LOGSTASH: PROGETTO OPEN PER L'ANALISI DEI LOG IN TEMPO REALE DI ARCHITETTURE CLOUD

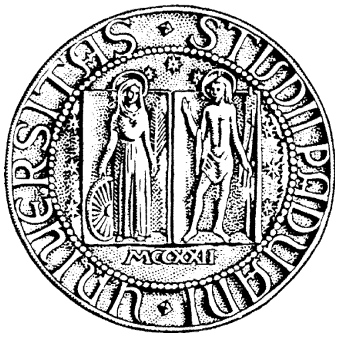
LOGSTASH: OPEN PROJECT FOR REAL-TIME LOG ANALYSIS OF CLOUD ARCHITECTURES

RELATORE: Carlo Ferrari

LAUREANDO: Mattia Peterle

A.A. 2012-2013

UNIVERSITÀ DEGLI STUDI DI PADOVA



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

LOGSTASH: PROGETTO OPEN PER L'ANALISI DEI LOG IN TEMPO REALE DI ARCHITETTURE CLOUD

LOGSTASH: OPEN PROJECT FOR REAL-TIME LOG ANALYSIS OF CLOUD ARCHITECTURES

RELATORE: Carlo Ferrari

LAUREANDO: *Mattia Peterle*

Padova, 22 Luglio 2013

Alla mia famiglia

Indice

Sommario

Nel settore informatico, oggigiorno, le aziende devono proporre soluzioni innovative di altissima qualità complice una clientela esigente e un mercato sempre più competitivo. La necessità di ridurre i tempi in fase di *debugging* ha spinto molte imprese ad adottare sistemi di Log Analisi per aiutare i propri sviluppatori o per monitorare, istantaneamente, applicativi e infrastrutture web.

L'obiettivo del tirocinio svolto presso *Zero12 s.r.l.* è stato quello di progettare un sistema di Analisi dei Log dinamico, scalabile e altamente portabile che fosse in grado di analizzare, in tempo reale, i messaggi dell'applicativo *DriveFarm* e recapitasse, attraverso sistemi intuitivi quali email o interfacce grafiche, i risultati agli amministratori del servizio.

Inizialmente si è deciso di non sviluppare un applicativo proprietario, ma di effettuare una *Software Selection* per identificare gli strumenti *Open Source* e stabilire un primo sistema di Log Analisi. La scelta finale è ricaduta sul software *LogStash* e una particolare gamma di strumenti correlati.

Il sistema è stato configurato su macchine Linux della famiglia Ubuntu, particolare attenzione è stata compiuta nell'ambito architetturale per supportare le tecnologie AMI di *Amazon*.

Sono stati realizzati dei *plugin* personalizzati, in linguaggio *Ruby*, per l'implementazione di sistemi temporizzatori di avvisi con successiva trasmissione via SNS e per la memorizzazione di eventi su *Bucket* S3.

L'ultimo passo è stato quello di scrivere degli script in Linux per stabilizzare ed avviare il sistema di Log Analisi; oltre a raccogliere una serie d'informazioni per l'ottimizzazione, il *debugging* e la gestione del sistema di Analisi dei Log utili per futuri aggiornamenti.

I risultati ottenuti dimostrano il raggiungimento degli obiettivi proposti, con un margine d'incertezza sulla stabilità del sistema e sulla consapevole individuazione di particolari eventi.

Introduzione

Sono passati più di trent'anni dall'introduzione di internet come mezzo di comunicazione eppure solamente dalla fine degli anni novanta si è evidenziato una crescita esponenziale dei suoi utilizzatori, complice una più rinnovata mole di applicativi web e un forte progresso tecnologico nell'ambito delle telecomunicazioni.
A partire dal nuovo secolo, con l'introduzione di servizi social, di intrattenimento e la massiccia messa in commercio di dispositivi mobili, si è cambiato radicalmente il modo di trattare le informazioni digitali e il rilascio del software.
Oramai sempre più imprese si lanciano nel mercato rilasciando applicativi aggiornabili nel tempo, grazie ad un buon supporto *feedback* da parte dell'utenza, e proponendo tariffe sul servizio offerto.
Altresì, la dinamicità del sistema nel complesso è tale che la domanda di infrastrutture per la gestione delle informazioni digitali è innegabile e tuttora molti produttori offrono servizi *Cloud* rivolti a privati o altresì ad aziende interessate.
La semplicità e la versatilità di questi sistemi permette un notevole risparmio nei *backup* indipendenti oltre a migliorare ulteriormente lo scambio d'informazioni tra dispositivi fisici. Nell'ambito finanziario queste tecnologie facilitano la gestione delle infrastrutture web rendendo, di fatto, i sistemi dinamici alla domanda dei *client* per i servizi messi a disposizione.
Tra le aziende che forniscono questo genere di consulenze vi è *zero12 s.r.l.* che propone soluzioni Cloud sulla base di tecnologie *Amazon*.
In particolare il rilascio dell'applicativo DriveFarm, realizzato attraverso un'infrastruttura Cloud di Amazon Web Services, consente ai beneficiari di condividere informazioni digitali in modo rapido e sicuro.
Tuttavia Zero12, come altre aziende del settore, vuole facilitare lo sviluppo dei suoi applicativi attraverso l'introduzione di sistemi di Log Analisi per aiutare i suoi programmatori nella fase di *debugging*, ora rivolta esplicitamente al servizio DriveFarm.
La Log Analisi si basa sull'interpretazione, la suddivisone in campi e l'osservazione di eventi generati da applicativi informatici.
I Log, che siano generati da un servizio web implementato attraverso un *application server* come il *JBoss, TomCat, Geronimo* o siano applicativi mobile per dispositivi *Android, iOS, ecc...*, hanno diverse formattazioni, a volte complesse, che richiedono spesso almeno alcuni secondi d'interpretazione simbolica.
Aggiungendo il sovraffollamento di operazioni da parte di uno o più applicativi, la risultante del flusso di eventi è enorme anche per la vista di un operatore assegnato solamente a svolgere quel determinato compito.

Da qui l'idea d'instaurare un sistema autonomo, scalabile e altamente portabile, che operi in un ambiente sincronizzato, in grado di interpretare i Log e di avvisare rapidamente gli sviluppatori in caso di necessità.

Il primo Capitolo contiene un'introduzione ai requisiti progettuali.
Il secondo fornisce una panoramica sul software *open-source* utilizzato nonchè gli applicativi correlati, nel terzo viene descritta la procedura di configurazione del server ricevente e la classificazione degli eventi.
Segue il Capitolo 4 nel quale viene descritta la possibilità di realizzare plugin personalizzati attraverso linguaggio Ruby oltre all'analisi implementativa di temporizzatori d'avvisi e di memorizzatori d'eventi con *Bucket S3*.
Nel Capitolo 5 si andranno ad analizzare script in Linux per l'avvio del sistema di Log Analisi e per la sua stabilità nel corso del tempo.
Infine, i risultati del collaudo, considerazioni e sviluppi futuri nel Capitolo 6.

Per completezza sono state aggiunte due appendici:
L'Appendice A contiene informazioni riguardanti il *File.conf* di LogStash nonchè il codice della configurazione del Server trasmettitore, ricevente e un listato delle espressioni regolari.
L'Appendice B, contiene il codice Ruby dei plugin *custom* modificati e realizzati per LogStash.

Capitolo 1

Progettazione Preliminare

1.1 Introduzione alla Log Analisi

La Log Analisi è una metodologia di osservazione e traduzione di messaggi rilasciati, attraverso diversi tipi di output, da software informatici.
Fin dagli albori, la programmazione di applicativi per calcolatori ha richiesto una certa arte di comunicazione tra uomo e macchina, affinché quest'ultima potesse svolgere determinati compiti. Gli sviluppatori, oltre ad affrontare la progettazione e la realizzazione di algoritmi, dovevano tutelarsi nella fase di sviluppo, poiché era facile smarrirsi nel proprio codice ed incorrere in errore.
Il problema nasceva, in particolare, nella necessità di prevenire una considerevole perdita di tempo nell'ispezionare minuziosamente una mole consistente di righe di codice, nel tentativo di risolvere qualche *bug* o aiutare il prossimo programmatore che avrebbe dovuto partire dal potenziale artistico sviluppato.
Tali procedure spaziano dal più semplice commento fra le righe di codice ai blocchi di *Exception* per gestire il verificarsi di controversie nel programma.
Nel tempo si accetto come uno *standard* istaurare un sotto sistema di messaggi che potesse aiutare lo sviluppo e oramai la maggior parte dei software rilascia delle stringhe di log sugli standard-output o attraverso file in memoria, le quali sono dei moniti diagnostici dell'applicativo.
Tuttavia nemmeno l'ISO[1] riuscì ad imporsi in un'unificazione del formato dei Log i quali, molto spesso, rispecchiano il fascino artistico degli stessi sviluppatori lasciando perplessi chi tenta di comprenderne il significato.
I Log, infatti, non sono altro che una porzione di frase criptica che riassume un particolare stato cui il software incorre.

[1] *ISO: Acronimo di **I**nternational **O**rganization for **S**tandardization.*

Detti anche Eventi, in assenza di metadati, i Log possono identificare diversi stati di programma: dalle semplici informazioni sul corretto funzionamento a severe avvisaglie di malfunzionamento interno. Con l'introduzione di codice complesso il flusso d'eventi può divenire consistente, generando un ulteriore stato confusionale e quindi un corrispettivo meccanismo tutelante.
Da qui l'introduzione della Log Analisi:
In principio un operatore il cui compito era quello di osservare e tradurre i messaggi che i software, importunati dai *tester*, rilasciavano sugli output, catalogarne le varie priorità e stendere rapide relazioni riassuntive agli sviluppatori.
Tuttavia quest'approccio portava alle imprese costi aggiuntivi di manodopera nonché, detta simpaticamente, visite oculistiche ai propri analisti di Log.
Il passo successivo è stato dunque quello di creare dei software appositi che, correttamente configurati, potessero osservare e catalogare gli eventi generati degli altri applicativi.
Parlando specificatamente dell'obbiettivo del tirocinio: Zero12, sviluppatrice di DriveFarm, era propensa ad utilizzare un software di Log Analisi *open-source* già realizzato, questo sia per estraniarsi dai costi di sviluppo di un software proprietario, sia per velocizzare i tempi di messa a punto di un sistema d'Analisi dei Log.
Il primo fondamentale punto è stato quello di trarre una panoramica generale dei requisiti di progetto e successivamente di effettuare una *Software Selection* per promuove l'utilizzo di un giusto applicativo.

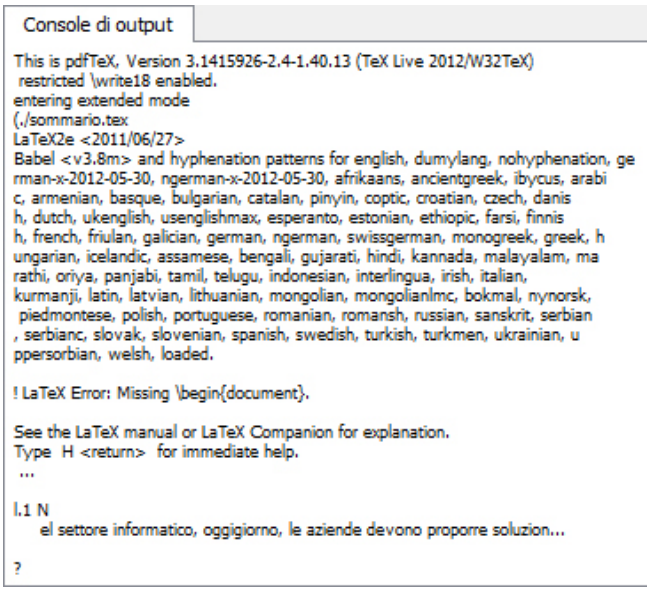


Figura 1.1: Esempio di Evento generato da LaTeX2e durante la stesura della tesi

1.2 Requisiti Progettuali

L'obiettivo fondamentale per eseguire una buona *Software Selection* è quello di stendere una serie di requisiti che l'azienda vuole dal programma di Log Analisi. Nelle premesse si elenca la ricerca di un applicativo dinamico, portabile, scalabile, che svolga i compiti assegnati in tempo reale e abbia la possibilità di avvisare gli amministratori attraverso sistemi quali mail o interfacce grafiche.
Quando parliamo di software dinamico intendiamo la capacità di un programma di poter svolgere diverse operazioni attraverso semplici configurazioni; nel nostro caso si necessitava di un applicativo in grado di analizzare diversi tipi di Log, nonché d'integrare funzionalità di supporto all'occorrenza.
Con applicativi portabili s'intende la capacità di un software di adattarsi a diversi sistemi operativi; da qui la prerogativa di scegliere un programma di Log Analisi realizzato attraverso linguaggio Java o linguaggi che comunque sarebbero stati indipendenti dall'ambiente d'esecuzione.
Infine con il termine scalabile indichiamo quel software in grado di poter operare efficacemente su diverse architetture di rete, riducendo i colli di bottiglia che potrebbero formarsi ed estendendo o riducendo le eventuali risorse disponibili a seconda del carico di lavoro; una delle caratteristiche proposte nei requisiti era quella di supportare la tecnologia AMI[2] di *Amazon*, descritta nel paragrafo 1.3.2.

1.2.1 Software Selection

Una *Software Selection* si basa sulla ricerca di determinati requisiti all'interno di una gamma di applicativi, producendo dapprima delle *Long List* dove si elencano i possibili candidati.
Da qui si passa ad una scrematura per ottenere una *Short List* contenente i candidati propensi e maggiori dettagli sulle caratteristiche positive e negative. Dopodiché si estraggono due o tre applicativi che rispecchiano i requisiti per effettuare la scelta finale.
Alla base di una buona *software selection*, oltre ai punti visti nel paragrafo precedente, vi è senza dubbio la ricerca di punti salienti che possano facilitare lo sviluppo, in particolare: il linguaggio di programmazione, la documentazione disponibile, il supporto dalle comunità, le aziende utilizzatrici e la facilità di utilizzo.
La disponibilità di software o framework open source in grado di svolgere la Log Analisi è ampia, da qui la ricerca di far chiarezza sui possibili candidati.

[2] *AMI: Acronimo di **A**mazon **M**achine **I**mage*

Un esempio di *Short List*

- Scribe (Apache License 2.0) https://github.com/facebook/scribe/wiki
- Flume (Apache License 2.0) http://flume.apache.org/
- LogStash (Apache License 2.0) http://www.logstash.net
- Splunk http://www.splunk.com/
- Cascading (Apache License) http://www.cascading.org/
- Pig (Apache License 2.0) http://pig.apache.org/
- Graylog2 (GNU General Public License v3) http://graylog2.org/home

Tra i seguenti *Splunk* è uno dei più utilizzati fra le aziende, in quanto supporta ampie risorse per l'analisi dei Log in tempo reale e interpreta i dati quando si cerca di fornirne un contesto più completo.
Il risultato è un approccio flessibile e completo ma, tuttavia, ha una licenza proprietaria in prova per 60 giorni e limitato ad una quota di dati fisici pari a 500MB al giorno. Non è quindi una scelta conforme alle nostre prerogative.
Un grande potenziale lo offre *Flume* che propone un'architettura semplice e affidabile basata su flussi di dati in streaming. Dispone di buoni meccanismi nel recupero dati e di una considerevole documentazione, nonché supporto della comunità. Per contro c'è da dire che ha un potenziale fin troppo ampio e necessita di studi approfonditi, essendo non rivolto soltanto alla Log Analisi.
Scribe è stato realizzato da *Facebook* e attualmente utilizzato come sistema di Log Analisi per la loro piattaforma. Versatile nel definire la tipologia di Log e nella localizzazione di quest'ultimi nei vari nodi che compongono l'architettura dell'infrastruttura. Purtroppo ha poca documentazione e aggiornamenti recenti, oltre ad essere stato scritto in linguaggio *Scala* e facente uso di librerie C++.
Un altro candidato è *Pig*: realizzato da *Yahoo!*, è ben documentato e propone una serie di script concisi ed efficaci per manipolare i Log, in particolare gli script possono anche essere scritti in *Python*. Purtroppo utilizza il linguaggio *Pig Latin* che è solamente un *Data Flow Language*[3], inoltre è necessario definire delle funzioni in Java se si vuole realizzare qualcosa che il linguaggio non dispone.

[3] *Data Flow Language: è un paradigma di programmazione che modella il programma come un grafo orientato al fine di gestire i dati che scorrono fra le operazioni.*

1.2.2 Candidati Favorevoli

Dopo un'attenta analisi, i candidati finali della *Software Selection* sono stati *LogStash* e *Cascading*.

- LogStash è un tool open souce flessibile e altamente portabile.
  Scritto in JRuby, processa eventi generati da specifici input che vengono filtrati e rilasciati sotto forma di Log, attraverso determinati Output.
  Può facilmente instaurare un sistema centralizzato, ovvero prevede l'invio di messaggi da più macchine verso un elaboratore principale che si fa carico degli eventi.
  Dispone di una struttura precisa di configurazione che viene lanciata insieme al programma attraverso un file nel formatol *JSON*[4] dove si specificano input output e filtri applicati all'analisi.
  Oltre ad avere un buon supporto dalla comunità, LogStash si dimostra vincente nella sua rapidità di instaurare un sistema efficace fin dal principio, utilizzando un pacchetto *Jar* pre-compilato e dei tool integrativi quali: *ElasticSearch*, *Kibana* e *Redis* che saranno trattati nel Capitolo 2.

- Cascading nasce come framework open source Java, affinché si possano sviluppare, facilmente, robusti sistemi d'analisi e gestioni dati sulla base di *Hadoop*[5]. Cascading è ancora un progetto giovane e poco supportato dalla comunità ma che non delude nel suo potenziale; è infatti possibile utilizzarlo anche come strumento per la Log Analisi.
  Uno dei fattori limitanti è senza dubbio la complessità di conoscenze necessarie per il suo corretto funzionamento, nonché una buona disciplina sui sistemi Hadoop. Si parte già con una certa difficoltà nell'istaurare un ambiente di sviluppo, essenzialmente dovuto alle librerie da contemplare.
  Il vero punto a favore è la possibilità di realizzare componenti riutilizzabili, sofisticati *pipelines* di dati e alto livello di astrazione dei processi.
  Tutto ciò viene implementato attraverso la definizione di costrutti come *pipes* e *filters* che interconnessi tra loro generano i cosiddetti *flow*, ovvero un flusso di dati che viene manipolato secondo i concetti stabiliti. Ovviamente un insieme di *flow* può generare strutture complesse alla base della gestione dati di grandi infrastrutture.

La scelta finale è ricaduta su LogStash, poiché permette d'istaurare un sistema di Log Analisi nell'immediato. Inoltre, la possibilità di una libreria open source ci

[4] *JSON: Acronimo di **J**ava**S**cript **O**bject **N**otation.*
[5] *Hadoop è un framework che supporta applicazioni distribuite con elevato accesso ai datii.*

ha permesso di sviluppare parti aggiuntive per gli obbiettivi che ci eravamo prefissati. Cascading prevedeva tempistiche elevate per comprenderlo ed instaurare un sistema di Log Analisi.

1.3 Attuazione di un Sistema di Log Analisi

Da questo punto in poi si parlerà esplicitamente di LogStash e strumenti correlati come software usati per la Log Analisi, mentre si analizzerà, in modo teorico fino alla fine del Capitolo 1, i passi per l'attuazione del sistema.
Ad ogni modo è fondamentale comprendere d'apprima l'applicativo DriveFarm e in successiva la tecnologia AMI che costituisce la base architteturale del sistema.

1.3.1 L'applicativo DriveFarm

DriveFarm è un file manager costruito su un'infrastruttura Cloud di Amazon Web Services che elimina tutte le problematiche relative alle infrastrutture hardware fisiche. Particolarmente adatto alle aziende che necessitano di gestire la propria documentazione, poiché prevede sistemi di controllo per la sicurezza, facilitazioni nella mobilità degli utenti, repository sempre accessibile e proliferazione degli utenti con gestione della regola di accesso sui file.
DriveFarm viene eseguito attraverso l'application server JBoss, il quale rilascia gli eventuali eventi dell'applicativo in un file server.log.
Ovviamente nei file di diagnostica del JBoss potremmo trovare i Log di altri applicativi istanziati insieme a DriveFarm.

