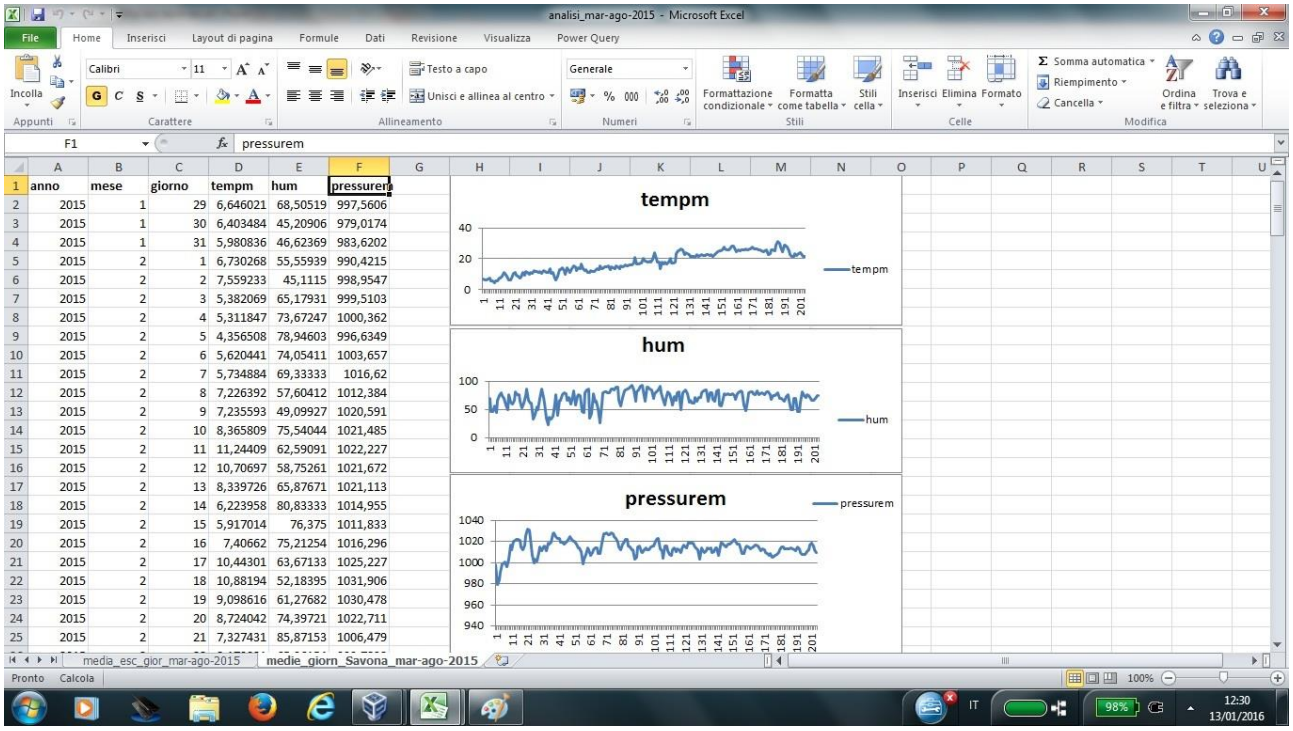
Sarà sufficiente importare il nuovo file in Excel per vedere i dati o produrre grafici. Ad esempio in Figura 14:



	code	sttype	city	location	lat	lon	var tempm	var dewptm	var hum	var wspdm	var wgustm	var vism	var pressurm	var windchillm	var heatindexm	var precipm
2	IPIEMONT209	pws	Barolo	Vergne_Barolo	44,61706	7,93052	16,83	6,36	55,24						11,30	2,90
3	IVILLAFA2	pws	Villafalletto	Unnamed_Road	44,56957	7,547578	15,65	7,60	36,80				4,66		12,17	0,00
4	ILIGURIA79	pws	Altare	Savona	44,33804	8,34181	15,53	11,16	48,67	0,52	0,52		7,93	4,60	7,04	0,00
5	IDOGLIAN2	pws	Dogliani	Via_Generale_Cappa	44,53197	7,949748	15,32	6,02	40,04				4,49		12,35	0,00
6	ITORRI2	pws	Torri	Ventimiglia_-_Torri	43,83864	7,553584	15,03	5,86	39,88	3,58	10,56		3,63		4,37	0,00
7	IPONTINV2	pws	Pontinvrea	Strada_Provinciale_50	44,47075	8,41694	15,02	9,48	34,97				4,20		9,46	0,72
8	LFMC	airport	Le_Luc		43,38306	6,386111	13,76	7,07	58,26	22,69	9,15	37,95	4,44	1,80	4,11	
9	IBRAGNO2	pws	Bragno	Borgata_San_Pietro	44,37031	8,322324	13,25	6,54	42,18				4,02		7,23	0,00
10	ISANREMO2	pws	Sanremo	Via_Palazzo	43,81803	7,776665	12,45	6,83	30,62				4,01		14,97	0,27
11	IMONDOV5	pws	Mondovi	Via_Vecchia_di_Monastero	44,36871	7,821933	11,32	6,60	28,01				4,27		7,37	0,00
12	IFOSSANO2	pws	Fossano	Via_Cuneo	44,53062	7,708721	10,87	5,22	28,93				4,50		4,34	0,00
13	ILIGURIA144	pws	Casanova_Lerrone	Casa_Ghiglione,_Marmoreo	44,04268	8,060061	10,81	5,16	41,45	15,19	25,15		4,56		5,53	0,59
14	IISOLA2	pws	Isola	Adrech_de_Barris	44,18615	7,16143	10,80	7,70	30,68				4,44		1,21	0,00
15	IPIEMONT110	pws	Entracque	Entracque	44,25079	7,38983	10,60	6,85	28,62	7,32	10,65		2,79		2,36	0,00
16	ILIGURIA144	pws	Casanova_Lerrone	Casa_Ghiglione,_Marmoreo	44,04268	8,060061	10,57	5,66	40,21	19,12	31,26		4,36		2,32	0,29
17	ICAGNESS6	pws	Cagnes-sur-Mer	Avenue_General_Leclerc	43,65799	7,164026	10,47	6,63	28,51				3,98		7,37	0,00
18	ISAVONAL1	pws	Loano	Loano_Meteo_di_IW1PPB	44,12475	8,254445	10,27	10,53	23,86	28,43	35,00		21,53	3,20	3,31	0,89
19	LFMD	airport	Cannes		43,54205	6,953478	9,99	6,25	47,93	17,92	11,90	41,76	4,50	2,94	3,68	
20	ISAVIGLI2	pws	Savigliano	Via_Luigi_Einaudi	44,64178	7,660029	9,68	4,28	27,16				4,76		4,69	0,00
21	IBORMIDA2	pws	Bormida-Genepro	Via_Soprana	44,38905	8,207798	9,62	5,30	27,06				4,30		2,64	0,00
22	IBUSCA2	pws	Busca	Corso_Papa_Giovanni_XXIII	44,5066	7,487621	9,53	5,15	31,96				4,59		4,16	0,65
23	ICUNEO6	pws	Cuneo	Corso_Francia	44,37656	7,534479	9,36	4,85	22,93				4,28		6,18	0,00
24	IMOROZZO2	pws	Morozzo	Strada_Provincale_422	44,39297	7,64058	9,34	5,25	28,87				4,54		4,32	0,00

Figura 14. La lista delle stazioni con massima escursione termica

In Figura 15 la schermata Excel con i dati e i grafici stagionali delle medie giornaliere di temperatura, umidità e pressione



	anno	mese	giorno	tempm	hum	pressurem
2	2015	1	29	6,646021	68,50519	997,5606
3	2015	1	30	6,403484	45,20906	979,0174
4	2015	1	31	5,980836	46,62369	983,6202
5	2015	2	1	6,730268	55,55939	990,4215
6	2015	2	2	7,559233	45,1115	998,9547
7	2015	2	3	5,382069	65,17931	999,5103
8	2015	2	4	5,311847	73,67247	1000,362
9	2015	2	5	4,356508	78,94603	996,6349
10	2015	2	6	5,620441	74,05411	1003,657
11	2015	2	7	5,734884	69,33333	1016,62
12	2015	2	8	7,226392	57,60412	1012,384
13	2015	2	9	7,235593	49,09927	1020,591
14	2015	2	10	8,365809	75,54044	1021,485
15	2015	2	11	11,24409	62,59091	1022,227
16	2015	2	12	10,70697	58,75261	1021,672
17	2015	2	13	8,339726	65,87671	1021,113
18	2015	2	14	6,223958	80,83333	1014,955
19	2015	2	15	5,917014	76,375	1011,833
20	2015	2	16	7,40662	75,21254	1016,296
21	2015	2	17	10,44301	63,67133	1025,227
22	2015	2	18	10,88194	52,18395	1031,906
23	2015	2	19	9,098616	61,27682	1030,478
24	2015	2	20	8,724042	74,39721	1022,711
25	2015	2	21	7,327431	85,87153	1006,479

Figura 15. I dati e i grafici in Excel

Questo esempio mostra quanto un utente che necessiti di una limitatissima quantità di dati da elaborare in un programma di produttività individuale, possa estrarli da un insieme estremamente vasto (Big Data), grazie alla potenza

di calcolo di un cluster Hadoop, e di uno strumento di programmazione ad alto livello (PIG) che traduca le esigenze applicative in istruzioni Map-Reduce.

5.4 HIVE

5.4.1 Un data-warehouse per Hadoop

Sulla base dei risultati fin qui ottenuti si potrebbe obiettare che la conoscenza di un linguaggio come PIG è molto specifica e poco diffusa. Per cui questo tipo di programmazione si presta bene per chi intenda investire nella conoscenza di questo linguaggio o per quelle elaborazioni periodiche e ripetitive per le quali sia plausibile scrivere programmi da utilizzare poi in maniera ripetitiva, per estrarre file di dati e report predeterminati. Abbiamo quindi immaginato un altro scenario in cui, accettando l'uso di PIG per la preparazione dei dati, questi vengano fatti confluire in un datawarehouse permanente interrogabile con il diffuso linguaggio SQL.

Si è quindi ricorso ad Hive, un SQL engine ormai storico nell'ecosistema Hadoop, in grado di ricevere programmi SQL e di tradurli in istruzioni Map-Reduce da eseguire su un cluster Hadoop. L'SQL di Hive, meglio detto HiveQL, non è perfettamente SQL compliant. Ha tuttavia raggiunto, una stabilità, diffusione ed economicità dei servizi cloud che lo mettono a disposizione, che lo rendono strumento ideale per chi intenda diffondere l'uso di queste tecnologie a fasce d'utenza ben più ampie delle aziende o degli istituti di ricerca che si sono finora affacciati al mondo Big Data.

5.4.2 L'utilizzo di Hive

Le esemplificazioni seguenti si riferiscono all'utilizzo di Hive sul cluster Hadoop basato sui server fisici e VMware [11]. Le stesse sarebbero anche eseguibili sulla workstation di sviluppo, preferibilmente sul cluster di Hadoop simulato localmente, basato sul Derby Metastore. Si è ritenuto comunque più significativo basare le spiegazioni sul cluster VMware con il Metastore su un Oracle DBserver, più simile a quello che sarebbe un ambiente operativo.

Presupposto all'utilizzo di Hive, oltre al fatto che il cluster Hadoop debba essere attivo, è che sia anche attivo il Metastore, cioè il database dei metadati relativi alle tabelle i cui dati risiedono sul file system distribuito HDFS.

Quindi dalla console VMware dopo aver lanciato la macchina virtuale sulla quale è stato preventivamente installato l'Oracle DB server (versione gratuita), ed aver acceduto alla console di suddetta macchina, lanciamo lo script `/home/oracle/metastore.sh`

A questo punto abbiamo attivato l'ambiente idoneo per attivare Hive: il cluster Hadoop è attivo e così Metastore come dalle righe appena precedenti.

Infatti il seguente semplice comando Hadoop lanciato dalla console del nodo master del cluster permette di vedere i contenuti del file-system HFDS:

```
[hadoop@HAD0 ~]$ hadoop dfs -ls /
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 7 items
drwxr-xr-x   - hadoop supergroup          0 2015-10-11 15:34 /MeteoSVL
drwxr-xr-x   - hadoop supergroup          0 2015-09-02 17:15 /hdfs
drwxr-xr-x   - hadoop supergroup          0 2015-06-04 14:00 /input
drwxr-xr-x   - hadoop supergroup          0 2015-06-04 14:13 /output01
drwxr-xr-x   - hadoop supergroup          0 2015-09-02 13:15 /prova
drwx-w----   - hadoop supergroup          0 2015-10-11 15:43 /tmp
drwxrwxr-x   - hadoop supergroup          0 2015-07-02 23:49 /user
[hadoop@HAD0 ~]$
```

Possiamo ora spostarci sulla workstation di sviluppo e far partire gli script che lanciano Hive e poi beeline, cioè la sua interfaccia di comando:

Se ripetiamo l'interrogazione sull'HDFS attraverso Hive e la confrontiamo con la precedente riscontriamo che Hive 'punta' proprio al cluster sui server VMware.

Eseguiamo ora i comandi che si connettono alle tabelle in HDFS attraverso il client Hive e poi accedono al data-base meteosvl.

"meteosvl" contiene i dati prodotti da PIG che sono stati caricati in una tabella Hive.

Si rimanda alla documentazione di Hive per la descrizione e l'uso dei singoli comandi.

```
Beeline version 1.2.0 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/meteosvl;
Connecting to jdbc:hive2://localhost:10000/meteosvl;
Enter username for jdbc:hive2://localhost:10000/meteosvl;: hadoop
Enter password for jdbc:hive2://localhost:10000/meteosvl;: ***
Connected to: Apache Hive (version 1.2.0)
Driver: Hive JDBC (version 1.2.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000/meteosvl> describe database meteosvl;
+-----------+----------+-----------------------------------------------------+----
--------+-------------+-------------+--+
|  db_name  | comment  |                      location                       | own
er_name  | owner_type  | parameters  |
+-----------+----------+-----------------------------------------------------+----
--------+-------------+-------------+--+
| meteosvl  |          | hdfs://HAD0:9000/user/hive/warehouse/meteosvl.db  | had
oop      | USER        |             |
+-----------+----------+-----------------------------------------------------+----
--------+-------------+-------------+--+
1 row selected (2,56 seconds)
0: jdbc:hive2://localhost:10000/meteosvl>
0: jdbc:hive2://localhost:10000/meteosvl> show tables;
+-------------+--+
|  tab_name   |
+-------------+--+
| svimcnni    |
+-------------+--+
1 row selected (0,191 seconds)
```

5.4.3 L’accesso da Excel

Anche da uno strumento estremamente diffuso come Excel è possibile lanciare query ed estrarre dati da Hive attraverso il driver ODBC che abbiamo precedentemente installato.

