Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

a.a. 2011 /2012

Corso di Sistemi Distribuiti e Cloud Computing
Prof. Valeria Cardellini

PuRo
Server Mail

Studenti:
Claudio Pupparo
Damiano Rossato

INDICE

Introduzione

In questa relazione verrà presentato il progetto **PuRo Mail Server**, un servizio di posta elettronica basato su **Amazon Web Services**, mirante a costruire un sistema distribuito scalabile ed altamente disponibile. La relazione è così strutturata:

Capitolo 1: Vengono riportati i requisiti del progetto, così come esposto nella traccia.
Capitolo 2: Fornisce una breve descrizione dei servizi AWS utilizzati.
Capitolo 3: Descrizione dei tool utilizzati per la realizzazione del server mail e per il testing del sistema.
Capitolo 4: Descrizione dettagliata del sistema; protocolli, architettura e dettagli implementativi.
Capitolo 5: Mostra i risultati del testing del sistema.
Capitolo 6: Riporta i costi del servizio, calcolati in base ai servizi AWS utilizzati.
Capitolo 7: Descrizione di possibili estensioni future del sistema.
Capitolo 8: Conclusioni finali sul progetto.

1. Requisiti progetto

Realizzare, in un linguaggio di programmazione a scelta, un servizio di email scalabile ed altamente disponibile basato su Amazon Web Services (AWS).
Il servizio di email deve soddisfare le caratteristiche elencate di seguito:

- Supporto dei protocolli SMTP e POP3. Opzionale: supporto del protocollo IMAP.
- Scalabilità. Adottare a tale scopo tecniche di replicazione, eventualmente integrate con tecniche di caching (consigliato un grado di replicazione minimo pari a 3, in modo da poter gestire eventuali failure). Discutere nella relazione quale tipo di scalabilità (rispetto alla dimensione, geografica) viene offerta dal servizio realizzato, motivando opportunamente la scelta e le soluzioni adottate.
- Capacità elastica (ovvero scale-in e scale-out dei nodi del sistema) automatica in base al traffico in arrivo al servizio di email. Si noti che, in base ai servizi AWS scelti, tale requisito può richiedere la definizione di un ciclo di controllo MAPE; una semplice soluzione consiste nell'aumentare (o diminuire) il numero dei nodi sui cui il servizio è replicato quando viene superata una soglia di carico dei nodi che offrono il servizio (ad es., si veda la soluzione in [1]).
- Modello di consistenza a scelta. Motivare nella relazione il modello di consistenza scelto ed il suo impatto sul servizio di email.
- Gestione di failure di tipo crash dei componenti del servizio di email.

Si progetti l'architettura del sistema che offre il servizio di email ponendo particolare cura al soddisfacimento dei requisiti sopra elencati e delle altre eventuali scelte effettuate dal gruppo e ponendo particolare attenzione nell'organizzazione dei dati. I componenti del sistema devono essere eseguibili nello spazio utente e senza richiedere privilegi di root. Si richiede inoltre che il servizio sia configurabile, ad es. tramite un file in cui sia possibile specificare i valori dei parametri di configurazione.
Si richiede inoltre di effettuare il testing delle funzionalità del servizio realizzato e di effettuare un testing delle prestazioni, analizzando in particolare il throughput del servizio realizzato. A tale scopo, si può usare Postal (http://doc.coker.com.au/projects/postal/).

2. Breve descrizione dei servizi utilizzati

Il progetto sviluppato costituisce un'implementazione di un servizio email basato su Amazon Web Services. L'utilizzo dei suddetti servizi consente di garantire alta disponibilità e scalabilità sotto diversi punti di vista. Allo scopo, abbiamo scelto di utilizzare i seguenti servizi:

- **Elastic Compute Cloud (EC2)**
  - **Auto Scaling**
- **DynamoDB**
- **Simple Storage Service (S3)**
- **Elastic Load Balancing**
- **CloudWatch**
- **Route 53**

EC2 - Elastic Cloud Computing

Amazon Elastic Compute Cloud (Amazon EC2) è un servizio web che fornisce capacità di calcolo dimensionabile su cloud.
Amazon EC2 riduce i tempi richiesti per ottenere e avviare una nuova istanza all'ordine dei minuti, permettendo rapidamente di scalare, sia verso l'altro che verso il basso, le capacità di calcolo di cui si ha bisogno per far fronte al cambiamento delle richieste.
Il servizio fornisce agli sviluppatori gli strumenti per costruire applicazioni resistenti ai guasti.
L'impegno indicato nello SLA di Amazon EC2 è del 99.95% di disponibilità per ogni Amazon EC2 Region.
Amazon EC2 fornisce un elevato numero di funzionalità al fine di potenziare la scalabilità, la resistenza ai guasti delle applicazioni aziendali, tra cui:

- **Auto Scaling:** permette di scalare automaticamente le capacità del tuo Amazon EC2, aumentandole oppure diminuendole in accordo alle condizioni definite. Auto Scaling, assicura che il numero di istanze di Amazon EC2 in uso aumentino durante i picchi di richiesta, evitando un degrado delle prestazioni, e diminuiscano quando non se ne ha più la necessità al fine di minimizzare i costi di utilizzo. Auto Scaling è particolarmente indicato per le applicazioni con carico variabile a seconda di diversi fattori.

DynamoDB

Amazon DynamoDB è un servizio di database NoSQL che fornisce alte prestazioni predicibili con grandi caratteristiche di scalabilità. Attraverso la AWS Management Console, il client può creare nuove tabelle, aumentarne o diminuirne la capacità in termini di prestazioni ed analizzare diverse metriche relative all'utilizzazione di tali tabelle.
Amazon DynamoDB consente agli utilizzatori di non occuparsi degli oneri amministrativi di gestione di un database distribuito, eliminando preoccupazioni sull''approvigionamento di hardware, della configurazione, della replicazione, dell'aggiornamento del software.
Amazon DynamoDB è progettato per affrontare i problemi chiave della gestione, delle performance, della scalabilità e dell'affidabilità di un database.
DynamoDB diffonde automaticamente i dati ed il traffico su un sufficiente numero di server al fine di gestire la capacità di richieste specificate dal client e l'ammontare di dati immagazzinati, mantenendo allo stesso tempo consistenza e alte prestazioni.

Tutti i dati sono salvati su SSD e sono automaticamente replicati su tre Availability Zones un una Region per fornire un alta disponibilità e la durata dei dati. Le latenze medie dalla parte del servizio per Amazon DynamoDB sono tipicamente nell'ordine del millisecondo. E' inoltre flessibile, poichè non ha uno schema fisso. Ogni entry può avere un differente numero di attributi. Diversi tipi di dato (stringhe, numeri, dati binari, e collezioni) aggiungono ricchezza al modello di dati. Consistenza forte: Amazon DynamoDB assicura lo sviluppatore che ciò che legge sia sempre l'ultimo valori scritto sul database.

S3 - Simple Storage Service

Amazon S3 fornisce una semplice interfaccia web che può essere usata per salvare o recuperare qualsiasi quantità di dati, in ogni momento, da qualsiasi punto sul web. Fornisce accesso ad ogni sviluppatore alla stessa infrastruttura altamente scalabile, affidabile, sicura, veloce e poco costosa che Amazon stessa utilizza per far girare la sua rete globale di siti web. Amazon S3 è intenzionalmente costruito con un insieme minimo di caratteristiche.

- Gli oggetti write, read, e delete contengono da 1 byte a 5 terabytes di dati ognuno. Il numero di oggetti che è possibile salvare è illimitato.
- Ogni oggetto è salvato in un bucket e recuperato attraverso una chiave unica definita dallo sviluppatore.
- Un bucket può essere salvato in una delle diverse Regioni. E' possibile scegliere una Regione per ottimizzare la latenza o minimizzare i costi. Amazon S3 è disponibile nelle Region: US Standard, US West (Oregon), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo), Asia Pacific (Sydney), South America (Sao Paulo), e GovCloud (US).
- Gli oggetti salvati in una Regioni non la lasciano mai a meno che non sia lo sviluppatore a deciderlo.
- I meccanismi di autenticazione sono forniti per assicurare che i dati non vengano acceduti da soggetti non autorizzati.

ELB - Elastic Load Balancing

Elastic Load Balancing distribuisce automaticamente il traffico in entrata delle applicazioni tra la istanze Amazon EC2 aumetando la tolleranza ai guasti nelle proprie applicazioni.
Elastic Load Balancing rileva le istanze non funzionanti all'interno di un pool ed automaticamente reindirizza il traffico su quelle funzionanti fino a quando le prime non vengono ripristinate. I clienti possono abilitare Elastic Load Balancing all'interno di una singola Availability Zone oppure su diverse zone per massimizzare la tolleranza ai guasti.

CloudWatch

Amazon CloudWatch è un servizio web che permette di monitorare, gestire e pubblicare diverse metriche, permettendo inoltre di configurare allarmi basati su dati provenienti da esse.
Grazie alla collezione ed analisi delle metriche è possibile prendere decisioni operative e di business più velocemente e con maggiore sicurezza. Amazon CloudWatch permette di gestire le metriche in diversi modi. Se si ha accesso ad un determinato prodotto AWS, il servizio aggiunge le metriche su CloudWatch su cui è poi possibile calcolare le statistiche. Gli allarmi di Amazon CloudWatch permettono di ricevere notifiche o di eseguire delle azioni, precedentemente stabilite, su determinate risorse, in maniera del tutto automatica.
Un utilizzo comune di Amazon CloudWatch è quello mantenere le applicazioni e i servizi "healthy" e in esecuzione in maniera efficiente.

Route 53

Amazon Route 53 è un servizio DNS altamente scalabile e disponibile. E' progettato per offrire agli sviluppatori un modo, estremamente affidabile ed efficace dal punto di vista dei costi, di indirizzare gli utenti verso una applicazione Internet, traducendo nomi leggibili dall'uomo in indirizzi IP. Route 53 connette efficacemente le richieste degli utenti alle infrastrutture che girano sugli AWS (istanze EC2, ELB) e può anche essere utilizzato per indirizzare gli utenti verso le infrastrutture esterne agli AWS.
Route 53 è progettato per essere veloce, facile da usare ed efficente dal punto di vista dei costi. Risponde con bassa latenza ad un query DNS attraverso l'utilizzo di un rete globale di server DNS. Le query per un dominio sono automaticamente indirizzate al server DNS più vicino il quale risponde con la migliore prestazione possibile.

3. Descrizione dei tool software utilizzati

Nell'affrontare la progettazione e quindi lo sviluppo di un nostro server di posta elettronica, ci siamo serviti di un progetto di riferimento, per il quale erano già state sviluppate delle funzionalità, ma che è stato necessario modificare ed estendere affinchè si adattasse ai nostri scopi e facesse uso dei servizi di Amazon descritti nella sezione precedente. Per la fase di testing delle funzionalità del servizio realizzato, oltre al software di benchmark *Postal*, sono stati utilizzati anche *Rabid*, per il benchmark di POP3, ed il client di posta Thunderbird. Di seguito un elenco più completo dei software utilizzati:

- **GreenMail**: Server SMTP, IMAP, POP3
- **Postal**: testing SMTP
- **Rabid**: testing POP3
- **Thunderbird**: testing SMTP/s, POP3/s
- **Telnet:** testing SMTP, POP3

GreenMail

GreenMail è una suite di server mail open source, intuitiva e facile da utilizzare, che supporta i protocolli SMTP, SMTPS, POP3, POP3S, IMAP, e IMAPS per scopi di test.
E' stata progettata secondo un'architettura di tipo stand-alone, infatti non fa uso di alcun tipo di database esterno, ma memorizza In RAM tutte le risorse su cui lavora. Al suo interno è presente una struttura per la gestione degli utenti, ma è priva di un meccanismo di autenticazione che possa essere almeno chiamato tale. I suoi vantaggi sono quelli di avere del codice ben documentato, di avere una logica strutturale di facile comprensione e quindi ben disposta a modifiche ed estensioni, al fine di essere adatta al proprio scopo.

Postal e Rabid

Postal è un programma, distribuito sotto licenza GPL che nasce con lo scopo di essere uno strumento di benchmark per server SMTP. Lo scopo dell'autore era quello di conoscere quale server di posta elettronica desse le migliori performance con più di un milione di utenti.
Il programma lavora prendendo una lista di indirizzi email da utilizzare come indirizzi per riempire i campi FROM e TO. In fase di invio la sezione body di una mail viene riempita con dati generati in maniera casuale. Viene aggiunto inoltre, nella sezione header, un campo **X-Postal** che puà essere usato per filtrare le mail generate da programma, nel caso il proprio indirizzo di posta venga accidentalmente aggiunto alla lista degli indirizzi di test. L'autore ha aggiunto all'interno della suite anche il programma di bechmark **Rabid**, utile per

testare server POP3. All'interno dell'header di tutti i messaggi che vengono inviati durante il test, è presente anche un campo **X-PostalHash** che non è altro che un checksum MD5 (effettuato sui campi Subject, Date, Message-ID, From, e To dell'header e sul body del messaggio, inclusi i caratteri "\r\n" che, nel protocollo SMTP, terminano ogni linea del testo). Rabid controlla il checksum MD5 e in caso genera un errore se il confronto fallisce.
Entrambi i programmi possono essere configurati, rispettivamente, per inviare uno specifico numero di messaggi ed effettuare uno specifico numero di connessioni POP per ogni minuto. Questo significa che è facile dimensionare il numero di risorse del sistema che vengono utilizzare per un determinato volume di traffico. E' anche possibile eseguire un'analisi delle prestazioni per determinare quali sono i colli di bottiglia nel proprio server di posta, configurando Postal e Rabid per utilizzare solo metà della massima velocità disponibile, così che la CPU e l'utilizzo del disco del software di analisi non impattino su server testato.

Thunderbird

Mozilla Thunderbird è un client di posta elettronica e news (in grado di gestire anche i feed RSS ed i Newsgroup) sviluppato da Mozilla Foundation ed è un software libero.
Il programma è disponibile per sistemi Microsoft Windows, GNU/Linux, Mac OS X e molti altri. Supporta estensioni (funzionalità aggiuntive da installare a seconda delle esigenze).