1.3.2 Amazon Machine Image

I server su cui è istanziato il JBoss sono gestiti dalla tecnologia AMI di Amazon. Amazon Machine Image è una particolare immagine descrittiva dello stato attuale del calcolatore, utilizzata per ricreare una macchina virtuale attraverso EC2[6].
Le Istanze EC2[9] sono macchine virtuali in esecuzione su hardware di Amazon e affinché l'istanza venga avviata, è necessario una quantità minima d'informazioni. Per questo motivo la descrizione dell'AMI[8] racchiude in se il tipo di virtualizzazione, l'architettura (32/64 bit), il kernel, e il dispositivo di root.
Grazie a questa tecnologia, quando il server centrale non riesce più a farsi carico del lavoro da svolgere, viene affiancata una macchina la cui struttura non è altro

[6] *EC2: Acronimo di Amazon* ***E****lastic* ***C****ompute* ***C****loud.*

che una copia esatta del calcolatore in difficoltà, il tutto attraverso l'attuazione dell'immagine AMI. Ciò permette di utilizzare server *On Demand* consentendo un notevole risparmio in ambiti economici e prestazionali.
Tale tecnologia permette anche la sostituzione di macchine poco utilizzate con controparti meno potenti in modo da ristabilire un bilancio tra consumi e servizio da tutelare.
Molte imprese cominciano ad avvicinarsi a questo nuovo modo di concepire la gestione dei propri web server; dal momento che svincolano dalla necessità di collocare macchine con elevate prestazioni per tutelare i picchi di clientela che possono verificarsi in alcuni frangenti temporali. La vecchia concezione rendeva il servizio eccellente alla domanda, ma indebitava le aziende che dovevano mantenere i server anche quando questi non venivano sfruttati.
DriveFarm riesce quindi ad essere scalabile e versatile a seconda delle domande o meno dei suoi client. (Fig 1.2)

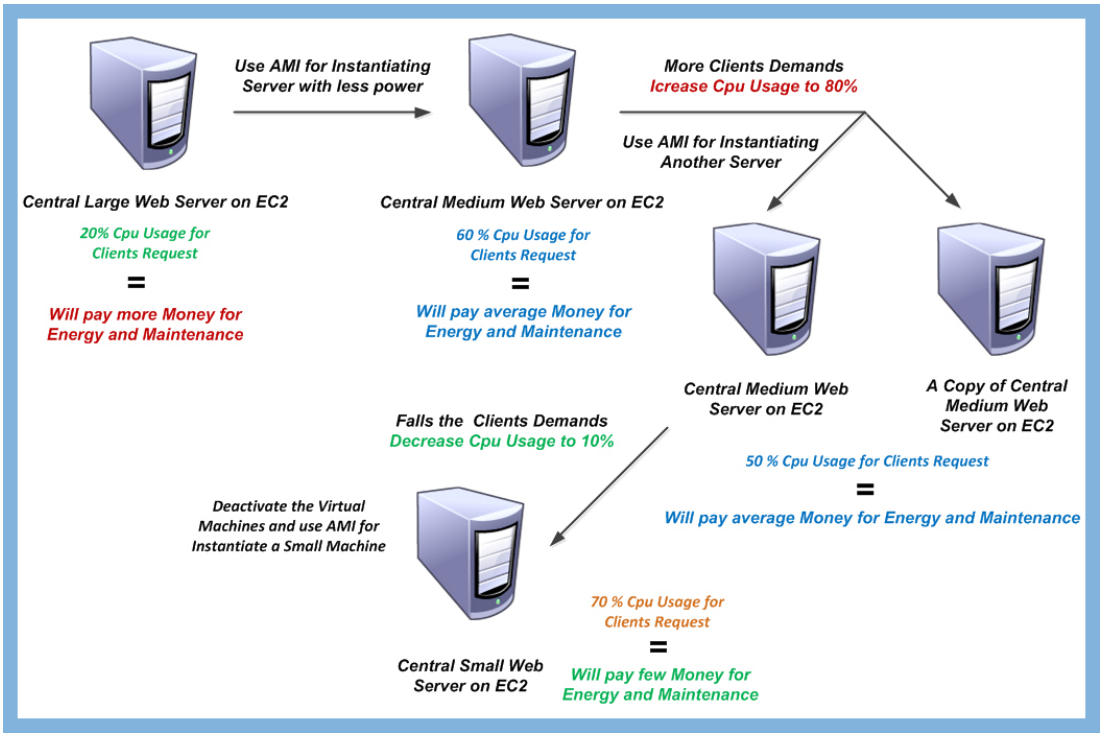


Figura 1.2: Esempio di applicazione dell'AMI su un'infrastruttura Web Server

1.3.3 Architettura del Sistema

Abbiamo definito, a grandi linee, quale sia la tecnologia con cui viene gestito DriveFarm. Il passo successivo è stato quello di realizzare, in parte, la stessa scalabilità per il sistema di Log Analisi.

Per iniziare abbiamo istanziato una versione di LogStash, precisamente la 1.9, sul Web Server centrale dov'è presente DriveFarm.
Attraverso la configurazione di LogStash siamo andati a leggere il file *central.log*, presente in */opt/jboss/standalone/log/server.log*, ovvero il Log del JBoss.
A questo punto abbiamo inviato gli eventi, ancora da manipolare, ad un server centrale destinato all'analisi. La configurazione di LogStash punta ad un'istanza del database Redis presente nel server ricevente con un IP fisso.
Grazie a questa configurazione i server trasmettitori, denominati *Shipper*, contengono un'istanza indipendente di LogStash, la quale legge la diagnostica del JBoss e quindi quella di DriveFarm. Nel caso in cui l'AMI entrasse in gioco, il gestore di rete non deve preoccuparsi di dover impostare qualche parametro sul software di Log Analisi, poiché quest'ultimo ha già i riferimenti necessari per svolgere i propri compiti indipendentemente.
Una volta che gli eventi arrivano alla macchina ricevente, denominata *Indexer*, questi vengono processati dalla configurazione del LogStash ricevitore (applicazione di filtri agli eventi, stoccaggio dei Log e invio di mail).
Lo stoccaggio avviene attraverso delle API[7] di Amazon, implementate in Ruby, per utilizzare un Bucket S3[8] e l'invio email attraverso quelle per il servizio SNS[9].
La memorizzazione di Log su Bucket avviene regolarmente e dipende dalla configurazione di LogStash. Il periodo di salvataggio dipende dal *LogLevel* dell'evento ricevuto, per esempio i Log d'errore vengono memorizzati ogni ora.
Le email, invece, vengono inviate all'amministratore con due prerogative: istantanee con il corpo del Log nella mail se nel LogStash ricevente incombono eventi con un LogLevel superiore o uguale all'errore, asincrone dopo un determinato tempo di configurazione per informare la quantità di diversi tipi di Log ricevuti.
Ovviamente tempistiche e grado di LogLevel possono essere modificate a piacimento nella configurazione di LogStash per tutelare il requisito di dinamicità
La maggior parte dei Log, tranne quelli informativi dei plugin custom, vengono inviata attraverso un istanza di ElasticSearch presente nel server Indexer per es-

[7] *API: Acronimo di **A**pplicationl **P**rogramming **I**nterface.*
[8] *S3: Acronimo di **S**imple **S**torage **S**ervice.*
[9] *SNS: Acronimo di **S**imple **N**otification **S**ervice.*

sere mappati e trattenuti brevemente. Un'istanza di Kibana è presente anch'essa nell'Indexer, la quale è in ascolto di eventi raccolti da ElasticSearch.
Quindi questi Log, infine, vengono prelevati e visualizzati attraverso una pagina Web per controllare in tempo reale il sistema di Log Analisi. (Fig 1.3)
Una nota importante da sottolineare è il fatto che il sistema è scalabile sulle macchine Shipper, tuttavia la macchina di Indexer ha un IP fisso per mantenere i giusti collegamenti con i server trasmettitori e comunicare con Redis.
Se la macchina ricevente si dovesse trovare appesantita, per esempio nel processare una grande mole di eventi, non sarà per ora possibile utilizzare l'AMI per suddividere il carico di lavoro.
Un buon punto di partenza per migliorare in futuro il sistema.
Ad ogni modo un'idea potrebbe essere quella di instaurare un controllo che, una volta attivata l'AMI, invii il nuovo IP alla configurazione di LogStash di alcuni Shipper, magari attraverso un algoritmo per bilanciare il carico di lavoro.

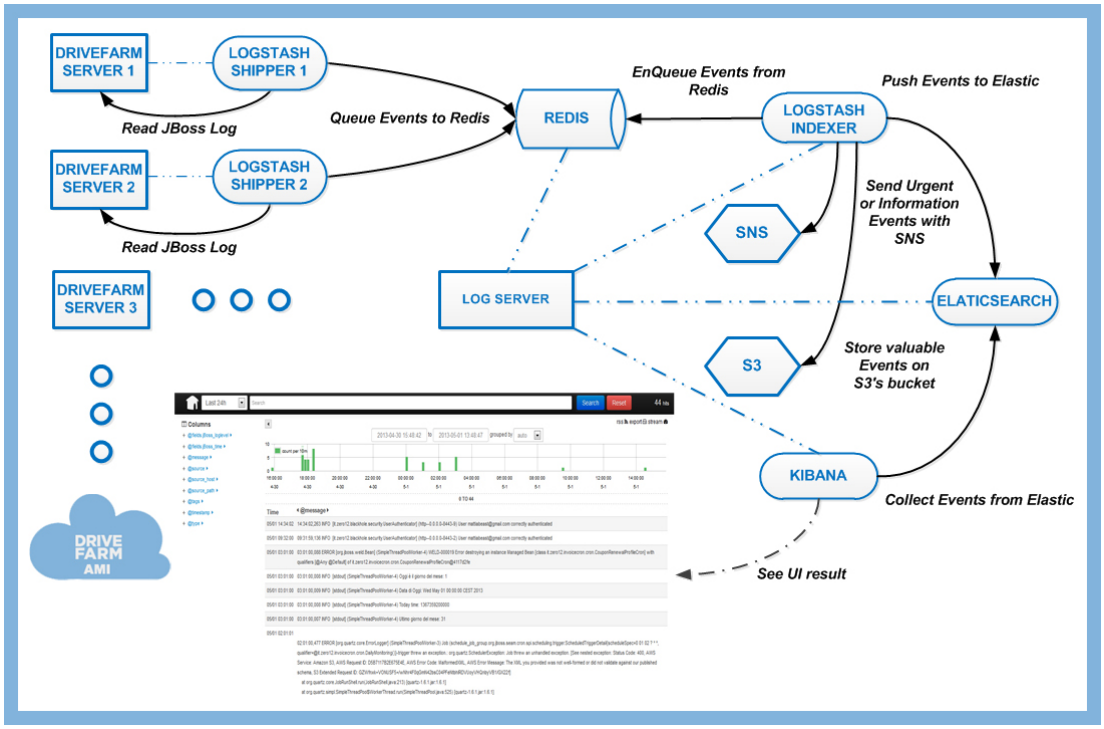


Figura 1.3: L'esempio di architettura realizza per il sistema di Log Analisi

Tutte le macchine del sistema di Log Analisi utilizzano il sistema operativo Linux della famiglia Ubuntu, in particolare il Server ricevente la versione 12.04.
La maggior parte dello sviluppo è stato effettuato sulla macchina di Indexer attraverso una connessione remota *ssh*.

Capitolo 2

Log Analisi Open Source

In questo capitolo tratteremo gli strumenti utilizzati per attuare il sistema di Log Analisi, nonché la loro configurazione in ambiente Linux.

2.1 Redis

Redis, *Remote Dictionary Server*, è un avanzato database open source che espone cinque differenti strutture dati delle quali solamente una è una tipica struttura chiave-valore. L'approccio di programmazione è di tipo procedurale e si fonda sugli stessi concetti di un database: persistenza di un insieme di dati, coerenza tra i dati di diversi applicativi.
Nel sistema di Log Analisi il suo utilizzo è stato cruciale per instaurare un collegamento fra i server trasmettitori e il ricevitore d'eventi.
La scelta è ricaduta su Redis, poiché rispetta le caratteristiche di dinamicità e portabilità ed ampiamente supportato dai plugin di output e input di LogStash.
Abbiamo detto che Redis utilizza una tipica struttura chiave-valore: le chiavi identificano le porzioni di dati e hanno un ruolo primario nella ricerca, mentre i valori rappresentano i dati effettivi associati alla chiave su cui non è possibile effettuare una ricerca esplicita.
L'aspetto interessante è il fatto che i valori possano assumere qualsiasi formato (stringhe, interi, oggetti, ecc...), poiché vengono considerati come array di byte.
Fra le differenti cinque strutture messe a disposizione (Stringe, Hash, Liste, Insiemi e Insiemi Ordinati) è stata utilizzata la lista per instaurare una FIFO[1] fra gli eventi trasmessi e quelli ricevuti fra i server.

[1] *FIFO: Acronimo di **F**irst **I**n **F**irst **O**ut.*
In questo caso s'intende una coda come struttura dati che implementa tale caratteristica.

La trattazione [4] definisce come le liste in Redis permettano di memorizzare e manipolare serie omogenee di valori associati a una data chiave.
È possibile infatti aggiungere valori ad una lista, ottenere il primo o l'ultimo elemento e infine manipolare valori a un dato indice. Le liste inoltre mantengono l'ordine d'inserimento e forniscono efficienti operazioni basate sull'indice.
Detto questo, ulteriori aspetti che hanno favorito il suo utilizzo sono senza dubbio la gestione della memoria persistente e volatile. Redis è un database in memoria persistente che esegue un'immagine su disco del *datastore* in base al numero di chiavi cambiate. In particolare, è possibile configurare in modo che se X è il numero di chiavi che cambiano, allora avvenga un salvataggio ogni Y secondi. Di default Redis salverà il database ogni 60 secondi se 1000 o più chiavi sono cambiate, oppure ogni 15 minuti se sono cambiate nel frattempo 9 chiavi o meno.
Per quanto riguarda la memoria volatile, Redis mantiene tutti i propri dati in memoria, tuttavia non è il caso di preoccuparsi, poiché la gestione di grandi dati può occupare pochi MB di spazio. Inoltre, per quanto riguarda la scalabilità, molte soluzioni tendono ad essere limitate dall'input-output o dalla potenza di calcolo più che dalla RAM nella gestione dei dati.
Altri aspetti importanti di Redis sono l'atomicità e la gestione dei backup.
Redis è *single-threaded*[2], ciò garantisce l'atomicità di ogni singola operazione e quindi i comandi verranno eseguiti sequenzialmente e potranno essere attuati solamente da un singolo client per volta.
Per quanto riguarda il backup, Redis salva l'istantanea del database nel file *dump.rdb*. Di fatto è possibile salvaguardare tale file attraverso meccanismi come un *ftp* o *scp* durante l'esecuzione di Redis.

2.1.1 Installazione e Configurazione

Diamo ora uno sguardo pratico all'attuazione di Redis nel sistema di Log Analisi. Con riferimento alla figura 1.3 si nota come la base di dati sia stata istanziata nella macchina Indexer, la ricevitrice d'eventi. Eseguendo semplici comandi in Linux si è scaricata una versione stabile dell'applicativo.

wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make

[2] *Con* ***single-threaded*** *s'intende la facoltà del programma di eseguire un compito alla volta.*

Notare come il comando *make* instauri già tutte quei meccanismi di auto start del server Redis all'avvio di sistema. Una volta eseguiti i comandi nella macchina di Indexer è sufficiente avviare Redis attraverso il comando: *redis-server*
Ora è possibile lanciare un client di Redis per assicurarci che il server risponda correttamente. Per farlo è sufficiente eseguire un ping alla macchina in locale dove risiede il server Redis.

redis-cli
redis 127.0.0.1:6379> ping
PONG

Il redis-cli è un ottimo strumento per svolgere alcune prove pratiche, per esempio potremmo creare una nuova chiave con un valore per poi richiederne quest'ultimo:

redis 127.0.0.1:6379> set chiave_di_prova ciao_mondo
redis 127.0.0.1:6379> get chiave_di_prova
"ciao_mondo"

Il server Redis resta, sotto forma di processo di sistema, in ascolto di domande o richieste da parte dei clienti. Le istanze di LogStash si comportano proprio come quest'ultimi.
Prendendo a prima vista il file di configurazione dei trasmettitori, nell'appendice A.1 alle righe 12-16, notiamo che i LogStash shipper mettono in coda gli eventi alla chiave "drivefarmtest_logstash" del server Redis hostato all'ip "ecXXXXXXX.eu-west-1.compute.amazonaws.com" (Per questioni di privacy aziendale).
Tratteremo LogStash nella sezione 2.2, ma quello che ci interessa sapere è la possibilità di eseguire una diagnostica via redis-client per osservare se effettivamente i trasmettitori inviano dati a Redis.

redis-cli
redis 127.0.0.1:6379> llen "drivefarmtest_logstash"
redis 127.0.0.1:6379> 10

Questo esempio ci permette di capire che i LogStash shipper hanno fatto una queue di 10 eventi alla chiave "drivefarmtest_logstash" e non sono stati ancora estratti dall'Indexer. Questo trumento si è rilevato utile in fase di debug per comprendere se vi era comunicazione tra il server centrale e i trasmettitori d'eventi.

Passando all'appendice A.2 alle righe 2-7, notiamo l'input che estrae gli eventi da Redis effettuando una dequeue. Una semplice prova con redis-cli ci consente di osservare che i valori sono stati estratti secondo la convenzione della FIFO e sono arrivati a destinazione.

redis-cli
redis 127.0.0.1:6379> llen "drivefarmtest_logstash"
redis 127.0.0.1:6379> 0

Una nota importante riguarda la sincronizzazione fra i server. Molto spesso si è lavorato sul server indexer, mentre i shipper raccoglieva gli eventi.
Due gli aspetti da tenere a mente:

- Prima di tutto, in un ambiente asincrono, si verificano spiacevoli congestioni, in particolare si è riscontrato che il server Redis riceve gli eventi trasmessi e quando si avvia il LogStash Indexer, allocato nella stessa macchina, gli eventi potrebbero non venir raccolti tutti, lasciando dei valori in memoria. Per ovviare a ciò si è dovuto forzare la loro rimozione attraverso il comando di redis-cli: *DEL "drivefarmtest_logstash"*

- Il secondo punto riguarda la lettura degli eventi che molto spesso non rispettano i filtri applicati dal file di configurazione di LogStash. Molto probabilmente si tratta di un'inconveniente dovuto al sovraffollamento d'eventi in ricezione non appena il LogStash effettua la lettura. Gli effetti più marcati si sono osservati nel filtro multiline che tronca prematuramente la concatenazione d'eventi.

2.2 LogStash

LogStash, intravisto nella sezione 1.2.2, è un software per la Log Analisi gratuito (*L. Apache 2.0*) sviluppato principalmente da Jordan Sissel della DreamHost. Scritto in JRuby, viene eseguito attraverso una Java Virtual Machine e proposto agli utilizzatori sotto forma di un pacchetto *Jar* pronto per l'utilizzo immediato. La sua particolare progettazione favorisce una semplice configurazione attraverso un file *.conf* ed estendibilità in ambienti scalabili, poiché è in grado di processare

eventi di svariato genere: Syslog, Microsoft EventLogs, STDIN ecc...
Questi eventi possono essere manipolati da particolari filtri all'interno di LogStash. Una volta modificati assumono la forma di veri e propri Log, caratterizzati da campi che ne indentificano la formattazione [1].
Terminato il processo di manipolazione i Log vengono espulsi sugli output, anche questi di diverso genere: TCP/UDP, email, files, HTTP, Nagios, ecc...
LogStash promuove la sua progettazione attraverso livelli di *Information Hiding*, i livelli architetturali sono separati e lavorano attraverso interfacce interconnesse. Il settore più alto è quello dei plugin (input, filter e output), in questo modo un programmatore alle prime armi può facilmente estendere le funzionalità di LogStash attraverso l'utilizzo di metodi predefiniti e semplici comandi Ruby.
Il file di configurazione *.conf*, detto agente o evento di pipeline, contiene la definizione delle fasi di input, filter e output che si vogliono utilizzare all'avvio di LogStash. Gli eventi che incombono vengono passati ad ogni fase attraverso delle code interne, in particolare delle *"SizedQueue"* in Ruby. Queste code fra fasi, gestite da Thread, hanno una capienza limitata e bloccano la scrittura di ulteriori eventi raggiunta la capacità limite. (Fig 2.1)
Logstash imposta ogni dimensione della coda a 20. Questo significa che solo 20 eventi possono essere in sospeso nella fase successiva, ciò aiuta a ridurre la perdita di dati ed in generale evita un sistema di *cache* che saturi la memoria.
Nella fase di output, se un plugin sta incorrendo malfunzionamenti, il Thread attende che l'uscita torni disponibile e quindi blocca la lettura da quell'uscita.
La conseguenza è il blocco anche in scrittura della coda d'uscita, se gli output non ritornano disponibili, forzando il lavoro sulla coda di filter e così in ridondanza fino agli input. In questo modo si instaura un meccanismo per salvaguardare la congestione dovuta ai malfunzionamenti interni.
Di fatto, in circostanze ideali, questo si comporta similmente alla finestra tcp quando chiude a 0, nessun nuovo dato viene inviato perchè il ricevitore non ha terminato l'elaborazione della coda di dati corrente.
LogStash assegna ad ogni input un Thread per fa si che gli ingressi più veloci non vengano bloccati dai più lenti. I filtri vengono applicati agli eventi in modo sequenziale alla loro definizione nel file di configurazione, questo per limitare l'utilizzo della CPU, poiché questi plugin tendenzialmente non utilizzano solamente un Thread ciascuno. Ovviamente anche per gli output vale la regola di un Thread con l'aggiunta di una coda per ciascuno.
Una nota importante è senza dubbio l'utilizzo della CPU e della memoria dovuta alla mole di Thread che svolgono le diverse operazione. Gli sviluppatori

raccomandano sempre di eseguire frequenti diagnostiche utilizzando *jstack* per comprendere se eventuali sovraccarichi di risorse sono dovuti al mal istanziamento dei svariati plugin.