Vediamo la sequenza delle operazioni eseguite.

Dal menù “dati” e poi “Da altre origini” si sceglie “Da Microsoft Query ODBC”.

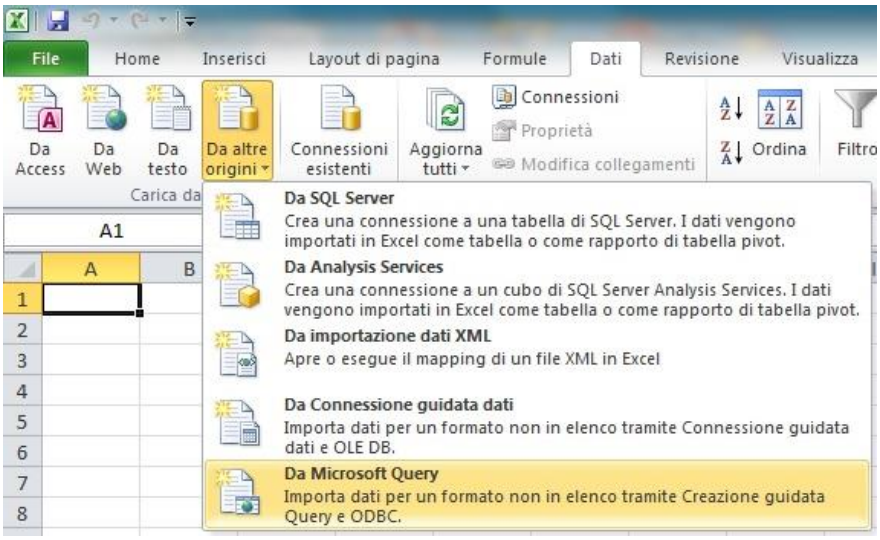


Figura 16. La connessione da Excel ad Hive

Quindi si seleziona la connessione “HiveVirtualBoxDevel” precedentemente creata:

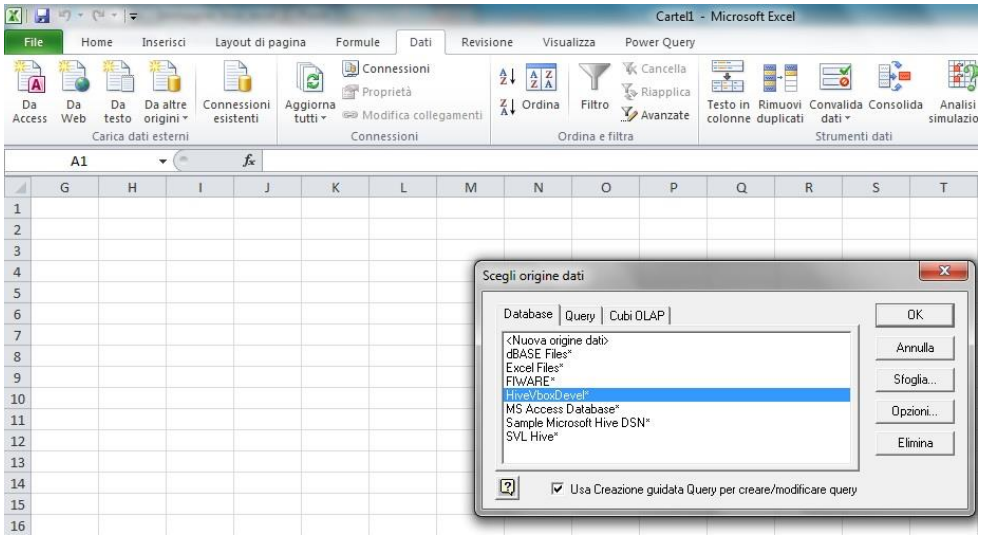


Figura 17. La connessione da Excel ad Hive

Proseguendo compaiono i dati della connessione ai quali aggiungiamo la password:

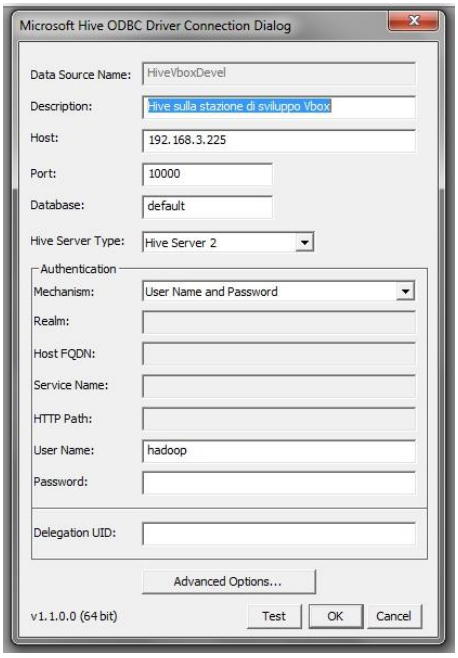


Figura 18. La configurazione del driver ODBC in Excel

Segue la scelta dei campi da importare: in questo caso li selezioniamo tutti:

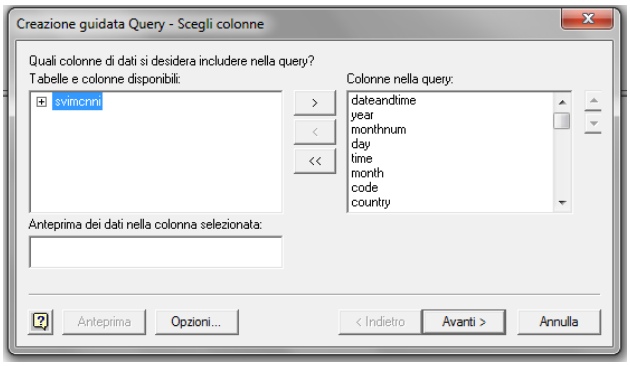


Figura 19. La query SQL da Excel ad Hive

Si ha quindi la possibilità di scegliere come ordinare i dati:

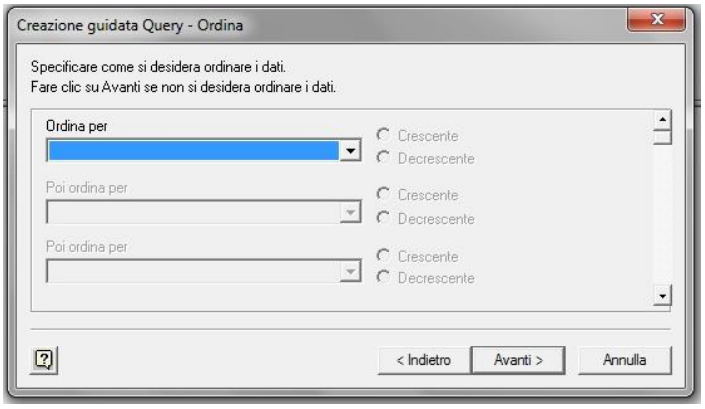


Figura 20. La query SQL da Excel ad Hive

E decidere a quale applicazione (i.e. Excel o Microsoft Query) restituire l'output:

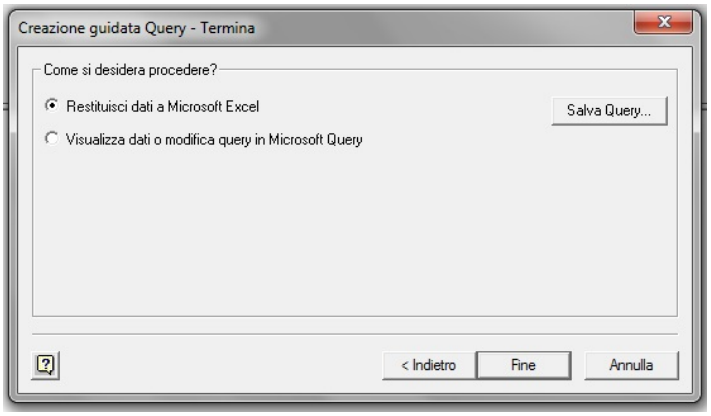


Figura 21. La query SQL da Excel ad Hive

In Figura 22 la schermata di invio della query di estrazione dei dati.

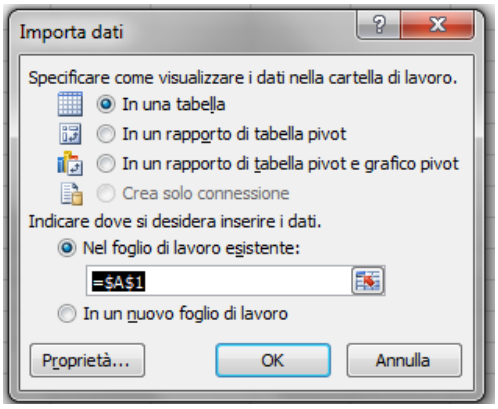
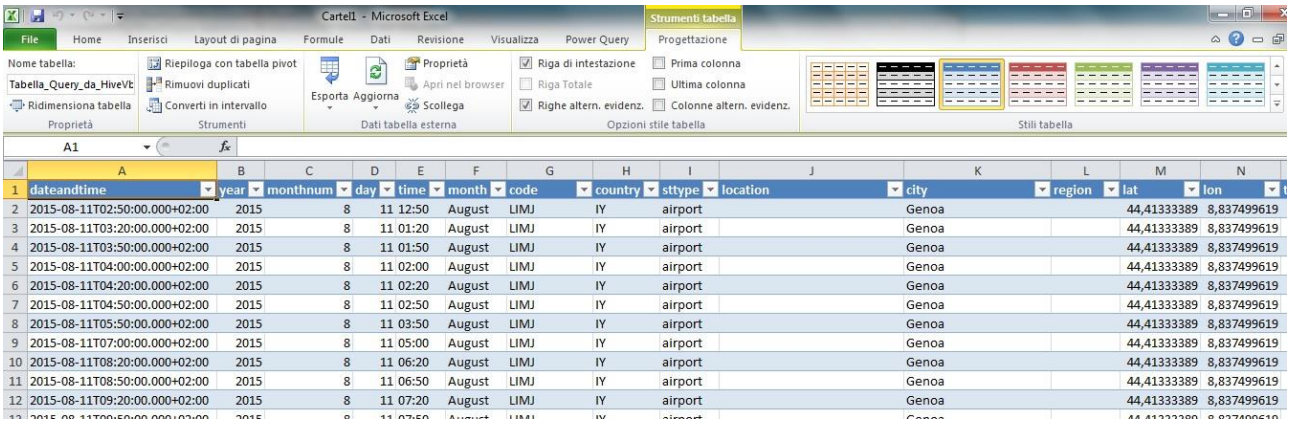


Figura 22. La query SQL da Excel ad Hive



dateandtime	year	monthnum	day	time	month	code	country	sttype	location	city	region	lat	lon
2015-08-11T02:50:00.000+02:00	2015	8	11	12:50	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T03:20:00.000+02:00	2015	8	11	01:20	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T03:50:00.000+02:00	2015	8	11	01:50	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T04:00:00.000+02:00	2015	8	11	02:00	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T04:20:00.000+02:00	2015	8	11	02:20	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T04:50:00.000+02:00	2015	8	11	02:50	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T05:50:00.000+02:00	2015	8	11	03:50	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T07:00:00.000+02:00	2015	8	11	05:00	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T08:20:00.000+02:00	2015	8	11	06:20	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T08:50:00.000+02:00	2015	8	11	06:50	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619
2015-08-11T09:20:00.000+02:00	2015	8	11	07:20	August	LIMJ	IY	airport		Genoa		44,41333389	8,837499619

Figura 23. I risultati della query in Excel

In Figura 23 sono visualizzati i risultati dell'estrazione ottenuti nella tabella Excel.

Non è oggetto di questo rapporto entrare nel merito dei contenuti né dei comandi SQL o di Excel, per i quali si rimanda ai rispettivi manuali, si è invece cercato di mettere in evidenza la possibilità di portare i concetti e i dati da un contesto Big Data, intrinsecamente abbastanza complesso al livello di uno strumento di produttività individuale di un utente finale.

Per concludere accenniamo alla definizione della connessione al database Hive da parte di Excel che abbiamo dato per già presente durante il precedente esempio. Per stabilire la connessione si utilizza il driver Microsoft per Hive. Mentre le connessioni sono state definite accedendo dal "Pannello di controllo" di Windows, agli "Strumenti di Amministrazione", quindi "Origine dati ODBC" per poi dare un nome alla connessione, definire l'host, la porta di accesso al server su cui è installato Hive, le modalità di connessione in sicurezza e con che utenza accedervi.

5.4.4 L'accesso da Oracle SQL Developer

Oracle SQL Developer è un'IDE (Integrated Development Environment) per il database Oracle, cioè un'interfaccia grafica che consente agli sviluppatori di operare velocemente sui propri dati.

Mentre con l'interfaccia Excel abbiamo avvicinato il mondo Big Data a degli utenti finali, vediamo ora come è possibile affermare lo stesso principio anche nei confronti degli abituali sviluppatori SQL lavorando con questo IDE estremamente diffuso e che da molto tempo supporta l'interfacciamento ad Hive.

Nell'immagine seguente mostriamo la definizione della connessione verso Hive per la quale è sufficiente indicare l'indirizzo del server su cui è presente Hive (la nostra worstation di sviluppo), la porta a cui risponde, l'utenza con cui si accede.