Altre caratteristiche del programma sono:

- un filtro bayesiano anti spam;
- la possibilità di creare filtri per "smistare" automaticamente la posta su diverse cartelle;
- raggruppamento dei messaggi secondo data, mittente, priorità o altre caratteristiche;
- capacità di importazione messaggi da altri programmi (tra cui anche Eudora e Microsoft Outlook);
- posta in arrivo singola per account multipli;
- la ricerca veloce;
- gestione account POP e IMAP;
- supporto di rubriche basate su LDAP:

4. Descrizione del Sistema

Protocolli supportati

POP3 - Post Office Protocol

Il **Post Office Protocol versione 3** è un protocollo di livello applicativo di tipo client-server che ha il compito di permettere, mediante autenticazione, l'accesso da parte del client ad un account di posta elettronica, presente su di un host server e scaricare le e-mail dell'account stesso.
Il POP (nella versione 3) rimane in attesa sulla porta 110 dell'host (di default, ma può anche essere diversa) per una connessione TCP da parte di un client.
Una delle differenze sostanziali con il protocollo IMAP è che i messaggi di posta elettronica, per essere letti, devono essere scaricati sul computer, anche se è possibile lasciarne una copia sull'host. Il POP3 non e' inteso per fornire operazioni aggiuntive di manipolazione della posta presente su un server; normalmente, la posta viene scaricata e poi cancellata.
Il protocollo POP3 non prevede alcun tipo di cifratura, quindi anche le password utilizzate per l'autenticazione fra server e client passano in chiaro. Per risolvere questo possibile problema è stata

sviluppata l'estensione APOP che utilizza MD5.

Comandi implementati:

- **USER**: necessario per l'autenticazione. Specifica nome utente.
- **PASS**: necessario per l'autenticazione. Specifica la password.
- **STAT**: restituisce il numero di messaggi nella inbox, affiancato dalla dimensione totale degli stessi.
- **LIST**: per ogni messaggio specifica UID e dimensione.
- **UIDL**: per ogni messaggio specifica message number e UID.
- **TOP**: per il messaggio di cui viene specificato il message number, viene recuperato l'header e il numero di linee del body specificate nel comando.
- **RETR**: recupera messaggio corrispondente al message number specificato.
- **DELE**: il messaggio di cui viene specificato il message number, viene contrassegnato come da eliminare.
- **NOOP**: No Operation, può essere usato come semplice ping.
- **RSET**: per tutti i messaggi impostati come da eliminare, viene eliminato il flag di cancellazione.
- **QUIT**: chiude la comunicazione.

Esempio sessione POP3:

```
S:    +OK <22593.1129980067@example.com>
C:    USER pippo
S:    +OK
C:    PASS pluto
S:    +OK
C:    LIST
S:    +OK
      1 817
      2 124
      .
C:    RETR 1
S:    +OK
      Return-Path: <pippo@example.org>
      Delivered-To: pippo@example.org
      Date: Sat, 22 Oct 2005 13:24:54 +0200
      From: Mario Rossi <mario@rossi.org>
      Subject: xxxx
      Content-Type: text/plain; charset=ISO-8859-1

      testo messaggio
      .
C:    DELE 1
S:    +OK
C:    QUIT
S:    +OK
```

SMTP - Simple Mail Transfer Protocol

Il **Simple Mail Transfer Protocol** (**SMTP**) è il protocollo standard per la trasmissione via internet di e-mail. È un protocollo relativamente semplice, testuale, nel quale vengono specificati uno o più destinatari di un messaggio, verificata la loro esistenza il messaggio viene trasferito. È abbastanza facile verificare come

funziona un server SMTP mediante un client telnet. Il protocollo SMTP utilizza il TCP come protocollo per il livello di trasporto.
Il client apre una sessione TCP verso il server sulla porta 25 (recentemente modificata in 587 da molti Provider per limitare lo spam). Per associare il server SMTP a un dato nome di dominio (DNS) si usa un Resource Record di tipo MX (Mail eXchange). Quando un server SMPT riceve una richiesta di invio di un messaggio che ha tra i destinatari indirizzi che appartengono a domini diversi dal suo, risolve il nome del dominio in un indirizzo IP, tramite un server DNS, e instaura una connessione con il server SMTP in questione, comportandosi da client verso quel server.
Poiché SMTP è un protocollo testuale basato sulla codifica ASCII, non è permesso trasmettere direttamente testo composto con un diverso set di caratteri e tantomeno file binari. Lo standard MIME permette di estendere il formato dei messaggi mantenendo la compatibilità col software esistente. Altri limiti di SMTP, quale la lunghezza massima di una riga, impediscono la spedizione di file binari senza transcodifica.

Comandi implementati:

- **AUTH**: autenticazione al server. Supportato unicamente il meccanismo **AUTH PLAIN**.
- **HELO**: il client si identifica al server.
- **EHLO**: extended HELO, il server risponde con le funzionalità supportate.
- **MAIL**: specifica mittente
- **RCPT**: specifica destinatario. Gli user agent inviano un comando RCPT per ogni indirizzo specificato nei campi TO, CC, CCN.
- **DATA**: il client comincia ad inviare il messaggio, prima header e a seguire il body.
- **NOOP**: No Operation. Può essere usato come un semplice ping.
- **RSET**: resetta i parametri della transazione mail. Se inviato dopo il comando DATA non può fermare la mail il cui procedimento di invio è già stato eseguito.
- **QUIT**: chiude la comunicazione con il server.

Esempio sessione SMTP:

```
C:    telnet localhost 25
S:    220 XXXXX.yyyyyy.telecomitalia.it ESMTP Sendmail 8.14.5/8.14.5;
Sun, 6 Jan 2013 11:48:02 +0100

C:    HELO
S:    501 5.0.0 HELO requires domain address
C:    HELO puro.tk
S:    250 XXXXX.yyyyyy.telecomitalia.it Hello localhost.localdomain
[127.0.0.1], pleased to meet you

C:    MAIL FROM:<mittente@puro.tk>
S:    250 2.1.0 <mittente@puro.tk>... Sender ok
C:    RCPT TO:<destinatario@servermail.it>
S:    250 2.1.5 <destinatario@servermail.it>... Recipient ok
C:    DATA
S:    354 Enter mail, end with "." on a line by itself
C:    Date: Thu, 21 May 1998 05:33:29 -0700
Subject: zzzzzz
      ...
testo del messaggio
      ...
.
S:    250 2.0.0 r06Am2C0002219 Message accepted for delivery
C:    QUIT
S:    221 2.0.0 XXXXX.yyyyyy.telecomitalia.it closing connection
Connection closed by foreign host.
```

Protocolli non implementati

IMAP - Internet Message Access Protocol

IMAP - Internet message access protocol - è un protocollo di livello applicativo per il recupero di email da server remoti. La porta predefinita del protocoll è la 143, mentre per IMAP su SSL la porta è la 993.
Le caratteristiche del protocollo sono:

- Connessioni persistenti
- Connessioni in parallelo alla stessa mailbox da più utenti
- Prelievo parziali di parti dell'header o del body delle mail
- Possibilità di aggiungere flag (predefiniti o creati dall'utente) alle mail
- Ogni utente può avere più cartelle

N.B. Le motivazioni per cui IMAP non è stato implementato sono mostrate nel paragrafo dedicato agli sviluppi futuri.

Comandi IMAP:

Ogni comando deve esser preceduto da un numero, indicante la sequenza progressiva di operazioni.

- **LOGIN**: autenticazione utente. Specifica username e password
- **LSUB**: mostra la lista delle cartelle (comprensiva di sottocartelle) a cui l'utente risulta "sottoscritto".
- (comprensiva di sottocartelle): mostra la lista di tutte le cartelle (comprensiva di sottocartelle) appartenenti all'utente, non solo quelle a cui risulta sottoscritto.
- **SELECT**: seleziona la mailbox specificata. Le successive operazioni avranno effetto su tale cartella.
- **EXAMINE**: stesso funzionamento del comando SELECT, ma la cartella viene restituita in formato READ ONLY.
- **FETCH**: preleva parti di un messaggio. E' possibile specificare numerosi attributi per ogni tipologia di informazione da recuperare.
- **STORE**: modifica un messaggio, in particolare i suoi FLAG.
- **COPY**: copia un messaggio della cartella correntemente selezionata nella cartella destinazione specificata.
- **CLOSE**: i messaggi contrassegnati come da eliminare vengono eliminati, senza segnalare al client quali messaggi sono stati eliminati.
- **EXPUNGE**: esegue le stesse azioni del comando CLOSE, ma in output mostra i messaggi eliminati.
- **UID**: associato ai comandi COPY, FETCH, STORE, SEARCH, permette di eseguire operazioni su un messaggio specificandone l'UID anzichè il relativo message number.
- **CREATE**: crea nuova mailbox.
- **DELETE**: cancella una cartella, ma solo se priva di figli.
- **RENAME**: rinomina una cartella. Nel caso in cui si tentasse di rinominare la cartella INBOX (che deve essere obbligatoriamente presente), viene creata una nuova cartella con il nome specificato, in cui vi vengono copiati i messaggi della folder INBOX.
- **SUBSCRIBE**: l'utente si sottoscrive alla cartella. Successive chiamate al comando LSUB, mostrerranno anche la suddetta cartella.
- **UNSUBSCRIBE**: l'utente ritira la sua sottoscrizione ad una cartella.

- **APPEND**: aggiunge alla fine delle cartella il messaggio specificato per intero (ossia non specificandone solamente il relativo UID o message number).
- **CHECK**: crea un checkpoint (specifico dell'implementazione).
- **SEARCH**: permette di ricercare messaggi specificando diversi criteri di ricerca.
- **NOOP**: No Operation. Può esser utilizzato come ping.
- **STATUS**: restituisce le informazioni richieste su una determinata mailbox.
- **LOGOUT**: chiude la connessione.

Esempio sessione IMAP:

```
C:    telnet imap.tiscali.it 143
        Trying xxx.xxx.xxx.xxx...
        Connected to imap.tiscali.it.
        Escape character is '^]'.
S:    * OK IMAP4 server ready
C:    1 LOGIN userA@tiscali.it plainpassword
S:    1 OK authentication successful
C:    3 LIST "" "*"
S:    * LIST (\HasNoChildren) "/" "cartella A"
        * LIST (\HasChildren) "/" "cartella B"
        * LIST (\HasNoChildren) "/" "cartella B/subB"
        * LIST (\HasNoChildren) "/" Draft
        * LIST (\NoInferiors) NIL INBOX
        * LIST (\HasNoChildren) "/" Sent
        * LIST (\HasNoChildren) "/" Spam
        * LIST (\HasNoChildren) "/" Trash
        * LIST (\HasNoChildren) "/" Trashcan
        3 OK LIST completed
C:    12 select "cartella A"
S:    * 2 EXISTS
        * 0 RECENT
        * FLAGS (\Answered \Flagged \Deleted \Seen \Draft $MDNSent)
        * OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft
$MDNSent)] flags can be changed
        * OK [UIDVALIDITY 1349630711] mailbox UID validity
        * OK [UIDNEXT 3] predicted next UID
        12 OK [READ-WRITE] SELECT complete
```

```
C:     14 UID fetch 1:* (FLAGS)
S:     * 1 FETCH (FLAGS (\Seen) UID 1)
14 OK FETCH complete
C:     15 UID fetch 1:1 (UID RFC822.SIZE FLAGS BODY.PEEK[HEADER.
FIELDS (From To Cc Bcc Subject Date Message-ID Priority X-
Priority References Newsgroups In-Reply-To Content-Type)])
S:     * 1 FETCH (UID 1 RFC822.SIZE 2037 FLAGS (\Seen) BODY
[HEADER.FIELDS (From To Cc Bcc Subject Date Message-ID Priority
X-Priority References Newsgroups In-Reply-To Content-Type)]
{287}
Date: Sun, 7 Oct 2012 19:24:30 +0200
Message-ID:
<CAM5rKvunj6uSD65h_8YNs6EwgG0TP=W7QyYW+q_DqG1=7cc9KQ@mail.
tiscali.it>
Subject: Prova soggetto
From: User B <userB@tiscali.it>
To: userA@tiscali.it
Content-Type: multipart/alternative;
boundary=e89a8f234855b4ea4a04cb7b60a4
)
15 OK FETCH complete
C:     16CLOSE
S:     16 OK CLOSE complete
C:     17 LOGOUT
S:          * BYE disconnecting
       17 OK LOGOUT complete
```

Architettura generale

Quando si progettano servizi web, una soluzione molto spesso adottata consiste nella Architettura Three-Tier:

- **Client**
- **Server**
- **Data Storage**

Per il servizio realizzato si è intrapresa una soluzione differente, di cui parte delle motivazioni verranno ora presentate.
Durante una comunicazione POP3, il client invia una serie di comandi il cui scopo è quello di ottenere informazioni circa lo status della mailbox dell'utente (cfr. paragrafo protocolli); una volta ottenute, lo user agent andrà a recuperare, fra tutte le mail, unicamente quelle risultanti nuove; separando i metadati dai dati stessi, è possibili limitare il recupero degli ultimi solo quando effettivamente necessario, evitando lo scambio di informazioni inutili. Altre motivazioni sono da ricercarsi nella natura dei servizi utilizzati per la gestione dei Metadati e dei dati stessi, come verrà successivamente descritto.
Si è quindi deciso di separare i metadati dai dati, portando ad una architettura a quattro stati, **Four Tier Architecture**:

- **Primo Tier**: Client - User Agent
- **Secondo Tier**: Mail Server - Server SMTP/s POP3/s
- **Terzo Tier**: Metadati - Informazioni su Utenti, Mail e Mailbox

- **Quarto Tier**: Dati - Storage Mail in formato MIME

Una rappresentazione schematica di quanto appena descritto è mostrata in fig. 1

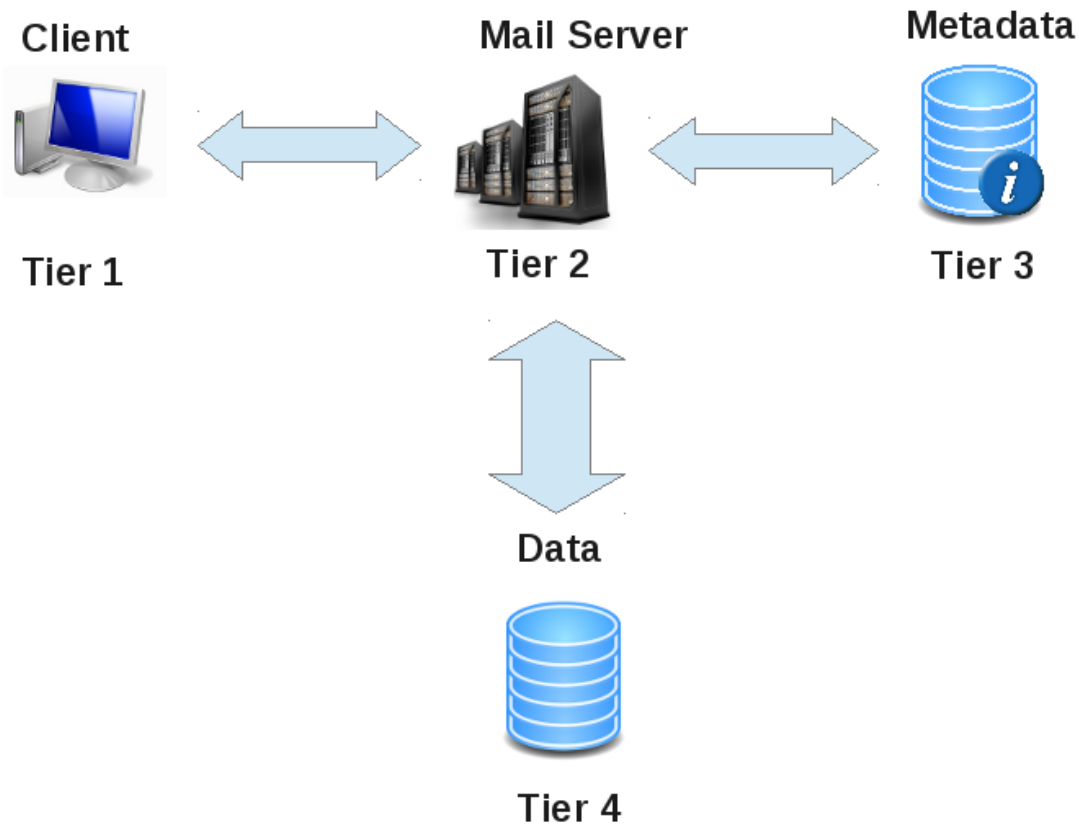

Fig. 1 - Schema generale architettura

Architettura dettagliata

Andando ad esplodere i singoli strati, passando così ad una rappresentazione dettagliata dell'intero sistema si giunge allo schema in fig. 2