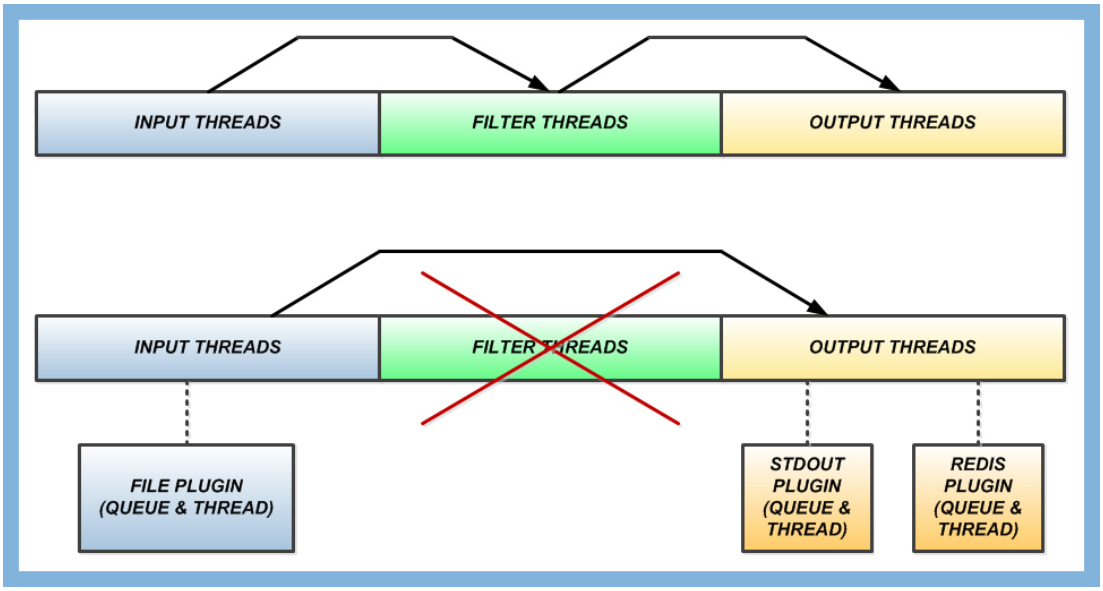


Figura 2.1: Nella figura sopra, la normale comunicazione fra i thread di Logstash. Sotto: l'esempio di comunicazione fra i thread sulla configurazione dei shipper.

2.2.1 Installazione e Configurazione

Definita l'importante logica che gestisce gli eventi in LogStash, ci accingiamo a definire la procedura, attraverso semplici comandi Linux, per l'attuazione del software sul sistema di Log Analisi.
Prima di tutto è necessario osservare che LogStash è un software molto giovane in costante evoluzione e la versione utilizzata per il progetto è la 1.1.9.
In particolare, gli sviluppatori, prima della versione 1.10, rilasciavano due versioni distinte del pacchetto jar: una di esse destinata a comprendere ogni singola feature, ma più lenta all'avvio, mentre l'altra, definita *flatjar*, più leggera e comoda nello sviluppo, ma assente di alcune parti di codice.
Dalla versione 1.10 gli sviluppatori hanno ben pensato di utilizzare il pacchetto performante *flatjar* con il codice completo inserito. Nel nostro caso la versione di LogStash completa è presente negli shipper, mentre nell'indexer vi è stata istanziata una versione flatjar, la spiegazione di questo passaggio verrà analizzata nel

capitolo 4 destinato allo sviluppo di parti aggiuntive per LogStash.
Il primo fondamentale passo è assicurarci di avere una versione di Java aggiornata nel nostro sistema. Dopodiché la scelta di quale pacchetto contemplare:

sudo wget http://logstash.objects.dreamhost.com/release/logstash-1.1.9-flatjar.jar

sudo wget https://logstash.objects.dreamhost.com/release/logstash-1.1.9-monolithic.jar

Arrivati a questo punto è necessario costruire un file .conf per testare il LogStash.
Sulla stessa directory dichiariamo il file simple.conf così:

```
1  input {
2    stdin {
3      type => "human"
4    }
5  }
6  output {
7    stdout {
8      debug => true
9    }
10 }
```

In questo codice si sono definiti due fasi: input e output.
Nella fase di input utilizziamo il plugin stdin per indicare al LogStash di leggere standard input (console di comando) gli eventi e dichiarali di tipologia *human*.
Nella fase output indichiamo al LogStash di espellere l'evento sullo standard output (console di comando), con la prerogativa di aggiungere della diagnostica di *debug*. Con questa semplice configurazione una stringa scritta sul terminale linux implica una risposta da parte di LogStash con il medesimo messaggio.
Ovviamente per lanciare LogStash con questa configurazione è sufficiente eseguire il seguente comando:

java -jar logstash-1.1.9.jar agent -v -f simple.conf

Il commando -f serve per caricare l'agente configurante, mentre il comando -v indica un verbose per aggiungere una lieve diagnosi interna del LogStash.
La diagnostica viene visualizzata attraverso la console di comando e per maggiori dettagli viene usata la prerogativa *-vv*. Con l'aggiunta di *–log diagnosi.log* alla fine della riga indichiamo di salvare gli eventi o diagnostica nel file *diagnosi.log*.

2.2.2 Analisi dello Shipper

Nell'appendice A.1 si trova il codice della configurazione dell'agente di LogStash per le macchine trasmettitrici, ne analizzeremo in dettaglio le singole parti.
Dapprima la fase di input (righe 1-7) e poi la fase di output(righe 8-17).
Nella fase d'input presentiamo *file* come plugin (righe 2-6).
Questo pacchetto consente di leggere i file contenenti stringhe di caratteri; particolarmente necessario per la lettura del server.log al percorso */opt/jboss/standalone/log/server.log*. Questo percorso punta al log del jBoss presente nelle macchine trasmettitrici dove è istanziato DriveFarm.
La voce *debug* consente di rilasciare eventuali messaggi nel Log di Logstash qual ora si verificassero inconvenienti, mentre il *type* definisce la tipologia di Log.
Tale tipologia è abbasta superflua, poiché non strettamente necessaria per un corretto funzionamento del sistema, ma è un identificativo abbastanza soggettivo. Definire alternativamente il tipo *foobar* non avrebbe influito, poiché il lavoro è svolto dai filtri applicati successivamente. Ciò nonostante si utilizza la tipologia *log4j* per correttezza del jBoss, anche se quest'ultimo non rilascia Log totalmente puri di questo formato. Le impurezze si riscontrano quando si analizzano software istanziati nell'Application Server che differiscono marginalmente dalla tipologia. Tuttavia, definire il tipo di Log consente di impartire un ordine insiemistico fra la grande mole di dati d'analizzare, in questo modo si facilita la modellazione degli eventi con i plugin più utili.
Nella fase di output vi è un *stdout* plugin (righe 9-11). Questa voce è accessoria, poiché veniva utilizzata in fase di debug. Ad ogni modo gli eventi, che vengono letti dal file, sono passati sia attraverso lo standard output sia attraverso gli altri plugin istanziati negli output a meno di particolari *tag* o *field* contemplati.
La voce interessante è *redis* (righe 12-15). Partendo dal resoconto fatto nella sezione 2.1.1 analizziamo il contenuto delle righe.

- *host* identifica la macchina indexer dove risiede l'istanza di Redis.
- *data_type* specifica il paradigma di trasmissione dati in Redis.
  List affronta una struttura di tipo FIFO fra trasmettitori e ricevitori.
  Effettua una enqueue dall'istanza dei shipper, mentre l'indexer incorre ad una dequeue per leggere i dati trasmessi.
- *key* identifica la chiave del database Redis.

Nella figura precedente, la 2.1, viene semplicemente esplicato il ciclo di vita degli eventi della configurazione attuata nei server trasmettitori.

2.3 ElasticSearch

ElasticSearch, sviluppato da Shay Banon e rilasciato ufficialmente nel febbraio 2010 sotto licenza open Apache 2.0, è un motore d'indicizzazione testo e di ricerca. La sua pecularietà nasce dall'instaurare un collegamento fra la mole dati testuali e dei termini specifici. Tali termini possono essere visti, banalmente, come l'indice di un libro e i loro riferimenti come le pagine di quest'ultimo. Le ricerche vengono compiute sui termini indicizzati consentendo un notevole risparmio prestazionale. Sotto la scocca ElasticSearch utilizza Apache Lucene; un API open source per il reperimento d'informazioni, implementata in Java da Doug Cutting e molto nota per la realizzazione di motori di ricerca.
In LogStash si utilizza ElasticSearch per indicizzare la mole di Log generati.
Gli indici di default sono giornalieri, in questo modo si preservano i dati d'interesse attraverso semplici sintassi. Gli eventi processati dal LogStash sono costituiti da campi, per esempio *@message*, *@tags* e *@type* e rappresentano un documento all'interno dell'indice. Più semplicemente, confrontando ElasticSearch ad un database relazionale, un indice rappresenta una tabella, un documento un tupla della tabella e un campo una colonna di quest'ultima [1], [3].
Come un database relazionale è possibile definire uno schema che in ElasticSearch viene definito *mappatura*. Ad ogni modo, gli schemi non sono strettamente necessari per compiere una ricerca all'interno degli eventi processati dal LogStash, ma consentono delle semplificazioni.
Entrando nel dettaglio nella struttura di ElasticSearch si denota che gli indici sono memorizzati in istanze di Lucene chiamati *shards* o frammenti.
Vi sono due tipologie di frammenti: *primario* e *replica*.

- I frammenti primari immagazzinano i documenti, ovvero gli eventi processati dal LogStash. Ogni nuovo indice dispone automaticamente di cinque frammenti primari ed è possibile variarne il numero prima della creazione degli indici, ma non dopo la loro creazione.

- I frammenti di tipo replica sono delle copie degli shards primari per conseguire due obiettivi.

  - Per proteggere i dati.
  - Per rendere le ricerche più veloci.

  Ogni frammento primario può disporre di una o più repliche e quest'ultimi possono variare dinamicamente anche dopo la loro creazione.

ElasticSearch è in grado di distribuire i frammenti fra i nodi disponibili e garantirne un indice nel caso fossero dislocati in settori differenti. I frammenti vengono memorizzati nei nodi, i quali appartengono ad un *cluster* di ElasticSearch.
I nodi sono interconnessioni d'informazioni riguardanti gli indici e possono utilizzare delle metodologie comunicative per investigare sul cluster in comune.
Se il cluster è condiviso fra i nodi vi è la particolarità di conseguire uno scambio d'informazioni fra istanze diverse di ElasticSearch. La distribuzione e la gestione di frammenti permettono ad ElasticSearch la riallocazione di quest'ultimi nel caso in cui i nodi dovessero incorrere a dei guasti. Di prassi vengono definiti un cluster e dei bizzarri nomi ai nodi (scelti fra 3000 candidati).

2.3.1 Installazione e Configurazione

Dopo un banale approccio alle trattazioni di ElasticSearch introduciamo i processi pratici che sono stati attuati nel server ricevente. Anche in questo caso diciamo che ElasticSearch evolve rapidamente ed è fondamentale che le versioni di LogStash siano bilanciate con quelle di ElasticSearch per non incorrere ad inconvenienti di trasmissione.
Attraverso semplici comandi Linux scarichiamo il pacchetto compresso.

sudo wget http://www.elasticsearch.org/download/

Una volta scompattato è sufficiente lanciare il processo da super user.

sudo ./elasticsearch

Da questo punto è possibile monitorare il processo da terminale.

ps aux |grep elastic

Oppure facendo una richiesta http alla porta locale 9200 (*http://LocalHost:9200*)
Nella cartella *config* di ElasticSearch vi è il file di configurazione *elasticsearch.yml*
In questo file è possibile gestire svariate feature quali memoria, connessione, numero dei frammenti, nodi e cluster. L'unica voce modificata è stata quella inerente al nome del cluster impostata come *DriveFarmElasticSearch*.
LogStash si connette ad ElasticSearch come un client, attraverso il cluster specificato ed il plugin istanziato nel file di configurazione, ed è possibile eseguire delle query da terminale per reperire delle informazioni dal database.

curl -XGET 'http://LocalHost:9200/_search?q=@type:human&pretty=true'

Questo comando reperisce i Log presenti in ElasticSearch di tipo *human*.
Ovviamente i comandi disponibili sono molteplici consentendo un buon lavoro anche in assenza d'interfaccia grafica.

2.3.2 Mappature

Sempre con l'aiuto della trattazione [1] si sono personalizzate le mappature.
In particolare si sono classificati gli eventi passati con un certo ordinamento.
Le mappature sono scritte in JSON e vengono applicate in ElasticSearch attraverso delle API.

```
curl –XPUT http://LocalHost:9200/_template/logstash_per_index -d '
{
    "template": "logstash*",
    "settings": {
      "index.query.default_field": "@message",
      "index.cache.field.type": "soft",
      "index.store.compress.stored": true },
    "mappings": {
      "_default_": {
         "_all": { "enabled": false },
         "properties": {
           "@message": { "type": "string", "index": "analyzed" },
           "@source": { "type": "string", "index": "not_analyzed"},
           "@source_host": { "type": "string", "index": "not_analyzed" },
           "@source_path": { "type": "string", "index": "not_analyzed" },
           "@tags": { "type": "string", "index": "not_analyzed" },
           "@timestamp": { "type": "date", "index": "not_analyzed" },
           "@type": { "type": "string", "index": "not_analyzed" }
         }
       }
    }
}'
RISULTATO {"ok":true,"acknowledged":true}
```

Nel Mapping Template si specificano tre parti fondamentali:

- *Template* rappresenta il nome dell'index da assegnare al template.
  Il fatto che sia specificato *logstash** sta ad indicare che tale nome sarà dinamico e quindi che si creerà un index a seconda delle giornate; ad esempio: *logstash27/07/13*.

- *Settings* rappresenta la configurazione da attuare al template.
  Prima di tutto si specifica che la ricerca di default dei tipi verrà svolta sulle tipologie *@message*. Si varia anche *field data cache* che verrà utilizzata soprattutto durante la ricerca o l'ordinamento su un campo.
  Tale voce può consumare molta memoria ed è responsabile in ElasticSearch, di tanto in tanto, della generazione di errori *OutOfMemory* nel LogStash. La si modifica con la voce *soft*, in questo modo ogni volta che ElasticSearch necessita di memoria verrà attivato il *garbage collector* della *cache*.
  Si imposta la *index.store.compress.stored* su *true*, ciò consente ad ElasticSearch di impostare il livello di compressione.
  In questo modo si memorizzeranno molti più eventi con una minimo utilizzo di prestazioni per la compressione/decompressione.

- *Mapping* istruisce ElasticSearch su come gestire alcuni particolari campi: il tipo di dato e suggerimenti su come i puntatori devono indicizzare e analizzare questi campi.
  Per impostazione predefinita LogStash svolge un buon lavoro, poiché interagisce con ElasticSearch, ma sono possibili delle migliorie. In primo luogo, ElasticSearch crea un campo chiamato *_all*. Questo campo facilita la gestione del tipo di dati per una mole onerosa di messaggi. Per esempio quando s'interroga ElasticSearch e non si conosce la struttura dei massaggi. Purtroppo questo meccanismo ha un costo di CPU, memoria e storage elevato. Per LogStash, in particolare tramite la sua interfaccia web o altri strumenti di query, questo campo non è necessario e la sua rimozione può estendere o migliorare le prestazioni di ElasticSearch. Si disabilita questa funzione, specificando il valore *false*.
  Essendo che ElasticSearch ricerca nel campo *_all*, di default, è necessario indicare ora quali campi si deve usare al suo posto. In setting si aggiorna *l'index.query.default_field* per specificare la tipologia *@message*; è necessario definire quindi la sua struttura all'interno di Mapping.
  Il fattore accomunante delle proprietà che si instaurano sono *type* e *index*.

  - Il *type* indica a quali dati ElasticSearch deve aspettarsi in quel campo. Questo migliora la pre-aspettative di ElasticSearch specificando in modo chiaro come debba comportarsi con alcuni campi.
  - L'*index* controlla se ElasticSearch analizzerà e *tokenizerà* i dati contenuti in quel campo. In questo caso li specifichiamo *not_analyzed*, in quanto questo lavoro lo svolgerà LogStash.

È possibile visualizzare il template appena creato alla pagina

http://LocalHost:9200/_template/logstash_per_index

Mentre per visualizzare la lista dei template presenti in ElasticSearch

http://LocalHost:9200/_all/_mapping?pretty=true

2.4 Kibana

Kibana, sviluppato da Rashid Khan, è un tool open source per la visualizzazione di eventi manipolati dal LogStash e indicizzati da ElasticSearch.
Scritto in Ruby e Java Script, utilizza il framework *Sinatra* per svolgere le proprie funzionalità di classificazione grafica. Purtroppo, fra i software visti fin ora per la Log Analisi, Kibana non dispone di documentazione tecnica esauriente.
Il software ricerca, attraverso delle query, le informazioni all'interno del cluster di ElasticSearch, mettendosi in ascolto della sua porta 9200.
I dati raccolti vengono visualizzati attraverso una pagina web sul proprio browser alla porta 5600 (*http:/LocalHost:5600*), fig.2.2. In tale pagina si possono svolgere delle query manualmente o effettuare ricerche in base al tempo o ai campi contemplati; inoltre si dispone di una sezione apposita per la visualizzazione in streaming di eventi raccolti da ElasticSearch.
In LogStash, inizialmente, si utilizzava un output plugin per la visualizzazione grafica degli eventi, tuttavia la continua fama di Kibana riscontrata come applicativo di supporto alla LogAnalisi ha indotto gli sviluppatori di LogStash ad operare sulla possibilità d'integrare Kibana all'interno della libreria di LogStash.
Kibana, per il progetto, ha rappresentato l'interfaccia in tempo reale per la visualizzazione dei Log, in questo modo i gestori aziendali possono essere al corrente di eventuali malfunzionamenti presenti o passati in cui è incorso DriveFarm.

2.4.1 Installazione e Configurazione

La procedura d'istanziamento di Kibana si è dimostrata più ostica del previsto. La causa è senza dubbio la corretta installazione di Ruby e Gems nel sistema e la preparazione dell'ambiente aggiornando le risorse disponibili.

sudo apt-get update

Nel caso in cui non si sia installato Java per ElasticSearch si esegue:

sudo apt-get install openjdk-7-jdk

Se non dovesse funzionare, utilizzare i comandi seguenti e ripetere il punto precedente. Con *java -version* ci assicuriamo che la versione giusta sia nel sistema.

sudo add-apt-repository "deb http://us.archive.ubuntu.com/ubuntu/ hardy multiverse"
sudo add-apt-repository "deb http://us.archive.ubuntu.com/ubuntu/ hardy-updates multiverse"
sudo apt-get update

Ora è necessario installare ruby, rvm (per gestire le varie versioni di ruby) e le gems; la prassi fondamentale è assicurarsi di eseguire i comandi che seguono con i privilegi di root. Il problema nasce dalla diversificazione di percorsi in cui si allocano i vari applicativi: ruby in un percorso utente, rvm in root e gem di nuovo in utente; tutto ciò causa innumerevoli disordini.
Con la prerogativa *sudo su* ci assicuriamo di eseguire i comandi con criterio.
Passiamo ad installare curl: *apt-get install curl*
Con il comando successivo si scarica una versione stabile di rvm sul percorso dello user (pur eseguendolo i comandi con il privilegio di root).
Insieme alla rvm vi è l'ultima versione di ruby disponibile e un set di gemme.

curl -L https://get.rvm.io |bash -s stable –rails

Se tutto si è svolto per il verso giusto allora il super user non troverà la rvm installata, ma una vecchia versione di ruby e nessuna gemma.
Nella parte user, invece, tutte i pacchetti appena scaricati e installati.
Con i comandi *rvm -v, ruby -v* ci assicuriamo dei risultati sperati, in particolare con *rvm list* troverete le varie versioni di ruby installate e quella in corretto uso.
Con il comando *gem list* la lista delle gemme installate, in caso di problematiche i comandi *gem env, ruby env, rvm env* osservano i percorsi d'installazione.
Per utilizzare Kibana sono necessarie le gemme *bundler* e *sinatra*.
La gem *bundler* dovrebbe essere stata installata con il set di gemme precedente, per la gems *sinatra* utilizzare il seguente comando (da user!)

rvmsudo gem install sinatra

Il comando *rvmsudo* consente d'installare le gems con i privilegi di user.
Ora è necessario procurarsi Kibana:

wget http://kibana.org/intro.html

Una volta scompattato, all'interno della cartella, è sufficiente lanciare:

Rvmsudo ruby kibana.rb

Da questo punto è possibile monitorare il processo da terminale.

ps aux |grep kibana

All'interno della cartella di Kibana vi è il file *KibanaConfig.rb*, aprendolo con un semplice editor di testo è possibile apportarvi delle modifiche.
La prima voce d'interesse è la direttiva ip del server di ElasticSearch.
Nel nostro caso abbiamo specificato ip della macchina di indexer, ma non è altro che la macchina locale. Pertanto è sufficiente specificare:

Elasticsearch = "127.0.0.1:9200"

Il passo successivo è definire su quale ip Kibana è in ascolto.
La scelta più consona sarebbe ascoltare solamente l'ip locale (dato che abbiamo ElasticSearch e Kibana sull'Indexer Server), tuttavia è meglio che sia in ascolto sulla maggior quantità di ip possibili.
In questo modo si potrebbero intercettare più eventi da altri fonti in futuro.

KibanaHost = '0.0.0.0'

L'ultimo passo fondamentale è modificare la voce *Primary_field*.
Nel template di ElasticSearch si è disabilitato il campo *_all* (vedi sezione 2.3.2), quindi è necessario avvertire Kibana del fatto che il campo principale della ricerca debba essere fatto su *@message*, ovvero il corpo del Log.