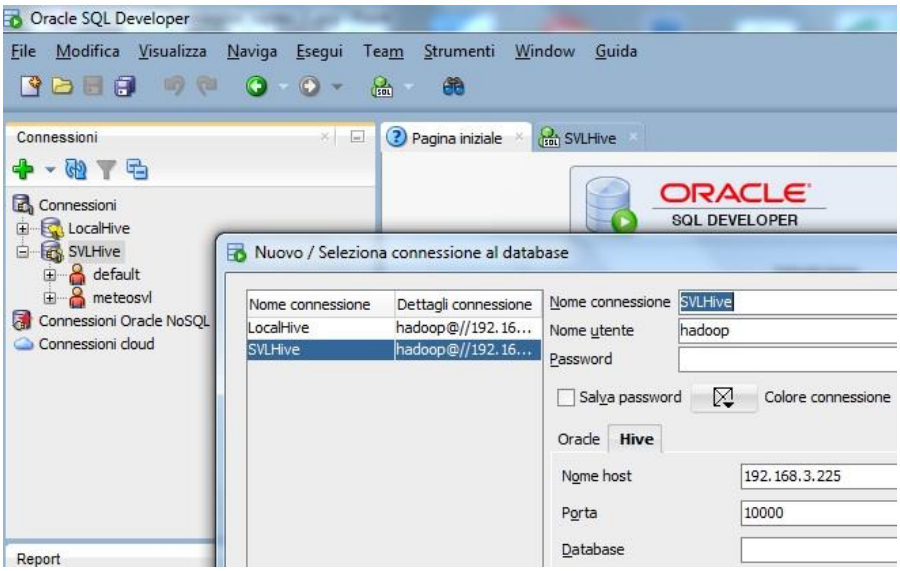


Figura 24. la connessione ad Hive da SQL Developer

Una volta indicata la password la connessione viene stabilita:

Figura 25. La query SQL da Oracle SQL Developer

Nell'immagine di Figura 26 a sinistra sono riportati il database e le sue tabelle accedute in Hive, attraverso i metadati contenuti sull'Hive Metastore che nel nostro caso è su un database Oracle, archiviate sul cluster Hadoop. Cliccando sul nome di tabella si vedono direttamente i nostri dati metereologici.

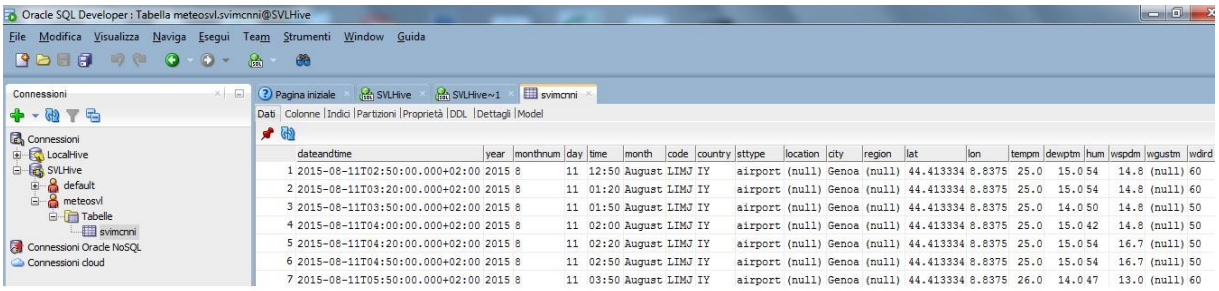


Figura 26. I risultati della query in Oracle SQL Developer

5.5 AWS

5.5.1 L'ambiente AWS - Elastic Map-Reduce

Amazon Web Services[9] è uno dei primi e più diffusi servizi cloud presenti sul mercato. Esso offre servizi di storage, calcolo e servizi applicativi gestiti. Della vasta offerta AWS abbiamo utilizzato il sistema di archiviazione S3 sul quale abbiamo appoggiato i nostri dati per la loro archiviazione permanente.

Del servizio EMR (Elastic Map-Reduce) abbiamo utilizzato il cluster Hadoop, Pig ed Hive. Tutti, nella configurazione prescelta, accedono direttamente ad S3, così abbiamo evitato di dover gestire in proprio server virtuali con un proprio HDFS. Non sono state tenute conto valutazioni di performance, ad esempio relativi ai tempi di elaborazione o ai tempi di accesso ai dati.

[9] https://aws.amazon.com/

In Figura 27 come appare la console di AWS per lo specifico account, a sinistra i servizi correntemente utilizzati a destra gli altri disponibili.

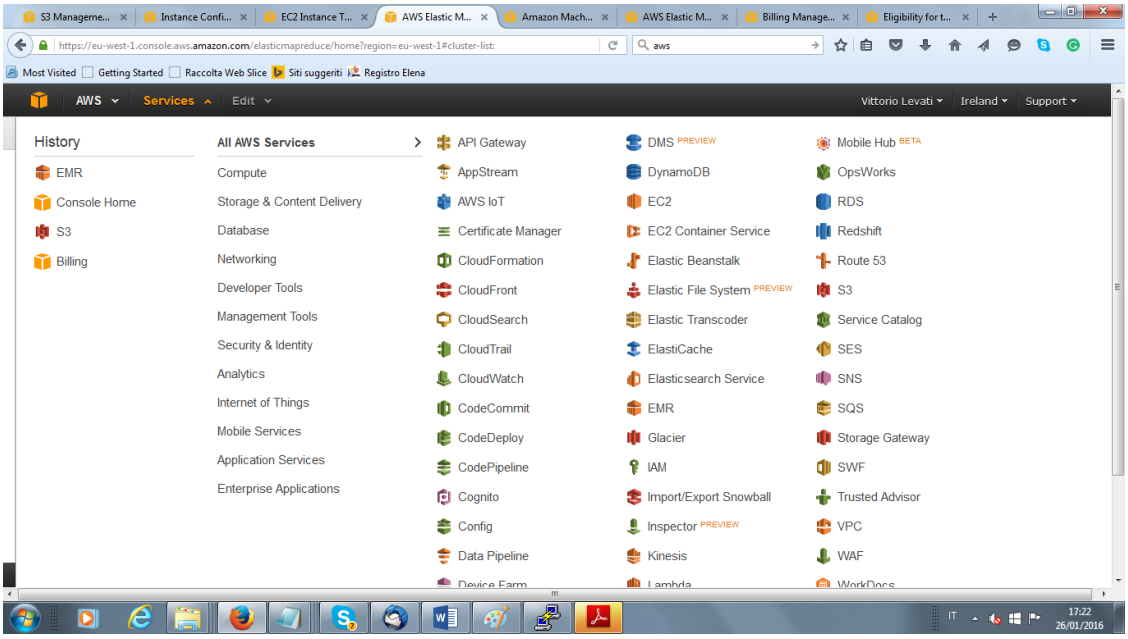


Figura 27. AWS Management Console

In Figura 28 mostriamo un paio di estratti delle videate con cui S3 presenta nel browser le directory e i files contenuti nel "bucket" assegnato. Il caricamento dei dataset avviene con un "upload" lanciato dall'interfaccia WEB AWS sfogliando i folder visti localmente dall'utente.

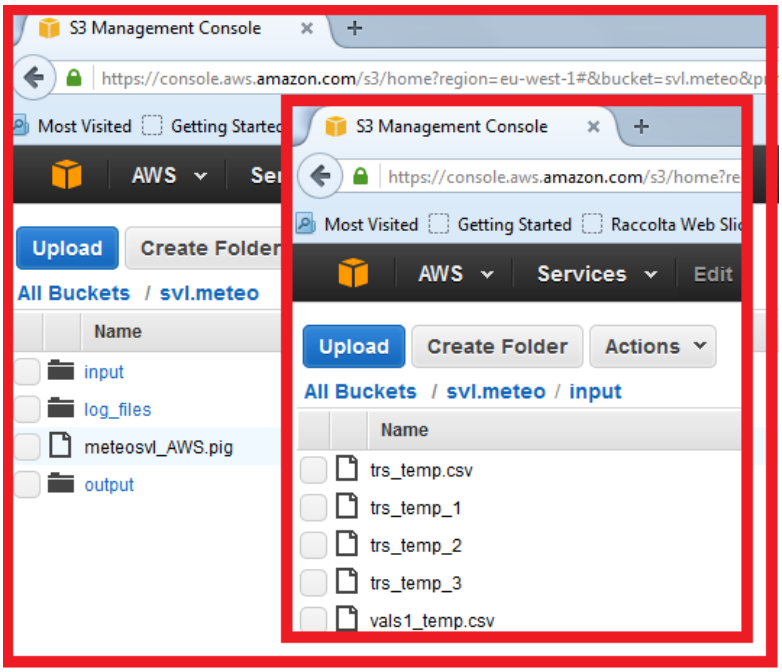


Figura 28. AWS S3

L’uso di EMR inizia con la creazione di un cluster Hadopp scegliendo fra diverse configurazioni di servizi desiderati.

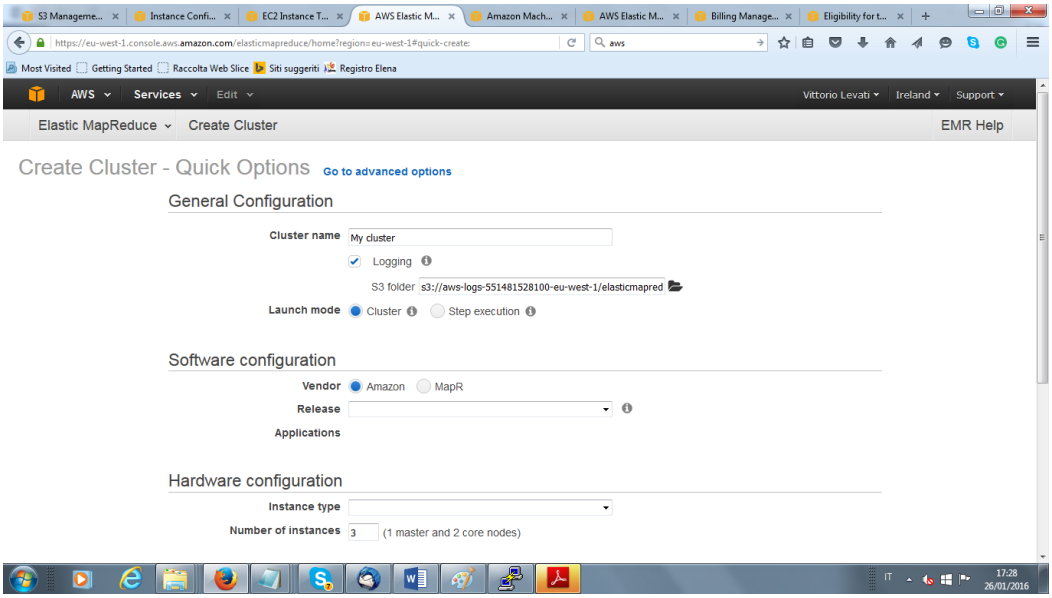


Figura 29. AWS cluster set-up

Si è scelto, come di seguito, la configurazione minima composta di un server master e due slave che possa offrire anche i servizi di Pig e Hive:

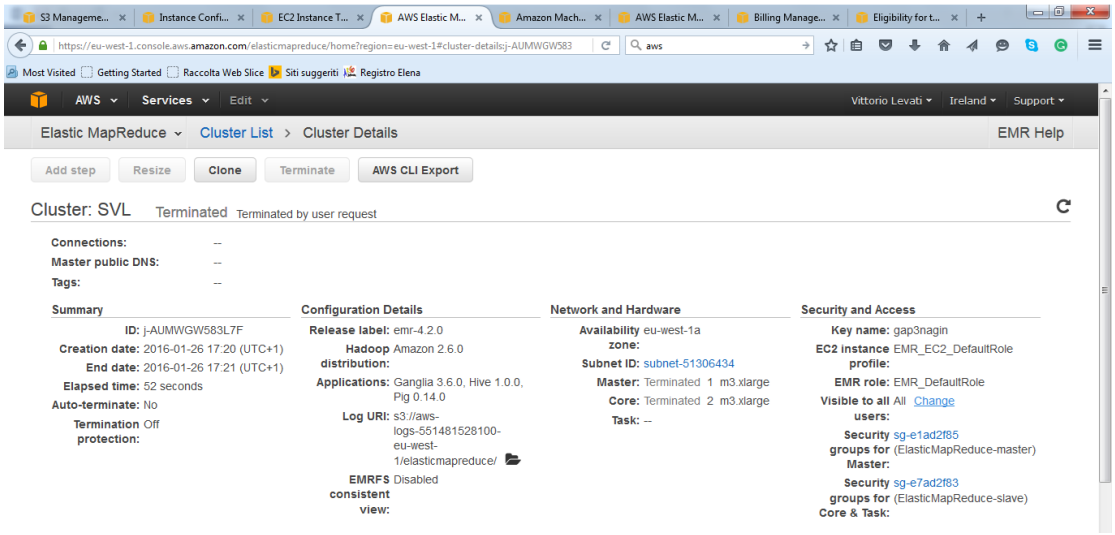


Figura 30. AWS cluster set-up

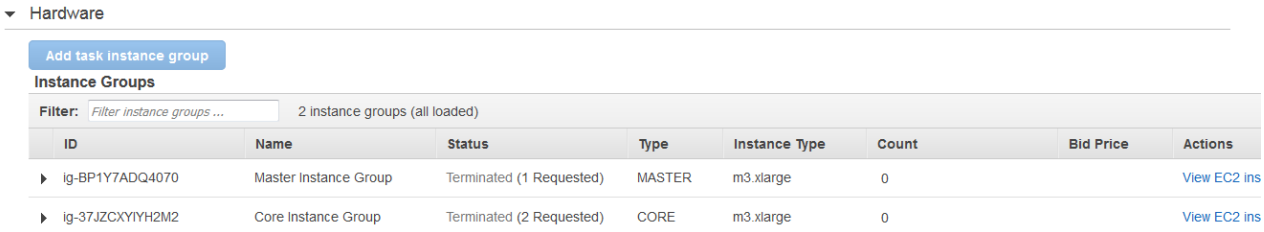


Figura 31. AWS Hadoop cluster nodes

Queste le caratteristiche dei server m3.large:

- SSD-based instance storage for fast I/O performance

m3.xlarge	4	15	2 x 40

Se il cluster è di nuova creazione per attivarlo si effettua un'operazione create se invece ne era già stato attivato uno in una precedente sessione si clona la precedente configurazione.