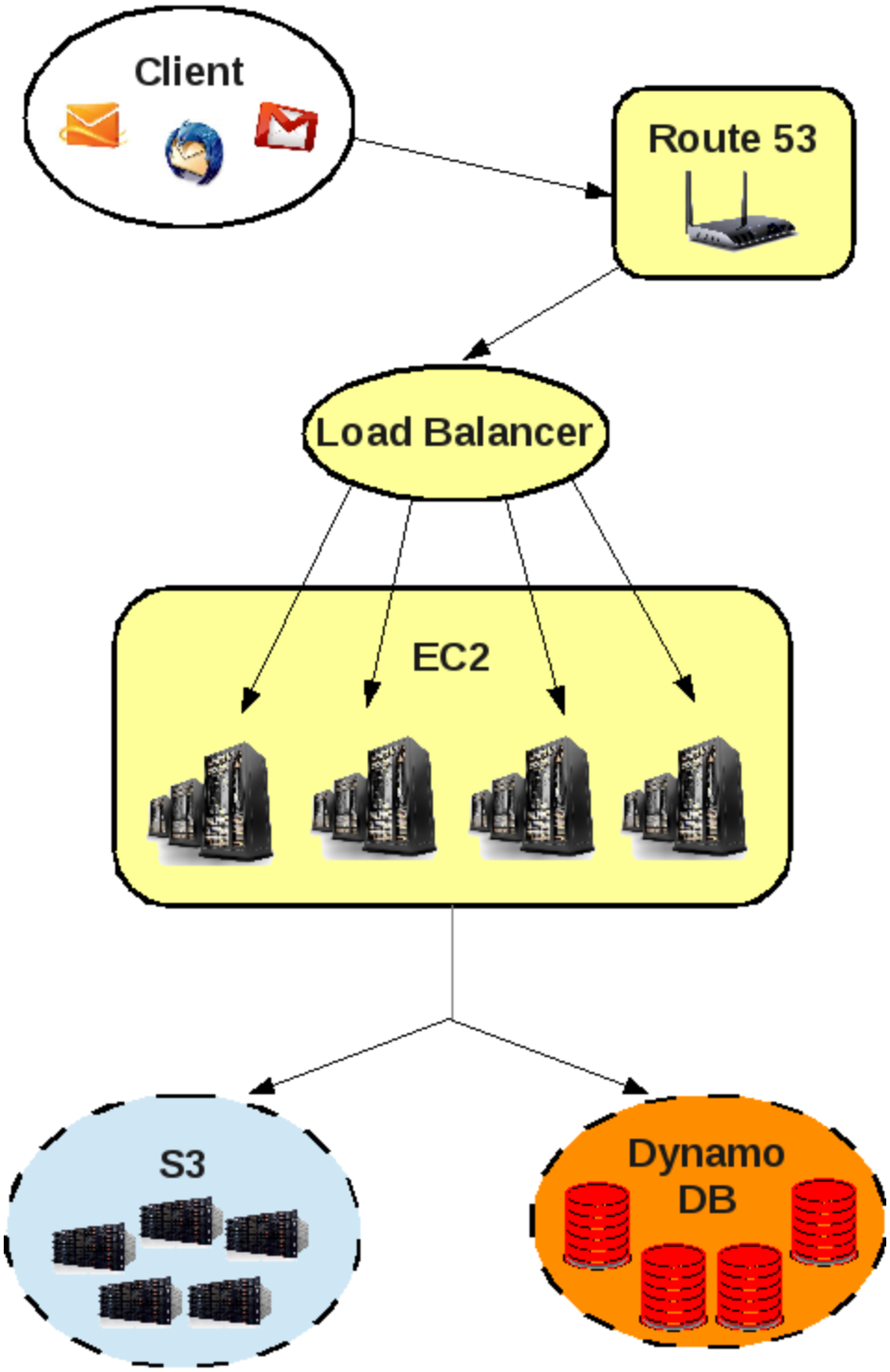

Fig. 2 - Schema architettura dettagliata. I servizi circondati da linee tratteggiate, nascondonoo agli utilizzatori la loro natura distribuita.

L’architettura risultante, vista con maggiore dettaglio, è quindi la seguente:

- **Primo Tier**: User Agent
- **Secondo Tier**: Route 53, Load Balancer, istanze EC2
- **Terzo Tier:** DynamoDB
- **Quarto Tier**: S3

Nei paragrafi che seguiranno verranno analizzate in dettaglio le singole componenti.

Tier 1

Client

Ogni client può utilizzare un qualsiasi user agent. In particolare test sono stati eseguiti con Thunderbird e Telnet. I client devono essere configurati per contattare il servizio **puro-mail.tk** rispettando il seguente le seguenti linee guida:

Server	**Porta**
SMTP	12365 (o 25)
SMTPS	12366
POP3	12450
POP3S	12451

Tier 2

Route 53

Route 53 mette a disposizione un servizio DNS per la risoluzione di indirizzi quali **puro-mail.tk**.
In particolare sono presenti quattro tipologie di record:

- **Record NS**: specifica uno o più server (alla creazione ne vengono specifici quattro) autorevoli per il dominio.
- **Record SOA**: restituisce informazioni autorevoli sulla zona DNS, incluso il server DNS principale, l'e-mail dell'amministratore, il numero seriale del dominio (utile per sapere se i dati della zona sono stati variati) e diversi timer che regolano la frequenza di trasferimento e la durata di validità dei record.
- **Record A**: associa a **puro-mail.tk** l’indirizzo (nome dns) del load balancer.
- **Record MX**: contiene informazioni relative al server SMTP responsabile del dominio **puro-mail.tk.** Collega un nome di dominio ad una lista di server di posta autorevoli per quel dominio. I record indicano anche la preferenza di un server rispetto ad un altro.

Elastic Load Balancer

Il Load Balancer agisce da dispatcher e distribuisce equamente le richieste fra le varie istanze EC2. Esso è impostato per inoltrare richieste in arrivo su una determinata porta, alle porte su cui sono in ascolto i server. La configurazione del Load Balancer è mostrata nello schema seguente:

Protocollo mail	**Protocollo entrante**	**Porta interna**	**Protocollo uscente**	**Porta esterna**
SMTP	TCP	12365	TCP	12365
SMTP	TCP	25	TCP	12365
SMTPS	TCP	12366	TCP	12366
POP3	TCP	12450	TCP	12450
POP3S	TCP	12451	TCP	12451

Il motivo per cui non sono state utilizzate le porte standard dei protocolli SMTP/s e POP3/s risiede in due motivazioni:

- E' possibile eseguire il servizio senza richiedere privilegi da super user, come da requisiti
- Il Load Balancer di Amazon può inoltrare richieste unicamente sulle porte 25 (SMTP), 80 (HTTP), 443 (HTTPS) e nell'intervallo [1024, 65535]

E' stata replicata la voce relativa al protocollo SMTP in modo da rendere il server SMTP del dominio, contattabile anche da server di altri domini, permettendo l'invio di email da utenti estranei al servizio verso utenti interni allo stesso.

Availability Zone - Tolleranza ai guasti

Il load balancer è configurato in modo da distribuire equamente le richieste fra Availability Zone di una stessa regione; una Availability Zone è una locazione all'interno di una regione progettata in modo da essere isolata dalle altre Availability Zone in modo da aumentare la resistenza alle failure. La politica di distribuzione delle istanze lungo più Availability Zone rende quindi il sistema più **tollerante ai guasti**.
E' importante sottolineare come il numero di istanze per Availability Zone debba essere equamente distribuito, in quanto il bilanciamento avviene fra le zone e non le istanze; se, per esempio, venissero create cinque istanze nella zona A e due nella zona B, equidistribuendo il carico fra A e B, le due istanze in B si troverebbero a servire lo stesso carico delle cinque istanze in A. Questa problematica è risolta in maniera automatica utilizzando il tool di **autoscaling**, di cui verranno mostrati gli aspetti salienti nel paragrafo relativo ad EC2 ed in quello dedicato al Testing.

Elastic Compute Cloud (EC2)

Amazon EC2 offre un servizio di tipo IaaS (Infrastructure as a Service), questo vuol dire che il cliente ha a disposizione una virtualizzazione di risorse hardware e software di un sistema (CPU, RAM, Network, OS,

etc.), che può utilizzare per eseguire il proprio software. In EC2 ogni macchina virtuale è rappresentata da una istanza che può essere replicata a piacere.
Al creazione di una nuova istanza, o di un insieme di istanze, è possibile decidere le relative caratteristiche tecniche. Tramite la definizione di una Key Pair è possibile connettersi alle istanze precedentemente create. Creando un Security Group si definiscono le porte aperte dell'istanza, ossia le porte su cui essa si mette in ascolto e su cui, di conseguenza, può rispondere.
Su ogni istanza EC2 è stato eseguito il deployment dell'applicazione, una versione pesantemente modificata ed adattata di GreenMail, come descritto nella sezione introduttiva.
In figura n. 3 è schematizzato il funzionamento dell'applicazione.

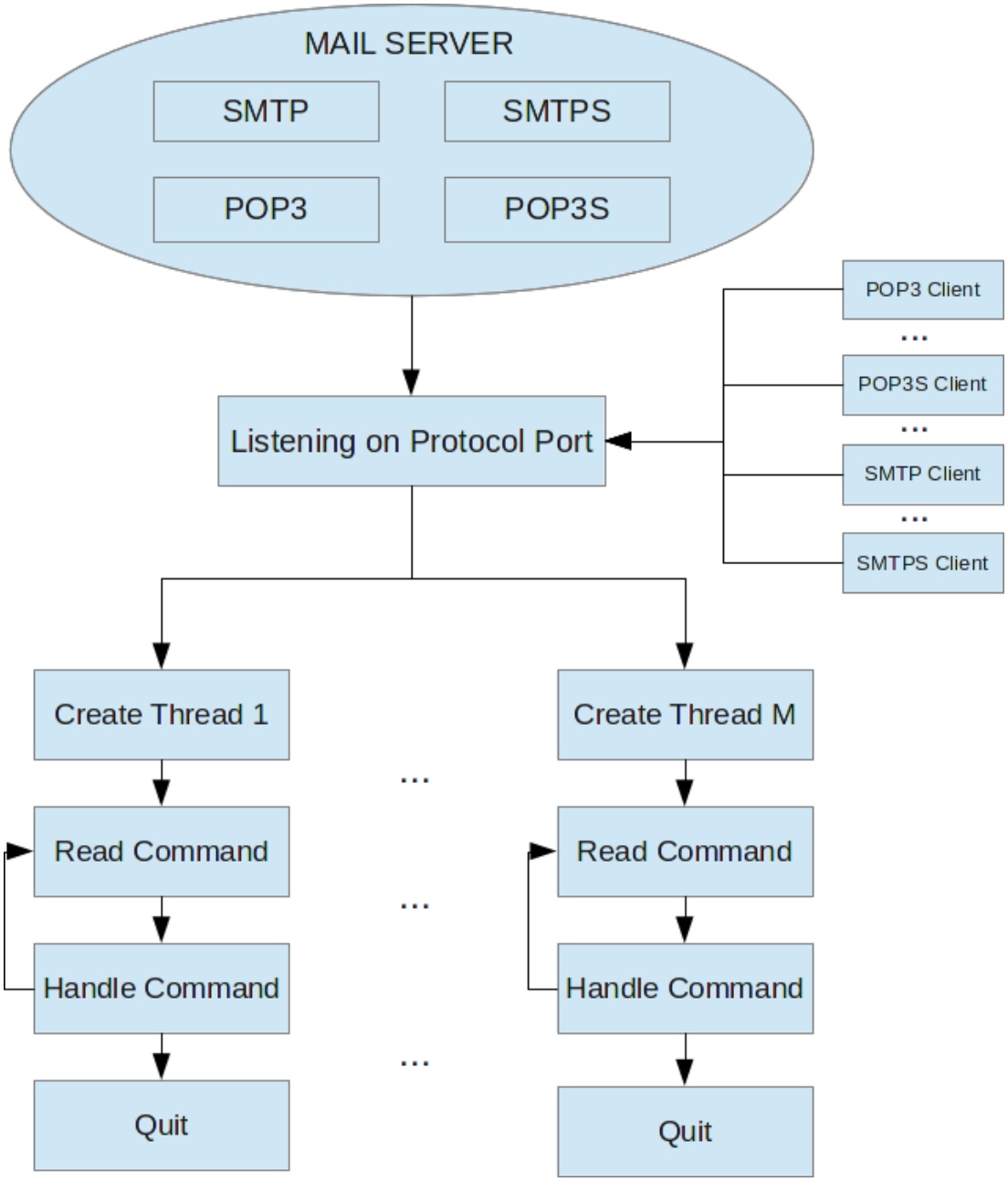

Fig.3 - Schema server mail

Il comportamento dell'applicazione può essere schematizzato nelle seguenti fasi:

1. Al lancio dell'applicazione, vengono creati quattro server, uno per protocollo supportato. I server si pongono in ascolto sulle porte precedentemente specificate.
2. All'arrivo di una richiesta di connessione su una specifica porta, viene creato un handler (della tipologia della comunicazione richiesta) che si occupa di gestire le richieste del client.
3. L'handler legge i comandi inviati dal client e risponde di conseguenza. Ogni comando può richiedere il contatto di DynamoDB o S3. Terminata la comunicazione col client l'handler termina.
4. Ricomincia dal passo 3.

Caratteristiche del server

Qui di seguito vengono elencate alcune scelte legate al funzionamento del servizio.