Primary_field = '@message'

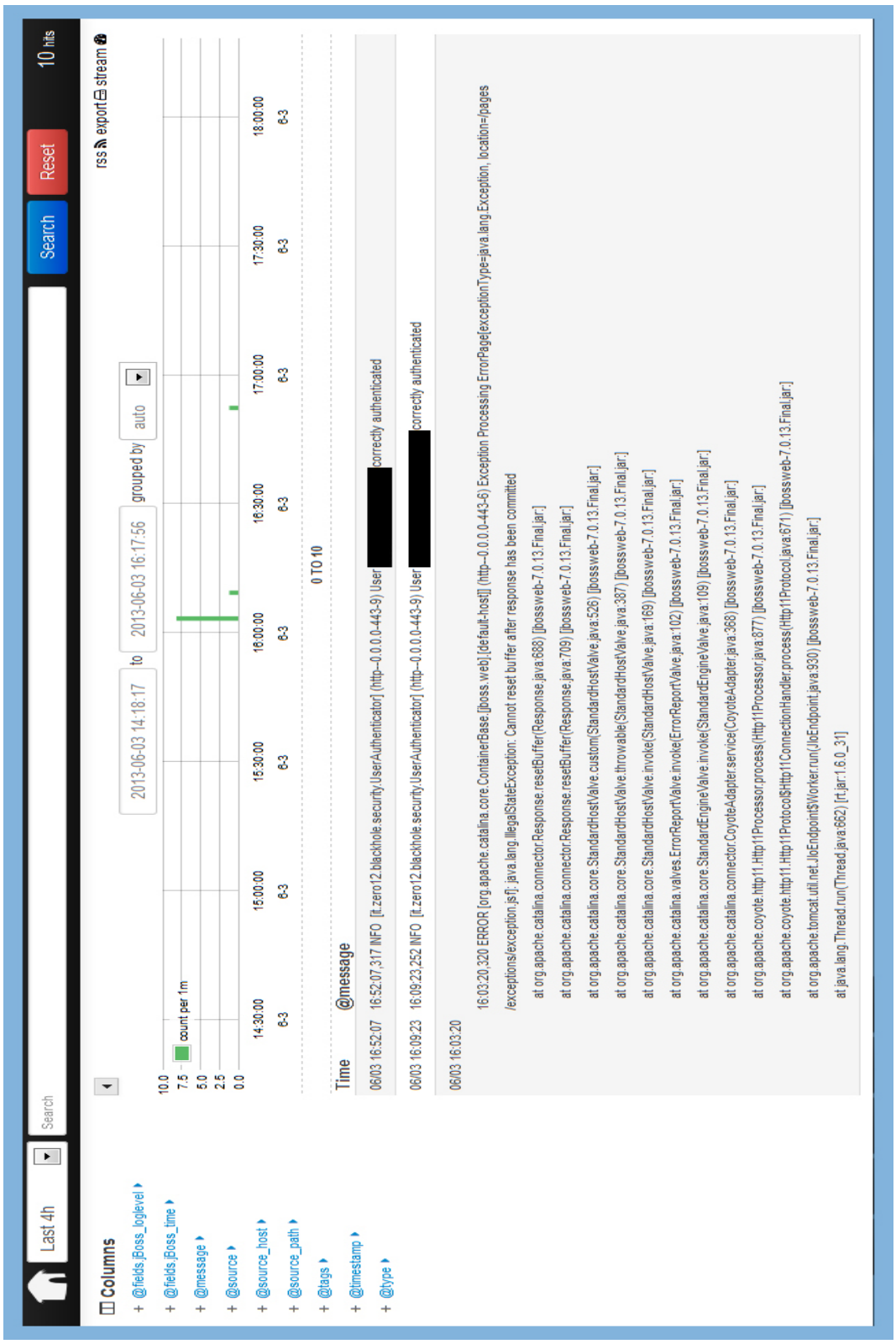


Figura 2.2: Un esempio di rappresentazione grafica dei Log in Kibana

Capitolo 3

Classificazione dei Log

3.1 Trasformazione dell'informazione

Durante il capitolo 2 si è parlato dei software utilizzati per la Log Analisi del progetto, tuttavia la vera problematica riguarda la classificazione dell'informazione che dobbiamo processare. In questa trattazione quando si parla di evento si definiscono quei dati informatici per la diagnostica, generati dagli applicativi, assenti di metadati identificatori e quindi affetti da incomprensione logica.
Nel caso generale i software espongono degli eventi già con una suddivisione abbastanza logica dei campi alla vista di un operatore, ma l'intelletto umano applica autonomamente dei filtri per stabilirne i campi. Gli applicativi per la Log Analisi devono essere istruiti per classificare gli eventi e quindi definirli Log.
In questo modo, oltre a recepire il concetto del messaggio di un evento, si possono effettuare delle ricerche sui campi attraverso ElasticSearch o Kibana.

3.1.1 Evento d'entrata

Fondamentalmente il progetto ha richiesto la suddivisione in campi degli eventi diagnostici generati dal jBoss. Tali eventi assumono una logica sequenziale che ne definiscono le caratteristiche principali quali: data e ora del messaggio, livello del Log, classe Java contemplata, ecc... Fortunatamente per noi questa suddivisione ci consente una più semplice identificazione dei campi, cosa che non capita con eventi poco dettagliati generati, per esempio, da applicativi amatoriali.
Si potrebbe definire il campo *@Tempo* che contempla la parte di stringa data e ora di un evento, oppure il *@Log_Level* che prende il *token* del livello del Log.
In LogStash questo lavoro viene svolto attraverso dei filtri programmati per compiere delle scelte sugli eventi. Tali filtri non servono solamente per definire i campi

di un evento, ma possono servire anche per marchiarli e indirizzarli in determinate uscite oppure eliminare o aggiungere informazioni supplementari.
LogStash, ElasticSearch e Kibana aiutano ulteriormente definendo automaticamente i campi: data e ora dell'arrivo del Log, la sorgente, host sorgente, il percorso della sorgente, il tipo, i marchi (detti *tags*) e il corpo del messaggio.
In questo modo possiamo già operare delle ricerche utili con delle query compiute attraverso Kibana o ElasticSearch.

3.1.2 Applicazione dei Filtri

Nel nostro caso gli eventi d'entrata, che vengono reperiti da Redis attraverso l'istanza del LogStash ricevitore, necessitano di essere sottoposti ad una serie di filtri per promuovere la classificazione desiderata.
Uno degli obiettivi proposti era quello di aggregare più eventi in successione in unico costrutto per salvaguardare l'integrità dell'informazione in ricezione.
Parliamo specificatamente di quegli eventi che descrivono uno *Stack Trace* in Java, ovvero una serie di riferimenti alle classi che incorono al malfunzionamento del programma. LogStash, senza filtri, analizza gli eventi e interfacciato con Kibana descrive lo Stack Trace come stringhe indipendenti e cronologicamente in successione. Attraverso l'attuazione di un filtro Multiline, descritto nella sezione 3.2, configurato attraverso un'appropriata espressione regolare abbiamo risolto questa necessità preliminare.
La trave portante di LogStash resta comunque l'utilizzo dei filtri Grok.
Attraverso questo plugin siamo in grado di suddividere in campi una particolare tipologia d'eventi. Ecco quindi come entri in gioco la definizione del tipo d'informazione da recepite nella fase di input. In questo modo un utilizzatore che debba svolgere una suddivisione in campi, di diversi eventi, potrebbe applicare il filtro Grok su una determinata tipologia, presa da uno specifico input, ed effettuarne un'altra diversificazione per gli altri tipi. Ovviamente s'incappa spesso nella diversità dell'evento stesso all'interno della tipologia e per ovviare a questo è necessario l'applicazione di pre-filtraggi di tipo Grep per attuare dei tag e instradare gli eventi marchiati in opportuni Grok specifici. Tuttavia l'applicazione del filtro Grok nella pratica ha portato ad una serie di problematiche:

- Il primo punto riguarda l'ambiente di test. Per svolgere le prove è necessario modificare il file.conf e rilanciare LogStash. Il suo avvio è una spesa di tempo consistente a scapito dell'utilizzatore, da qui l'idea di utilizzare la versione performante di LogStash per restringere le tempistiche.

- In secondo luogo la costruzione del *pattern*[1], per la suddivisione dei campi, richiede una buona conoscenza sulla tipologia d'analizzare e sull'utilizzo delle espressioni regolari.
- Infine si è notato come la restrizione in dettaglio della suddivisione, in aggiunta alla diversificazione degli eventi, nella tipologia, sia proporzionale al numero di *grok_failure*.

Nel progetto la tipologia diagnostica utilizzata da jBoss è la *log4j*, si veda [7], pertanto è stato necessario modellare un filtro Grok per indentificarne i campi d'interesse. Fortunatamente è possibile risolvere le prime due problematiche attraverso un *Grok Debbuger* presente alla pagina: *http://grokdebug.herokuapp.com/* Questo strumento consente di testare preventivamente i corretti *pattern* applicati ai nostri eventi senza incorrere al *grok_failure* e risparmiando le tempistiche dovute al lancio di LogStash.
Prendiamo, ad esempio, il seguente evento rilasciato da DriveFarm:

20:41:31,761 INFO [com.spaceprogram.simplejpa.EntityManagerFactoryImpl] (http–0.0.0.0-443-3) Scanning: file:/opt/jboss/jboss-modules.jar

Inserendolo nell'input del Grok Debbuger è possibile applicarne alcuni pattern. In primo luogo abbiamo un tempo ed è quindi possibile istaurare il pattern *%{TIME: jBoss_Time}*. *TIME* rappresenta il nome del pattern, mentre *jBoss_Time* è un nome scelto per il campo. Appena dopo vi si trova un LogLevel che identifica il grado diagnostico dell'evento scaturito da DriveFarm in esecuzione sul jBoss. Per questo campo applichiamo il pattern *%{LOGLEVEL: jBoss_loglevel}*. Potremmo procedere l'analisi applicando [*%{JAVACLASS}*], tuttavia continuando la suddivisione si entra nel punto tre delle problematiche, ovvero l'incremento di grok_failure che il filtro potrebbe incorrere (Non tutti gli eventi potrebbero avere questo formato). Inoltre, i requisiti progettuali non necessitavano un campo per ricerche sulla classe Java contemplata negli eventi, né dei campi successivi. Il risultato del Grok Debbuger, applicando la concatenazione dei pattern *%{TIME: jBoss_Time}* *%{LOGLEVEL: jBoss_loglevel}*, è la seguente:

```
1 {
2   "TIME": [
3     [
4       "20:41:31,761"
5     ]
```

[1] *Pattern identifica un insieme di espressioni regolari, corellate, che identificano un campo.*

```
6     ],
7     "HOUR": [
8       [
9         "20"
10      ]
11    ],
12    "MINUTE": [
13      [
14        "41"
15      ]
16    ],
17    "SECOND": [
18      [
19        "31,761"
20      ]
21    ],
22    "LOGLEVEL": [
23      [
24        "INFO"
25      ]
26    ]
27  }
```

3.1.3 Log d'uscita

Completato il processo di filtraggio, gli eventi mutano in Log e richiedono di essere espulsi attraverso opportuni output. Anche in questa fase è possibile instradare gli eventi manipolati attraverso la tipologia o i tag contemplati. In questo modo si suddivide il flusso di Log nelle uscite più consone. Le uscite sono scelte in base ai requisiti progettuali desiderati, per esempio viene usato, solitamente, *STDOUT* per la fase di debug. Nel nostro caso specifico si necessitava, oltre al comune output di ElasticSearch, un'uscita in grado di avvisare gli operatori rapidamente. Un primo plugin candidato fu *email*, ma sfortunatamente accantonato per future problematiche di spam con il server mail di Google. Si è deciso quindi di appoggiarsi ad Amazon per il servizio SNS, affinché inviasse le mail desiderate. Ovviamente questo è solo un esempio delle uscite contemplate dal progetto, un utilizzatore può scegliere un'ampia gamma di output. Per esempio *graphite* è un'uscita destinata ad utilizzare il tool suddetto per la visualizzazione di grafici statistici sulla quantità di Log ricevuti nell'arco del tempo.

3.2 LogStash: Analisi dell'Indexer

Nell'appendice A.2 si trova il codice della configurazione dell'agente di LogStash per le macchina ricevitrice, ne analizzeremo in dettaglio le singole parti. (Fig.3.1) Dapprima la fase di input (righe 1-8), poi la fase di filter (righe 9-100) e infine la fase di output (righe 101-183).
Nel blocco di input troviamo certamente il plugin *redis* (righe 2-7), poiché, come abbiamo detto, è necessario raccogliere eventi dal trasmettitore.
Al suo interno specifichiamo:

- *host =>"127.0.0.1"* Indichiamo che il Server Redis su cui reperire le informazioni di input risiede nella macchina locale, ovvero la macchina ricevitrice.

- *data_type =>"list"* Specifica il paradigma di trasmissione dati in Redis. List affronta una struttura di tipo FIFO fra trasmettitori e ricevitori. In questo caso s'incorre ad una dequeue per leggere i dati trasmessi.

- *key =>"drivefarmtest_logstash"* Specifica il database, in Redis, dove estrarre gli eventi.

- *type =>"redis-input"* Specifica la tipologia di input, ma in questo caso è forzato dal trasmettitore e viene imposto in log4j.

In filter, in ordine sequenziale, vi si trova il plugin *multiline* (righe 10-15).
Multiline è un filtro che consente di aggregare più righe di eventi, adiacenti, in un unico blocco. Solitamente questi filtri sono gestiti da una regola che consente un giusto raggruppamento degli eventi.
Nel caso particolare la sua funzione è strettamente legata all'unione di eventi riguardanti la generazione di uno Stack Strace in Java. Senza questo filtro il risultato di uno Stack Trace, in ricezione, sarebbe quello di eventi singoli distinti.
Al suo interno specifichiamo:

- *type =>"log4j"* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.

- *what =>"previous"* Indica che la regola di pattern dev'essere eseguita sulle righe ricevute precedentemente, in ordine temporale.

- *pattern => ["(^.+Exception: .+) |(^\s+at .+) |(^\s+... \d+ more) | (^\s*Caused by:.+)"]* Rappresenta la parte fondamentale del filtro.

Attraverso il pattern si decide la regola di raggruppamento tra gli eventi adiacenti. In questo caso, utilizzando l'appendice A.3, si sono definiti quattro blocchi:

- *(^.+Exception: .+)* Il carattere ″^″ indica l'inizio di una stringa, il ″.″ indica qualsiasi carattere, il ″+″ si riferisce ad una o più occorrenze di ″.″, ovvero di qualsiasi carattere. ″Exception: ″ la sottostringa per identificare l'inizio di uno Stack Trace. Detta in modo informale, questa espressione regolare, ricerca quelle stringhe che contengono la sottostringa ″Exception: ″ e partono da una nuova riga.
- *(^\s+at .+)* Il carattere ″\s″ ricerca uno spazio e il ″+″ che segue indica una o più occorrenze di spazi. ″at″ è la sottostringa per identificare la sequenza di uno Stack Trace, mentre ″.+″ indica una o più occorrenze di qualsiasi carattere. Detta in modo informale, questa espressione regolare, ricerca quelle stringhe caratterizzate da una sottostringa ″at″ preceduta da spazi e seguita da qualsiasi carattere, ma che non partano dall'inizio, cioè sono la continuazione di un a capo.
- *(^\s+... \d+ more)* Il carattere ″\s″ ricerca uno spazio e il ″+″ che segue indica un o più occorrenze di spazi. ″...″ è una sottostringa presente nella stringa analizzata, ″\d″ ricerca un numero e il ″+″ che segue indica un o più occorrenze di numeri. ″more″ è una sottostringa presente nella stringa analizzata. Detta in modo informale, questa espressione regolare, ricerca quelle stringhe caratterizzate da uno o più spazi seguiti da tre ″.″, uno o più numeri e una sottostringa ″more″. Non partono dall'inizio, ovvero sono la continuazione di un a capo.
- *(^\s*Caused by:.+)* Il carattere ″^″ indica l'inizio di una stringa, il carattere ″\s″ ricerca uno spazio e seguito da ″*″ indica 0 o più occorrenze di spazi. ″Caused by: ″ indica una sottostringa presente nella stringa analizzata, ″.+″ indica una o più occorrenze di qualsiasi carattere. Detta in modo informale, questa espressione regolare, ricerca quelle stringhe che partono dall'inizio, che possono avere degli spazi iniziali, ma che contengono la sottostringa ″Caused by: ″ seguita da una o più occorrenze di qualsiasi carattere.

L'unione dei blocchi genera un'espressione regolare composta, in grado d'indentificare eventuali Stack Trace e di classificarli con il tag *multiline*.
Ovviamente, per quanto sia specifica la sua costruzione, si può incorrere

ad un margine d'incertezza in alcuni dati che non rispecchiano i caratteri cercati, causando spiacevoli conseguenze all'unione degli eventi adiacenti.

In successione, appena dopo, vi si trova il filtro *grep* (righe 16-20).
Questo plugin è utile per rimuovere alcuni eventi o per aggiungere un tag specifico. In questo caso particolare è stato utilizzato per rimuovere alcuni eventi generati, apparentemente, dalla trasmissione dati in Redis. Tali eventi erano caratterizzati da un messaggio vuoto ed erano particolarmente fastidiosi visualizzati in Kibana. Al suo interno specifichiamo:

- *type =>"log4j"* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.
- *exclude_tags =>"multiline"* Specifica di non manipolare gli eventi caratterizzati dal tag multiline, ovvero eventi che sono stati raggruppati da un filtro multiline. Si è notato che così facendo il filtro grep non comprometteva l'integrità degli eventuali Stack Trace che si presentavano in ricezione.
- *math => ["@message", "\d"]* Indica quali eventi dobbiamo considerare, in questo caso è necessario rimuovere tutti quegli eventi caratterizzati dallo spazio all'interno del campo *@message*.

Passiamo quindi al filtro *Grok* (righe 21-24) descritto nella sezione 3.1.2 e specifichiamo:

- *type =>"log4j"* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.
- *pattern => ["%{TIME:jBoss_time} %{LOGLEVEL:jBoss_loglevel}"]* Indica che gli eventi ricevuti sono caratterizzati da un tempo seguiti da un loglevel. Così facendo si generano due nuovi campi: *@JBoss_time* e *@JBoss_loglevel* (presenti su Kibana). Ovviamente, eventi che non hanno tali caratteristiche saranno classificati *grok_failure* sul campo tag.

Notare com'è stato preceduto il filtro *multiline*. In questo modo si limita il più possibile il rischio di *grok_failure* causato esplicitamente da quegli eventi che non coincidono con l'espressione regolare (per esempio gli "...at..." che non hanno all'inizio un tempo e un loglevel).
I sei filtri grep successivi (righe 25-72) si differenziano dal predecessore, poiché non rimuovono particolari eventi, ma aggiungono un tag specifico. Questa implementazione può risultare superflua per i seguenti motivi:

- Si è usato un filtro grok per i loglevel, perchè utilizzare i grep per classificare gli errori attraverso il campo tag?
- Definire il loglevel sia su un campo che sul tag risultata ridondante.
- Usare il campo @Jboss_loglevel sarebbe stato più corretto.

Questi sono tutti punti veritieri. L'utilizzo dei grep è dovuta principalmente ai filtri e output che si sono realizzati successivamente e che contemplano il tag e non altri campi. Si è così fatto sia per semplicità, sia per contribuire con la comunità di LogStash e realizzare plugin altamente indipendenti e scalabili.
Inoltre, inizialmente, si era pensato d'identificare i loglevel in kibana e grok sembrava la soluzione più rapida ed efficiente.
Tornando al compito svolto da questi filtri grep, analizzeremo il primo di essi e specifichiamo:

- *type =>"log4j"* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.
- *match => ["@message","GRAVE"]* Indica che stiamo considerando tutti quegli eventi che contengono la sottostringa *GRAVE* all'interno del messaggio dell'evento .
- *exclude_tags => ["SEVERE", "ERROR", "WARN", "WARNING", "INFO"]* Indica che non si considerino gli eventi caratterizzati da questi tag.
  Questa tecnica ci permette di classificare i loglevel in modo gerarchico a seconda della sequenza di grep.
  Aggiungendo *exlude_tags* ci assicuriamo che se un evento contiene *GRAVE*, *INFO* o una sottostringa importante per la nostra classificazione, allora venga necessariamente classificata *GRAVE*, ovvero il loglevel maggiore.
  Per il grep successivo, *SEVERE* risulta il log level maggiore e così via.
- *remove_tag =>"multiline"* Indica che si va a rimuovere il tag multiline, essenziale per la classificazione da parte di alcuni output.
- *add_tag => ["GRAVE"]* indica che stiamo classificando l'evento con il tag *GRAVE*.
- *drop =>"false"* Indica che non si elimina l'evento combaciante con il match.

Notare come, nel definire la tipologia *log4j*, si escluda automaticamente gli eventi generati dal filtro *advisor*.

Infine, troviamo una serie di filtri advisor (righe 73-100).
Advisor è un plugin custom realizzato durante lo svolgimento del progetto e descritto nella sezione 4.3.3.
Il suo compito è quello di rilasciare, eventualmente, una copia del primo evento differente da quelli già ricevuti nell'arco del tempo ″time_adv″ e un secondo evento che riassuma la quantità di diversi eventi ricevuti una volta scaduto il ″time_adv″. Tutto ciò ripetendosi in modo ciclico allo scadere del tempo stabilito. Essendo le istanze di advisor molto simili fra loro ne analizzeremo una di esse e specifichiamo:

- *tags =>″WARN″* Indica che si considerino solamente quegli eventi contrassegnati con il tag WARN.
- *time_adv => 720* Indica il tempo, in minuti, dopo la quale inviare un evento riassuntivo sulla quantità dei diversi eventi ricevuti dal filtro.
- *send_first =>″false″* Indica di non creare una copia del primo evento differente ricevuto nell'arco di tempo *time_adv*.

In output, in ordine sequenziale, troviamo il plugin elasticsearch (righe 102-105). Elasticsearch ha la facoltà d'inviare gli eventi ad un sua istanza presente nel sistema. Specifichiamo:

- *type =>″log4j″* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.
- *cluster => DriveFarmElasticSearch* Si riferisce ad un cluster specifico di ElasticSearch.