In Figura 32 la schermata di monitoraggio dello stato del processo.

Una volta che il cluster è avviato è possibile collegarsi al nodo master in ssh. Noi lo faremo con Putty dopo aver generato la chiave di autenticazione per la cui generazione si rimanda alle spiegazioni in linea di AWS.

Per stabilire la connessione indicare il "Master public DNS" del master server, come indicato sulla console del cluster e seguendo le istruzioni al link "SSH" in evidenza immediatamente alla sua destra o "Enable Web Connection":

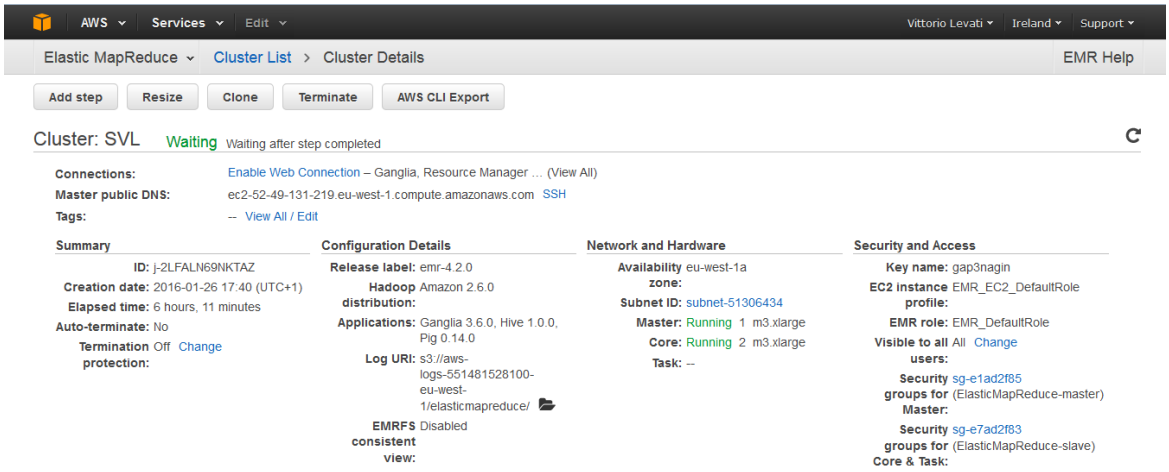


Figura 32. AWS cluster set-up

Nella finestra di Putty che si apre alla connessione ci identifichiamo con lo username hadoop:

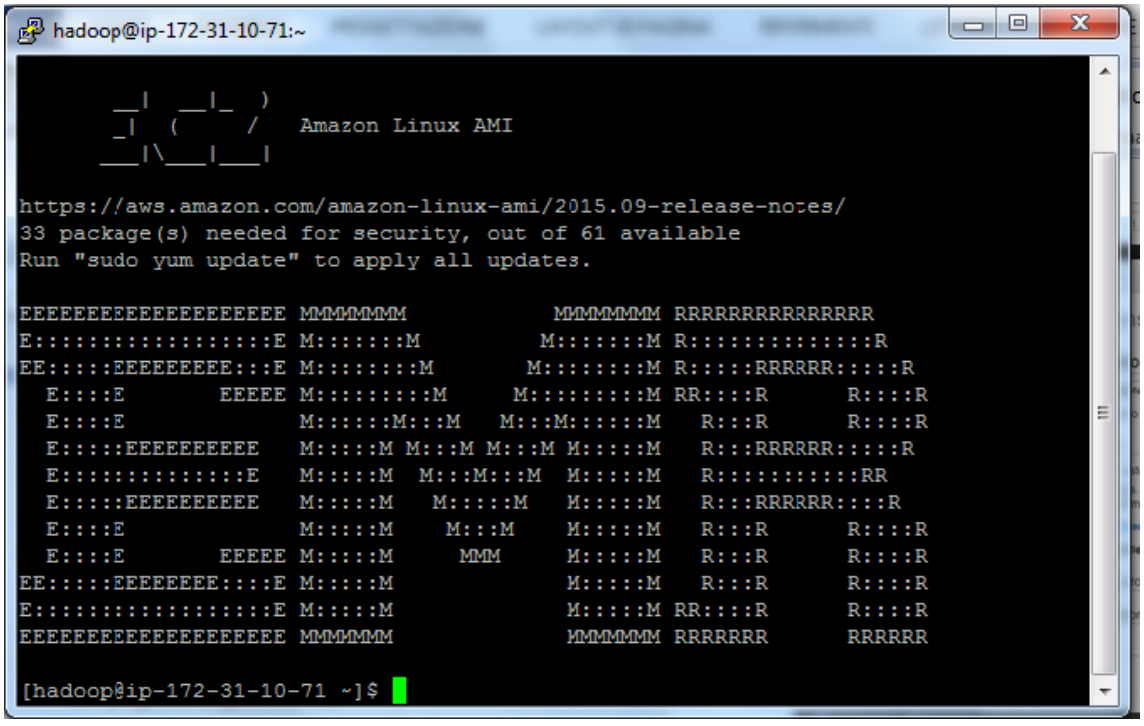


Figura 33. AWS master node

5.5.2 Pig in AWS

Come primo esempio di utilizzabilità applicativa possiamo lanciare PIG in maniera interattiva (prompt “grunt”), per eseguire i programmi che avevamo usato in precedenza sui cluster locali:

```
[hadoop@ip-172-31-10-71 ~]$ pig -x mapreduce
16/01/26 23:06:28 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
16/01/26 23:06:28 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
16/01/26 23:06:28 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
42   [main] INFO  org.apache.pig.Main  - Apache Pig version 0.14.0-amzn-0 (r: un
known) compiled Nov 16 2015, 21:51:33
16/01/26 23:06:28 INFO pig.Main: Apache Pig version 0.14.0-amzn-0 (r: unknown) c
ompiled Nov 16 2015, 21:51:33
43   [main] INFO  org.apache.pig.Main  - Logging error messages to: /mnt/var/log
/pig/pig_1453849588107.log
16/01/26 23:06:28 INFO pig.Main: Logging error messages to: /mnt/var/log/pig/pig
_1453849588107.log
63   [main] INFO  org.apache.pig.impl.util.Utils  - Default bootup file /home/ha
doop/.pigbootup not found
16/01/26 23:06:28 INFO util.Utils: Default bootup file /home/hadoop/.pigbootup n
ot found
16/01/26 23:06:28 INFO Configuration.deprecation: mapred.job.tracker is deprecat
ed. Instead, use mapreduce.jobtracker.address
16/01/26 23:06:28 INFO Configuration.deprecation: fs.default.name is deprecated.
 Instead, use fs.defaultFS
677  [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine
  - Connecting to hadoop file system at: hdfs://ip-172-31-10-71.eu-west-1.comput
e.internal:8020
16/01/26 23:06:28 INFO executionengine.HExecutionEngine: Connecting to hadoop fi
le system at: hdfs://ip-172-31-10-71.eu-west-1.compute.internal:8020
16/01/26 23:06:29 INFO Configuration.deprecation: fs.default.name is deprecated.
 Instead, use fs.defaultFS
grunt>
```

Figura 34. PIG in AWS

5.5.3 Hive in AWS

Dalla console del cluster acceduta con Putty come descritto nel paragrafo precedente abbiamo lanciato beeline e stabilito una connessione ad Hive utilizzando l’utenza e la password di AWS; infine si è chiesta la lista dei database:

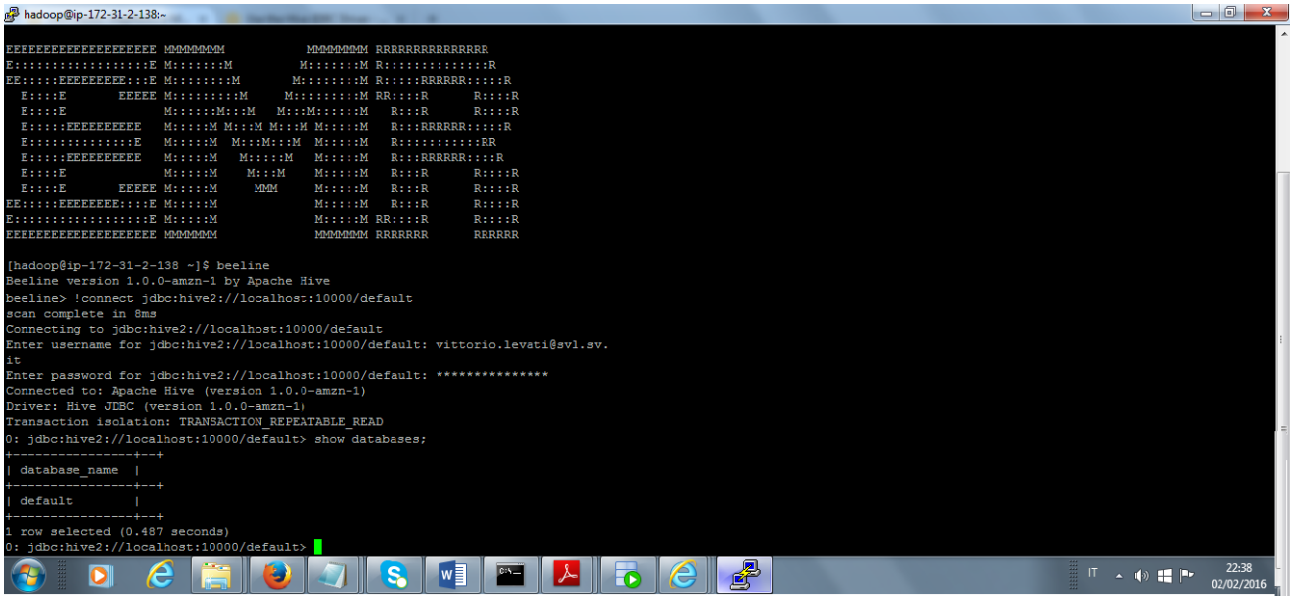


Figura 35. Hive in AWS

A questo punto è possibile eseguire i comandi Hive, come già effettuato nei cluster Hadoop locali facendo riferimento, quando necessario ai file in S3 anziche al filesystem HDFS dei filelocali.

È anche possibile connettersi da Excel ad AWS Hive accertandosi di aver aperto la porta 10000 sul nodo master dell'Hadoop cluster.

In Figura 36, la videata acceduta dalla console del cluster per gestire le regole di sicurezza.

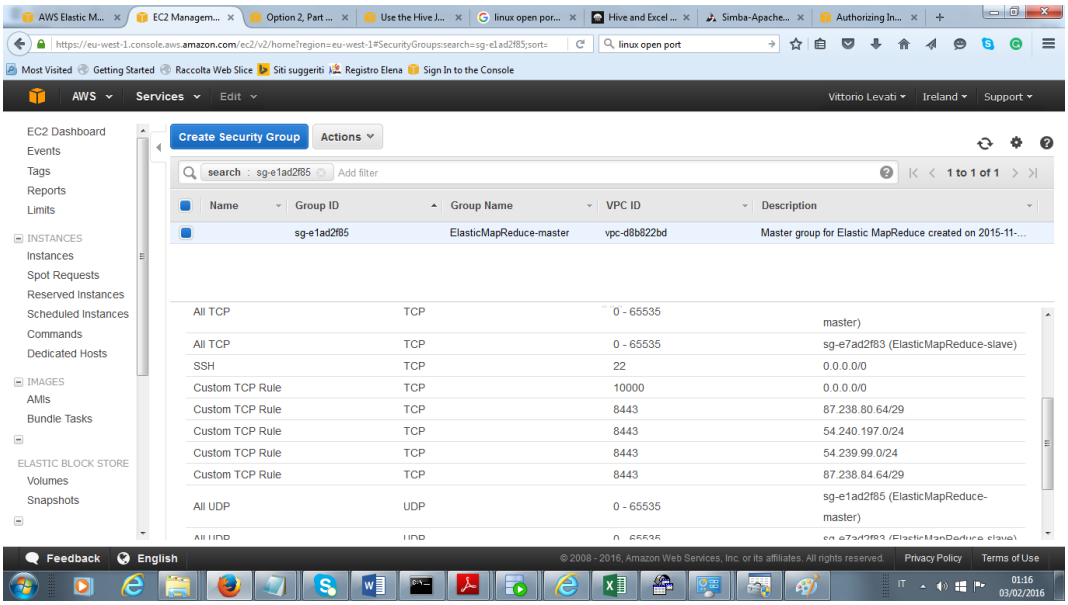


Figura 36. Il firewall del master node in AWS

Per completezza, in analogia agli accessi Hive fatti dai cluster non in cloud si è acceduto AWS Hive anche da Oracle SQL developer con la configurazione mostrata in Figura 37:

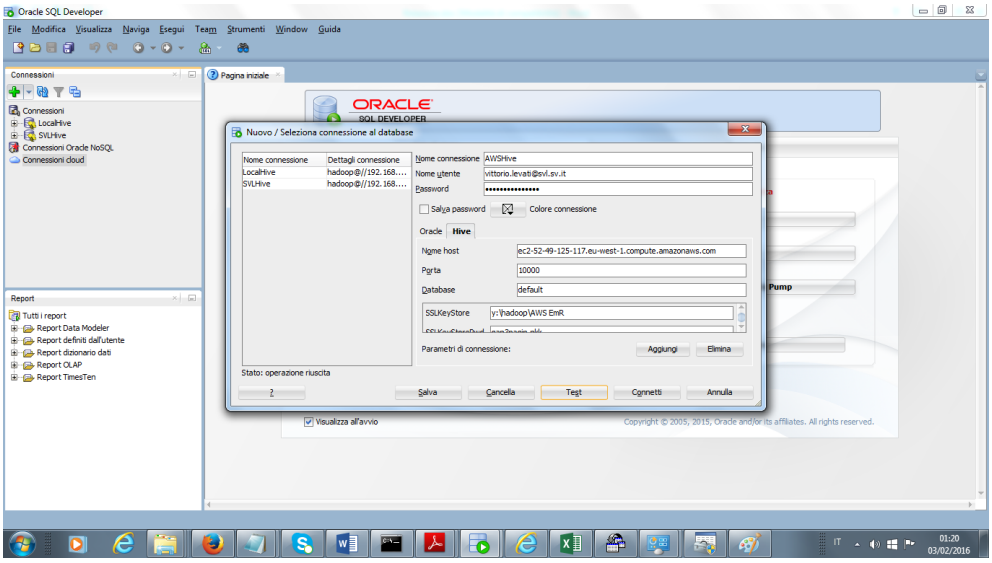


Figura 37. Configurazione connessione Oracle SQL Developer da AWS Hive

Concludendo si è mostrato come sia possibile eseguire le stesse applicazioni e sistemi di connessione di più alto livello sia in cloud che nell'ambiente presente sulla rete locale.