- **I comandi sono tutti sincroni**; poichè il client prima di inviare il successivo comando, aspetta la risposta del comando appena inviato, utilizzare chiamate asincrone non comporterebbe alcun vantaggio.
- Gli oggetti che si interfacciano con i servizi DynamoDB e S3 (cfr. Descrizione e Struttura codice - **Servizi**), ed i controllori che richiamano i metodi di tali oggetti (cfr. Descrizione e Struttura codice - **Manager**) sono stati modellati come oggetti **singleton**; ogni handler e quindi ogni connessione condivide gli stessi oggetti. In questo modo si diminuisce l'occupazione di memoria, evitando la replicazione di tali oggetti.
- I comandi **STAT** e **LIST** richiedono informazioni su tutte le mail salvate nella INBOX, in particolare per ogni mail nella cartella - anche per quelle già scaricate dal client - viene richiesto il relativo UID e la dimensione. Questi comandi sono seguiti da RETR, che recupera uno specifico messaggio; il client invia un comando RETR per ogni mail riconosciuta come nuova. Per evitare di caricare molte informazioni inutili, all'inivio dei comandi STAT e LIST il server carica da DynamoDB per ogni elemento solamente i campi UID e SIZE, tralasciando gli altri metadati e la mail intera presente in S3. Solo quando il client effettivamente richiede il recupero intero di un messaggio, vengono recuperate tutte le informazioni dello stesso presenti su DynamoDB e S3. Tale politica è stata scelta per ridurre il più possibile le chiamate ai servizi, aumentando le prestazioni e riducendo i costi.
- Non è stato implementato alcun meccanismo di contollo che vieti la possibilità a due o più utenti di connettersi con lo stesso account tramite POP3. DynamoDB di base non offre alcun meccanismo di locking, lasciato agli svilluppatori del lato applicativo. Si è ragionato su una possibile soluzione: appena un utente viene servito, viene valorizzato un campo su DynamoDB (un booleano); nel momento in cui un altro client tenta di fare una richiesta per il medesimo utente, leggendo quel campo si accorge che attualmente quell'account è già in uso, e quindi la comunicazione viene interrotta. Tale soluzione è sicuramente facile da implementare, ma nasconde un grande problema legato alla resistenza alle failure; se dopo aver scritto su DynamoDB il campo di cui sopra, il server subisse un crash, l'utente sarebbe irraggiungibile a tempo indeterminato perchè considerato sempre come attualmente servito da qualche altro server. Per superare tale problema si potrebbe creare un meccanismo più complesso, in cui sulla tabella User viene aggiunto non un semplice booleano, ma una stringa contenente l'ip del client e il nome dns (è sconsigliato utilizzare gli ip perchè non statici) della istanza EC2 servente tale client, la quale si salvi su un log la medesima stringa; quando un'altra istanza EC2 legge questo campo, contatta l'istanza servente per verificare effettivamente che questa stia servendo l'utente. Tale soluzione risulta piuttosto complicata ed è stata scartata. Infatti non risulta una limitazione troppo grande l'aver rilassato il vincolo su utenti POP3

contemporanei. Innanzitutto è una situazione molto rara, che non dovrebbe mai verificarsi; anche nel caso in cui si verifichi, se il messaggio non viene cancellato una volta prelevato, entrambi gli utenti verrebbero serviti; l'unico problema si verificherebbe se un utente appena finito di scaricare il messaggio, lo cancellasse; in tale situazione semplicemente al secondo utente arriverebbe un messaggio di errore, sull'impossibilità di scaricare il messaggio.

Resistenza a failure nella chiamate a S3 e DynamoDB

Alcune operazioni eseguite dai server mail, possono far nascere delle inconsistenze tra le informazioni salvato nelle tabelle di DynamoDB e quelle salvate in S3. I metodi che, in particolare, possono far nascere questi scenari sono la *append* e la *expunge*. La prima viene eseguita dal server SMTP quando riceve il comando DATA, la seconda eseguita dal server POP3 quando riceve il comando QUIT.

Il comando DATA, in particolare, precede l'invio da parte del client del corpo della mail; il server quindi una volta ricevuto il messaggio per intero, attraverso il metodo *append* procede con il salvataggio, su S3 e DynamoDB, dello stesso e delle informazioni che lo riguardano.
Questa operazione può essere vista come composta da due fasi:

- salvataggio su DynamoDB
- salvataggio su S3.

Ovviamente, è possibile che, queste due procedure non vengano entrambe portate a termine con successo, a causa di errori di varia natura (es. crash); se non fosse presente nessuna logica nella gestione di queste due operazioni, avremmo due possibili scenari critici:

1. il primo, che corrisponde allo stato in le informazioni riguardanti la mail sono state correttamente salvate nella tabella di DynamoDB mentre non lo è stato il salvataggio dell'intera mail in S3;
2. il secondo in cui la mail è stata correttamente salvata su S3 ma su DynamoDB non sono presenti le informazioni che la riguardano.

L'ultimo scenario è quello meno auspicabile poichè, le informazioni presenti nella tabella Metamail di DynamoDB, ed in particolare i campi *bucketname* e *objectname*, sono gli unici riferimenti che possono essere utilizzati nel sistema per recuperare la mail presente in uno dei bucket di S3.
Proprio per questo motivo, la logica che si è scelta per eseguire queste operazioni è stata quella di inserire, prima di tutto, il contenuto in DynamoDB e poi procedere con l'inserimento della mail nel bucket di S3.
In questo modo, il secondo scenario non può mai realizzarsi, perchè il sistema è stato implementato per interrompere l'intera procedura nel caso in cui la prima fase non venga portata a termine con successo.
Nel caso possibile, in cui il sistema raggiunga lo stato descritto dal primo scenario, lo stato di inconsistenza presente verrà corretto alla prima invocazione della procedura per il recupero della mail incriminata. Infatti in questo caso, letti i campi *bucketname* e *objectname* da DynamoDB (fase1), il server tenta di recuperare il file da S3, ma non trovandolo, lancia una *NotFileFoundException* ed elimina la voce all'interno della tabella di DynamoDB, risolvento così l'inconsistenza, e procede quindi con l'invio di un messaggio di avvenuto errore all'utente.
Il client che ha ricevuto l'errore mentre stava inviando la mail, è quindi consapevole che qualcosa è andato storto e che quindi la sua mail potrebbe non essere recuperabile dal destinatario.

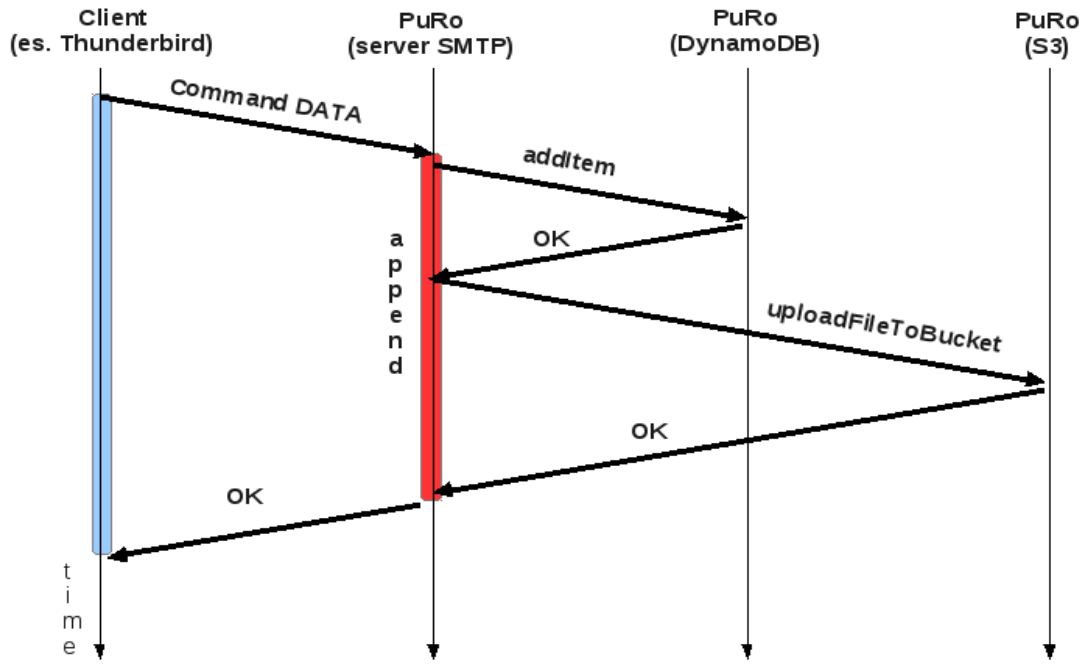

Fig.4 - Esecuzione metodo append

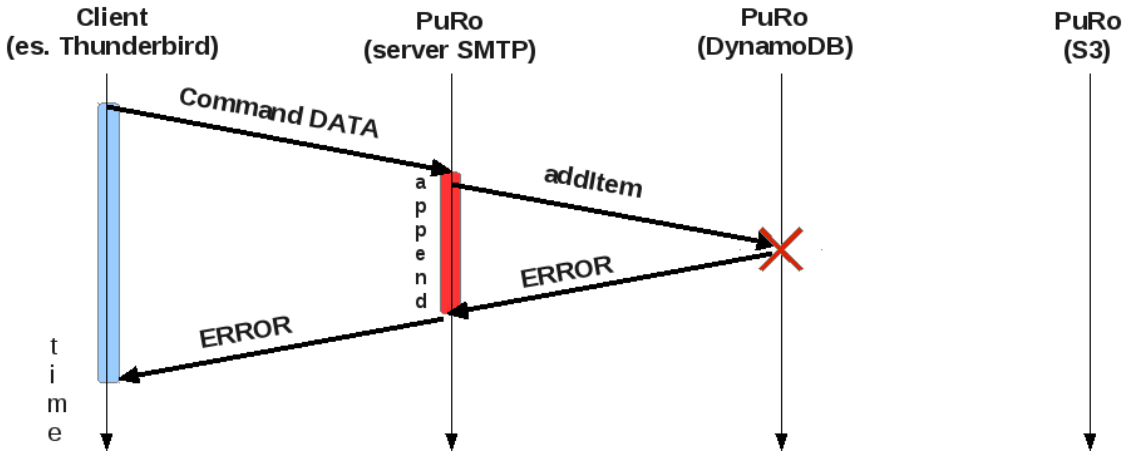

Fig.5 - Errore esecuzione metodo append

Per quanto riguarda il comando QUIT abbiamo la situazione opposta, in quanto con questo comando, oltre a terminare la sessione con il server, si procede anche con l'eliminazione lato server dei messaggi che sono stati contrassegnati come *eliminabili*. Chiaramente anche l'eliminazione può essere vista come composta da due fasi distinte che quindi, posso andare incontro ad errori distinti. La logica di eliminazione è, come si ci può aspettare, inversa a quella di inserimento di un nuovo messaggio. Infatti ciò che è stato scelto di fare, è stato di implementare il sistema in maniera tale che venga prima eseguita la rimozione del messaggio dal bucket di S3 per poi procedere con la rimozione della voce dalla tabella *MetaMail* di DynamoDB. Per il resto valgono le stesse considerazioni fatte precedentemente.

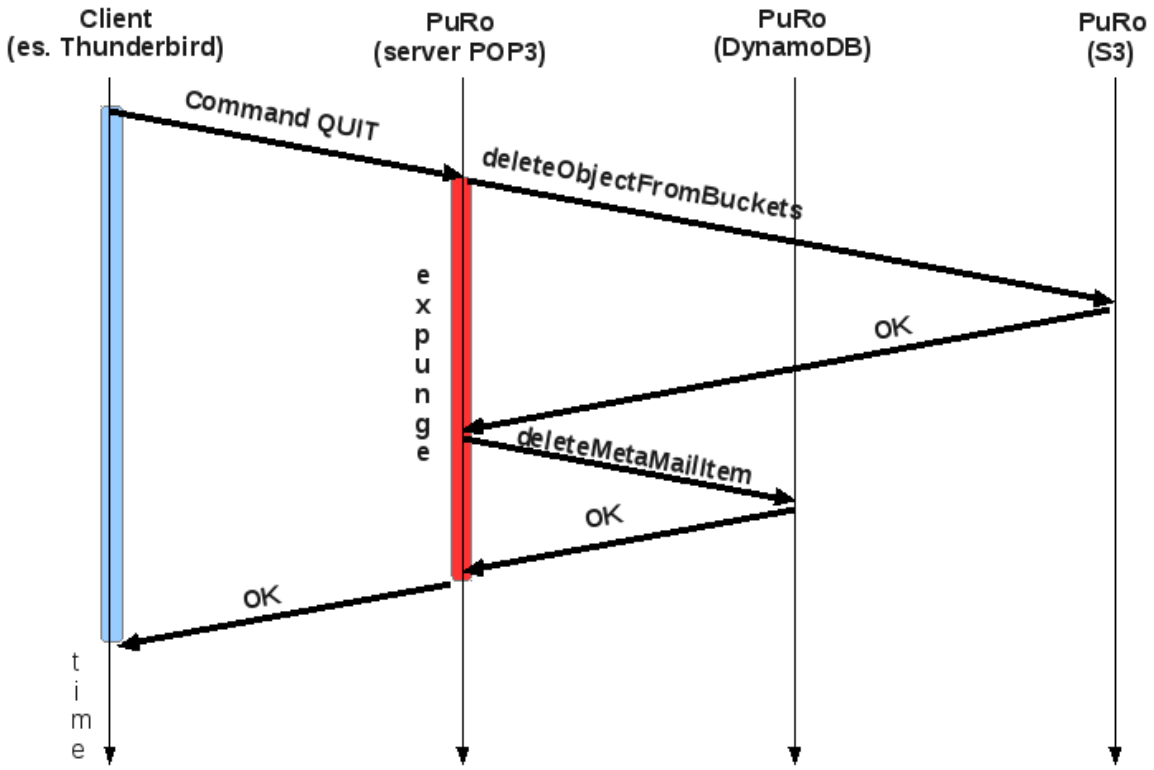

Fig.6 - Esecuzione metodo expunge

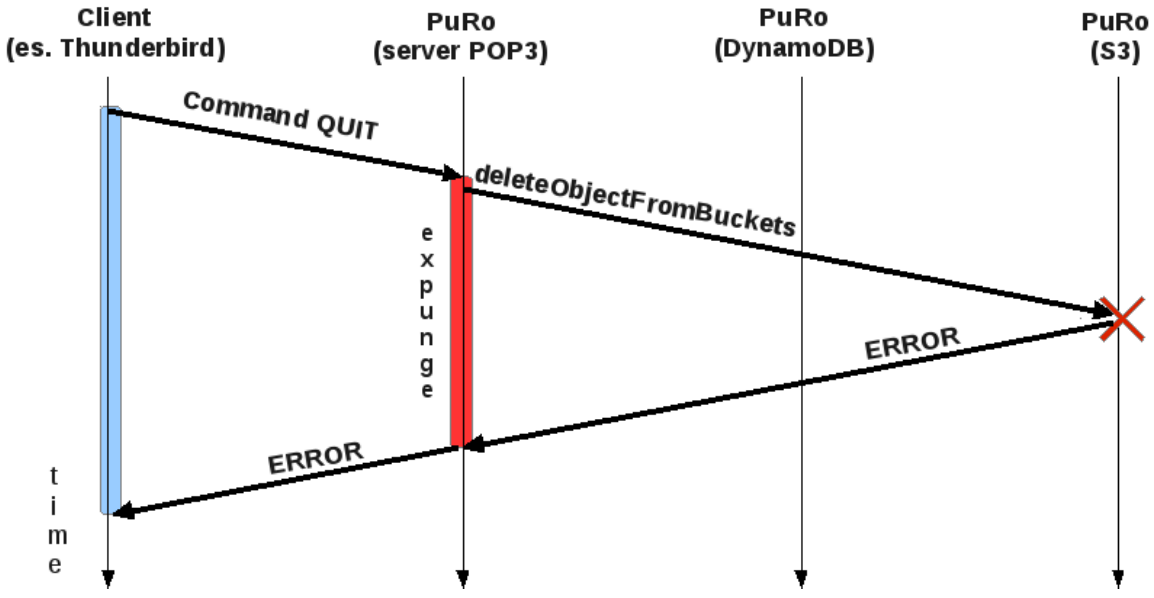

Fig.7 - Errore esecuzione metodo expunge

Why Greenmail?