Notare come, nel definire la tipologia *log4j*, si escluda automaticamente gli eventi generati dal filtro *advisor*.
Di seguito vi si trova una sequenza di outputs S3 (righe 106-165).
S3 è un plugin custom realizzato durante lo svolgimento del progetto e descritto nella sezione 4.3.4. Il suo scopo è quello di collocare, periodicamente, gli eventi in un bucket S3 di Amazon. Andiamo ad analizzarne uno di essi e specifichiamo:

- *access_key_id => XXX* Indica una chiave fornita da Amazon per usufruire dei suoi servizi.
- *secret_access_key => XXX* Indica la chiave segreta fornita da Amazon per usufruire dei suoi servizi.
- *endpoint_region =>″ eu-west-1″* Indica l'endpoint dei Server su cui si fanno le richieste di servizio.

- *bucket =>″it.zero12.logstash″* Indica il nome del bucket sul quale si vogliono collocare gli eventi.
- *size_file => 10000000* Indica in byte (circa 10 MB in questo caso) la dimensione massima dei vari pacchetti da spedire sul bucket. Se la taglia del file supera *size_file*, allora quest'ultimo verrà spedito sul bucket.
- *time_file => 60* Indica, in minuti, il tempo dopo il quale gli eventi, presenti nel file, vengono spediti sul bucket.
- *type =>″log4j″* Indica al filtro di manipolare solamente gli eventi riguardanti il tipo di dato specificato, ovvero log4j in questo caso.
- *tags => [″SEVERE″]* Indica che si considerino solamente quegli eventi contrassegnati con il tag SEVERE.

Infine troviamo una serie di output SNS (righe 167-182). SNS è un plugin per l'invio di mail attraverso i servizi Amazon ed è stato leggermente modificato nella sezione 4.3.2. In questo caso particolare il loro scopo è quello di inviare gli eventi generati dai filtri Advisor. Andiamo ad analizzarne uno di essi e specifichiamo:

- *access_key_id => XXX* Indica una chiave fornita da Amazon per usufruire dei suoi servizi.
- *secret_access_key => XXX* Indica la chiave segreta fornita da Amazon per usufruire dei suoi servizi.
- *region =>″ eu-west-1″* Indica l'endpoint dei Server su cui si fanno le richieste di servizio.
- *arn => XXX* Indica il topic di riferimento.
- *tags => [″advisor_info″]* Indica che si considerino solamente quegli eventi contrassegnati con il tag *advisor_info*, ovvero il resoconto del plugin. Per quando riguarda *advistor_first*, indica le copie d'eventi che vengono generate da advisor.
- *kiba_ip_port => [″xxx:5601″]* Indica un riferimento *http* al servizio kibana nel corpo del messaggio.

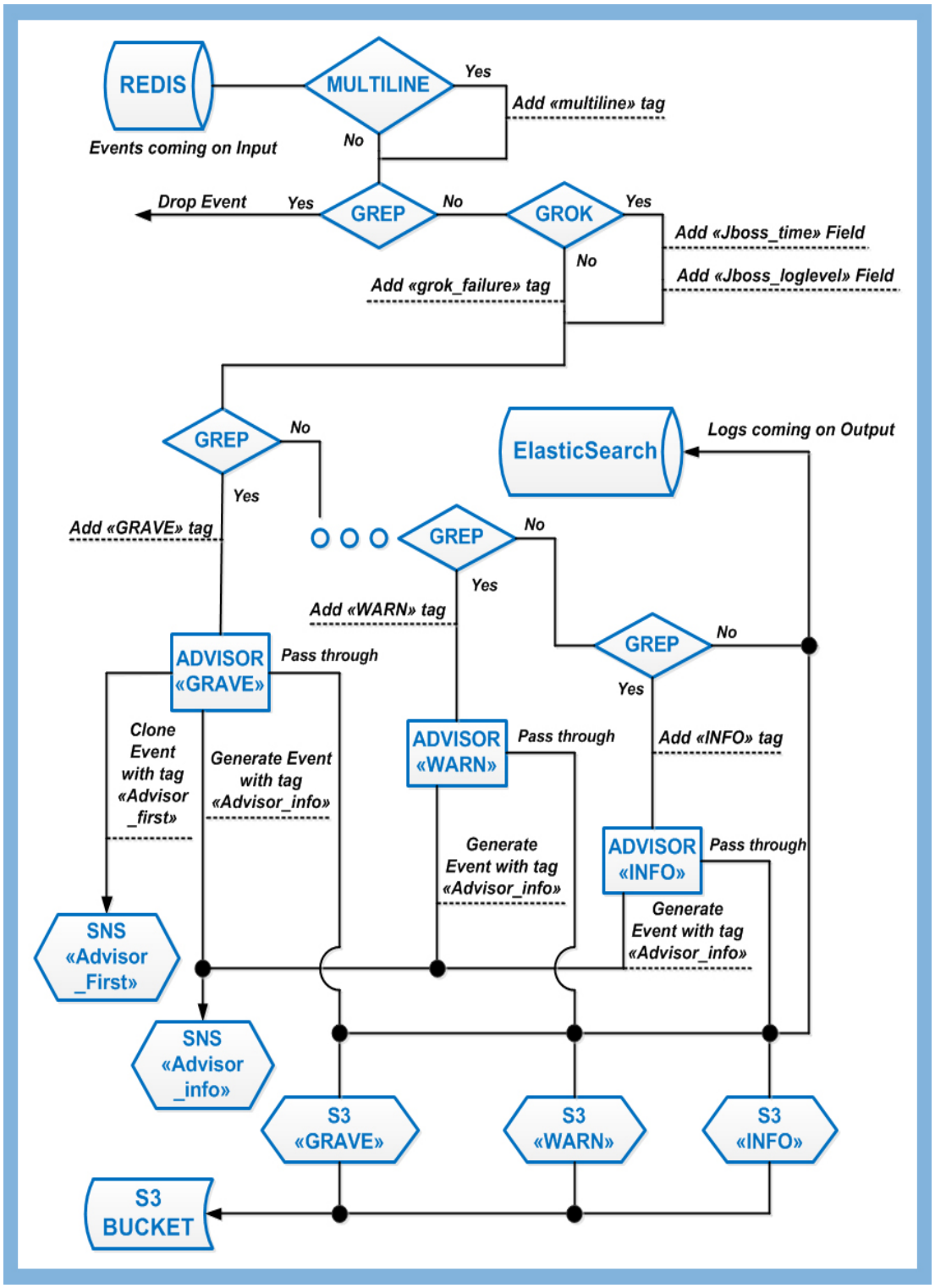


Figura 3.1: Rappresentazione della configurazione del LogStash Indexer.

Capitolo 4

Personalizzazione dell'ambiente

4.1 Scopo delle modifiche

Nella sezione 3.2 è stata analizzata la configurazione del LogStash ricevitore, introducendo i plugin realizzati per il progetto. Tali estensioni sono la conseguenza di due obiettivi:

- La necessità d'introdurre un sistema intelligente d'invio email, basato sulla priorità del LogLevel, che limitasse il sovraffollamento d'informazioni superflue.
- L'esigenza d'integrare un meccanismo per lo stoccaggio dei Log, affinché fosse possibile recuperare le informazioni in un secondo tempo.

Inoltre, essendo un progetto open source, si è pensato di realizzare le estensioni più generali e indipendenti possibili, affinché in un futuro fossero disponibili per tutta la comunità di LogStash.

4.1.1 Problematiche

Estendere LogStash, tuttavia, ha portato una serie di problematiche.
Dapprima la scelta della metodologia di sviluppo. Utilizzare il codice sorgente pre-compilato per attuare le modifiche e in un secondo tempo, una volta compilato il pacchetto, costatare i risultati ottenuti. Oppure, attraverso l'attuazione di una gerarchia di cartelle e un comando di lancio, consentire la lettura delle estensioni di LogStash al pacchetto già compilato.
Si è deciso di utilizzare quest'ultima soluzione, rinvenuta dalla trattazione [1], che specifica di creare una cartella denominata *logstash* e al suo interno una tripletta

di cartelle con i seguenti nominativi: *filters*, *outputs* e *inputs*. Ovviamente, i plugin scritti in Ruby, debbono essere collocati nelle cartelle suddette, rispettando il nominativo con le facoltà cui ricoprono. Una volta conseguito questo punto (per esempio *etc/logstash/(filters, outputs e inputs))* è possibile effetturare le prove con il comando:

java -jar logstash.jar agent -vv -f configurazione.conf –pluginpath /etc/

Una conseguente problematica, introdotta da questa scelta, riguarda la velocità d'esecuzione del LogStash. Più in generale: l'idea di utilizzare costantemente il pacchetto performante per i test da svolgere. Ovviamente tutto ciò ha portato alla modificazione del plugin SNS. In primo luogo perchè affetto da un bug riguardante la localizzazione della Region attraverso la chiamata ad una classe esterna *LogStash::PluginMixins::AwsConfig*. E, in un secondo tempo, estraniarsi da questa chiamata perchè non presente nel pacchetto performante della versione del LogStash utilizzata.

4.2 Introduzione al linguaggio Ruby

I plugin di LogStash sono scritti attraverso il linguaggio Ruby.
Ruby è un linguaggio di programmazione di alto livello, creato da *Yukihiro Matsumoto*, che combina la programmazione funzionale con quella imperativa.
La sua naturale semplità nasconde una considerevole complessità, poiché ogni costrutto viene considerato un oggetto. I tipi primitivi vengono considerati oggetti e nella dichiarazione di variabili non si necessita, obbligatoriamente, di specificare le informazioni riguardanti la loro tipologia. Una variabile di questo tipo assume forma una volta assegna ad un contesto. Altri aspetti di Ruby, che lo differenziano dagli altri linguaggi, sono: l'assenza di un carattere terminale di comando, ad esempio $''; ''$ e il potenziale rivolto alla manipolazione di stringhe e array.
Ulteriori peculiarità d'interesse sono: la definizione di metodi attraverso la clausola *def*, i quali ricevono come parametri delle variabili senza una tipologia specifica e la visibilità di quest'ultime attraverso il carattere speciale $''@''$.
Essendo Ruby sprovvisto di un carattere terminatore, ogni sezione deve terminare con il comando *end*.
Per lo svolgimento delle estensioni di progetto si sono utilizzate alcune guide, in particolare la trattazione [5].

4.2.1 Agilità di sviluppo

Nella sezione 4.1.1 si è discusso della necessità di utilizzare la versione performante di Logstash e il sistema gerarchico di sottocartelle per richiamare i plugin realizzati. Tuttavia, aggiungendo che lo sviluppo è stato conseguito su macchine in remoto senza interfaccia grafica, il ritardo di test è stato comunque considerevole. Per procedere più rapidamente si è utilizzato *irb*[1] per sviluppare, in console di comando, alcune parti del codice da integrare in un secondo tempo nelle estensioni.

4.3 Estendibilità del LogStash

La realizzazione delle estensioni per LogStash, attraverso classi scritte in Ruby, si basa sull'utilizzo di alcuni metodi specifici che dipendono dalla tipologia di fase che si vuole realizzare. Queste classi ereditano le funzionalità di base dalle classi che descrivono la loro fase. Per esempio nella fase di filter si utilizza *require "logstash/filters/base"* e *require "logstash/namespace"* per riferire alle classi che caratterizzano il filtro generico e la definizione dei costrutti come tag o type; chiaramente il plugin deve estendere la classe base per utilizzare quella riferita. Tutte le fasi sono contraddistinte dal metodo *register* che viene processato una volta istanziato ed eseguito il plugin. Al suo interno, solitamente, si specifica il codice che deve essere inizializzato; si possono specificare eventuali Thread che rimangono vigilanti fino alla chiusura, inaspettata o meno, del LogStash.
Nella fase di output si definisce il metodo *receive(event)*. Al suo interno si specifica quel codice che deve essere eseguito ogni qual volta il plugin riceva un evento. Molto similmente, nella fase di filter, si definisce il metodo *filter(event)* che attua il codice al suo interno ogni qual volta il plugin riceva un evento da filtrare.
Ovviamente quando si parla d'evento ricevuto s'intende quell'evento che rispetta i parametri del file di configurazione per quell'estensione.
In particolare, utilizzando la chiamata *return unless output?(event)* oppure *return unless filter?(event)*, a seconda della fase scelta, si scartano quegli eventi che non rispettano i parametri comuni, ad esempio tags e type.
Un metodo che si è dimostrato fondamentale per la costruzione del plugin Advisor è il metodo *flush*. Il codice definito al suo interno viene eseguito ogni cinque secondi ed è stato appositamente realizzato per incapsulare un nuovo evento nella coda principale di LogStash. Tale metodo può essere utilizzato solamente nella

[1] *irb: Acronimo di Interactive Ruby Shell*

fase di filter. Una possibile alternativa al metodo flush, asicrona, può essere attuata utilizzando la chiamata *yield* all'interno del metodo *def filter(event)* con cui è possibile clonare un evento nella coda principale.
La creazione di eventi da introdurre nella coda principale da parte delle altre fasi non è contemplata in Logstash. Una scelta poco consona consiste nel realizzare degli input e output che si appoggino ad una coda secondaria. In questo modo s'istaura una sorta di loop d'eventi fra le fasi, consentendo a quest'ultime la capacità di consumare gli eventi di ogni stadio; chiaramente scardinando il concetto stesso di passaggio fra fasi.
Nella costruzione di un'estensione resta lecito raccomandarsi di non svolgere del lavoro già realizzato visitando periodicamente il repository GitHub del codice sorgente [2]. In questo modo è possibile sia comprendere la struttura interna del pacchetto jar, sia apprendere le tecniche di realizzazione di un'estensione.

4.3.1 Modifica del filtro Multiline

Nell'appendice B.1.1 si trova il codice Ruby del filtro Multiline.
Il suo funzionamento è stato descritto nella sezione 3.2, mentre in questa parte si descrivono gli aggiornamenti apportati per il progetto.
Il fattore scatenante delle modifiche è dovuto, essenzialmente, all'arrivo di due o più Stack Trace simultaneamente e al delay che si manifesta nel conglomerare gli eventi causando un troncamento prematuro.
Grazie alla segnalazione *https://logstash.jira.com/browse/LOGSTASH-893*
su *Jira*, da parte dell'utente *luoxinwei1*, sono state aggiunte alcune righe di codice (le righe 120, 233 e modificata la 241). Risultato alla Fig.4.1
Dando uno sguardo al codice, notiamo come sia necessario, nei plugin, definire il riferimento alle variabili utilizzate nel file di configurazione (le righe 68,71 e 74).
La prima di esse: *config :pattern, :validate => :string, :require => true* definisce una variabile di tipo *string*, denominata *pattern* e deve essere definita obbligatoriamente, nel file di configurazione, affinché il plugin possa essere utilizzato.
La terminologia *default => false* indica che se non si specifica la variabile, nel file di configurazione, allora le verrà assegnato il valore *false*.

4.3.2 Modifica dell'output SNS

Nell'appendice B.1.2 troviamo il codice Ruby dell'output SNS.
Simple Notification Service è un servizio di Amazon per l'invio mail[10]; in particolare si possono utilizzare un massimo di 100 caratteri per il soggetto e 32768

byte per il corpo del messaggio, mentre non è supportato l'aggiunta di allegati. Queste limitazioni hanno favorito uno studio progettuale sulla concezione del filtro Advisor e sulla gestione degli errori del sistema di Log Analisi.
Il motivo delle modificazioni sono, essenzialmente, quelle citate nella sezione 4.1.1. La variabile che gestisce la *region* del *topic*, all'interno di *AwsConfig*, assumeva di default *US_EAST_1* e il plugin SNS non riusciva ad estrarre la localizzazione dall'*arn*, chiaramente generando un *exception* non trovando il topic negli USA. In conseguenza all'utilizzo del pacchetto performante non avente *AwsConfig*, si è deciso di non appoggiarsi più a quest'ultima classe presente in *PluginMixis*. *AwsConfig* viene ereditata da tutti quei plugin che fanno uso dei servizi Amazon e non è altro che una configurazione preliminare per stabilire la connessione. Quindi sono stati ridefiniti i parametri del plugin quali: *Aws access_key, Aws secret_access_key, etc...* che erano ereditati in *AwsConfig* e stabilita la connessione nel metodo *register* (righe 65-71, 76). Per ovviare ad inconvenienti di connessione, si controlla il riferimento all'oggetto ogni qualvolta arriva un evento da spedire (righe 114-116). Inoltre è stata aggiunta una variabile *kiba_ip_port* per introdurre, dal file di configurazione, l'ip e la porta dell'interfaccia grafica (righe 55, 116-119). In questo modo, una volta spedita una mail, ci si assicura che nel messaggio vi sia presente un http che punta alla pagina di kibana.
Un'altra modifica è stata compiuta nel formato *plain* (righe 138-144) dove è stata rimossa la variabile *field*, non utilizzata per il progetto. Notare come gli sviluppatori carpino il messaggio attraverso il metodo *slice* (righe 99,122, 132 e 142); ovviamente per rispettare la capienza di una mail SNS.

4.3.3 Realizzazione del filtro Advisor

Il filtro Advisor è un'estensione per LogStash, realizzata per il progetto, il cui scopo è quello di generare alcuni eventi nella coda principale di LogStash.
Il codice di realizzazione è presente nell'appendice B.1.3 ed è stato accuratamente commentato per essere condiviso con la comunità di LogStash.
Advisor ha la facoltà di classificare e contare una serie di eventi; svolge efficacemente le sue funzioni attraverso un filtraggio preliminare di tipo grep, poichè si ricevono degli eventi marchiati attraverso un tag. L'utilizzo del filtro prevede, in primo luogo, l'impostazione di un tempo *time_adv*. Nell'incombenza di un evento si controlla se è presente in un array; qual ora così non fosse e la modalità *send_first* fosse attiva, allora Advisor genera un clone dell'evento ricevuto e lo marchia con il tag *advisor_first*. L'evento generato verrà aggiunto alla coda principale di LogStash, nell'array e verrà aggiornato il contattore per quell'evento.

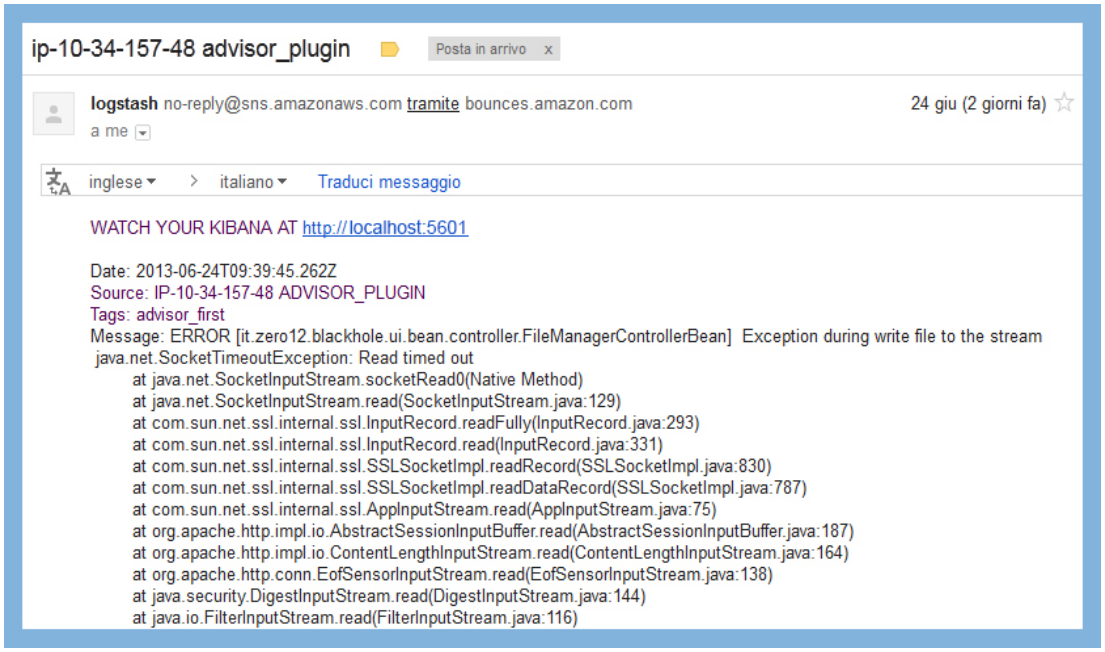


Figura 4.1: Esempio di mail generata dal plugin SNS per la clausola ″Advisor_first″, include Stack Trace gestito da Multiline.

Nel caso in cui la modalità *send_first* non fosse attiva, allora l'evento verrà aggiunto all'array e il contattore aggiornato. Di contro se l'evento incombente fosse già presente nell'array, allora verrà aggiornato semplicemente il suo contattore. Nel momento in cui il tempo *time_adv* scade, Advisor generarà un evento con il tag *advisor_info*. Il messaggio finale generato sarà un elenco del numero di eventi diversi ricevuti nel tempo *time_adv* con l'aggiunta di una descrizione carpiata del messaggio per quell'evento (questo per facilitare il limite di capienza di SNS).
L'evento generato verrà aggiunto alla coda principale di LogStash e sarà pronto per essere manipolato da altri filtri o output. Dopo che il tempo *time_adv* è scaduto, Advisor si comporterà come se fosse stato appena istanziato; re-inizializzando gli array ed interpretando un evento incombente come il primo di essi e quindi ripetendo la tessa procedura sopra descritta.
Nel caso specifico, qual ora si generasse un *ERROR*, allora l'idea è quella di impostare un filtro grep che aggiunga un tag all'evento, in primo luogo.
Successivamente una volta che l'evento marchiato incombe in Advisor, esso verrà clonato e spedito nella coda con il tag *advisor_first*. (Fig.4.1)
Allo scadere del *time_adv* si generarà un evento riassuntivo dei diversi errori con il tag *advisor_info*. Questi eventi verranno raccolti e spediti attraverso i plugin SNS in ascolto di log con i tag sopra descritti. (Fig.4.2)

Sono state aggiunte alcune righe personalizzate non presenti nella versione rilasciata alla comunità (righe 102-104). Il problema principale si riscontrava nel comprendere quando due log fossero stati uguali. Gli eventi ricevuti dal jBoss presentano un tempo che è diverso per ogni log, ovviamente. Quindi, banalmente, ogni evento è diverso perchè l'ora di generazione è differente. Si potrebbe usare un filtro *mutate* per rimuovere il tempo del jBoss nei log, ma questo eliminerebbe il tempo, anche graficamente, sul Kibana e sugli altri output. Utilizzando uno stratagemma, informale, si tronca la parte iniziale del messaggio; ovvero il tempo e si parte dal resto del testo. A questo punto si dispone di messaggi che potrebbero essere uguali fra loro, ma non lo sono perchè è possibile che essi siano generati da client diversi. Quindi è necessario rimuovere anche la richiesta tra parentesi, attraverso il codice della riga 103, con un'espressione regolare. Con il codice della riga 104 si rimuove la sottostringa *http* trovata ed a questo punto i messaggi potrebbero essere definiti uguali.
Notare come, infine, si potrebbe incappare in altri particolari che rendono dei messaggi apparentemente uguali necessariamente diversi. Un esempio concreto è l'utilizzo di versioni diverse di DriveFarm, in ambiente di Test o di produzione, le quali potrebbero generare eventi differenti solamente dalla disposizione di una riga di codice nel punto x, piuttosto che in un punto y.
Altre modifiche, non presenti nella release delle comunità, sono attuate nel corpo dell'evento generato *advisor_info* (righe 163-167), poiché è stato aggiunto l'eventuale percorso dei file nel bucket S3.