5.6 Machine Learning

5.6.1 Le motivazioni e le caratteristiche del sistema

In occasione del lancio dei servizi di Machine Learning offerti da AWS (AWS ML), abbiamo preso in considerazione proprio questa possibilità sia per verificare l'applicabilità degli algoritmi di machine learning alla nostra applicazione, sia per provare il nuovo approccio di AWS che mira a semplificare l'uso di questi complessi algoritmi.

Attraverso un processo assistito l'utente viene guidato in un percorso che consente al sistema di individuare ed applicare gli algoritmi migliori rispetto al set di dati da analizzare. Nel complesso il sistema è risultato abbastanza duttile ed immediato, soprattutto rispetto alle soluzioni come Mahout, per rimanere in ambito Hadoop Open Sourse, dove gli utenti hanno un controllo assai maggiore sull'utilizzo degli algoritmi, ma devono possedere competenze più specializzate.

5.6.2 Gli aspetti applicativi

Fra la cinquantina di stazioni monitorate è stata scelta per l'esemplificazione su machine learning, la città di Savona, ipotizzando la perdita delle misurazioni di una giornata, che abbiamo pensato di ricostruire secondo un modello di regressione basato sui dati storici. Quindi si è proceduto secondo i seguenti passi ed assunzioni:

- solo le stazioni attorno a Savona portano informazione utile all'elaborazione del modello (scelte 6 stazioni: Varazze, Altare, Albisola, Loano, Sassello, Quiliano);
- è stata scelta la temperatura come variabile persa e target da stimare;
- si sono raggruppate le osservazioni complete (con tutti i dati del giorno), pulite e formattate in CSV;
- il data set (file CVS) è stato ripartito in due: un grosso training set e un piccolo validation set;
- si è fatto elaborare il modello al sistema con il training set e poi è stato valutato con il validation set;
- infine si è ripetuto questo processo diverse volte, con aggiustamenti successivi, fino ad una situazione stabile.

In pratica all'inizio sono stati presi in considerazione tutti i dati: coordinate geografiche, ora del giorno, tutti i parametri metereologici ecc. Poi si è cercato di ottimizzare il modello in base alle indicazioni ricevute dal sistema come la correlazione delle variabili sul target, il grafico della distribuzione dei residui e i parametri di regolarizzazione dei dati

Le operazioni di selezione dei dati utili, fino a pervenire ad una situazione ritenuta ottimale sono state fatte con l'ausilio di programmi PIG. In particolare:

- sono stati raggruppati i tempi in cui sono state fatte le osservazioni, diversi da stazione a stazione, per quarti d'ora e poi si è assunto come valore la media dei valori registrati per una certa stazione in quel quarto d'ora. Questo perché nei dati originali gli orari fra le stazioni erano sfalsati e rilevati con frequenze diverse;
- è stata ri-espressa l'ora di osservazione da hh:mm (considerate etichette a carattere) in hh,centesimi di ora;
- è stata ri-espressa la direzione del vento da punti cardinali (es: NNE) a gradi, ponendo particolare attenzione al calcolo delle medie nelle direzioni vicine al Nord (contiguità fra angolazioni vicine a 0° e vicine a 360°);
- sono state ridotte le variabili metereologiche rilevate, fino a comprendere che solo le temperature circostanti influenzano il calcolo della temperatura di Savona da stimare. La direzione vento, l'umidità, la pressione introducevano solo molto rumore che peggiorava i risultati;
- sono infine state eliminate le coordinate geografiche, le date e gli orari fino a lasciare solo temperature;
- infine si è validato il modello con le osservazioni di due giornate di stagioni opposte maggio, novembre.

La tabella seguente è un esempio di come si presentavano alcune righe dei dati del nostro data set usati per il modello finale.

```
linename,tempm1,tempm2,tempm3,tempm4,tempm5,tempm6,tempm7
2015-11-25_10:30,7.3,9.8,3.6,7.2,9.8,1.0,7.7
2015-11-25_10:45,7.2,9.4,3.8,7.3,9.8,1.0,7.8
2015-11-25_11:00,7.2,9.5,4.0,7.4,9.8,1.4,7.6
2015-11-25_11:15,7.2,9.5,4.3,7.4,9.8,2.2,7.8
2015-11-25_11:30,7.2,9.8,4.4,7.6,9.8,2.4,8.0
```

5.6.3 Come si presenta AWS ML

Figura 38 riprende la Dashboard AWS ML con l'elenco di modelli elaborati, i datasource dei training set e delle valutazioni:

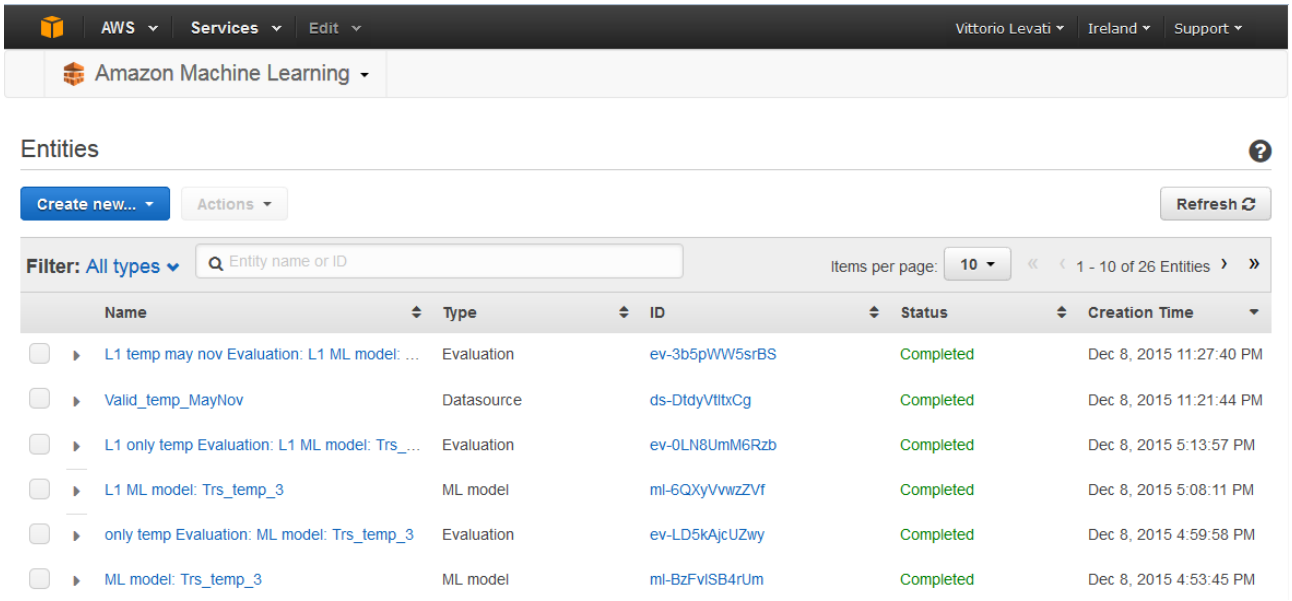


Figura 38. La dashboard di AWS ML

Figura 39 riporta invece le variabili, features nel lessico AWS ML, e il grado di correlazione che hanno con il target.

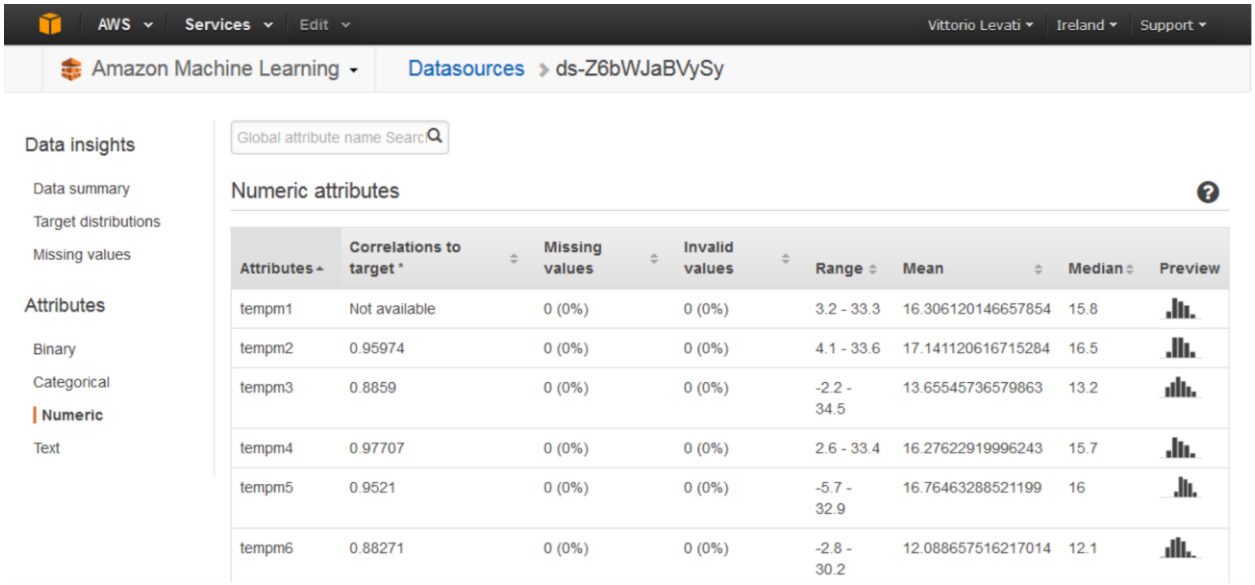


Figura 39. Le features del modello in AWS ML

La parte significativa di Figura 40 è l'istogramma, che si è dimostrato essere lo strumento più utile con il quale il sistema comunica all'utente la bontà o meno del modello. Quanto più la campana è centrata meglio è.

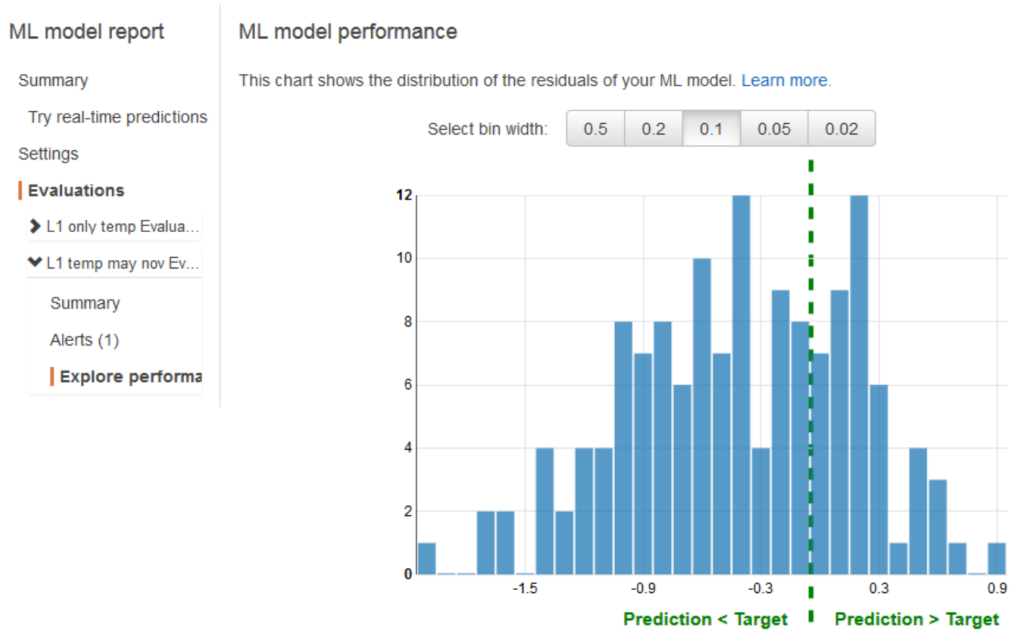


Figura 40. L'istogramma delle performance in AWS ML

Ricordiamo che il modello ha riguardato la stima delle temperature di tutta una giornata (quarto d'ora per quarto d'ora) di una località sulla base delle temperature nelle località circostanti, e che come validation set si sono prese due giornate di due stagioni opposte.

Il grafico, sulla base del confronto fra i dati veri del validation set e i dati stimati dal modello, ci dice sul lato sinistro, il numero di osservazioni (ordinata) vere che erano più basse di 1,5 0,9 0,3 C°, (ascisse) del dato stimato; a destra il modello ha stimato temperature più fredde di meno di 0,3 0,9 C°.

5.6.4 Conclusioni sul test di machine learning.

Amazon Machine learning semplifica molto la teoria del ML, rendendola effettivamente fruibile da un maggior numero di persone e in un grande numero di ambiti applicativi.

Il test si è concentrato soprattutto sulla possibilità di utilizzare queste tecniche e questa metodologia semplificata a completamento del ciclo di raccolta elaborazione ed analisi di dati in un contesto e con tecnologie Big Data, ed effettivamente si è dimostrato affrontabile ed utilizzabile.

Per quanto concerne la specificità commerciale di AWS ML, il costo del servizio è fondamentalmente basato sul numero di predizioni, quindi il rapporto/beneficio è fortemente dipendente e variabile da applicazione ad applicazione. Nel nostro caso i costi sono stati irrisori (pochi euro), sia perché, per quanto riguarda i servizi di supporto ad ML come S3 si è usufruito del periodo gratuito di prova, sia perché, essendo lo scopo sperimentale e non operativo le elaborazioni sono state molto poche. Per chiarire meglio l'aspetto costi, diverso sarebbe se in un ambiente operativo si dovessero stimare giornalmente molte migliaia di valori.