Le motivazioni che ha portato alla scelta di GreenMail tra i vari server di posta elettronica disponibili sul web, sono legate alla comprensibilità della sua logica strutturale. Nonostante la documentazione non sia del tutto completa, grazie ai commenti presenti nel codice, è stato possibile sfruttare le potenzialità e la flessibilità di questa suite in maniera abbastanza semplice.
Un problema, che riteniamo sia necessario portare all'attenzione, è la presenza di interfacce; sebbene lo scopo delle stesse sia incrementare la modularità e l'estendibilità del codice, in tale situazione più che un vantaggio hanno rappresentato un limite, forzando lo sviluppatore a seguire una struttura e uno stile implementativo (in particolare nel lancio di eccezioni specifiche) troppo legati alla versione originale di Greenmail. Nello sviluppo si è quindi deciso di eliminare tali interfacce.

Estensione di GreenMail

La prima modifica che è stata effettuata, è quella volta ad eliminare la limitazione presente in GreenMail legata all'utilizzo esclusivo della memoria RAM per il salvataggio dei messaggi degli utenti e delle informazioni sugli utenti stessi. Questo ovviamente comportava la perdita di qualsiasi informazione ogni volta che l'esecuzione del sistema venisse interrotta. Per evitare ciò, sono state implementate delle funzionalità per il salvataggio persistente degli utenti registrati al sistema e dei messaggi inviati e ricevuti, attraverso l'uso dei servizi di Amazon S3 e DynamoDB. Ciò permette una maggiore scalabilità e resistenza alla failure, poichè attraverso questi servizi, viene assicurato in modo automatico, sia un determinato grado di replicazione, sia un disponibilità dei dati definibile grande a piacere, attraverso la modifica dei valori di throughput in lettura e scrittura.
In relazione alla gestione degli utenti, è da sottolineare inoltre il fatto che che GreenMail perma l'accesso al sistema anche ad utenti non registrati, creando nel momento del "login" un riferimento in memoria. Questo, ovviamente non è ciò che ci si aspetta da un server di posta, quindi è stato implementato un semplice meccanismo di login, il quale prevede una precedente fase di registrazione in cui, l'utente ha la possibilità di scegliere un proprio username ed una password. La password viene criptata con un hash MD5 a 128bit e salvata nella tabella User in DynamoDB insieme alle altre info fornite dall'utente. Nella fase di login, oltre a verificare la presenza dell'utente tra quelli registrati, l'accesso viene consentito solo dopo che il confronto tra la password inserita e quella presente nel sistema resituisca esito positivo.
E' stato inoltre aggiunto il supporto per i comandi EHLO e AUTH PLAIN, come meccanismo, seppur semplice, di autenticazione al server SMTP.

Autoscaling

Il punto cruciale dell'applicazione realizzata è chiaramente costituito dai server mail, localizzati sulle istanze EC2; essi si occupano di smaltire le richieste dei client instaurando all'occorrenza comunicazioni con altri servizi. Si giunge quindi ad uno dei principali problemi dei servizi web: il dimensionamento degli stessi, ossia la quantità di risorse da allocare per poter fornire un livello di servizio accettabile in ogni scenario. Una delle soluzioni adottate in passato consisteva nel sovradimensionare le risorse; sebbene questo garantisca elevata disponibilità del servizio, tutto ciò avviene a fronte di un costo molto elevato. La soluzione ideale consiste nell'allocare/deallocare risorse in maniera automatica all'occorrenza in base alle variazioni di carico. **Autoscaling** è il servizio Amazon addetto a questo compito.
E' possibile effettuare autoscaling delle istanze EC2. Affinchè si possa effettuare tale operazione è necessario ottenere statistiche su alcune metriche ritenute importanti per la qualità del sistema. La cattura ed analisi di tali dati avviene tramite il servizio **CloudWatch**, il quale può inoltre eseguire azioni (definite dall'utente) al superare di valori impostati come soglia di allarme.
E' possibile catturare le metriche di diverse dimensioni, singole istanze, istanze di un certo tipo ecc. Si è

deciso di analizzare le metriche appartenenti allo stesso AutoScaling Group; un AutoScaling Group definisce un insieme minimo e massimo di istanze contemporaneamente attive. Se un'istanza fallisce e si scende sotto al limite minimo, un'altra istanza viene creata. Altre istanze si aggiungono dinamicamente, in base ad polizze definite secondo le metriche di cui sopra.
Al fine dello scaling delle istanze è importante analizzare metriche relative ad EC2 ed al Load Balancer. A seguire verrà effettuato un'analisi di tali metriche, di volta in volta facendo considerazioni sulla relativa utilità o meno nel contesto del dominio in questione.

CloudWatch EC2

- **Metriche incluse**
  - **CPUUtilization**: percentuale di CPU utilizzata. Poichè questa metrica è ben rappresentativa dello stato di una istanza, si è deciso di prenderla in considerazione. In particolare viene analizzato il valor medio.
- **Metriche escluse**
  - **NetworkIn**: Numero di byte ricevuti su tutte le interfacce di rete. Se una istanza satura completamente la banda a disposizione, la qualità percepita del servizio diminuisce, e quindi costituisce un parametro critico del sistema. Inizialmente tale metrica era stata inclusa, ma a seguito della fase di testing si è ritornati indietro sui propri passi; infatti non conoscendo la banda disponibile alle istanze, non è possibile definire una soglia critica in base a criteri oggettivi.
  - **NetworkOut**: Numero di byte inviati su tutte le interfacce di rete. Similmente alla metrica precedente, questo parametro è stato escluso.
  - **DiskReadOps**: Numero di operazioni di lettura da disco completate. Poichè i server utilizzano in maniera trascurabile il disco, non è considerata utile.
  - **DiskWriteOps**: Numero di operazioni di scrittura su disco completate. I motivi sono i medesimi sopra esposti.
  - **DiskReadBytes**: Numero byte letti da disco.
  - **DiskWriteBytes**: Numero byte scritti su disco.
  - **StatusCheckFailed_System**: Indica se l'istanza ha fallito lo StatusCheck_System, ossia se non è stato possibile inviare pacchetti all'instanza negli ultimi 5 minuti, problema dovuto ad AWS stesso. Poichè se un'istanza fallisce, le istanze esistenti vedono aumentare il carico, se il carico è eccessivo, l'utilizzo della cpu supera la soglia critica e viene effettuato auto scaling in base alla metrica **CPUUtilization**. Non sono quindi da prendere altre azioni.
  - **StatusCheckFailed_Instance**: Indica se l'istanza ha fallito lo StatusCheck_Instance, ossia se non è stato possibile inviare pacchetti al sistema operativo dell'istanza negli ultimi 5 minuti. Generalmente dovuto a problemi di malconfigurazione. Per i motivi sopra esposti legati alle metrica precedente, non viene considerata.
  - **StatusCheckFailed**: Indica se un controllo fra StatusCheck_Instance e StatusCheck_System sia fallito. Avendo ecluso le due metriche precedenti, stesso discorso vale per questa metrica.

Ogni metrica si può analizzare in relazione ad una specifica **dimensione**:

- **AutoScalingGroupName**: vengono analizzate unicamente le metriche in relazione alle istanze appartenenti al gruppo di AutoScaling definito, dove per gruppo di AutoScaling si intende l'insieme di istanze su cui vengono applicate le polizze di Scaling In/Out.
- **ImageId**: metriche relative ad istanze basate sulla stessa AMI, Amazon Machine Imagine.

- **InstanceId**: metriche relative ad una istanza con l'id specificato.
- **InstanceType**: metriche relative a tutte le istanze del tipo designato.

Si è deciso di effettuare autoscaling in relazione ai valori osservati delle metriche delle istanze appartenenti al gruppo di **AutoScaling**, in modo da avere una visione complessiva della situazione di tutte le istanze in funzione.

CloudWatch Load Balancer

- **Metriche incluse**
  - Nessuna
- **Metriche escluse**
  - **Latency**: Tempo trascorso tra l'arrivo di una richiesta e la ricezione della relativa risposta al Load Balancer. In un primo momento era stata considerata utile per tenere sotto controllo i tempi di risposta percepiti dall'utente; a seguito di test si è verificato di come rimanesse sempre ampiamente sotto la soglia critica, e quindi si è deciso in definitiva di scartarla.
  - **RequestCount**: Numero di richieste gestite dal Load Balancer. Non fornisce informazioni sul carico delle singole istanze.
  - **HealthyHostCount**: Numero di istanze risultati "healthy", cioè che hanno superato gli HealthCheck descritti nel paragrafo precedente. Per le medesime considerazioni sulle metriche in questione, questa non è stata presa in considerazione.
  - **UnHealthyHostCount**: Numero di istanze risultate "unhealthy", cioè non hanno superato gli HealthCheck descritti nel paragrafo precedente. Per le medesime considerazioni sulle metriche in questione, questa non è stata presa in considerazione.
  - **HTTPCode_ELB_4XX**: Numero di risposte generate dal Load Balancer con codice 4xx (client error). Se il client ha inviato una richiesta errata, ciò non porta informazioni sullo stato delle istanze.
  - **HTTPCode_ELB_5XX**: Numero risposte generate dal Load Balancer con codice 5xx, ossia errore del server. Tali errori vengono generati qualora non fossero presenti istanze EC2 "healty" (o non fossero presenti per nulla), o se la capacità del Load Balancer è stata superata.
  - **HTTPCode_Backend_2XX**: Numero risposte generate dal BackEnd con codice 2xx (successo). Non indicativo di failure.
  - **HTTPCode_Backend_3XX**: Numero risposte generate dal BackEnd con codice 3xx (user action required). Non indicativo di failure.
  - **HTTPCode_Backend_4XX**: Numero risposte generate dal BackEnd con codice 4xx (client error). Se il client ha commesso un errore, non devono essere eseguite azioni di scaling.
  - **HTTPCode_Backend_5XX**: Numero risposte generate dal BackEnd con codice 5xx (client error). Un errore del server non è necessariamente correllato alla situazione del carico.

- **Dimensioni**
  - **LoadBalancerName**: le metriche vengono analizzate solo in relazione alle istanze assegnate al Load Balancer. Si è deciso di scegliere questa dimensione, in quanto adeguata a fornire misure globali.
  - **AvailabilityZone**: le metriche vengono analizzate solo in relazione alle istanze assegnate al Load Balancer e presenti nella AvailabilityZone specificata.

Tier 3

DynamoDB

Descrizione

DynamoDB è un servizio di gestione di database **NoSQL** (**N**ot **O**nly **SQL**).
In un database relazionale, una tabella ha uno schema predefinito che descrive le proprietà come il nome della tabella, la chiave primaria, il nome delle colonne ed i tipi di dati. Tutti i record salvati nella tabella devono avere lo stesso insieme di colonne.
In un database NoSQL, ad eccezione della chiave primaria, le tabelle sono libere da uno schema prefissato; i vari elementi contenuti in una tabella posso avere un numero di attributi qualsiasi.
Quando si procede con la creazione di una tabella su DynamoDB, è necessario fornire la sua primary key e i valori del throughput in lettura e scrittura che si pensa debba soddisfare la tabella in questione.
Amazon DynamoDB supporta i seguenti tipi di chiavi primarie:

- **Hash**: la chiave primaria è costituita da un solo attributo, di tipo hash. Amazon DynamoDB costruisce un indice hash non ordinato su questo attributo della chiave primaria.
- **Hash e Range**: la chiave primaria è costituita da due attributi, il primo di tipo hash ed il secondo di tipo range. Amazon DynamoDB costruisce un indice hash non ordinato e un secondo indice ordinato sull'attributo indicato come range.

Amazon DynamoDB è costruito per supportare carichi di lavoro di ogni grandezza con tempi di risposta predicibili e con bassa latenza. Per assicurare alta disponibilità e risposte a bassa latenza, Amazon DynamoDB richiede di specificare i valori di throughput che si richiedono alla tabella e usa tale informazione per riservare sufficienti risorse hardware e per partizionare i dati in modo appropriato su diversi server per raggiungere il throughput richiesto. Questi valori possono essere alterati, se la propria applicazione richiede una diversa capacità di accesso. I valori di throughput richiesti devono essere specificati in termini delle seguenti capacity units:

- **Read capacity units**: è il numero di letture di oggetti, consistenti, fino ad 1 KB al secondo.
- **Write capacity units**: è il numero di scritture di 1 KB in un secondo.

DynamoDB vs SimpleDB

Amazon offre anche un altro servizio di gestione di database NoSql, ossia SimpleDB. I servizi offerti da entrambi sono simili, ma SimpleDB è caratterizzato da restrizioni non presenti in DynamoDB; in particolare ogni tabella ha una dimensione massima di 10 GB e un tetto massimo sul numero di richieste in parallelo supportate (request capacity limit); la dimensione massima delle tabelle non è un problema immediato, in quanto i metadati non costituiscono un gran peso in termini di dimensione; il limite sul numero di richieste in parallelo, invece, comporta la necessità di creare più tabelle simili per poter gestire carichi di lavoro maggiori, funzionalità che invece Dynamo offre di base. In fase di testing, sono stati riprodotti scenari molto stressanti per il sistema, scenari che SimpleDB non sarebbe stato in grado di gestire se non con necessità di interventi manuali. Di conseguenza SimpleDB risulta più utile nel supporto di servizi con basso carico previsto. DynamoDB si è rilevato più flessibile da questo punto di vista, e di conseguenza è stato eletto come il più adatto per questo sistema.