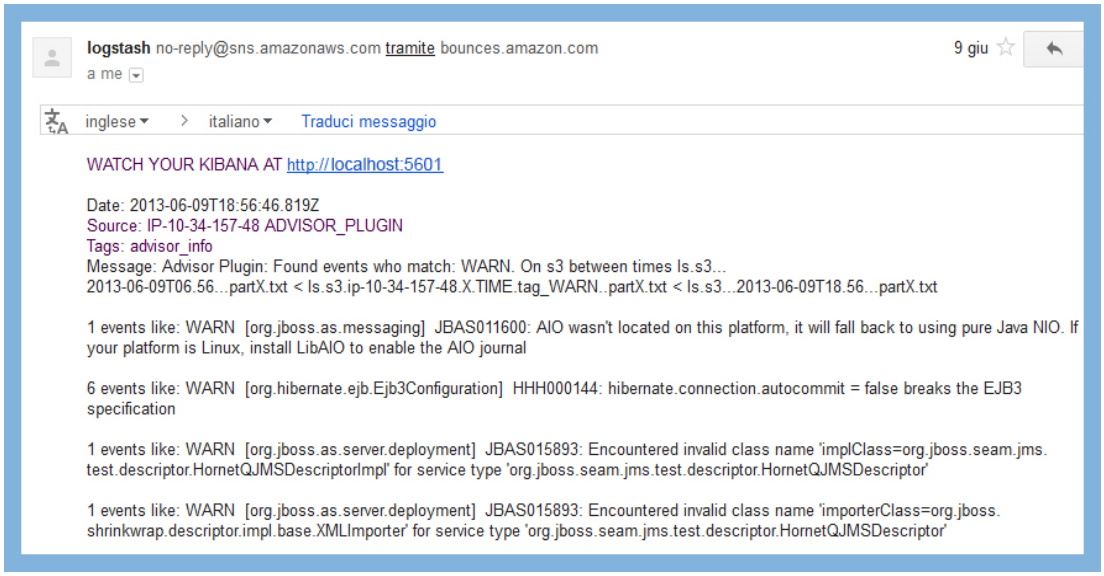


Figura 4.2: Esempio di mail generata dal plugin SNS per la clausola "Advisor_info"

4.3.4 Realizzazione dell'output S3

La necessità di realizzare un sistema per lo stoccaggio dei Log ha condotto alla realizzazione di un estensione che utilizzasse il servizio S3 di Amazon.
Simple Storage Service consente di impilare una serie di dati in bucket, residenti in server gestiti da tecnologia Cloud di Amazon [11].
Il codice di realizzazione è presente nell'appendice B.1.4 ed è stato accuratamente commentato per essere condiviso con la comunità di LogStash.
Nella realizzazione, come nel caso delle modifiche apportate ad SNS, è stata ridefinita la configurazione preliminare per stabilire la connessione. L'idea principale dell'estensione consiste nel generare dei file temporanei, in una sotto cartella presente in *opt/logstash/S3_temp*, dove scrivere i messaggi degli eventi ricevuti. Attraverso il file di configurazione si può stabilire una dimensione massima del file e un tempo dopo le quali il file stesso verrà inviato nel bucket.
Una volta spediti, i file temporanei verranno eliminati e sostituiti con nuove controparti. I file generati assumono una denominazione definita da uno standard, un esempio:

ls.s3.ip-10-228-27-95.2013-04-18T10.00.tag_hello.part0.txt

Pertanto la nuove controparti saranno caratterizzate dalla stessa denominazione delle precedenti, formato e data, se quest'ultime hanno fuorviato la dimensione massima stabilita, ovviamente *part0* verrà aggiornata con *part1*.
Viceversa, se l'ora stabilita è stata oltrepassata, verrà aggiornata l'ora o la data di creazione dei file. Naturalmente entrambi i concetti possono verificarsi e l'istanziare più estensioni di s3, che contemplano differenti tag, permette d'identificare i file più facilmente attraverso il marchio inciso appena dopo data e ora. (Fig.4.3)
Nel caso in cui il file fosse vuoto, esso rispetta la descrizione precedente, ma non verrà inviato nel bucket.
In aggiunta è stato implementato un meccanismo di *restore*, non obbligatorio. Tale sistema consente al plugin, una volta istanziato, di ricercare eventuali file temporanei nella cartella e di spedirli nel bucket. In questo modo, nel caso in cui si verificassero degli imprevisti che debilitassero il LogStash, il plugin sarà in grado di ripescare gli utili dati della vecchia sessione.
Una nota importante, per ragioni di inefficienza nello sviluppo, specifica di impostare la clausola *restore* solamente ad una istanza del plugin, nel caso in cui se ne utilisassero più di una, poiché si sono riscontrate delle interferenze di questo meccanismo con la generazione dei nuovi file temporanei all'avvio del sistema.

Problematiche di latenza o sequenziamento inducono inconsapevolmente alla cancellazione dei nuovi file generati, scambiati per vecchie sessioni).

```
?> AWS::S3::Base.establish_connection!(
?>   :access_key_id     => '                    ',
?>   :secret_access_key => '                                        '
>> )
=> #<AWS::S3::Connection:0x00000001c58568 @options={:server=>"s3-eu-west-1.amazonaws.com", :port=>80,
ecret_access_key=>"                                          "}, @access_key_id="                    ",
                  ", @http=#<Net::HTTP s3-eu-west-1.amazonaws.com:80 open=false>>
>>
?> AWS::S3::Bucket.find('it.zero12.logstash').each do |object|
?>         puts "#{object.key}\t#{object.about['content-length']}\t#{object.about['last-modified']}"
>> end
MyObjectKey     135     Fri, 12 Apr 2013 08:57:10 GMT
ls.s3.ip-10-228-178-176.2013-04-22T07.26.tag_INFO.part0.txt     100039  Mon, 22 Apr 2013 07:28:00 GMT
ls.s3.ip-10-228-178-176.2013-04-22T07.26.tag_INFO.part1.txt     100250  Mon, 22 Apr 2013 07:30:05 GMT
ls.s3.ip-10-228-178-176.2013-04-22T14.32.tag_ERROR.part0.txt    5947    Mon, 22 Apr 2013 14:34:33 GMT
ls.s3.ip-10-228-178-176.2013-04-22T14.49.tag_ERROR.part0.txt    2177    Mon, 22 Apr 2013 15:08:07 GMT
ls.s3.ip-10-229-131-136.2013-04-24T07.02.tag_ERROR.part0.txt    438     Wed, 24 Apr 2013 07:59:50 GMT
ls.s3.ip-10-229-131-136.2013-04-24T07.02.tag_INFO.part0.txt     85746   Wed, 24 Apr 2013 07:59:50 GMT
ls.s3.ip-10-229-131-136.2013-04-24T07.02.tag_WARN.part0.txt     8034    Wed, 24 Apr 2013 07:59:50 GMT
ls.s3.ip-10-229-131-136.2013-04-24T07.59.tag_ERROR.part0.txt    22526   Wed, 24 Apr 2013 08:59:51 GMT
ls.s3.ip-10-229-131-136.2013-04-24T08.59.tag_ERROR.part0.txt    29397   Wed, 24 Apr 2013 09:59:51 GMT
ls.s3.ip-10-229-131-136.2013-04-24T13.06.tag_ERROR.part0.txt    29880   Wed, 24 Apr 2013 14:06:23 GMT
ls.s3.ip-10-229-131-136.2013-04-24T13.06.tag_INFO.part0.txt     3204    Thu, 25 Apr 2013 07:03:20 GMT
ls.s3.ip-10-229-131-136.2013-04-24T16.06.tag_ERROR.part0.txt    34334   Wed, 24 Apr 2013 17:06:23 GMT
ls.s3.ip-10-229-131-136.2013-04-24T17.06.tag_ERROR.part0.txt    36748   Wed, 24 Apr 2013 18:06:23 GMT
ls.s3.ip-10-234-217-82.2013-04-26T10.43.tag_ERROR.part0.txt     32700   Fri, 26 Apr 2013 11:39:54 GMT
ls.s3.ip-10-234-217-82.2013-04-26T10.43.tag_INFO.part0.txt      44633   Fri, 26 Apr 2013 11:39:54 GMT
ls.s3.ip-10-234-217-82.2013-04-26T10.43.tag_WARN.part0.txt      1261    Fri, 26 Apr 2013 11:39:54 GMT
ls.s3.ip-10-234-217-82.2013-04-26T11.39.tag_ERROR.part0.txt     22369   Fri, 26 Apr 2013 12:39:54 GMT
ls.s3.ip-10-234-217-82.2013-04-26T11.39.tag_INFO.part0.txt      13584   Sat, 27 Apr 2013 07:03:46 GMT
ls.s3.ip-10-234-217-82.2013-04-26T13.39.tag_ERROR.part0.txt     33233   Fri, 26 Apr 2013 14:39:54 GMT
ls.s3.ip-10-235-45-134.2013-04-23T07.02.tag_ERROR.part0.txt     10913   Tue, 23 Apr 2013 08:02:50 GMT
```

Figura 4.3: Rappresentazione con s3sh dei file inviati, attraverso il plugin S3, al Bucket ″it.zero12.logstash″.

Capitolo 5

Automatizzare il sistema

5.1 Scripting in Linux

Dopo che sono state descritte le metodologie che hanno permesso il corretto svolgimento del progetto è importante instaurare quei processi che si occupano dell'autosufficienza del sistema di Log Analisi.
Il lancio di un processo, gergalmente demone, avviene attraverso degli script che richiamano il programma contemplato, affinché venga processato in background.
La scrittura di uno script avviene attraverso il linguaggio Bash[1][6].
Si tratta di un interprete di comandi che permette all'utente di comunicare col sistema operativo attraverso una serie di funzioni predefinite.
Bash permette di svolgere compiti complessi utilizzando variabili, funzioni e strutture di cotrollo di flusso; in grado di ricevere ed elaborare parametri esterni.
Una volta descritti gli script è necessario che essi vengano lanciati a livello di boot di sistema. In questo modo ci si assicura che il sistema di Log Analisi possa essere autosufficiente alla partenza del sistema e sia in grado di elaborare i dati nell'immediato

5.2 Intaurare i demoni

5.2.1 Processo LogStash

L'attuazione del processo LogStash prevede l'utilizzo del seguente script:

http://logstashbook.com/code/3/logstash-central.init

[1] *Bash: Acronimo di **B**ourne **a**gain **sh**ell.*

Collocato nella cartella *etc/init.d/* con la denominazione *logstash-central*.
All'interno dello script è fondamentale modificare i percorsi che referenziano al LogStash nel sistema:

logstash_conf = /etc/logstash/il_mio_file.conf
logstash_log = /var/log/logstash/log_del_logstash.log
logstash_bin = /usr/bin/java – -jar /opt/logstash/logstash.jar

Attraverso i seguenti comandi si abilita lo script delegandolo allo root user.

sudo chmod 0755 /etc/init.d/logstash-central
sudo chown root:root /etc/init.d/logstash-central

Da questo punto è possibile eseguire la procedura per caricare lo script sul boot di sistema. Nel caso in cui sia necessario eseguire operazioni manualmente in fase di sviluppo, allora questa serie di comandi possono rivelarsi utili per lanciare, fermare e monitorare il processo:

sudo /etc/init.d/logstash-central start
sudo /etc/init.d/logstash-central stop
etc/init.d/logstash-central status

5.2.2 Processo ElasticSearch

L'attuazione del processo richiamante ElasticSearch prevede l'esecuzione del comando *install* all'interno della cartella *bin/service*. In questo modo si andrà a collocare e caricare automaticamente lo script nella cartella *etc/init.d/*.
Da questo punto è possibile eseguire la procedura per caricare lo script sul boot di sistema. Nel caso in cui sia necessario eseguire operazioni manualmente in fase di sviluppo, allora questa serie di comandi possono rivelarsi utili per lanciare, fermare e monitorare il processo:

sudo /etc/init.d/elasticsearch start
sudo /etc/init.d/elasticsearch stop
sudo /etc/init.d/elasticsearch console

5.2.3 Processo Kibana

L'attuazione del processo Kibana prevede, come prerequisito, l'installazione delle gemme: sinatra, bundler e daemons nel caso in cui non fossero presenti.

rvmsudo gem install daemons
rvmsudo gem install sinatra
rvmsudo gem install bundler

Nel caso in cui si verificasse l'errore *tmp does not exist* è sufficiente creare una cartella denominata *tmp* all'interno di Kibana.
Ora è possibile lanciare il demone con il comando *rvmsudo ruby kibana-daemon.rb*.
Naturalmente è necessario scrivere un semplice script in *etc/init.d/* per la sua autonoma esecuzione:

#!/bin/bash
KIBANA_PATH = /etc/kibana
su - ubuntu rvmsudo ruby $KIBANA_PATH/kibana-daemon.rb start
RETVAL = $?
exit $RETVAL

Notare che si è usato *-ubuntu* e poi la chiamata rvmsudo; questo per rispettare le prerogative definite nella sezione 2.4.1.
Da questo punto è possibile eseguire la procedura per caricare lo script sul boot di sistema:

chmod +x kibana.sh
update-rc.d kibana.sh defaults

5.3 Autosufficienza del sistema

Il passo conclusivo riguarda la stesura degli script che consentono al sistema di rimanere operativo durante l'arco del tempo.
I processi potrebbero incorrere a malfunzionamenti e terminare la loro esecuzione, per questo motivo l'idea è quella di utilizzare il servizio *cron* di Linux affinché riabiliti il sistema in caso di necessità.
Nelle macchine trasmettitrici di progetto si necessita la stesura di uno script ri-

guardante solamente l'autosufficienza di LogStash.
Denominato *restore_logstash.sh*, in *etc/init.d*, si definisce:

```
#!/bin/sh
PROCESSFILE = 'logstash.jar'
if !([ `ps ax |grep -v grep |grep -ic $PROCESSFILE` -gt 0 ])
then sudo /etc/init.d/logstash-central start
fi
exit
```

Conseguentemente si attiva lo script: *chmod +x restore_logstash.sh.*
Attraverso la modifica di *crontab -e* si aggiunge la chiamata allo script ogni cinque minuti:

```
*/5 * * * * /etc/init.d/restore_logstash.sh
```

Nella macchina ricevitrice di progetto si necessita la stesura di uno script riguardante l'autosufficienza di LogStash, ElasticSearch e Kibana:

```
#!/bin/sh
PROCESSFILE1 = 'logstash.jar'
PROCESSFILE2 = 'elasticsearch'
PROCESSFILE3 = 'kibana'
if !([ `ps ax |grep -v grep |grep -ic $PROCESSFILE1` -gt 0 ])
then sudo /etc/init.d/logstash-central start
fi
if !([ `ps ax |grep -v grep |grep -ic $PROCESSFILE2` -gt 0 ])
then sudo /etc/init.d/elasticsearch start
fi
if !([ `ps ax |grep -v grep |grep -ic $PROCESSFILE3` -gt 0 ])
then sudo /etc/init.d/./kibana.sh
fi
exit
```

Infine si ripete la stessa procedura descritta precedentemente.
Notare come non si contempli nessuno script per Redis, poiché quest'ultimo istaura già un demone, auto avviante, particolarmente stabile nel lungo periodo.

Capitolo 6

Risultati

6.1 Collaudo del Sistema

Il giorno 06/04/2013 il sistema di Log Analisi per l'applicativo DriveFarm è entrato ufficialmente in funzione. I server trasmettitori, gestiti dall'AMI, hanno iniziato l'invio degli eventi all'istanza di Redis nella macchina ricevitrice (Fig.1.3). Una volta estratti dalla coda essi sono stati processati dal LogStash ricevitore ed espulsi attraverso le corrispettive uscite (Fig.3.1).
Una prima analisi, attraverso l'interfaccia Kibana, ha confermato il successo della classificazione degli eventi apportata attraverso i plugin Multiline, Grep e Grok. I filtri Advisor hanno clonato gli eventi con priorità, i quali sono stati inviati attraverso l'output SNS senza lasciare alcuna traccia in Kibana; avrebbero creato ridondanza d'informazione. Il resoconto ciclico dei diversi eventi, apportato da Advisor, è stato trasmesso anch'esso da SNS alla scadenza delle tempistiche stabilite, rimuovendo la traccia su Kibana sempre per motivi di ridondanza d'informazione. Distinto beneficio ha avuto l'introduzione del link nel plugin SNS affinché referenziasse alla pagina di Kibana, in questo modo si è potuto monitorare il sistema dai dispositivi, sia fissi che mobili, con la semplice osservazione delle mail segnalatrici. Gli stessi output S3 si sono dimostrati vincenti nel conseguimento dello stoccaggio degli eventi sul Bucket, definiti secondo le tempistiche indicate dalla priorità del Log.
Nelle successive due settimane si sono riscontrati alcuni problemi marginali.
In primo luogo si è osservato come il plugin Multiline, in rare occasioni, esegui l'aggregazione degli eventi in modo errato, in particolare nell'integrare eventi che contengono le parole chiavi delle espressioni regolari. Nel campione individuato si tratta di un Log Level INFO, contenente nel massaggio la sigla *exception*, che viene aggregato con un Log Level GRAVE incorso a Stack Trace.

Il secondo problema incombe ciclicamente ogni quattro-cinque giorni e si manifesta più rapidamente in proporzione allo scarso numero di Log ricevuti in un lasso di tempo considerevole. Tale problematica arresta accidentalmente il LogStash ricevitore fermando il flusso di dati; non è ancora certamente chiaro quale sia la causa scatenante.
Tra le supposizioni più accreditate si potrebbero indicare: un mancato riferimento alla connessione dei servizi Amazon S3 e SNS dei plugin. Oppure una chiusura inaspettata della connessione di Redis tra i server trasmettitori ed il ricevitore.
Attraverso il sistema di autosufficienza, descritto nella sezione 5.3, il processo *cron* riesce a riabilitare il LogStash efficacemente, ma la sua chiusura inaspettata si ripercuote sui filtri Advisor, specificatamente dal lato del temporizzatore dei messaggi aggregati. Se da un lato S3, con il sistema restore costruito, consente d'inviare la sessione precedente dei Log senza perdere le informazioni raccolte; dall'altro si smarrisce l'invio della mail riguardante i diversi eventi che si sono manifestati in quel lasso temporale. Tecnicamente, per bypassare questa problematica, s'imposta d'inviare la mail di resoconto su tempistiche più marginali, dell'arco di alcune ore, poiché in questo modo si abbassa la probabilità di perdita nel caso di chiusure inaspettate. L'effetto si è dimostrato più marcato per i resoconti dei Log Level INFO, poiché inviati ogni 24 ore di esecuzione.

6.1.1 Manutenzione

Allo scadere delle prime tre settimane di prova, oltre ai due problemi scritti nella sezione precedente, si è osservata un'altra anomalia del sistema.
Ogni mese l'interfaccia Kibana perde i riferimenti all'istanza di ElasticSearch. Dopo un'analisi superficiale si è dedotto che la causa scatenante fosse la considerevole mole di mapping effettuata da ElasticSearch. La procedura temporanea adottata prevede la cancellazione, nella cartella *data/* e *log/* di ElasticSearch, dei file e cartelle riguardanti il cluster specifico (*DriveFarmElasticSearch*).
Una volta conseguito questo passaggio è necessario riavviare l'istanza di ElasticSearch e secondariamente quella di Kibana per risolvere il problema originale.
Altri particolari non richiedono una manutenzione da parte di un operatore.
Resta sempre lecito considerare la quantità di file caricati nel Bucket S3 nel medio-lungo periodo e definire politiche che modifichino il file di configurazione per limitare o aumentare l'invio di mail.

6.2 Sviluppi futuri

Naturalmente il sistema di Log Analisi può essere esteso o migliorato ulteriormente. Tra i miglioramenti più apprezzabili:

- L'integrazione del sistema AMI per il server ricevitore, attraverso l'attuazione di un meccanismo per la gestione della referenza in Redis.
- Una procedura di sicurezza per l'autentificazione della pagina http di Kibana.
- La ricercare di soluzioni che possano limitare i malfunzionamenti riscontrati.
- La modifica del plugin Advisor affinché utilizzi una politica non volatile nella gestione delle informazioni, necessario per risolvere la chiusura inaspettata di LogStash.