Si ritiene inoltre utile sottolineare anche l'aspetto didattico e di prototipazione di questo approccio facilitato, dopo il quale si può eventualmente passare a strumenti di machine learning più complessi o che necessitino anche di attività di programmazione. In tal senso si può pensare di testare inizialmente una problematica applicativa semplificata con AWS ML e poi progettare un ambiente operativo ad hoc.

AWS ML è l'apripista di questi servizi in cloud, ma quanto appena detto potrebbe applicarsi anche ad altri analoghi servizi di altre aziende che verranno sempre più rilasciati sul mercato. In ciò si è cercato di identificare non un servizio specifico ma una categoria di prodotti/servizi che agevoleranno certamente un maggior diffusione di queste tecnologie.

6 Discussione e note conclusive

Anche se buona parte della sperimentazione presentata nelle sezioni precedenti è avvenuta su hardware locale, la filiera di strumenti per il trattamento dei Big Data che abbiamo sperimentato ha come naturale evoluzione le piattaforme Cloud. Nel corso del presente progetto si è quindi indagato l'offerta di servizi Cloud dedicati ai Big Data in questo momento di mercato. Va comunque tenuto conto che il settore è in pieno fermento e non passa giorno senza che si incrementi l'offerta, che mutino servizi e prezzi o addirittura si vedano nuovi attori entrare o uscire da questo mercato.

Nella nostra analisi sono stati presi in considerazione, pur senza entrare nei dettagli delle singole proposte commerciali diverse tipologie di servizi.

La tabella seguente è un file di lavoro utilizzato nel confronto, la cui obsolescenza è stata però quotidiana. Solo per citare alcuni fatti recenti e successivi alla raccolta di queste informazioni ricordiamo l'ingresso tardivo ma deciso nel mercato cloud di Oracle e il disimpegno da parte di HP, piuttosto che il nascere e il morire di iniziative locali e minori di cui non è praticamente possibile tenere traccia.

name	countr	distri	O.S.	descriprion	free user	web link
Amazon Elastic MapReduce (Amazon EMR)	US, EU, FE	MAPR	Linux	Amazon EMR provides a managed Hadoop framework to distribute and process vast amounts data across dynamically scalable Amazon EC2 (Elastic Compute Cloud) instances. You can also run other distributed frameworks such as Spark and Presto in Amazon EMR, and interact with data in other AWS data stores such as Amazon S3 (Simple Storage Service) and Amazon DynamoDB.Amazon EMR handles such big data use cases as log analysis, web indexing, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics.	AWS free tier: 750hrs of Micro DB Instance each month for one year, 20GB of Storage, and 20GB for Backups with Amazon Relational Database Service (RDS). Per EMR c'è gratis solo il loro test del getting started e lo spazio S3 utilizzato per salvare log e output; comunque ho chiesto conferma ed attendo risposta.	http://aws
Microsoft Azure HDinsight	US, EU, FE	Hortonworks	Win, Linux	Microsoft's Hadoop cloud service scales to petabytes on demand; processes unstructured and semi-structured data; deploys on Windows or Linux; integrates with on-premises Hadoop clusters (if needed); and supports multiple development languages including Java and .Net, Microsoft says.	Free one-month trial with €170 to spend on all Azure services	http://azu
IBM BigInsights on Cloud				IBM BigInsights on Cloud provides Hadoop-as-a-service on IBM's SoftLayer global cloud infrastructure – a bare metal design. The service requires no on-premises infrastructure; and it supports Big SQL, Big Sheets, text analytics and more, IBM asserts		http://ww
HP Cloud with Hadoop				HP Cloud provides an elastic cloud computing and cloud storage platform to analyze and index large data volumes in the hundreds of petabytes in size, HP asserts. Distributed queries run across multiple data sets and are then returned in near real time, the company says. HP Helion Public Cloud provides the underlying infrastructure required to process big data. The company partners with third-party solution providers that enable enterprises to better configure, manage, manipulate, and analyze data affordably.		HP Cloud p
Verizon Cloud				Verizon's Enterprise business inked a Cloudera partnership in 2013, and the IT services giant now offers Cloudera atop its cloud infrastructure. A Cloudera distribution supporting billions of records can be deployed on the Verizon Cloud in a matter of hours, significantly faster than deployments on generic public clouds, Verizon claims.		
Altiscale				Altiscale has developed a purpose-built, petabyte-scale infrastructure that delivers Apache Hadoop as a cloud service. An operational support team monitors jobs and system tuning for customers. Instead of charging by the node, Altiscale charges by monthly usage.		https://w

Ci limitiamo quindi a valutazioni qualitative sulla base delle esperienze fatte, cercando di raggrupparle per caratteristiche omogenee.

L'offerta di servizi cloud dedicati ai Big Data vede grandi protagonisti come AMAZON, antesignana dell'offerta di servizi cloud in genere, e i grandi nomi dell'informatica come Microsoft e IBM.

Tutti questi hanno in portafoglio un'ampia gamma di soluzioni. Quasi certamente Amazon possiede il portafoglio più esteso e diversificato che va dai servizi totalmente gestiti di prodotti Open Source come AWS EMR, da noi utilizzato, ai servizi gestiti di distribuzioni specifiche come MapR, ma si sta via via arricchendo di sue proprie soluzioni come AWS ML. A ciò si aggiunge la possibilità di utilizzare servizi infrastrutturali (IaaS) sui cui ogni cliente può installare e mantenere a sua cura cluster e prodotti applicativi.

IBM e Microsoft si presentano in maniera analoga ma caratterizzata dalla loro storia, dai loro prodotti, dai propri ambienti software, dalla tipologia dei grandi clienti che servono e dalle partnership con i fornitori di distribuzioni specifiche (es: Microsoft-Hortonworks, Oracle-Cloudera ecc.)

Esiste poi una categoria di fornitori meno noti come ad esempio Altiscale, che potremmo definire "boutique" che si presentano come servizi di alta qualità e personalizzazione.

Le offerte cloud dei piccoli fornitori locali, spesso legati alla vicinanza geografica e al rapporto diretto con i propri clienti si limitano in genere ad un'offerta infrastrutturale e applicativa standard, senza offrire specifici servizi Big Data.

Riassumendo le opzioni per chi intenda usufruire di servizi orientati ai Big Data in cloud sono quindi o di orientarsi in base al fornitore di servizi:

1) I grandi nomi dell'informatica, in una gradazione di soluzioni da quelle più economiche e generiche (es: Amazon e Microsoft) a quelle più specifiche, tecnologicamente sofisticate e personalizzate (es: Amazon, Microsoft, IBM, Oracle…).
2) Le aziende "boutique", quasi tutte all'estero e poco note in Italia (Altiscale)

Oppure orientarsi in funzione della distribuzione o della tecnologia prescelta, che potrebbe essere selezionata sulla base di esperienze on premise o in un cloud infrastrutturale, con cluster Hadoop implementati in proprio. Queste le distribuzioni più note: Cloudera, Hortonworks, MapR, Pivotal, Apache nativo. Infine selezionare il fornitore di servizi in funzione della distribuzione desiderata.

Dal punto di vista economico i fornitori di servizi hanno politiche e formulazione dei listini molto diverse fra di loro e in continua evoluzione e competizione, per cui ogni analisi avulsa dallo specifico momento e dalle specificità delle esigenze funzionali ed applicative, è di fatto un esercizio sterile e potenzialmente deviante.

Possiamo però dire che tutti i tipici benefici del cloud: nessun investimento iniziale, granularità delle risorse utilizzate e meglio dimensionate sulle reali esigenze, scalabilità, sono stati confermati. Inoltre, la natura stessa dei Big Data esige la disponibilità di risorse hardware di cui quasi sempre i fruitori non dispongono in proprio, quindi è frequente, dopo un'iniziale sperimentazione, se non fin dall'inizio, il ricorso ai servizi cloud.

Fra i costi collaterali all'utilizzo di applicazioni Big Data in cloud vanno certamente considerate le problematiche di networking, anche se sempre più spesso si confida, specie all'estero, di un costo costantemente decrescente delle telecomunicazioni a fronte di un incremento delle prestazioni.

L'idea di elaborare in siti remoti grandi quantità di dati suscita spesso il timore che il trasporto degli stessi rappresenti un collo di bottiglia in termini di prestazioni o per altro verso di costi.

In verità vi è quasi sempre la possibilità di effettuare 'pesanti' caricamenti di dati iniziali, come nel caso di dati storici, trasportando dei media fisici, sempre che si sia veramente in presenza di grandissimi quantitativi. Ma quasi sempre i dati nascono e crescono giorno per giorno e assai spesso sono già remoti rispetto ad un centro di raccolta, magari il centro stella di una rete aziendale. Non cambia quindi molto se questi dati vengono direttamente concentrati in cloud nelle quantità giornaliere di produzione. Il networking non rappresenta quindi, a nostro avviso, un reale problema o un problema tale da intimorire chi desideri avvicinarsi alle applicazioni Big Data.

Infine, sulla base delle precedenti considerazioni riteniamo comunque utile una iniziale sperimentazione che consenta la formulazione di una consapevole definizione dei requisiti prima di effettuare una scelta di tecnologia, distribuzione e fornitore, altrimenti difficilmente e costosamente reversibile.

Riferimenti

[1] A. Quarati, A. Clematis, G. Paschina, A. Parodi and T. Bedrina, Lightweight ICT Approaches to Hydro-Meteorological Data Issues, 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, 2015, pp. 759-763, (2015).

[2] T. Bedrina, A. Parodi, A. Clematis, A. Quarati, Mashing-Up Weather Networks Data to Support Hydro-Meteorological Research. In F. Cipolla-Ficarra (Ed.), Advanced Research and Trends in New Technologies, Software, Human-Computer Interaction, and Communicability (pp. 245-254). Hershey, PA: IGI Global. doi:10.4018/978-1-4666-4490-8.ch023, (2014).

[3] Deliverable del progetto TCUBE, D2.1 Specifica dei requisiti funzionali e non-funzionali della piattaforma, (2015).

[4] D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, 6 Feb 2001, https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf, (2001) .

[5] J. Lerner and J. Tirole, The Scope of Open Source Licensing. J Law Econ Organ 2005; 21 (1): 20-56. doi: 10.1093/jleo/ewi002, (2005).

[6] M. Bengtsson and S. Kock, Coopetition—Quo vadis? Past accomplishments and future challenges, Industrial Marketing Management, Volume 43, Issue 2, February 2014, Pages 180-188, ISSN 0019-8501, (2014).

[7] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113. DOI=http://dx.doi.org/10.1145/1327452.1327492, (2008).

[8] T. Bedrina, A. Parodi, A. Quarati and A. Clematis, ICT approaches to integrating institutional and non-institutional data services for better understanding of hydro-meteorological, Nat. Hazards Earth Syst. Sci., 12, pp. 1961–1968, (2012).

[9] Project Open Data, Metadata Schema v1.1, https://project-open-data.cio.gov/v1.1/schema/

[10] Apache PIG Documentation - http://pig.apache.org/docs/r0.14.0/

[11] Apache Hive Getting Started - https://cwiki.apache.org/confluence/display/Hive/GettingStarted