Metadati

Come precedentemente anticipato, si è scelto di separare la logica dei metadati da quella dei dati. DynamoDB è il servizio addetto alla gestione dei primi. Spostare la logica dei dati su questo servizio è una soluzione non proponibile per via dei limite imposti sulla grandezza dei singoli oggetti, 64 KB, che per mail contenenti allegati è sicuramente un limite troppo restrittivo. Per altre motivazioni si è deciso di assegnare S3 unicamente alla gestione dei dati (cfr. paragrafo S3).
Le informazioni salvate in DynamoDB sono di tre tipo:

- **Tabella User**: informazioni sugli utenti del servizio
- **Tabella Folder**: informazioni sulle mailbox
- **Tabella MetaMail**: informazioni sulle singole mail

Tabella User

E' la tabella che contiene le informazioni riguardanti gli utenti che sono iscritti al servizio di posta elettronica. Viene utilizzata in fase di registrazione per salvare le credenziali e viene richiamata ogni volta che un utente prova ad autenticarsi per accedere al servizio.

Nome Campo	**Tipo**	**Descrizione**
email	String (S)	**HASHKEY** Tabella. Contiene l'email in lower case dell'utente.
password	String (S)	Digest MD5 password (per evidenti questioni di sicurezza non sono state salvate in plaintext)
firstname	String (S)	Nome utente
lastname	String (S)	Cognome utente
folder	String Set (SS)	Insieme cartelle e sottocartelle appartenenti all'utente
lastupdate	*Number (N)*	*Data ultimo aggiornamento. Estensione **IMAP***
region	*String Set (SS)*	*Insieme delle regioni su cui sono disponibili i servizi per l'utente. Al momento della registrazione ne viene indicata solamente una, ma in seguito è possibile aggiungerne delle altre al fine di fornire all'utente un servizio più performante. Estensione **Futura**.*

Il campo **lastupdate** è stato inserito per una futura integrazione del protocollo IMAP. Questo protocollo fornisce la possibilità di accessi paralleli al medesimo account. Di conseguenza i metadati sull'utente potrebbero venir modificati da qualcun'altro (si pensi in particolare all'aggiunta di una nuova cartella all'insieme delle **folder**) e per garantire la visione di tali modifiche in tempo reale, è necessario confrontare la data di aggiornamento in memoria con quella nel database, ed eventualmente se discordanti aggiornare le prime.
Il campo **region** è stato previsto per un'eventuale estensione futura. Assegnare più regioni ad un utente implica replicare le sue informazioni (metadati e dati) su servizi (che fanno uso di risorse apparenenti ad altre regioni) dislocati in più regioni, in modo da diminuire la latenza in caso di accesso da altre regioni

(accesso che verrebbe indirizzato alla regione più vicina tramite Route53). Una tale estensione introdurrebbe problemi di consistenza fra repliche dei servizi dislocate in regioni differenti (cfr. Sviluppi future).

Tabella Folder

E' la tabella che contiene implicitamente la struttura gerarchica delle folder. In cima alla gerarchia è presente la root *#mail*, i cui diretti figli sono i namespace delle cartelle di ogni singolo utente (strutturati come *#mail.hash-mail-utente*). La root e i namespace sono presenti unicamente in memoria, in quanto soggetti a modifiche frequenti che avrebbero aumentato gli accessi al database in maniera ingiustificata (non è possibile salvare mail direttamente nella root e nei namespace ed ogni aggiunta eliminazione di una cartella o creazione di un utente avrebbe richiesto la modifica delle relative voci nel database).

Nome Campo	**Tipo**	**Descrizione**
name	String (S)	**HASHKEY** Tabella. Nome folder così strutturato: *#mail.hash-mail-utente.nome-folder*
nextUID	Number (N)	Identificativo del prossimo messaggio salvato nella folder. E' univoco relativamente alla stessa folder, ma non fra più folder.
parent	String (S)	Nome folder genitore, presente in un'altra entry della tabella (se diverso dalla root e dal namespace)
children	String Set (SS)	Insieme delle subfolder
selectable	*String (S)*	*Specifica se una folder possa o no contenere messaggi. Cartelle non selezionabili sono la root e i namespace. Presente nella versione base di GreenMail, è stata lasciata per un'eventuale estensione **futura**.*
lastupdate	*Number (N)*	*Data ultimo aggiornamento. Estensione **IMAP***

Come nel caso della tabella User, anche qui il campo lastupdate è stato inserito per supporto ad un'eventuale implementazione del protocollo IMAP. In questo caso bisogna considerare non solo la possibilità a più utenti di connettersi in contemporanea utilizzando il medesimo account, ma anche la condivisione di una stessa folder fra più utenti di diversi account. Le motivazioni sono rintracciabili nelle stesse fornite precedentemente.
Il campo selectable, presente nella versione originale di Greenmail, sebbene non abbia trovato un'utilità immediata con le implementazioni di POP3 ed SMTP, è stato lasciato per garantre retrocompatibilità e per un eventuale sviluppo futuro.

Tabella MetaMail

E' la tabella contenente tutti i riferimenti che riguardano ogni email che viene gestita dal sistema e che sono necessari al fine di recuperare l'intero corpo del messaggio ed i suoi allegati attraverso il servizio Amazon S3.

Nome Campo	**Tipo**	**Descrizione**
folder	String (S)	**HASHKEY** Tabella. Nome folder in cui è salvato il messaggio

uid	Number (N)	**RANGEKEY** Tabella. Numero che identifica univocamente un messaggio all'interno della stessa folder (cfr. nextUID nella tabella Utenti)
bucket	String (S)	Nome bucket S3 (cfr sezione S3)
objectname	String (S)	Nome oggetto salvato nel bucket S3, il cui contenuto rappresenta intera mail in formato MIME
size	Number (N)	Dimensione in byte del messaggio
header	*String (S)*	*Header Messaggio. Estensione **IMAP***
timestamp	*Number (N)*	*Data inserimento messaggio. Estensione **IMAP***
flags	*String Set (SS)*	*Flag (di sistema e user defined) assegnati al messaggio. Flag di sistema: ANSWERED, DELETED, DRAFT, FLAGGED, SEEN. Estensione **IMAP***
lastupdate	*Number (N)*	*Data ultimo aggiornamento. Estensione **IMAP***

Associando il nome del bucket al nome dell'oggetto, si ottiene una chiave univoca naturale per identificare il messaggio salvato nello storage S3, i cui metadati sono presenti in DynamoDB.
I campi header, timestamp, flags e lastupdate sono presenti per supportare l'eventuale successiva implementazione di **IMAP**. Header contiene l'header della mail, i cui singoli campi possono essere ottenuti tramite specifiche opzioni del comando Fetch di IMAP. Timestamp è utilizzata nei comandi Fetch e Search di IMAP. Flags permette l'inserimento di altri flag. Lastupdate è analogo ai medesimi campi delle tabelle precedenti.

Caratteristiche servizio distribuito

Quando un client (istanza EC2) accede alle funzionalità di DynamoDB, viene mascherata la struttura distribuita del servizio e mostrato come se l'accesso avvenisse su un database in un singolo server. Per gestire carichi variabile DynamoDB è gestito per allocare in automatico risorse hardware. Inoltre garantisce affidabilità aumentanto la resistenza alle failure tramite la replicazione dei dati. Tale replicazione non è definibile in maniera manuale dall'utente.
Essendo un servizio distribuito si pongono le usuali problematiche in termini di consistenza. DynamoDB offre due modalità in relazione alle operazioni di **READ**:

- **Consistent READ** (READ consistenti): una READ restituisce sempre l'informazione più aggiornata, ossia quella derivante dall'ultima WRITE in ordine temporale.
- **Eventual Consistent READ** (READ a consistenza finale): una READ non restituisce necessariamente l'ultima versione dell'oggetto. E' necessario un certo tempo affinchè le varie repliche dell'oggetto siano aggiornate alla medesima ultima versione. Questa opzione raddoppia il throughput messo a disposizione dalla tabella in termini di READ CAPACITY UNITS.

La scelta fra le due opzioni è da prendere tenendo conto il tradeoff prestazioni-consistenza.
Analizziamo il quesito tenendo presente la natura del dominio del problema.
Read consistenti restituiscono subito una mail appena scritta su database. Sulle specifiche di DynamoDB vi è descritto come l'aggiornamento fra più repliche generalmente si propaghi entro un secondo di tempo. Non

essendo un servizio mail un'applicazione realtime, un simile delay non costituisce assolutamente un problema. Quindi sembrerebbe che una tale opzione non risulti necessaria. D'altro canto andando ad utilizzare Read a consistenza finale, si ottiene un throughput delle letture raddoppiato. Di conseguenza si può dedurre come la scelta della seconda piuttosto che della prima opzione, comporti un aumento delle prestazioni in maniera gratuita, ossia senza andare in qualche modo a lesionare l'affidabilità del servizio. Per queste motivazioni **si è deciso di utilizzare Eventual Consistent READ**.

CloudWatch DynamoDB

Si è precedentemente descritto come sia possibile effettuare Scaling In e Out delle istanze EC2, ossia della possibilità di aumentarne o diminuirne il numero in relazione a determinate metriche. Non vengono purtroppo forniti tool simili in relazione a DynamoDB; i valori di throughput attesi specificati alla creazione di una tabella possono essere modificati solo manualmente.
Nel decidere il throughput da impostare per le tabelle, sono state analizzate varie metriche in fase di testing:

- **Metriche**:
    - **ConsumedReadCapacityUnits**: Numero di unità di lettura "consumate" nel periodo di tempo specificato. Fornisce importanti informazioni sul carico.
    - **ConsumedWriteCapacityUnits**: Numero di unità di scrittura "consumate" (sul totale predisposto) nel periodo di tempo specificato. Fornisce importanti informazioni sul carico.
    - **ThrottledRequests**: Numero di richieste eseguite oltre i limiti di throghput impostati fra le proprietà della tabella. E' indice di carico eccessivo e quindi di necessità di aumentare il throughput.
    - **ProvisionedReadCapacityUnits**: Numero di unità di lettura riservate per la tabella. Dà unicamente informazioni sul dimensionamento attuale della tabella.
    - **ProvisionedWriteCapacityUnits**: Numero di unità di scrittura riservate per la tabella. Dà unicamente informazioni sul dimensionamento attuale della tabella.
    - **ReturnedItemCount**: Numero di item restituiti da un'operazione di Scan o Query (descritte successivamente).
    - **SuccessfulRequestLatency**: Numero di richieste con successo nel periodo specificato. E' una misura che dà solo fornisce poche informazioni, se non associata al numero totale di richieste inviate.
    - **UserErrors**: Numero di richieste generanti errore 400 (client error). Non sintomatico di carico eccessivo.
    - **SystemErrors**: Numero di richieste generanti errore 500 (client error). Non sintomatico di carico eccessivo.

- **Dimensioni**
    - **TableName**: le metriche sono relative alla tabella specificata
    - **Operation**: le metriche sono relative alla singola operazione
        - **PutItem**: crea/rimpiazza elemento
        - **DeleteItem**: elimina elemento
        - **UpdateItem**: aggiorna elemento
        - **GetItem**: restituisce elemento
        - **BatchGetItem**: restituisce più elementi
        - **Scan**: restituisce un elemento eseguendo una scansione completa della tabella
        - **Query**: restituisce un elemento in base alla chiave primaria

Tier 4

S3

Il servizio di Storage S3 è destinato alla gestione di file di qualunque dimensione (il limite è di 5 TB per file, ma nel contesto di un servizio mail si può considerare una capacità illimitata); per questo motivo è stato scelto per la memorizzazione dei dati piuttosto che dei metadati, funzione assegnata a DynamoDB come precedentemente descritto.
Come mostrato nello schema in figura n. 2, il servizio cela all'utilizzatore una natura distribuita, andando a replicare i dati per garantire scalabilità, disponibilità e resistenza a failure, nel rispetto delle SLA. In quanto sistema distribuito, S3 offre diverse opportunità, le quali una ad una sono state analizzate ed eventualmente integrate nel progetto:

- **Reduced Redundancy Storage (RRS)**: opzione che permette di abbassare i costi di mantenimento dei dati diminuendo il grado di replicazione degli stessi. Si è deciso di **scartare** questa opzione, in modo da garantire un servizio più affidabile all'utente finale.
- **Versioning**: garantisce il salvataggio di più versioni dello stesso oggetto, in modo da semplificare il recupero di informazioni a seguito di operazioni di write o delete errate. Poichè S3 salva le mail dell'utente, che tralasciando i metadati non possono venir modificate, questa opzione non aggiunge in alcun modo beneficio al servizio, e quindi è stata **scartata**.
- **Region**: In S3 gli oggetti vengono salvati all'interno di contenitori, chiamati Bucket, disposte in specifiche regioni. Ogni regione offre diversi livelli di consistenza. Per garantire minore latenza in fase di testing, la scelta è ricaduta sull'unica regione europea, ossia l'**Irlanda**. Questa è caratterizzata dal seguente modello di consistenza:
    - Read After Write per PUT di nuovi elementi
    - Consistenza Finale per PUT di overwrite di elementi già esistenti
    - Consitenza Finale per DELETE

Relativamente alla memorizzazione dei dati, si è scelto di salvare nello storage le mail nel formato MIME. Di conseguenza vengono memorizzati header e body nello stesso oggetto.
Sebbene questo possa sembrare ridondante, in quanto le informazioni dell'header sono già presenti in Dynamo, in realtà precise motivazioni sono alla base di questa scelta:

- L'header in un servizio quale S3 occupa uno spazio non importante (in dimensione)
- Tenere header e body uniti in un unico file semplifica le operazioni di recupero delle mail. Una scelta diversa avrebbe richiesto un processo di merging fra le parti della mail prima di poterle inviare all'utente.
- Si potrebbe eliminare la ridondanza dell'header, andando ad utilizzare S3 per la gestione di dati e metadati eliminando DynamoDB. Una tale scelta oltre a risultare poco pulita, avrebbe anch'essa complicato il processo di recupero dei file; andando a creare file dedicati al mantenimento dei metadati, i singoli campi sarebbero stati recuperati solo dopo aver prelevato l'intero file (a dispetto di quanto avviene ora tramite dynamo) ed effettuato il parsing dello stesso.

Descrizione della struttura del codice

Struttura

In questa sezione verranno discussi dettagli inerenti l'implementazione vera e propria. Verranno descritte le classi principali e il loro scopo, rimanendo comunque ad un livello di astrazione alto, senza scendere troppo in dettaglio.