Tra le estensioni più interessanti:

- La rielaborazione del file di configurazione di LogStash affinché possa elaborare tipologie di Log differenti da quelle contemplate dal jBoss.
- L'utilizzo di plugin, come *Graphite*, che consentano di stendere delle statistiche sulla quantità di eventi ricevuti nell'arco del tempo e sul utilizzo di CPU del sistema.
- Realizzare un meccanismo sicuro che nelle mail SNS referenzi al Bucket S3 e sia possibile la lettura dei file caricati dal sistema di Log Analisi.

6.3 Considerazioni

Il lavoro conseguito per l'attuazione del sistema di LogAnalisi presso *Zero12 s.r.l.* ha concesso la possibilità di apprendere una considerevole quantità d'informazioni in un ampio spettro di argomentazioni. Oltre alla necessità di comprendere i sistemi di LogAnalisi è stato fondamentale concepire i meccanismi alla base di Linux, la programmazione attraverso il linguaggio Ruby e l'utilizzo dell API per i servizi Amazon. Il bagaglio culturale inziale non era adeguato alle necesità del progetto e lo sviluppo ha dovuto subire dei rallentamenti, tuttavia le difficoltà hanno concesso una curva di apprendimento considerevole e i benefici si sono osservati al termine delle operazioni di collaudo. Ottimo il successo personale, poichè i plugin realizzati sono stati approvati come sperimentali dalla comunità di LogStash e saranno disponibili nelle prossime release ufficiali dell'applicativo.

Appendice A

Logstash File.conf

Il file di configurazione di LogStash è utilizza il formato JSON.
JavaScript Object Notation, specificato da Douglas Crockford (RFC 4627), è una notazione di facile lettura per l'interscambio di dati che deriva dal JavaScript, il linguaggio di scripting per la rappresentazione di strutture dati, chiamati oggetti.
Nel file di configurazione di Logstash viene ulteriormente semplificato con l'introduzione di tre blocchi specifici: Input, filter e output dove vengono elencati i requisiti che si vogliono contemplare dal software.

A.1 Shipper Configuration

```
1  input {
2    file {
3      type => "log4j"
4      path => ["/opt/jboss/standalone/log/server.log"]
5      debug => true
6    }
7  }
8  output {
9    stdout {
10     debug => true
11   }
12   redis {
13     host => "ecXXXXXXX.eu-west-1.compute.amazonaws.com"
14     data_type => "list"
15     key => "drivefarmtest_logstash"
16   }
17 }
```

A.2 Indexer Configuration

```
1  input {
2    redis {
3      host => "127.0.0.1"
4      type => "redis-input"
5      data_type => "list"
6      key => "drivefarmtest_logstash"
7    }
8  }
9  filter {
10   multiline {
11     type => "log4j"
12     pattern => ["(^.+Exception: .+)|(^\s+at .+)|(^\s+... \d+ more)|
13                 (^\s*Caused by:.+)"]
14     what => "previous"
15   }
16   grep{
17     type => "log4j"
18     exclude_tags => "multiline"
19     match => [ "@message", "\d" ]
20   }
21   grok{
22     type => "log4j"
23     pattern => ["%{TIME:jBoss_time} %{LOGLEVEL:jBoss_loglevel}"]
24   }
25   grep{
26     match => ["@message","GRAVE"]
27     exclude_tags => ["SEVERE","ERROR","WARN","WARNING","INFO"]
28     remove_tag => "multiline"
29     add_tag => ["GRAVE"]
30     drop => "false"
31     type => "log4j"
32   }
33   grep{
34     match => ["@message","SEVERE"]
35     exclude_tags => ["GRAVE","ERROR","WARN","WARNING","INFO"]
36     remove_tag => "multiline"
37     add_tag => ["SEVERE"]
38     drop => "false"
39     type => "log4j"
40   }
41   grep{
42     match => ["@message","ERROR"]
43     exclude_tags => ["SEVERE","GRAVE","WARNING","WARN","INFO"]
```

```
44      remove_tag => "multiline"
45      add_tag => ["ERROR"]
46      drop => "false"
47      type => "log4j"
48    }
49    grep{
50      match => ["@message","WARN"]
51      exclude_tags => ["SEVERE","GRAVE","WARNING","ERROR","INFO"]
52      add_tag => ["WARN"]
53      remove_tag => "multiline"
54      drop => "false"
55      type => "log4j"
56    }
57    grep{
58      match => ["@message","WARNING"]
59      exclude_tags => ["SEVERE","ERROR","GRAVE","WARN","INFO"]
60      remove_tag => "multiline"
61      add_tag => ["WARNING"]
62      drop => "false"
63      type => "log4j"
64    }
65    grep{
66      match => ["@message","INFO"]
67      exclude_tags => ["SEVERE","GRAVE","WARNING","WARN","ERROR"]
68      add_tag => ["INFO"]
69      remove_tag => "multiline"
70      drop => "false"
71      type => "log4j"
72    }
73    advisor {
74      tags => "GRAVE"
75      time_adv => 60
76    }
77    advisor {
78      tags => "SEVERE"
79      time_adv => 60
80    }
81    advisor {
82      tags => "ERROR"
83      time_adv => 60
84    }
85    advisor {
86      tags => "WARN"
87      time_adv => 720
88      send_first => "false"
```

```
89    }
90    advisor {
91      tags => "WARNING"
92      time_adv => 720
93      send_first => "false"
94    }
95    advisor {
96      tags => "INFO"
97      time_adv => 1440
98      send_first => "false"
99    }
100 }
101 output {
102   elasticsearch {
103     cluster => DriveFarmElasticSearch
104     type => "log4j"
105   }
106   s3 {
107     access_key_id => "XXX"
108     secret_access_key => "XXX"
109     endpoint_region => "eu-west-1"
110     bucket => "it.zero12.logstash"
111     size_file => 10000000
112     time_file => 60
113     restore => "true"
114     type => "log4j"
115     ags => ["GRAVE"]
116   }
117   s3 {
118     access_key_id => "XXX"
119     secret_access_key => "XXX"
120     endpoint_region => "eu-west-1"
121     bucket => "it.zero12.logstash"
122     size_file => 10000000
123     time_file => 60
124     type => "log4j"
125     tags => ["SEVERE"]
126   }
127   s3 {
128     access_key_id => "XXX"
129     secret_access_key => "XXX"
130     endpoint_region => "eu-west-1"
131     bucket => "it.zero12.logstash"
132     size_file => 10000000
133     time_file => 60
```

```
134       type => "log4j"
135       tags => ["ERROR"]
136     }
137     s3 {
138       access_key_id => "XXX"
139       secret_access_key => "XXX"
140       endpoint_region => "eu-west-1"
141       bucket => "it.zero12.logstash"
142       size_file => 10000000
143       time_file => 720
144       type => "log4j"
145       tags => ["WARN"]
146     }
147     s3 {
148       access_key_id => "XXX"
149       secret_access_key => "XXX"
150       endpoint_region => "eu-west-1"
151       bucket => "it.zero12.logstash"
152       size_file => 10000000
153       time_file => 720
154       type => "log4j"
155       tags => ["WARNING"]
156     }
157     s3 {
158       access_key_id => "XXX"
159       secret_access_key => "XXX"
160       endpoint_region => "eu-west-1"
161       bucket => "it.zero12.logstash"
162       size_file => 10000000
163       time_file => 1440
164       type => "log4j"
165       tags => ["INFO"]
166     }
167     sns {
168       access_key_id => "XXX"
169       arn => "arn:aws:sns:eu-west-1:924228905104:logstash"
170       secret_access_key => "XXX"
171       region => "eu-west-1"
172       tags => "advisor_info"
173       kiba_ip_port => "XXX:5601"
174     }
175     sns {
176       access_key_id => "XXX"
177       arn => "arn:aws:sns:eu-west-1:924228905104:logstash"
178       secret_access_key => "XXX"
```

```
179       region => "eu-west-1"
180       tags => "advisor_first"
181       kiba_ip_port => "XXX:5601"
182     }
183 }
```

A.3 Sintassi Espressioni Regolari

Metacaratteri	Descrizione
.	Indica qualsiasi carattere ad eccezione di quelli che identificano una riga nuova (\r e \n)
$	Identifica la fine di una riga
|	Indica una condizone OR
()	Identificano dei gruppi di caratteri
[]	Identificano intervalli e classi di caratteri
\	Annulla l'effetto del successivo metacarattere
^	Identifica l'inizio di una riga

Tabella A.1: I metacarattere delle espressioni regolari.

Quantificatori	Descrizione
*	Indica 0 o più occorrenze
+	Indica 1 o più occorrenze
?	Indica 1 o 0 occorrenze
{N}	Ricerca esattamente n occorrenze
{N,}	Ricerca al minimo n occorrenze
{N,M}	Ricerca al minimo n e massimo m occorenze

Tabella A.2: I quantificatori delle espressioni regolari.

Modificatori	Descrizione
i	Indica che la ricerca sarà case-insensitive
m	Indica che la ricerca verrà considerata per ogni riga
s	Indica che il testo verrà considerato come un'unica riga
u	Indica che verranno abilitati i caratteri Unicode estesi

Tabella A.3: I modificatori delle espressioni regolari.

Classi	**Corrispondente**	**Descrizione**
\w	[a-zA-Z0-9_]	Ricerca un carattere (numeri, lettere e '_')
\W	[^a-zA-Z0-9_]	Ricerca un non carattere, è la negazione di \w
\d	[0-9]	Ricerca un numero
\D	[^0-9]	Ricerca un non numero, è la negazione di \d
\s	[\t \n \v \f]	Ricerca uno spazio, tabulazioni e caratteri di fine riga
\S	[^\t \n \v \f]	È la negazione di \s
[:alpha:]	[a-zA-Z]	Ricerca caratteri alfabetici
[:blank:]	[\t]	Ricerca solamente spazi e tabulazioni
[:lower:]	[a-z]	Ricerca lettere minuscole
[:upper:]	[A-Z]	Ricerca lettere maiuscole
[:graph:]	[\x21-x7E]	Ricerca tutti i caratteri della tabella ascii [dal 33 (!) al 126 (~)]
[:punct:]	-	Ricerca tutti i caratteri di punteggiatura

Tabella A.4: Le classi di caratteri e i POSIX, delle espressioni regolari, utilizzati per specificare una serie di caratteri senza utilizzare i gruppi.

Ancora	**Descrizione**
$	Identifica la fine della stringa; l'aggiunta del modificatore /m indica la fine di ogni riga
\A	Identifica solamente l'inizio della stringa (Il modificatore /m è irrilevante)
\Z	Identifica solamente la fine della stringa (Il modificatore /m è irrilevante)
\b	Indentifica il punto tra due caratteri, che siano \w a sinistra e non \w a destra
\B	Indica l'opposto di \b
^	Identifica l'inizio della stringa; l'aggiunta del modificatore /m indica l'inizio di ogni riga

Tabella A.5: Le ancore delle espressioni regolari identificano la posizione in cui ricercare il testo desiderato.

Appendice B

LogStash Custom Plugin

B.1 Plugin Modificati

B.1.1 Multiline

```
1  # multiline filter
2  #
3  # This filter will collapse multiline messages into a single event.
4  #
5
6  require "logstash/filters/base"
7  require "logstash/namespace"
8  require "set"
9
10 # The multiline filter is for combining multiple events from a single source
11 # into the same event.
12 #
13 # The original goal of this filter was to allow joining of multi-line messages
14 # from files into a single event. For example - joining java exception and
15 # stacktrace messages into a single event.
16 #
17 # TODO(sissel): Document any issues?
18 # The config looks like this:
19 #
20 #     filter {
21 #       multiline {
22 #         type => "type"
23 #         pattern => "pattern, a regexp"
24 #         negate => boolean
25 #         what => "previous" or "next"
26 #       }
27 #     }
28 #
29 # The 'regexp' should match what you believe to be an indicator that
30 # the field is part of a multi-line event
31 #
32 # The 'what' must be "previous" or "next" and indicates the relation
33 # to the multi-line event.
```

```
34 #
35 # The 'negate' can be "true" or "false" (defaults false). If true, a
36 # message not matching the pattern will constitute a match of the multiline
37 # filter and the what will be applied. (vice-versa is also true)
38 #
39 # For example, java stack traces are multiline and usually have the message
40 # starting at the far-left, then each subsequent line indented. Do this:
41 #
42 #     filter {
43 #       multiline {
44 #         type => "somefiletype"
45 #         pattern => "^\s"
46 #         what => "previous"
47 #       }
48 #     }
49 #
50 # This says that any line starting with whitespace belongs to the previous line.
51 #
52 # Another example is C line continuations (backslash). Here's how to do that:
53 #
54 #     filter {
55 #       multiline {
56 #         type => "somefiletype "
57 #         pattern => "\\$"
58 #         what => "next"
59 #       }
60 #     }
61 #
62 class LogStash::Filters::Multiline < LogStash::Filters::Base
63
64   config_name "multiline"
65   plugin_status "stable"
66
67   # The regular expression to match
68   config :pattern, :validate => :string, :require => true
69
70   # If the pattern matched, does event belong to the next or previous event?
71   config :what, :validate => ["previous", "next"], :require => true
72
73   # Negate the regexp pattern ('if not matched')
74   config :negate, :validate => :boolean, :default => false
75
76   # The stream identity is how the multiline filter determines which stream an
77   # event belongs. This is generally used for differentiating, say, events
78   # coming from multiple files in the same file input, or multiple connections
79   # coming from a tcp input.
80   #
81   # The default value here is usually what you want, but there are some cases
82   # where you want to change it. One such example is if you are using a tcp
83   # input with only one client connecting at any time. If that client
84   # reconnects (due to error or client restart), then logstash will identify
85   # the new connection as a new stream and break any multiline goodness that
86   # may have occurred between the old and new connection. To solve this use
87   # case, you can use "%{@source_host}.%{@type}" instead.
88   config :stream_identity , :validate => :string, :default => "%{@source}.%{@type}"
89
```

```
90    # logstash ships by default with a bunch of patterns, so you don't
91    # necessarily need to define this yourself unless you are adding additional
92    # patterns.
93    #
94    # Pattern files are plain text with format:
95    #
96    #     NAME PATTERN
97    #
98    # For example:
99    #
100   #     NUMBER \d+
101   config :patterns_dir, :validate => :array, :default => []
102
103   # Detect if we are running from a jarfile, pick the right path.
104   @@patterns_path = Set.new
105   if __FILE__ =~ /file:\/.*\.jar!.*/
106     @@patterns_path += ["#{File.dirname(__FILE__)}/../../patterns/*"]
107   else
108     @@patterns_path += ["#{File.dirname(__FILE__)}/../../../patterns/*"]
109   end
110
111   public
112   def initialize(config = {})
113     super
114
115     @threadsafe = false
116
117     # This filter needs to keep state.
118     @types = Hash.new { |h,k| h[k] = [] }
119     @pending = Hash.new
120     @capture = Hash.new
121   end # def initialize
122
123   public
124   def register
125     require "grok-pure" # rubygem 'jls-grok'
126
127     @grok = Grok.new
128
129     @patterns_dir = @@patterns_path.to_a + @patterns_dir
130     @patterns_dir.each do |path|
131       # Can't read relative paths from jars, try to normalize away '../'
132       while path =~ /file:\/.*\.jar!.*\/\.\.\//
133         # replace /foo/bar/../baz => /foo/baz
134         path = path.gsub(/[^\/]+\/\.\.\//, "")
135       end
136
137       if File.directory?(path)
138         path = File.join(path, "*")
139       end
140
141       Dir.glob(path).each do |file|
142         @logger.info("Grok loading patterns from file", :path => file)
143         @grok.add_patterns_from_file(file)
144       end
145     end
```

```
146
147       @grok.compile(@pattern)
148
149       @logger.debug("Registered multiline plugin", :type => @type,
150       :config => @config)
151     end # def register
152
153     public
154     def filter(event)
155       return unless filter?(event)
156
157       if event.message.is_a?(Array)
158         match = @grok.match(event.message.first)
159       else
160         match = @grok.match(event.message)
161       end
162       key = event.sprintf(@stream_identity)
163       pending = @pending[key]
164
165       @logger.debug("Multiline", :pattern => @pattern, :message => event.message,
166                     :match => match, :negate => @negate)
167
168       # Add negate option
169       match = (match and !@negate) || (!match and @negate)
170
171       case @what
172       when "previous"
173         if match
174           event.tags |= ["multiline"]
175           # previous previous line is part of this event.
176           # append it to the event and cancel it
177           if pending
178             pending.append(event)
179           else
180             @pending[key] = event
181           end
182           event.cancel
183         else
184           # this line is not part of the previous event
185           # if we have a pending event, it's done, send it.
186           # put the current event into pending
187           if pending
188             tmp = event.to_hash
189             event.overwrite(pending)
190             @pending[key] = LogStash::Event.new(tmp)
191           else
192             @pending[key] = event
193             event.cancel
194           end # if/else pending
195         end # if/else match
196       when "next"
197         if match
198           event.tags |= ["multiline"]
199           # this line is part of a multiline event, the next
200           # line will be part, too, put it into pending.
201           if pending
```

```
202             pending.append(event)
203           else
204             @pending[key] = event
205           end
206           event.cancel
207         else
208           # if we have something in pending, join it with this message
209           # and send it. otherwise, this is a new message and not part of
210           # multiline, send it.
211           if pending
212             pending.append(event)
213             event.overwrite(pending.to_hash)
214             @pending.delete(key)
215           end
216         end # if/else match
217       else
218         # TODO(sissel): Make this part of the 'register' method.
219         @logger.warn("Unknown multiline 'what' value.", :what => @what)
220       end # case @what
221
222       if !event.cancelled?
223         filter_matched(event)
224       end
225     end # def filter
226
227     # Flush any pending messages. This is generally used for unit testing only.
228     public
229     def flush
230       events = []
231       @pending.each do |key, value|
232         # flushes keys that're not changed in last 5s
233         next unless value == @capture[key]
234
235         @pending.delete(key)
236         value.uncancel
237         events << value
238       end
239
240       # capture current keys in @pending
241       @capture = @pending.clone
242       return events
243     end # def flush
244   end # class LogStash::Filters::Date
```

B.1.2 SNS

```
1  require "logstash/outputs/base"
2  require "logstash/namespace"
3  require "aws-sdk"
4
5  # SNS output.
6  #
7  # Send events to Amazon's Simple Notification Service, a hosted pub/sub
8  # framework.  It supports subscribers of type email, HTTP/S, SMS, and SQS.
9  #
10 # For further documentation about the service see:
```

```
11 #
12 #   http://docs.amazonwebservices.com/sns/latest/api/
13 #
14 # This plugin looks for the following fields on events it receives:
15 #
16 #  * sns - If no ARN is found in the configuration file, this will be used as
17 #  the ARN to publish.
18 #  * sns_subject - The subject line that should be used.
19 #  Optional. The "%{@source}" will be used if not present and truncated at
20 #  MAX_SUBJECT_SIZE_IN_CHARACTERS.
21 #  * sns_message - The message that should be
22 #  sent. Optional. The event serialzed as JSON will be used if not present and
23 #  with the @message truncated so that the length of the JSON fits in
24 #  MAX_MESSAGE_SIZE_IN_BYTES.
25 #
26 class LogStash::Outputs::Sns < LogStash::Outputs::Base
27
28   MAX_SUBJECT_SIZE_IN_CHARACTERS  = 100
29   MAX_MESSAGE_SIZE_IN_BYTES       = 32768
30
31   config_name "sns"
32   plugin_status "experimental"
33
34   # Aws access_key.
35   config :access_key_id, :validate => :string
36
37   # Aws secret_access_key
38   config :secret_access_key, :validate => :string
39
40   # The `credentials` option is deprecated, please update your config to use
41   #`aws_credentials_file` instead
42   config :credentials, :validate => :string, :deprecated => true
43
44   # Message format.  Defaults to plain text.
45   config :format, :validate => [ "json", "plain" ], :default => "plain"
46
47   # SNS topic ARN.
48   config :arn, :validate => :string
49
50   config :region, :validate => ["us_east_1", "us-west-1", "us-west-2",
51                                 "eu-west-1", "ap-southeast-1", "ap-southeast-2",
52                                 "ap-northeast-1", "sa-east-1", "us-gov-west-1"],
53                                 :default => "eu-west-1"
54
55   config :kiba_ip_port, :validate => :string, :default => nil
56
57   # When an ARN for an SNS topic is specified here, the message
58   # "Logstash successfully booted" will be sent to it when this plugin
59   # is registered.
60   #
61   # Example: arn:aws:sns:us-east-1:770975001275:logstash-testing
62   #
63   config :publish_boot_message_arn, :validate => :string
64
65   def setUp
66     @sns = AWS::SNS.new(
```

```
67        :access_key_id => @access_key_id,
68        :secret_access_key => @secret_access_key,
69        :sns_endpoint => "sns.#{region}.amazonaws.com"
70      )
71    end
72
73    public
74    def register
75
76      setUp
77
78      # Try to publish a "Logstash booted" message to the ARN provided to
79      # cause an error ASAP if the credentials are bad.
80      if @publish_boot_message_arn
81        @sns.topics[@publish_boot_message_arn].publish("Logstash successfully booted",
82        :subject => "Logstash booted")
83      end
84    end
85
86    public
87    def receive(event)
88      return unless output?(event)
89
90      arn      = Array(event.fields["sns"]).first || @arn
91
92      raise "An SNS ARN required." unless arn
93
94      message = Array(event.fields["sns_message"]).first
95      subject = Array(event.fields["sns_subject"]).first || event.source
96
97      # Ensure message doesn't exceed the maximum size.
98      if message
99        message = message.slice(0, MAX_MESSAGE_SIZE_IN_BYTES)
100     else
101       if @format == "plain"
102         message = self.class.format_message(event)
103       else
104         message = self.class.json_message(event)
105       end
106     end
107
108     # Log event.
109     @logger.debug("Sending event to SNS topic [#{arn}]
110     with subject [#{subject}] and message:")
111     message.split("\n").each { |line| @logger.debug(line) }
112
113     # Publish the message.
114     if (@sns == nil)
115       setUp
116     end
117     if (@kiba_ip_port != nil)
118             message = "WATCH YOUR KIBANA AT http://"+@kiba_ip_port+"\n"+message
119     end
120
121     @sns.topics[arn].publish(message, :subject =>
122       subject.slice(0, MAX_SUBJECT_SIZE_IN_CHARACTERS))
```

```
123     end
124
125     def self.json_message(event)
126       json       = event.to_json
127       json_size = json.bytesize
128
129       # Truncate only the message if the JSON structure is too large.
130       if json_size > MAX_MESSAGE_SIZE_IN_BYTES
131         event.message = event.message.slice(0,
132         (event.message.bytesize - (json_size - MAX_MESSAGE_SIZE_IN_BYTES)))
133       end
134
135       event.to_json
136     end
137
138     def self.format_message(event)
139       message =  "\nDate: #{event.timestamp}\n"
140       message << "Source: "+("#{event.source}\n").upcase
141       message << "Tags: #{event.tags.join(', ')}\n"
142       message << "Message: #{event.message}
143       message.slice(0, MAX_MESSAGE_SIZE_IN_BYTES)
144     end
145 end
```