APPENDICI

6.1 WScircle.java

Programma WScircle.java per la ricerca ed estrazione delle stazioni attorno alle stazioni di partenza per poter coprire al meglio il territorio

```
package SearchWS;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Scanner;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/* A partire da una lista di città contenute nel file citySearch.txt ha estratto to le stazioni di tipo eroporto e PWS in un certo
 * raggio da ciascuna città dell'elenco. Vien poi creata una lista di tutte le stazioni trovate, questa verrà ripulita a mano
 * fino a creare un file di imput per il programma che leggere le osservazioni
 *
 */
public class WScircle {
            public static void main(String[] args) throws FileNotFoundException,
   ParserConfigurationException, SAXException,IOException, InterruptedException
 {
                        String [] st_value = new String[20];
                        String [] citySearch= new String [12];
                        String country;
                        country="Italy";
      int count=0;
      int temp=0;
      int temp2=0;
      int nairport=0;

           for (int i=0; i<19;i++)
        st_value[i]=null;
      for (int i=0; i<12;i++)
        citySearch[i]=null;
      File file = new File("C:/Meteo/citySearch.txt");
      Scanner input = new Scanner(file);
      PrintWriter output = new PrintWriter
           (new BufferedWriter(new
      FileWriter("c:/Meteo/station_from.txt",true)));
      while(input.hasNext())
      {
        citySearch[count] = input.nextLine();
        count++;
      }
      input.close();
      for (int i=0; i<count;i++)
        {
          try
           {
              country = "Italy";
              if (citySearch[i].replaceAll("\\s+","").equals("Nice"))
                        {
                        country = "France";
```

```
                    }
                      URL                    googleWeatherXml                    =                    new
URL("http://api.wunderground.com/api/980cd527b87d38cd/geolookup/q/"+country+"/"+citySearch[i]+".xml");
                URLConnection uc=googleWeatherXml.openConnection();
                uc.connect();
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document doc = db.parse(uc.getInputStream());
                 doc.getDocumentElement().normalize();
                 NodeList nList = doc.getElementsByTagName("airport");
                 nairport = nList.getLength();
                 for (temp = 0; temp < nairport; temp++)
                 {
                    NodeList stlist = nList.item(temp).getChildNodes();
                    for (temp2 = 0; temp2 < stlist.getLength();
                          temp2++)
                    {
                          Node stNode = stlist.item(temp2);
                          if (stNode.getNodeType() == Node.ELEMENT_NODE)
                          {
                                    Element eElement = (Element) stNode;
                                    st_value[2]=eElement.getElementsByTagName("city").item(0).getTextContent().toString();
                                    if (!st_value[2].equals(""))
                                    {
                          st_value[0]="null";      // neighborhood
                          st_value[1]="/";
                          st_value[2]=eElement.getElementsByTagName("city").item(0)
                                              .getTextContent().toString();
                          st_value[3]="/";
                          st_value[4]="airport";   //tipo airport-pws
                          st_value[5]="/";
                          st_value[6]=eElement.getElementsByTagName("state").item(0).getTextContent().toString();
                          if (st_value[6] == "") st_value[6]="null";
                          st_value[7]="/";
                          st_value[8]=eElement.getElementsByTagName("country").item(0).getTextContent().toString();
                          st_value[9]="/";
                          st_value[10]=eElement.getElementsByTagName("icao").item(0).getTextContent().toString();
                          st_value[11]="/";
                          st_value[12]=eElement.getElementsByTagName("lat").item(0).getTextContent().toString();
                          st_value[13]="/";
                          st_value[14]=eElement.getElementsByTagName("lon").item(0).getTextContent().toString();
                          st_value[15]="/";
                          st_value[16]=citySearch[i];    // città da cui si è cercato
                          st_value[17]="/";
                          st_value[18]="null";  // distance km.
                        /*   try
                          { */
                             for (int j=0; j<19; j++)
                                output.print(st_value[j]);
                                    output.println();
                             }  //chiude l'if che esclude se  la città non è presente
                          /* } */
                           /*  catch (IOException e)
                              {
                              System.out.println("Error: " + e);
                              System.exit(1);
                           }  */
                                  }    //chiude if elemento valido
                    }   // chiude lista figli
                 }       //loop aiport
                NodeList pwsList = doc.getElementsByTagName("pws");
                for ( temp = 0; temp < pwsList.getLength(); temp++)
                {
                    Node pwsNode = pwsList.item(temp);
                    NodeList stList = pwsList.item(temp).getChildNodes();
                    for ( temp2 = 0; temp2 < stList.getLength();
                          temp2++)
                    {
                          Node stNode = stList.item(temp2);
                          if (stNode.getNodeType() == Node.ELEMENT_NODE)
                          {
                                    Element eElement = (Element) stNode;
                                    st_value[0]=eElement.getElementsByTagName("neighborhood").item(0)
```

```
                        .getTextContent().toString();
                            if (st_value[0].replaceAll("\\s+","").equals("")) st_value[0]="null";
                            st_value[1]="/";
                            st_value[2]=eElement.getElementsByTagName("city").item(0)
                    .getTextContent().toString();
                            if (st_value[2].equals("")) st_value[2]="null";
                            st_value[3]="/";
                            st_value[4]="pws";   //tipo airport-pws
                            if (st_value[4] .equals ("")) st_value[4]="null";
                            st_value[5]="/";
                            st_value[6]=eElement.getElementsByTagName("state").item(0).getTextContent().toString();
                            if (st_value[6] .equals ("")) st_value[6]="null";
                            st_value[7]="/";
                            st_value[8]=eElement.getElementsByTagName("country").item(0).getTextContent().toString();
                            if (st_value[8] .equals ("")) st_value[8]="null";
                            st_value[9]="/";
                            st_value[10]=eElement.getElementsByTagName("id").item(0).getTextContent().toString();
                            if (st_value[10] .equals ("")) st_value[10]="null";
                            st_value[11]="/";
                            st_value[12]=eElement.getElementsByTagName("lat").item(0).getTextContent().toString();
                            if (st_value[12] .equals ("")) st_value[12]="null";
                            st_value[13]="/";
                            st_value[14]=eElement.getElementsByTagName("lon").item(0).getTextContent().toString();
                            if (st_value[14] .equals ("")) st_value[14]="null";
                            st_value[15]="/";
                            st_value[16]=citySearch[i];    // città da cui si è cercato
                            if (st_value[16] .equals ("")) st_value[16]="null";
                            st_value[17]="/";
                            st_value[18]=eElement.getElementsByTagName("distance_km").item(0).getTextContent().toString();
                            if (st_value[18] .equals ("")) st_value[18]="null";
                    /*          try
                              {  */
                                 for (int j=0; j<19; j++)
                                 output.print(st_value[j]);
                                 output.println();
                             /*   }
                               catch (IOException e)
                                 {
                                  System.out.println("Error: " + e);
                                  System.exit(1);
                                 } */
                             }      //chiude blocco if dato buono
                        }     //chiude loop figli di pws
                    }      // chiude loop pws
                }    // chiude blocco try citySearch
               catch (IOException e)
               {
                  System.out.println("Error: " + e);
                  System.exit(1);
               }
                     Thread.sleep(60*1000*2);
            }    // chiude loop città di ricerca
        output.close();
    }
    }
```

6.2 Observations.java

Programma Observations.java per l'estrazione delle osservazioni per tutte le stazioni prese in considerazione nelle date prescelte rispetto alla data di esecuzione del programma stesso.

```
/**********************************************************************************************************************
 * Sulla base di un elenco di stazioni metereologiche private e non, contenute in SVIMCNNIstations.csv
 * interroga per 8 volte ogni stazione Weather Unerground chiedendo l'history delle osservazioni che ciascuna di loro ha effettuato
 * negli 8 giorni presi in considerazione.
 * La lista delle stazioni viene estratta con WScircle.java che cerca tutte le stazioni prossime entro un certo raggio
 * alle città contenute in un elenco opportunamente preparato. Quanto estratto viene poi rivisto a mano in excel per selezionare le stazioni
 * di reale interesse. Vengono anche sostituiti i blank con underscore nel campo della località.
 *
```

```
* La prima volta che si avvia il servizio di estrazione delle Observations è necessario preparare il file olddates.txt co i seguenti
* campi su una singola riga: 8 zeri, ";" la data precedente l'esecuzione del programma nel formato YYYYMMGG es:
* 00000000;20150720
* Dalle successive escuzioni il programma provvederà ad aggiornare questo file con le nuove date che corrispondono a:
* La data di ultima estrazione (la più recente nel tempo ricercata con l'ultima precedente esecuzione e dopo il ";" la da più vecchia
* che sia stata estratta.
*
* Per mantenere la continuità delle estrazioni è necessario far girare il programma almeno ogni 8 giorni.
* iniziera col cercare di estrarre le osservazioni del giorno precedente, poi le date più recenti non ancora lette nei giorni precedenti,
* poi se gli rimangono ancora delle letture da fare (entro le 8 disponibili) risale alla data di più vecchia estrazione e va avanti
* da li per leggere le osservazioni passate.
* Quindi se lo si fa girare più frequentemente di ogni 8 giorni, poco alla volta ricostruisce lo storico andando all'indietro.
*
* Il limite di 8 letture per 57 stazioni è per rientrare nella disponibilità di 500 interrogazioni gratuite al giorno di
*  weather underground: 57*8=456 con l'avanzo di 44 per eventuali test o nuovi sviluppi.
*
*  Il programma accoda le osservazioni in weather_SVIMCNNI.txt, quando le si preleva per passarle in hadoop è consigliato cancellare
*  il file per limitarne le dimensioni ed evitare doppioni ad ogni reload in hadoop
*
* Il programma produce anche un log delle date trattate "dates_log.txt" che via via accumula su tante righe la data di esecuzione e la
* data di cui si è fattala ricerca.
*
*
*/
package Whistory;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.GregorianCalendar;
import java.util.Scanner;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class Observations
{
   public static void main(String[] args) throws FileNotFoundException,
                                    ParserConfigurationException,
                                    SAXException,IOException,
                                       InterruptedException
   {
            String [] weather_value = new String[57];
      String [][] station= new String [3][8]; // codice, nazione, tipo
      String history_date, stationstring, last_exec_date, oldest_date, exec_date, this_last_date, today;
      history_date = "";
      stationstring = "";
      last_exec_date = "";
      this_last_date = "";
      oldest_date = "";
      exec_date = "";
      today = "";
      GregorianCalendar current_date=new GregorianCalendar();
      int before=-1;
      int count=0;
      int calls=0;
      int lunghlist=0;
      int first=0;   // per riconoscere la prima execuzione del run e attribuire la data di estrazione a exec_date
      for (int i=0; i<57;i++)
      {
         weather_value[i]=null;
      }
      for (int j=0; j<3;j++)
```

```
{
        for (int i=0; i<8;i++)
        {
   station[j][i]=null;
        }
}
File file = new File("/Meteo/SVIMCNNIstations.csv");
Scanner input = new Scanner(file);
while(input.hasNext())
{
   String stationline = input.nextLine();
        String delims = "[;]";
        String [] tokens = stationline.split(delims);
        for (int i=0; i<8;i++)
        {
                        station[count][i] = tokens [i];  //codice stazione
        }
                        count++;     //ha riempito l'array delle città da interrogare con i dati letti dal file
} //chiude il while ci sono righe da leggere nel csv
input.close();
DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");
today=dateFormat.format(current_date.getTime());
PrintWriter date_log = new PrintWriter(new BufferedWriter
      (new FileWriter("/Meteo/dates_log.txt",true)));        //apre il file log delle date cercate in data
File file_olddates = new File("/Meteo/olddates.txt");  // legge ultima data esecuzione e data più vecchia di lettura
Scanner input_olddates = new Scanner(file_olddates);
while(input_olddates.hasNext())
        {
        String datesline = input_olddates.nextLine();
        String delims = "[;]";
        String [] tokens = datesline.split(delims);
        last_exec_date = tokens[0];    // legge l'ultimo giorno di esecuzione
        oldest_date = tokens[1];    //legge il giorno più vecchio di esecuzione
        }
input_olddates.close();
PrintWriter output = new PrintWriter(new BufferedWriter
                        (new FileWriter("/Meteo/weather_SVIMCNNI.txt",true)));
for (int i=0; i<3;i++)
   {
      try
      {
         current_date=new GregorianCalendar();
        while(calls<8)
          {
            current_date.add(GregorianCalendar.DATE, before);
            history_date=dateFormat.format(current_date.getTime());
            if (history_date.equals(last_exec_date))
            {
                        while(!history_date.equals(oldest_date))
                          {    // salta le date già fatte
                           current_date.add(GregorianCalendar.DATE, before);
                           history_date=dateFormat.format(current_date.getTime());
                          }
                    current_date.add(GregorianCalendar.DATE, before);
                    history_date=dateFormat.format(current_date.getTime());
             }
            if (i==0)
            {
               date_log.print(today+";"+history_date); //con per tutte le date della prima città segna le date cercate
               date_log.println();
            }

            if (first==0)
                                {
                        exec_date=history_date;
                        first=1;
                                }
            if (station[i][2].equals("airport")) {
            stationstring = station[i][0];
            } else {
                        stationstring = "pws:"+station[i][0];
            }
```

```
URL googleWeatherXml = new URL("http://api.wunderground.com/api/980cd527b87d38cd/history_"+history_date+"/q/"+station[i][1]+"/"+stationstring+".xml");
URLConnection uc=googleWeatherXml.openConnection();
uc.connect();
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(uc.getInputStream());
doc.getDocumentElement().normalize();
NodeList nList = doc.getElementsByTagName("observation");
lunghlist = nList.getLength();
for (int temp = 1; temp < lunghlist; temp++)
{
   Node nNode = nList.item(temp);
   if (nNode.getNodeType() == Node.ELEMENT_NODE)
   {
      Element eElement = (Element) nNode;
      weather_value[0]=eElement.getElementsByTagName("pretty").item(0)
            .getTextContent().toString();
      weather_value[1]="/";
      weather_value[2]=station[i][0];
      weather_value[3]="/";
      weather_value[4]=station[i][2];
      weather_value[5]="/";
      weather_value[6]=station[i][3];
      weather_value[7]="/";
      weather_value[8]=station[i][4];
      weather_value[9]="/";
      weather_value[10]=station[i][5];
      weather_value[11]="/";
      weather_value[12]=station[i][6];
      weather_value[13]="/";
      weather_value[14]=station[i][7];
      weather_value[15]="/";
      weather_value[16]=eElement.getElementsByTagName("tempm").item(0)
            .getTextContent().toString();
      weather_value[17]="/";
      weather_value[18]=eElement.getElementsByTagName("dewptm").item(0)
            .getTextContent().toString();
      weather_value[19]="/";
      weather_value[20]=eElement.getElementsByTagName("hum").item(0)
            .getTextContent().toString();
      weather_value[21]="/";
      weather_value[22]=eElement.getElementsByTagName("wspdm").item(0)
            .getTextContent().toString();
      weather_value[23]="/";
      weather_value[24]=eElement.getElementsByTagName("wgustm").item(0)
                  .getTextContent().toString();
      weather_value[25]="/";
      weather_value[26]=eElement.getElementsByTagName("wdird").item(0)
            .getTextContent().toString();
      weather_value[27]="/";
      weather_value[28]=eElement.getElementsByTagName("wdire").item(0)
            .getTextContent().toString();
      weather_value[29]="/";
      weather_value[30]=eElement.getElementsByTagName("wdird").item(0)
            .getTextContent().toString();
        weather_value[31]="/";
      if (station[i][3].equals("airport"))
        {
        weather_value[32]=eElement.getElementsByTagName("vism").item(0)
          .getTextContent().toString();
         weather_value[33]="/";
        }
      else
      {
        weather_value[32]="-9999";
            weather_value[33]="/";
      }
      weather_value[34]=eElement.getElementsByTagName("pressurem").item(0)
            .getTextContent().toString();
      weather_value[35]="/";
      weather_value[36]=eElement.getElementsByTagName("windchillm").item(0)
          .getTextContent().toString();
```

```
                weather_value[37]="/";
                weather_value[38]=eElement.getElementsByTagName("heatindexm").item(0)
                      .getTextContent().toString();
                weather_value[39]="/";
                if (station[i][3].equals("airport"))
                  {
                  weather_value[40]=eElement.getElementsByTagName("precipm").item(0)
                                    .getTextContent().toString();
                weather_value[41]="/";
                weather_value[42]=eElement.getElementsByTagName("conds").item(0)
                      .getTextContent().toString();
                weather_value[43]="/";
                weather_value[44]=eElement.getElementsByTagName("icon").item(0)
                            .getTextContent().toString();
                weather_value[45]="/";
                weather_value[46]=eElement.getElementsByTagName("fog").item(0)
                      .getTextContent().toString();
                weather_value[47]="/";
                weather_value[48]=eElement.getElementsByTagName("rain").item(0)
                      .getTextContent().toString();
                weather_value[49]="/";
                weather_value[50]=eElement.getElementsByTagName("snow").item(0)
                      .getTextContent().toString();
                weather_value[51]="/";
                weather_value[52]=eElement.getElementsByTagName("hail").item(0)
                      .getTextContent().toString();
                weather_value[53]="/";
                weather_value[54]=eElement.getElementsByTagName("thunder").item(0)
                      .getTextContent().toString();
                weather_value[55]="/";
                weather_value[56]=eElement.getElementsByTagName("tornado").item(0)
                      .getTextContent().toString();
                  }
                else
                  {
                  weather_value[40]="-99999";
                   weather_value[41]="/";
                   weather_value[42]="-99999";
                   weather_value[43]="/";
                   weather_value[44]="-99999";
                   weather_value[45]="/";
                   weather_value[46]="-99999";
                   weather_value[47]="/";
                   weather_value[48]="-99999";
                   weather_value[49]="/";
                   weather_value[50]="-99999";
                   weather_value[51]="/";
                   weather_value[52]="-99999";
                   weather_value[53]="/";
                   weather_value[54]="-99999";
                   weather_value[55]="/";
                   weather_value[56]="-99999";
                  }
                 for (int j=0; j<57; j++)
                 {
                   output.print(weather_value[j]);
                 }
                   output.println();
              }  // chiude la lettura dei dati di una singola observation, if nodo buono
            }    // chiude la lettura di tutte le observations: la lista
            Thread.sleep(60*1000);
            calls++;
            this_last_date=history_date;
        }   // chiude loop delle 8 lettture per città
        calls=0; // riazzera il flag delle chiamate per città
    }    // chiude il try del corpo loop una delle 57 città
    catch (IOException e)
    {
      System.out.println("Error: " + e);
      System.exit(1);
    }  //chiude il try n letture per città
}   // chiude il loop delle 57 città
try
```

```
        {
          output.close();
          PrintWriter output_date = new PrintWriter(new BufferedWriter
                    (new FileWriter("/Meteo/olddates.txt",true)));
          output_date.print(exec_date+";"+this_last_date);
          output_date.println();
          output_date.close();
          date_log.close();  //chiude il file log delle date cercate in data
        }
    catch (IOException e)
        {
          System.out.println("Error: " + e);
          System.exit(1);
        }
    } // chiude il main
} // chiude la classe observations
```