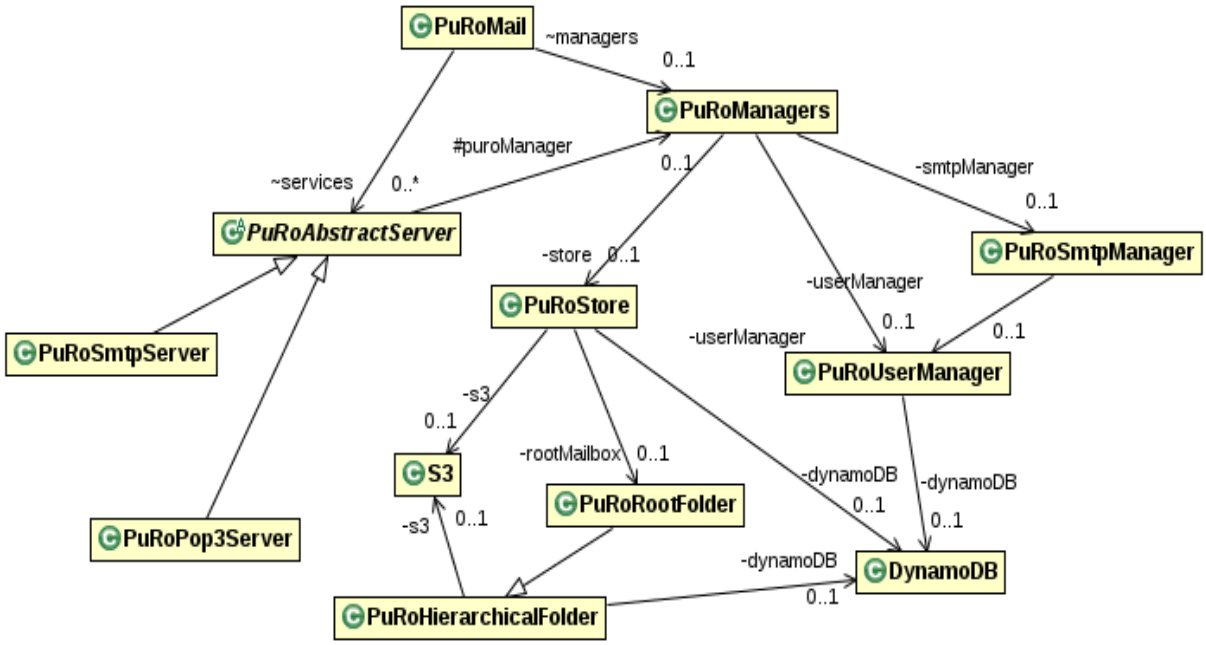

Fig.8 Class Diagram principale

Server

Le classi descritte in questo paragrafo sono dedicate alla gestione della comunicazione client-server, quindi in particolare vanno ad implementare le funzionalità di un server mail.

- PuRoMail
- PuRoAbstractServer
  - PuRoPop3Server
    - Pop3Handler
  - PuRoSmtpServer
    - SmtpHandler
  - PuRoImapServer
    - ImapHandler

PuRoMail

Package *it.prms.greenmail.util*
Descrizione Si occupa di far partire i server POP3/S, SMTP/S e IMAP/S
Metodi principali ● *start*: avvia tutti i server ● *stop*: interrompe tutti i server

PuRoAbstractServer

Package *it.prms.greenmail*
Superclasse *it.prms.greenmail.util.service*
Descrizione Costituisce lo schema di un server generico.
Metodi principali ● *openServerSocket*: crea la socket da cui il server accetterà connessioni

PuRoPop3Server

Package *it.prms.greenmail.pop3*
Superclasse *it.prms.greenmail.PuRoAbstractServer*
Descrizione Rappresenta il server Pop3
Metodi principali ● *run*: resta in ascolto sulla socket creata tramite *openServerSocket di PuRoAbstractServer*. All'arrivo di una richiesta di connessione, viene creato un *Pop3Handler* addetto a servirla. ● *quit*: termina ogni handler e chiude la socket.

Pop3Handler

Package *it.prms.greenmail.pop3*
Superclasse *Thread*

Descrizione Thread addetto alla gestione delle richieste di un singolo client POP3
Metodi principali ● *run*: inizializza le strutture dati e richiama *handleCommand* ● *handleCommand*: legge l'input del client, identifica i comandi e richiama oggetti specifici in base al comando rilevato. I comandi sono presenti nel package *it.prms.greenmail.pop3.commands* ● *quit*: chiude la socket

PuRoSmtpServer

Package *it.prms.greenmail.smtp*
Superclasse *PuRoAbstractServer*
Descrizione Rappresenta il server SMTP
Metodi principali ● *run*: resta in ascolto sulla socket creata tramite *openServerSocket di PuRoAbstractServer*. All'arrivo di una richiesta di connessione, viene creato un *Smtp3Handler* addetto a servirla. ● *quit*: termina ogni handler e chiude la socket.

SmtpHandler

Package *it.prms.greenmail.smtp*
Superclasse *Thread*
Descrizione Thread addetto alla gestione delle richieste di un singolo client SMTP
Metodi principali ● *run*: inizializza le strutture dati e richiama *handleCommand* ● *handleCommand*: legge l'input del client, identifica i comandi e richiama oggetti specifici in base al comando rilevato. I comandi sono presenti nel package *it.prms.greenmail.smtp.commands* ● *quit*: chiude la socket

PuRoImapServer

Package *it.prms.greenmail.imap*
Superclasse *PuRoAbstractServer*

Descrizione Rappresenta il server IMAP. Poichè GreenMail di base implementa una versione parziale del protocollo IMAP, sono presenti varie classi relative al suddetto protocollo.
Metodi principali ● *run*: resta in ascolto sulla socket creata tramite *openServerSocket di PuRoAbstractServer*. All'arrivo di una richiesta di connessione, viene creato un *ImapHandler* addetto a servirla. ● *quit*: termina ogni handler e chiude la socket.

ImapHandler

Package *it.prms.greenmail.imap*
Superclasse *Thread*
Descrizione Thread addetto alla gestione delle richieste di un singolo client IMAP
Metodi principali ● *run*: inizializza le strutture dati e delega la gestione dei comandi all'oggetto *ImapRequestHandler.* ● *resetHandler*: effettua le operazioni di chiusura socket.

Manager

Le classi di questa sezione costituiscono il punto centrale di differenza fra la versione base di GreenMail, e la versione finale pesantemente modificata presentata in questa relazione. La maggior parte del lavoro si è quindi concentrato nella loro implementazione. Tali classi sono ad un livello di astrazione minore delle classi della sezione precedente, in quanto arrivano ad invocare le funzionalità di DynamoDB e S3, sebbene richiamando non in maniera diretta.

- PuRoManagers
  - PuRoSmptManager
  - PuRoImapHostManager
  - PuRoUserManager
  - PuRoStore
    - PuRoHierarchicalFolder
      - PuRoRootFolder

PuRoManagers

Package *it.prms.greenmail*
Descrizione Contiene riferimenti agli altri manager, che gestiscono la logica degli utenti e delle cartelle. Esiste un unico manager in tutto il servizio, in modo da condividere le stesse informazioni fra più handler,

eliminando la replicazione di contenuti inutili.

PuRoSmptManager

Package *it.prms.greenmail.smtp*
Descrizione Manager addetto all'invio delle mail tramite la classe privata *Incoming.*
Metodi principali ● *Incoming.enqueue*: per ogni destinatario specificato con il comando RCPT richiama le funzioni di invio della mail *handle* e *handleExt* ● *Incoming.handle*: invia una mail ad un destinatario **interno** al dominio **puro-mail.tk** ● *Incoming.handleExt*: invia una mail ad un destinatario esterno al dominio. Contattando un server dns si ottiene un record mx che contiene l'indirizzo di un server autoritativo per il dominio della mail del destinatario. Viene quindi instaurata una connessione con tale server. Tale funzione, non presente nella versione originale di GreenMail, non svolge correttamente il suo lavoro, in quanto i server SMTP esterni rifiutano di instaurarare una connessione con il server SMTP corrente. La Motivazione risiede nell'impossibilità di effettuare relaying, ossia di inviare una mail tramite un server SMTP diverso da quello autoritativo per il dominio del mittente.

PuRoImapHostManager

Package *it.prms.greenmail.imap*
Descrizione Interfaccia fra i comandi dei protocolli (non solo IMAP a dispetto del nome) e le funzioni dello store, che si occupa della gestione delle cartelle.
Metodi principali ● *getInbox*: restituisce la cartella INBOX. Richiamata dal comando PASS di POP3 e DATA di SMTP. ● *createMailbox*: crea una mailbox in DynamoDB. Implementata per futura estensione IMAP.

PuRoUserManager

Package *it.prms.greenmail.user*
Descrizione Interfaccia con DynamoDB necessaria per la gestione degli utenti.
Metodi principali ● *getUser*: restituisce utente. E' stata predisposta per IMAP; se l'utente in memoria è aggiornato, verificato chiamato la funzione *userIsUpdated*, viene restituito lo stesso senza doverlo prelevare da DynamoDB ● *userIsUpdated*: verifica che l'utente in memoria sia aggiornato. Estensione IMAP. ● *test*: verifica che l'utente e la password specificata diano luogo ad un login valido. Estensione IMAP per comando LOGIN.

- *addConnectedUser*: incrementa il contatore del numero di connessioni interessate all'utente in questione.
- *removeConnectedUser*: decrementa il contatore di cui sopra. Quando raggiunge zero, è possibile eliminare l'utente dalla memoria.

PuRoStore

Package
it.prms.greenmail.store

Descrizione
Interfaccia con DynamoDB dedicata alla gestione della gerarchia delle mailbox.

Metodi principali
- *addFolderAndParentViewer:* incrementa contatore numero di connessioni interessate alla cartella in questione e ai relativi genitori (fino al namespace, esclusa la root).
- *removeFolderAndParentViewer:* decrementa contatore numero di connessioni interessate alla cartella in questione e ai relativi genitori (fino al namespace, esclusa la root). Se il contatore arriva a zero, la cartella e i genitori vengono eliminati dalla memoria.
- *getInMemoryFolder:* restituisce la cartella (di cui si specifica il nome) presente in memoria. Utilizzato per i namespace, che non vengono salvati su DynamoDB per i motivi descritti nel paragrafo dedicato al servizio. Può essere utilizzata con IMAP per prelevare la versione in memoria di una cartella, se rilevata aggiornata dopo aver richiamato *folderIsUpdated*.
- *getPop3Inbox:* restituisce la inbox di un utente.
- *addInMemoryFolderToRoot:* aggiunge un nuovo namespace in memoria.
- *getMailbox:* restituisce la mailbox specificata, recuperandola da DynamoDB. Estensione IMAP.
- *updateFolder:* aggiorna una cartella presente in memoria, recuperando le informazioni da Dynamo. Estensione IMAP.
- *folderIsUpdated:* verifica che una cartella in memoria sia aggiornata rispetto a DynamoDB. Estensione IMAP.
- *createMailbox:* crea una nuova cartella in DynamoDB. Estensione IMAP.

PuRoHierarchicalFolder

Package
it.prms.greenmail.store

Descrizione
Detiene le informazioni di una cartella salvata in memoria. Informazioni recuperate da un item della tabella Folder in DynamoDB. Ogni cartella contiene i messaggi presenti nella tabella MetaMail e su S3. Eventualmente è possibile che un messaggio sia stato caricato a metà, ossia siano stati caricati solo alcuni campi da DynamoDB (uid e size) e niente body da S3. Il motivo è spiegato nella sezione di EC2.

Metodi principali
- *getMessages*: restituisce i messaggi salvati in memoria. Se necessario viene recuperato totalmente. Richiamata da comandi DELE, RETR e TOP di POP3.
- *getSingleMessageFromDB*: recupera i metadati di un messaggio dalla tabella MetaMail.
- *deleteMessageInDB*: cancella i metadati di un messaggio dalla tabella MetaMail.
- *getUidAndSizeOnly*: recupera UID e Size di un messaggio. Richiamata da comandi LIST e UIDL di POP3.
- *appendMessage*: aggiunge, per un dato messaggio, i metadati nella tabella MetaMail e i dati S3.
- *expunge*: elimina dalla tabella MetaMail e da S3 tutte le voci riferite a messaggi contrassegnati con il flag DELETE. Richiamata da comando QUIT di POP3.

PuRoRootFolder

Package *it.prms.greenmail.store*
Superclasse *it.prms.greenmail.store.PuRoHierarchicalFolder*
Descrizione Cartella root di tutto il server. E' identificata dal nome #mail.
Metodi principali Gli stessi di *PuRoHierarchicalFolder.*

Servizi

I servizi utilizzati in questo tier sono:

- DynamoDB
- S3

DynamoDB

Package *it.prms.amazon.services*
Descrizione Interfaccia diretta con le funzionalità di DynamoDB.
Metodi principali ● *createTableDynamoDB*: crea una nuova tabella. ● *deleteTableDynamoDB*: elimina una tabella. ● *setCurrentEndPoint*: modifica la destinazione (in termini di regione) delle successive chiamate. ● *addItem*: aggiunge l'item specificato nella tabella specificata. ● *retrieveItem*: recupera un item da una tabella ● *query*: esegue una query su una tabella con chiave (hash, range) ● *updateAttributeList*: aggiorna campi di item ● *updateTable*: aggiorna le informazioni di una tabella ● *deleteItem*: elimina item. ● *waitForTableToBecomeAvailable*: la funzione *createTableDynamoDB* dopo la creazione della tabella si ponte in attesa del completamente di tale operazione, tramite questa funzione. ● *retrieveSingleMetaMailItem*: recupera un item della tabella MetaMail ● *retrieveMultipleMetaMailItem*: recupera più item della tabella MetaMail, con stessa hash (stessa folder) ● *retrieveUserItem*: recupera un item dalla tabella User ● *retrieveFolderItem*: recupera un item dalla tabella Folder ● *newMailItem*: crea un nuovo item nella tabella MetaMail ● *newFolderItem*: crea un nuovo item nella tabella Folder ● *newUserItem*: crea un nuovo item nella tabella User

S3

Package *it.prms.amazon.services*
Descrizione Interfaccia diretta con le funzionalità di S3.
Metodi principali ● *createNewBucket*: crea nuovo bucket ● *deleteBucket*: elimina bucket ● *setCurrentEndPoint*: modifica la destinazione (in termini di regione) delle successive chiamate. ● *uploadingFileToBucket*: esegue l'upload di un oggetto in un bucket ● *downloadObjectFromBucket*: esegue il download di un oggetto da un bucket ● *deleteObjectFromBuckets*: elimina un oggetto da un bucket

Configurazione

PuroConfig

Package *it.prms.greenmail*
Descrizione Permette la configurazione di alcuni parametri del servizio. Nella tabella User è salvata una riga contenente le informazioni di configurazione.
Metodi principali ● *loadConfiguration*: carica dalla tabella User le impostazioni di configurazione del servizio. ● *setThroughputParameters*: setta i valori di throughput in lettura e scrittura delle tabelle User, Folder e MetaMail.