B.2 Plugin Realizzati

B.2.1 Advisor

```
1  require "logstash/filters/base"
2  require "logstash/namespace"
3
4  # INFORMATION:
5  # The filter Advisor is designed for capture and confrontation the events.
6  # The events must be grep by a filter first,
7  # then it can pull out a copy of it, like clone, whit tags "advisor_first",
8  # this copy is the first occurrence of this event verified in time_adv.
9  # After time_adv Advisor will pull out an event tagged "advisor_info"
10 # who will tell you the number of same events verified in time_adv.
11
12 # INFORMATION ABOUT CLASS:
13
14 # For do this job, i used a thread that will sleep time adv.
15 # I assume that events coming on advisor are tagged,
16 # then i use an array for storing different events.
17 # If an events is not present on array, then is the first
18 # and if the option is activate then  advisor push out a copy of event.
19 # Else if the event is present on array, then is another same event
20 # and not the first, let's count it.
21
22 # USAGE:
23
24 # This is an example of logstash config:
25
26 # filter{
27 #  advisor {
```

```
28 #      time_adv => 1                    #(optional)
29 #      send_first => true               #(optional)
30 #  }
31 # }
32
33 # We analize this:
34
35 # time_adv => 1
36 # Means the time when the events matched and collected are pushed on outputs
37 # with tag "advisor_info".
38
39 # send_first => true
40 # Means you can push out the first events different who came in advisor
41 # like clone copy and tagged with "advisor_first"
42
43 class LogStash::Filters::Advisor < LogStash::Filters::Base
44
45  config_name "advisor"
46  plugin_status "experimental"
47
48  # If you do not set time_adv the plugin does nothing.
49  config :time_adv, :validate => :number, :default => 0
50
51  # If you want the first different event will be pushed out like a copy
52  config :send_first, :validate => :boolean, :default => true
53
54  public
55  def register
56
57   # Control the correct config
58   if (!(@time_adv == 0))
59
60     @flag = false
61     @first = false
62     # Is used for store the different events.
63     @sarray = Array.new
64     # Is used for count the number of equals events.
65     @carray = Array.new
66
67     @thread = time_alert(@time_adv.to_i*60) do
68      # if collected any events then pushed out a new event after time_adv
69      if (@sarray.size !=0)
70         @flag = true
71      end
72     end
73
74   else
75    @logger.warn("Advisor: you have not specified Time_adv. Do nothing!")
76   end
77
78  end
79
80  # This method is used to manage sleep and awaken threads
81   def time_alert(interval)
82      Thread.new do
83       loop do
```

```
84          start_time = Time.now
85          yield
86          elapsed = Time.now - start_time
87          sleep([interval - elapsed, 0].max)
88        end
89      end
90    end
91
92    public
93    def filter(event)
94     return unless filter?(event)
95
96     # Control the correct config
97     if(!(@time_adv == 0))
98
99       new_event = true
100
101      # It's only for me
102      @message = event.message.slice(13, event.message.size-13)
103      submessage = @message[/\(.*?\)/]
104      @message.slice! submessage
105
106      # control if the events are new or they are came before
107      for i in (0..@sarray.size-1)
108        if (@message == @sarray[i].to_s)
109          @logger.debug("Avisor: Event match")
110          # if came before then count it
111          new_event = false
112          @carray[i] = @carray[i].to_i+1
113          @logger.debug("Advisor: "+@carray[i].to_s+" Events matched")
114          break
115        end
116      end
117
118      if (new_event == true)
119         # else is a new event
120
121         @sarray << @message
122         @carray << 1
123         if (send_first == true)
124             @logger.debug("Advisor: is the first to send out")
125             @first = true
126         end
127      end
128
129    else
130     @logger.warn("Advisor: you have not specified Time_adv. Do nothing!")
131    end
132   end
133
134
135   # This method is used for generate events every 5 seconds (Thanks Jordan Sissel).
136   # In this case we generate an event when advisor thread trigger the flag
137   # or is the first different event.
138
139   def flush
```

```
140
141           if (@first == true)
142             event = LogStash::Event.new
143             event.source_host = Socket.gethostname
144             event.message = @message
145             event.tags << "advisor_first"
146             event.source = Socket.gethostname+" advisor_plugin"
147             filter_matched(event)
148
149             @first = false
150             return [event]
151           end
152
153            if (@flag == true)
154
155             if (@tags.size != 0)
156               @tag_path = ""
157               for i in (0..@tags.size-1)
158                 @tag_path += @tags[i].to_s+"."
159               end
160             end
161
162             # Prepare message
163             message = "Advisor: Found events who match: "+@tag_path.to_s+"On s3 between
164             times ls.s3...\n"+(Time.now-@time_adv.to_i*60).strftime("%Y-%m-%dT%H.%M")+
165             "...partX.txt < ls.s3."+Socket.gethostname+".X.TIME.tag_"
166             +@tag_path.to_s+".partX.txt < ls.s3..."+(Time.now).strftime("%Y-%m-%dT%H.%M")+
167             "...partX.txt\n\n"
168
169             # See on messagge partial part of different events
170             for i in (0..@sarray.size-1)
171               message = message+@carray[i].to_s+" events like: "+
172               (@sarray[i].to_s).slice(0, 300)+"\n\n"
173             end
174
175             event = LogStash::Event.new
176             event.source_host = Socket.gethostname
177             event.message = message
178             event.tags << "advisor_info"
179             event.source = Socket.gethostname+" advisor_plugin"
180             filter_matched(event)
181
182             # reset flag and counter
183             @flag = false
184             @carray = nil
185             @sarray = nil
186             @carray = Array.new
187             @sarray = Array.new
188
189             # push the event
190             return [event]
191            end
192       return
193     end
194
195 end
```

B.2.2 S3

```
1  require "logstash/outputs/base"
2  require "logstash/namespace"
3  require "aws-sdk"
4
5  # TODO integrate aws_config in the future
6  #require "logstash/plugin_mixins/aws_config"
7
8  # INFORMATION:
9
10 # This plugin was created for store the logstash's events
11 # into Amazon Simple Storage Service (Amazon S3).
12 # For use it you needs authentications and an s3 bucket.
13 # Be careful to have the permission to write file on S3's bucket
14 # and run logstash with super user for establish connection.
15
16 # S3 plugin allows you to do something complex, let's explain:)
17
18 # S3 outputs create temporary files into "/opt/logstash/S3_temp/".
19 # If you want, you can change the path at the start of register method.
20 # This files have a special name, for example:
21
22 # ls.s3.ip-10-228-27-95.2013-04-18T10.00.tag_hello.part0.txt
23
24 # ls.s3 : indicate logstash plugin s3
25
26 # "ip-10-228-27-95" : indicate you ip machine, if you have more logstash
27 #                     and writing on the same bucket for example.
28 # "2013-04-18T10.00" : represents the time whenever you specify time_file.
29 # "tag_hello" : this indicate the event's tag, you can collect events.
30 # "part0" : this means if you indicate size_file then it will generate
31 #           more parts if you file.size > size_file. When a file is full
32 #           it will pushed on bucket and will be deleted in temporary directory.
33 #           If a file is empty is not pushed, but deleted.
34
35 # This plugin have a system to restore the previous temporary files if something crash.
36
37 ##[Note] :
38
39 ## If you specify size_file and time_file then it will create file for each tag
40 ## (if specified), when time_file or their size > size_file, it will be triggered
41 ## then they will be pushed on s3's bucket and will delete from local disk.
42
43 ## If you don't specify size_file, but time_file then it will create only one file
44 ## for each tag (if specified). When time_file it will be triggered then the files
45 ## will be pushed on s3's bucket and delete from local disk.
46
47 ## If you don't specify time_file, but size_file  then it will create files
48 ## for each tag (if specified), that will be triggered when their size > size_file,
49 ## then they will be pushed on s3's bucket and will delete from local disk.
50
51 ## If you don't specific size_file and time_file you have a curios mode.
52 ## It will create only one file for each tag (if specified).
53 ## Then the file will be rest on temporary directory and don't will be pushed
54 ## on bucket until we will restart logstash.
```

```
55
56 # INFORMATION ABOUT CLASS:
57
58 # I tried to comment the class at best i could do.
59 # I think there are much thing to improve,
60 # but if you want some points to develop here a list:
61
62 # TODO Integrate aws_config in the future
63 # TODO Find a method to push them all files when logtstash close the session.
64 # TODO Integrate @field on the path file
65 # TODO Permanent connection or on demand? For now on demand. Use a while or
66 #      a thread to try the connection before break a time_out and signal an error.
67 # TODO If you have bugs report or helpful advice contact me,
68 # but remember that this code is much mine as much as yours,
69 # try to work on it if you want :)
70
71 # s3 not allow special characters like "/" "[,]", very useful in date format,
72 # because if you use them s3 dosen't know no more the key and send you to hell!
73 # For example "/" in s3 means you can specify a subfolder on bucket.
74
75 # USAGE:
76
77 # This is an example of logstash config:
78
79 # output {
80 #    s3{
81 #      access_key_id => "crazy_key"             (required)
82 #      secret_access_key => "monkey_access_key" (required)
83 #      endpoint_region => "eu-west-1"           (required)
84 #      bucket => "boss_please_open_your_bucket" (required)
85 #      size_file => 2048                        (optional)
86 #      time_file => 5                           (optional)
87 #      format => "plain"                        (optional)
88 #      restore => "false"                       (optional)
89 #    }
90 # }
91
92 # We analize this:
93
94 # access_key_id => "crazy_key"
95 # Amazon will give you the key for use their service if you buy it or try it.
96 # (not very much open source anyway)
97
98 # secret_access_key => "monkey_access_key"
99 # Amazon will give you the secret_access_key for use their service
100 # if you buy it or try it . (not very much open source anyway).
101
102 # endpoint_region => "eu-west-1"
103 # When you make a contract with Amazon, you should know where the services you use.
104
105 # bucket => "boss_please_open_your_bucket"
106 # Be careful you have the permission to write on bucket and know the name.
107
108 # size_file => 2048
109 # Means the size, in Byte, of files who can store on temporary directory before
110 # you will be pushed on bucket. Is useful if you have a little server with poor space
```

```
111 # on disk and you don't want blow up the server with unnecessary temporary log files.
112
113 # time_file => 5
114 # Means, in minutes, the time  before the files will be pushed on bucket.
115 # Is useful if you want to push the files every specific time.
116
117 # format => "plain"
118 # Means the format of events you want to store in the files
119
120 # restore => "false"
121 # Means that s3 will not look crash occurred
122
123 # LET'S ROCK AND ROLL ON THE CODE!
124
125 class LogStash::Outputs::S3 < LogStash::Outputs::Base
126  #TODO integrate aws_config in the future
127  #  include LogStash::PluginMixins::AwsConfig
128
129  config_name "s3"
130  plugin_status "experimental"
131
132  # Aws access_key.
133  config :access_key_id, :validate => :string
134
135  # Aws secret_access_key
136  config :secret_access_key, :validate => :string
137
138  # S3 bucket
139  config :bucket, :validate => :string
140
141  # Aws endpoint_region
142  config :endpoint_region, :validate => ["us_east_1", "us-west-1", "us-west-2",
143                                         "eu-west-1", "ap-southeast-1", "ap-southeast-2",
144                                         "ap-northeast-1", "sa-east-1", "us-gov-west-1"],
145                                         :default => "us_east_1"
146
147  # Set the size of file in KB, this means that files on bucket when have dimension
148  # > file_size, they are stored in two or more file.
149  # If you have tags then it will generate a specific size file for every tags
150  ## NOTE: define size of file is the better thing, because generate a local temporary
151  # file on disk and then put it in bucket.
152  config :size_file, :validate => :number, :default => 0
153
154  # Set the time, in minutes, to close the current sub_time_section of bucket.
155  # If you define file_size you have a number of files in consideration of the
156  # section and the current tag. 0 stay all time on listerner, beware if you specific 0
157  # and size_file 0, because you will not put the file on bucket,
158  # for now the only thing this plugin can do is to put the file when logstash restart.
159  config :time_file, :validate => :number, :default => 0
160
161  # The event format you want to store in files. Defaults to plain text.
162  config :format, :validate => [ "json", "plain", "nil" ], :default => "plain"
163
164  ## IMPORTANT: if you use multiple instance of s3, you should specify on one of them
165  #the "restore=> true" and on the others "restore => false".
166  ## This is hack for not destroy the new files after restoring the initial files.
```

```
167  ## If you do not specify "restore => true" when logstash crashes or is restarted,
168  # the files are not sent into the bucket,
169  ## for example if you have single Instance.
170  config :restore, :validate => :boolean, :default => false
171
172  # Method to set up the aws configuration and establish connection
173  def aws_s3_config
174
175   @logger.debug "S3: waiting for establishing connection..."
176   AWS.config(
177     :access_key_id => @access_key_id,
178     :secret_access_key => @secret_access_key,
179     :s3_endpoint => 's3-'+@endpoint_region+'.amazonaws.com'
180   )
181   @s3 = AWS::S3.new
182
183  end
184
185  # This method is used to manage sleep and awaken thread.
186  def time_alert(interval)
187
188    Thread.new do
189     loop do
190       start_time = Time.now
191       yield
192       elapsed = Time.now - start_time
193       sleep([interval - elapsed, 0].max)
194     end
195    end
196
197  end
198
199  # this method is used for write files on bucket.
200  # It accept the file and the name of file.
201  def write_on_bucket (file_data, file_basename)
202
203   # if you lose connection with s3, bad control implementation.
204   if ( @s3 == nil)
205     aws_s3_config
206   end
207
208   # find and use the bucket
209   bucket = @s3.buckets[@bucket]
210
211   @logger.debug "S3: ready to write "+file_basename+" in bucket "+@bucket+",
212                 Fire in the hole!"
213
214   # prepare for write the file
215   object = bucket.objects[file_basename]
216   object.write(:file => file_data, :acl => :public_read)
217
218   @logger.debug "S3: has written "+file_basename+" in bucket "+@bucket
219
220  end
221
222  # this method is used for create new path for name the file
```

```
223   def getFinalPath
224
225     @pass_time = Time.now
226     return @temp_directory+"ls.s3."+Socket.gethostname+"."+
227     (@pass_time).strftime("%Y-%m-%dT%H.%M")
228
229   end
230
231   # This method is used for restore the previous crash of logstash or to prepare
232   # the files to send in bucket. Take two parameter: flag and name.
233   # Flag indicate if you want to restore or not, name is the name of file
234   def upFile(flag, name)
235
236     Dir[@temp_directory+name].each do |file|
237       name_file = File.basename(file)
238
239       if (flag == true)
240        @logger.warn "S3: have found temporary file: "+name_file+",
241        something has crashed before... Prepare for upload in bucket!"
242       end
243
244       if (!File.zero?(file))
245         write_on_bucket(file, name_file)
246
247         if (flag == true)
248            @logger.debug "S3: file: "+name_file+" restored on bucket "+@bucket
249         else
250            @logger.debug "S3: file: "+name_file+" was put on bucket "+@bucket
251         end
252       end
253
254       @logger.debug "S3: let's destroying the temporary shit file "+name_file
255       File.delete(file)
256
257     end
258   end
259
260   # This method is used for create new empty temporary files for use.
261   # Flag is needed for indicate new subsection time_file.
262   def newFile (flag)
263
264     if (flag == true)
265       @current_final_path = getFinalPath
266       @sizeCounter = 0
267     end
268
269     if (@tags.size != 0)
270       @tempFile = File.new(@current_final_path+".tag_"+@tag_path+"part"+
271                   @sizeCounter.to_s+".txt", "w")
272     else
273       @tempFile = File.new(@current_final_path+".part"+@sizeCounter.to_s+".txt", "w")
274     end
275
276   end
277
278   public
```

```
279  def register
280    @temp_directory = "/opt/logstash/S3_temp/"
281
282    if (@tags.size != 0)
283        @tag_path = ""
284        for i in (0..@tags.size-1)
285            @tag_path += @tags[i].to_s+"."
286        end
287    end
288
289    if !(File.directory? @temp_directory)
290     @logger.debug "S3: Directory "+@temp_directory+" doesn't exist, let's make it!"
291     Dir.mkdir(@temp_directory)
292    else
293     @logger.debug "S3: Directory "+@temp_directory+" exist, nothing to do"
294    end
295
296    if (@restore == true )
297      @logger.debug "S3: is attempting to verify previous crashes..."
298
299      upFile(true, "*.txt")
300    end
301
302    newFile(true)
303
304    if (time_file != 0)
305       first_time = true
306       @thread = time_alert(@time_file*60) do
307        if (first_time == false)
308          @logger.debug "S3: time_file triggered,
309            let's bucket the file if dosen't empty and create new file "
310          upFile(false, File.basename(@tempFile))
311          newFile(true)
312        else
313          first_time = false
314        end
315      end
316    end
317
318  end
319
320  public
321  def receive(event)
322   return unless output?(event)
323
324   # Prepare format of Events
325   if (@format == "plain")
326      message = self.class.format_message(event)
327   elsif (@format == "json")
328      message = event.to_json
329   else
330      message = event.to_s
331   end
332
333   if(time_file !=0)
334      @logger.debug "S3: trigger files after "+((@pass_time+60*time_file)-Time.now).to_s
```

```
335     end
336
337     # if specific the size
338     if(size_file !=0)
339
340       if (@tempFile.size < @size_file )
341
342         @logger.debug "S3: File have size: "+@tempFile.size.to_s+" and size_file is: "+
343                       @size_file.to_s
344         @logger.debug "S3: put event into: "+File.basename(@tempFile)
345
346         # Put the event in the file, now!
347         File.open(@tempFile, 'a') do |file|
348           file.puts message
349           file.write "\n"
350         end
351
352       else
353
354         @logger.debug "S3: file: "+File.basename(@tempFile)+" is too large,
355                       let's bucket it and create new file"
356         upFile(false, File.basename(@tempFile))
357         @sizeCounter += 1
358         newFile(false)
359
360       end
361
362     # else we put all in one file
363     else
364
365       @logger.debug "S3: put event into "+File.basename(@tempFile)
366       File.open(@tempFile, 'a') do |file|
367         file.puts message
368         file.write "\n"
369       end
370     end
371
372   end
373
374   def self.format_message(event)
375       message = "Date: #{event.timestamp}\n"
376       message << "Source: #{event.source}\n"
377       message << "Tags: #{event.tags.join(', ')}\n"
378       message << "Fields: #{event.fields.inspect}\n"
379       message << "Message: #{event.message}"
380   end
381 end
```

Bibliografia

[1] James Turnbull *The LogStash Book*, February 2013.

[2] Jordan Sissel *LogStash Source Code*, [Online].
Available: https://github.com/logstash/logstash

[3] Rafał Kuć and Marek Rogoziński *ElasticSearch Server*, February 2013

[4] Karl Seguin, Sandro Conforto *The Little Redis Book*, [Online].
Available: http://openmymind.net/2012/1/23/The-Little-Redis-Book/

[5] Gianluigi Spagnuolo *Ruby Guide*, [Online].
Available: http://www.html.it/pag/17653/introduzione53/

[6] Machtelt Garrels *Bash Guide for Beginners*, [Online]. Available:
http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf

[7] *JBossLogLevel*, Guide, [Online].
Available: http://docs.jboss.org/process-guide/en/html/logging.html

[8] *Amazon-Machine-Image*, User-Guide, [Online].
Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

[9] *Amazon-Elastic-Compute-Cloud*, User-Guide, [Online]. Available:
http://docs.aws.amazon.com/AWSEC2/2011-05-15/UserGuide/index.html?concepts.html

[10] *Amazon-Simple-Notification-Service* , Documentation, [Online].
Available: http://docs.aws.amazon.com/sns/latest/dg/welcome.html

[11] *Amazon-Simple-Storage-Service*, Documentation, [Online].
Available: http://aws.amazon.com/documentation/s3/

Ringraziamenti

Ringrazio tutti coloro che mi hanno appoggiato e sopportato durante questi anni. Un ringraziamento particolare ai miei genitori che hanno sempre creduto in me, al professore Carlo Ferrari, mio relatore, a Stefano Dindo di Zero12 s.r.l che mi ha dato la possibilità di lavorare a questo progetto, ai mie cugini e amici che mi hanno consolato e divertito nelle difficoltà.