6.3 MeteoSVL.pig

```
-- letti tutti come chararray per pulire prima i dati numerici che hanno qualche valore
null scritto come stringa


rawdata = load '/home/hadoop/MeteoSVL/input' using PigStorage(' ') AS (time, apm, cest, on,
month, day, year, code, country, sttype, location:chararray, city, region:chararray, lat,
lon, tempm:chararray, dewptm:chararray, hum:chararray, wspdm:chararray, wgustm:chararray,
wdird:chararray,        wdire:chararray,        vism:chararray,        pressurem:chararray,
windchillm:chararray,     heatindexm:chararray,     precipm:chararray,     conds:chararray,
icon:chararray,     fog:chararray,     rain:chararray,     snow:chararray,     hail:chararray,
thunder:chararray, tornado:chararray);


-- ricavo un sample se necessario operare su pochi dati
-- rawdatasample = sample rawdata 0.01;


-- per pulire i dati prima cerco le uguaglianze tra chararray
cleaned0    =    foreach    rawdata    generate    (chararray)year,(month    ==    'January'?
'01':(month=='February'?'02':(month=='March'?'03':(month=='April'?'04':(month=='May'?'05':(
month=='June'?'06':(month=='July'?'07':(month=='August'?'08':(month=='September'?'09':(mont
h=='October'?'10':(month=='November'?'11':(month=='December'?'12':''))))))))))))          AS
monthnum,  SUBSTRING(day,0,2)  AS  (day:chararray),(SIZE(time)==4?CONCAT('0',time):time)  AS
time,           (chararray)month,           (chararray)code,           (chararray)country,
(chararray)sttype,(location=='null'or                              SUBSTRING(location,0,4)=='-
999'?null:(chararray)location)    AS    location,    (chararray)city,    (region=='null'or
SUBSTRING(region,0,4)=='-999'?null:(chararray)region)  AS  region,  (float)lat,  (float)lon,
(tempm=='null'?null:(float)tempm)  AS  tempm,  (dewptm=='null'?null:(float)dewptm)  AS  dewptm,
(hum=='null'?null:(int)hum)  AS  hum,  (wspdm=='-9999'  or  wspdm=='null'?null:(float)wspdm)AS
wspdm, (wgustm=='-9999' or wgustm=='null'?null:(float)wgustm)AS wgustm, (wdird=='-9999' or
wdird=='null'    ?null:(int)wdird)AS    wdird,    (wdire=='null'or    SUBSTRING(wdire,0,4)=='-
999'?null:(chararray)wdire)AS  wdire,  (vism=='-9999'  or  vism=='null'  ?null:(int)vism)  AS
vism,                  (pressurem=='null'?null:(int)pressurem)AS                  pressurem,
(windchillm=='null'?null:(float)windchillm)AS                                  windchillm,
(heatindexm=='null'?null:(float)heatindexm)AS          heatindexm,          (precipm=='null'
?null:(float)precipm)AS  precipm,  (conds  ==  'null'or  SUBSTRING(conds,0,4)=='-999'or  conds
=='unknown'  or  conds  ==  'Unknown'?null:(chararray)conds)AS  conds,  (icon=='null'or
SUBSTRING(icon,0,4)=='-999'  or  icon=='unknown'  or  icon=='Unknown'?null:(chararray)icon)AS
icon,   (fog=='null'    ?null:(int)fog)AS   fog,   (rain=='null'?null:(int)rain)   AS   rain,
(snow=='null'?null:(int)snow)AS      snow,      (hail=='null'?null:(int)hail)AS      hail,
(thunder=='null'?null:(int)thunder)AS     thunder,     (tornado=='null'?null:(int)tornado)AS
tornado;


-- proseguo la pulizia confrontando i valori numerici
cleaned1         =         foreach         cleaned0         generate         year,monthnum,
day,time,month,code,country,sttype,location,city,region,lat,lon,                  (tempm<=-
999?null:(float)tempm)  AS  tempm,  (dewptm<=-999?null:(float)dewptm)  AS  dewptm,  (hum<=-
999?null:(int)hum)        AS        hum,        (wspdm<0?null:(float)wspdm)AS        wspdm,
(wgustm<0?null:(float)wgustm)AS   wgustm,   (wdird==-9999?null:(int)wdird)AS   wdird,wdire,
(vism==-9999?null:(int)vism)  AS  vism,  (pressurem<=-999  ?null:(int)pressurem)AS  pressurem,
(windchillm<=-999?null:(float)windchillm)AS            windchillm,            (heatindexm<=-
9999?null:(float)heatindexm)AS      heatindexm,      (precipm<=-9999?null:(float)precipm)AS
precipm,conds,icon,  (fog<=-9999?null:(int)fog)AS fog,  (rain<=-9999?null:(int)rain)  AS rain,
```

```
(snow<=-9999?null:(int)snow)AS snow, (hail<=-9999?null:(int)hail)AS hail, (thunder<=-9999?null:(int)thunder)AS thunder, (tornado<=-9999?null:(int)tornado)AS tornado;

-- aggiusto la data nei vari formati datetime, interi e nome mese
cleaned2 = foreach cleaned1 generate ToDate(CONCAT(day,'/',monthnum,'/',year,' ',time,':00Z'),'dd/MM/yyyy hh:mm:ssZ') AS dateandtime, (int)year,(int)monthnum,(int)day,(chararray)time,(chararray)month, code..;

-- creo una tabella pulita su disco
store cleaned2 INTO '/home/hadoop/MeteoSVL/SVIMCNNItable' USING PigStorage (';');

-- filtro i dati da marzo ad agosto per fare esempi su ipotetica stagione agricola
season = filter cleaned2 by (monthnum>2 and monthnum<9);
-- istruzioni utilizzabili per verificare i dati
-- airportseason = filter season by (sttype matches 'airport');
-- months = group season by (monthnum);
-- justmonths = foreach months generate FLATTEN(group) as monthnum;

-- raggruppo per giornata e localita'
dateplace = group season by (code,sttype,year,monthnum,day,city,location,lat,lon);

-- calcolo max min ecc.
maxmindata = foreach dateplace generate FLATTEN(group) as (code,sttype,year,monthnum,day,city,location,lat,lon), MAX(season.tempm) AS maxtempm, MIN(season.tempm) AS mintempm, MAX(season.dewptm) AS maxdewptm, MIN(season.dewptm) AS mindewptm, MAX(season.hum) AS maxhum, MIN(season.hum) AS minhum, MAX(season.wspdm) AS maxwspdm, MIN(season.wspdm) AS minwspdm, MAX(season.wgustm) AS maxwgustm, MIN(season.wgustm) AS minwgustm,MAX(season.vism) AS maxvism, MIN(season.vism) AS minvism, MAX(season.pressurem) AS maxpressurem,MIN(season.pressurem) AS minpressurem, MAX(season.windchillm) AS maxwindchillm, MIN(season.windchillm) AS minwindchillm, MAX(season.heatindexm) AS maxheatindexm,MIN(season.heatindexm) AS minheatindexm, MAX(season.precipm) AS maxprecipm, MIN(season.precipm) AS minprecipm;

-- istruzioni utilizzabili per verificare i dati
-- singlecase = filter cleaned2 by (code matches 'IPIEMONT209');
-- store singlecase INTO '/home/hadoop/MeteoSVL/SVIMCNNItable3' USING PigStorage (';');

-- calcolo l'escursione giornaliera di ciascuna variabile
deltadata = foreach maxmindata generate code,sttype,year,monthnum,day,city,location,lat,lon,maxtempm-mintempm AS deltatempm, maxdewptm-mindewptm AS deltadewptm, maxhum-minhum AS deltahum, maxwspdm-minwspdm AS deltawspdm, maxwgustm-minwgustm AS deltawgustm, maxvism-minvism AS deltavism, maxpressurem-minpressurem AS deltapressurem, maxwindchillm-minwindchillm AS deltawindchillm, maxheatindexm-minheatindexm AS deltaheatindexm, maxprecipm-minprecipm AS deltaprecipm;

-- raggruppo le escursioni stagionali per ogni località
deltaxplace = group deltadata by (code,sttype,city,location,lat,lon);

-- calcolo le medie stagionali
meandata = foreach deltaxplace generate FLATTEN(group) as (code,sttype,city,location,lat,lon), AVG(deltadata.deltatempm) AS avgdeltatempm, AVG(deltadata.deltadewptm) AS avgdeltadewptm, AVG(deltadata.deltahum) AS avgdeltahum, AVG(deltadata.deltawspdm)AS avgdeltadeltawspdm, AVG(deltadata.deltawgustm) AS avgdeltawgustm, AVG(deltadata.deltavism) AS avgdeltavism, AVG(deltadata.deltapressurem) AS avgdeltapressurem, AVG(deltadata.deltawindchillm) AS avgdeltawindchillm, AVG(deltadata.deltaheatindexm) AS avgdeltaheatindexm, AVG(deltadata.deltaprecipm) AS avgdeltaprecipm;

-- ordina per la variabile temperatura
srtdeltatempm = ORDER meandata BY avgdeltatempm desc;

-- stampo/archivio la classifica ottenuta
-- dump srtdeltatempm;
store srtdeltatempm INTO '/home/hadoop/MeteoSVL/SVIMCNNIescterm' USING PigStorage (';');
```

```
-- Analisi medie per Savona
-- riparto dai soli dati di savona
savona = filter cleaned2 by (code matches 'I90579414');

-- raggruppo tutti i dati di ogni giorno
savonaxday = group savona by (year,monthnum,day);

-- produco la tabella con le medie giornaliere della stagione agricola per le variabili principali temperatura, umidità, pressione
avgday = foreach savonaxday generate FLATTEN(group) as (year,monthnum,day), AVG(savona.tempm) AS avgtempm, AVG(savona.hum) AS avghum, AVG(savona.pressurem) AS avgpressurem;

-- ordina alfabeticamente per data
srtavgday = ORDER avgday BY year,monthnum,day;

-- archivio i dati ottenuti
store srtavgday INTO '/home/hadoop/MeteoSVL/Savona' USING PigStorage (';');
```

Recent titles from the IMATI-REPORT Series:

2017

17-01: *BPX preconditioners for isogeometric analysis using analysis-suitable T-splines,* D. Cho, R. Vázquez.

17-02: *Initial-boundary value problems for nearly incompressible vector fields, and applications to the Keyfitz and Kranzer system*, A. P. Choudhury, G. Crippa, L.V. Spinolo.

17-03: *Quantitative estimates on localized finite differences for the fractional Poisson problem, and applications to regularity and spectral stability*, G. Akagi, G. Schimperna, A. Segatti, L.V. Spinolo.

17-04: *Optimality of integrability estimates for advection-diffusion equations*, S. Bianchini, M. Colombo, G. Crippa, L.V. Spinolo.

17-05: *A mathematical model for piracy control through police response*, G.M. Coclite, M. Garavelllo, L.V. Spinolo.

17-06: *Uncertainty Quantification of geochemical and mechanical compaction in layered sedimentary basins*, I. Colombo, F. Nobile, G. Porta, A. Scotti, L. Tamellini.

17-07: *VVS medical workflows: definition of a static workflow for part-based annotation of wrist bones & web service oriented architecture for executable workflows*, M. Pitikakis, F. Giannini.

17-08: *Computational Methods for the Morphological Analysis and Annotation of Segmented 3D Medical Data*, G. Patanè, F. Giannini, M. Attene.

17-09: *Applying Functional Principal Components to Structural Topology Optimization,* G. Alaimo, F. Auricchio, I. Bianchini, E. Lanzarone.

17-10: *Convergence of Sparse Collocation for Functions of Countably Many Gaussian Random Variables (with Application to Elliptic PDEs),* O. G. Ernst, B. Sprungk, L. Tamellini

17-11: *Soluzioni Open Source per la scalabilità di servizi di raccolta ed utilizzo di dati ambientali da sensore*, A. Quarati, A. Clematis, V. Levati.