Eccezioni: propagazione e gestione

La gestione della maggior parte delle eccezioni viene gestita al livello della classi PuRoStore e PuRoHiearchicalFolder, che possono essere viste come la classi principale di collegamento tra la parte derivata da GreenMail e quella nuova legata ai servizi di Amazon. Sebbene le classi di DynamoDB e S3 restuiscano molte eccezioni, presenti nello schema a seguire, ai livelli superiori ne viene inviata solo una generica contenente il messaggio generato dalle eccezioni sottostanti; in particolare si fà riferimento alla *FolderException* e alla *UserException*.

La tabella seguente illustra il legame presente tra le eccezioni e i messaggi che esse producono:

Eccezione	**Messaggio restituito al client**
ProvisionedThroughputExceededException	Throughput exceeded
InternalServerErrorException	Service has a problem when trying to process the request

ResourceNotFoundException	Referencing a resource that does not exist
AmazonServiceException	Service was not able to process the request PuRo unable to retrieve message
AmazonClientException	PuRo unable to get a response from a service
WrongTypeException	Wrong error type
NoSuchUserException	Username doesn't exist
WrongActionException	Wrong action type error
NumberFormatException	Wrong number format error
MessagingException	Corrupted message

5. Testing

Nel seguente capitolo verranno esposti i risultati della sessione di testing più significativa effettuata sul sistema realizzato. Per lo scopo sono state utilizzate istanze EC2 di tipo T1.micro. Il Load Balancer è stato impostato per avere di base, ossia a carico zero, tre istanze distribuite su tre differenti availability zone. La regione di appartenenza del Load Balancer e delle istanze è eu-west-1, Irlanda.

Postal 0.72

Essendo un software di benchmark, Postal offre la possibilità di configurare la tipologia di test che si desidera effettuare. Di seguito vengono riportati i valori e il significato dei parametri scelti per un test adatto a portare alla luce le funzionalità dell'autoscaling e degli allarmi ad esso associati.

- **-t threads**: il numero di thread che devono essere creati per provare connessioni separate
  - Fase1: 2x30 thread = 60 thread
  - Fase2: 2x60 thread = 120 thread
- **-c messages-per-connection "3":** numero di messaggio che è possibile mandare in una connessione SMTP; il valore è scelto in maniera casuale tra 1 e message-per-connection
- **smtp-server**: indirizzo server di posta da testare
- **user-list-filename**: lista degli utenti presenti nel sistema

Come si può notare in figura, durante il test, il numero di errori riscontrati da Postal è sempre molto al di sotto del numero totali di connessioni instaurate con il server PuRo. Purtroppo i messaggi di errore che vengono generati dal server sembrano essere completamente filtrati dal software di testing, non è stato quindi possibile risalire all'origine di tali errori.
L'analisi a posteriori dei grafici di Amazon CloudWatch hanno mostrato comunque un comportamento in linea con le politiche di autoscaling impostate.
Nella figura 9 si può notare quanto accennato ad inizio paragrafo e cioè la presenza di due fasi distinte ma consegutive nel test effettuato. La prima in cui il server PuRo è stato sottoposto alle richieste di due gruppi di 30 thread ciascuno, e la seconda, nella seconda metà, in cui sono stati aggiunti altri 2 gruppi identici ai precendenti, che hanno portato a 120 il numero di processi leggeri che tentavano l'invio di messaggi al server.

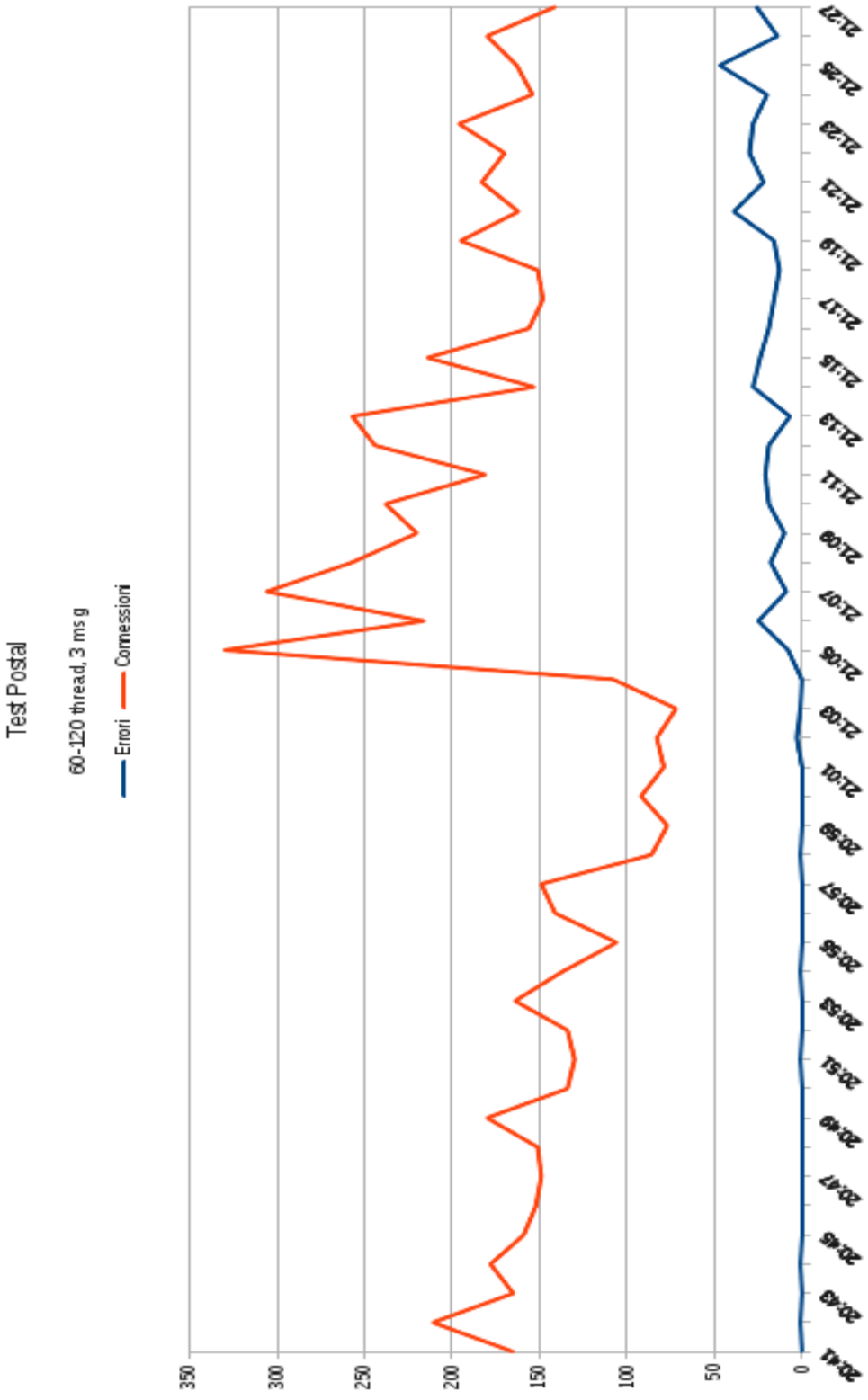

Fig. 9 Grafico risultati test postal

I risultati del test verrano ora analizzati dal punto di vista delle metriche di Amazon CloudWatch.
Tra le metriche disponibili, quella che si è rivelata di più grande utilità ai fini dell'autoscaling è stata la percentuale di utilizzo della CPU. In particolare, in relazione alla politica di autoscaling, cioè che è stato preso in considerazione è stata il valore della media di tale metrica. La politica di autoscaling è stata impostata al fine di compiere le seguenti azioni allo scattare dell'allarme:

- scale up di una istanza EC2, nel caso in cui il valore medio della metrica CPUUtilization superasse l'80% nei 3 periodi di osservazione di 120 secondi ciascuno
- scale down di una istanza EC2, nel caso in cui il valore medio della metrica CPUUtilization fosse inferiore del 20% nei 3 periodi di osservazione di 120 secondi ciascuno

In figura 10 è possibile notare il comportamento della cpu durante le due fasi del test precedentemente specificate; in particolare il primo incremento di utilizzo di cpu avviene nella Fase1 del test (19:40-20:00) e il secondo incremento nella Fase2 (20:00-20:30). In quest'ultima si può osservare l'attivazione della policy di scale up, al seguito della quale l'utilizzo della cpu subisce un decremento medio per l'aggiunta di una nuova istanza (20:12 circa).

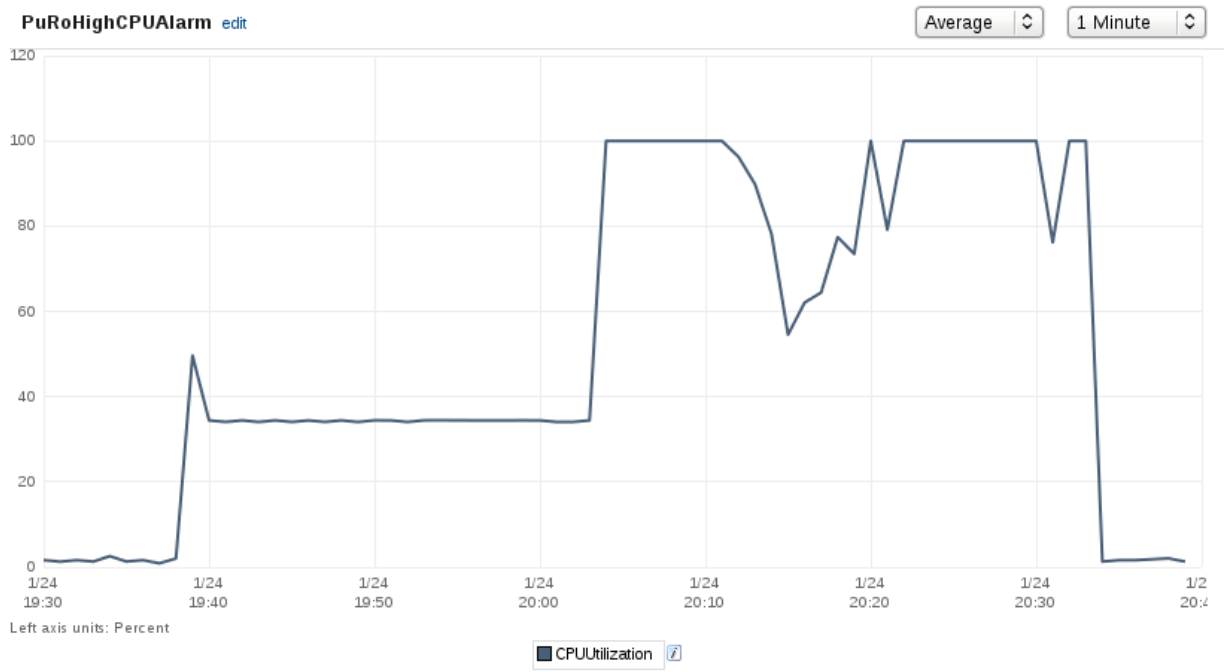

Fig. 10 Allarme per il sovraccarico della CPU oltre l'80%

Le metriche NetworkIn e NetworkOut illustrate nelle figure 11 e 12, anche se inizialmente prese in considerazione come valori indicativi del carico a cui sono sottoposte le istanze durante la loro esecuzione, sono state poi escluse perchè il limite della larghezza di banda a cui hanno accesso le singole istanze non è un valore noto e quindi non è possibile calcolare un livello massimo/minimo di carico di rete su cui poi ragionare per impostare un soglia per gli allarmi. Sono stati comunque riportati i risultati prodotti durante il test al fine di illustrare il quantitativo di dati scambiati con la rete dalle singole istanze EC2.

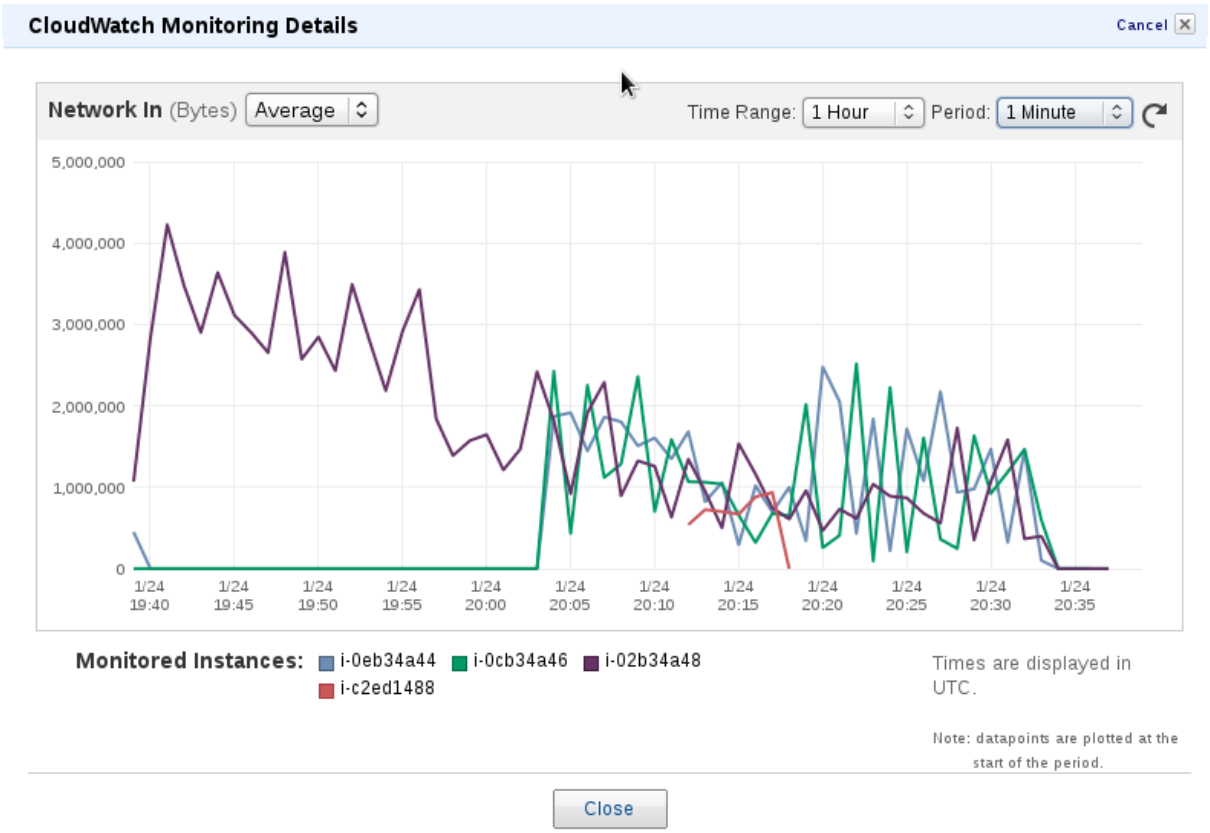

Fig.11 Metrica NetworkIn delle istanze nel gruppo si autoscaling

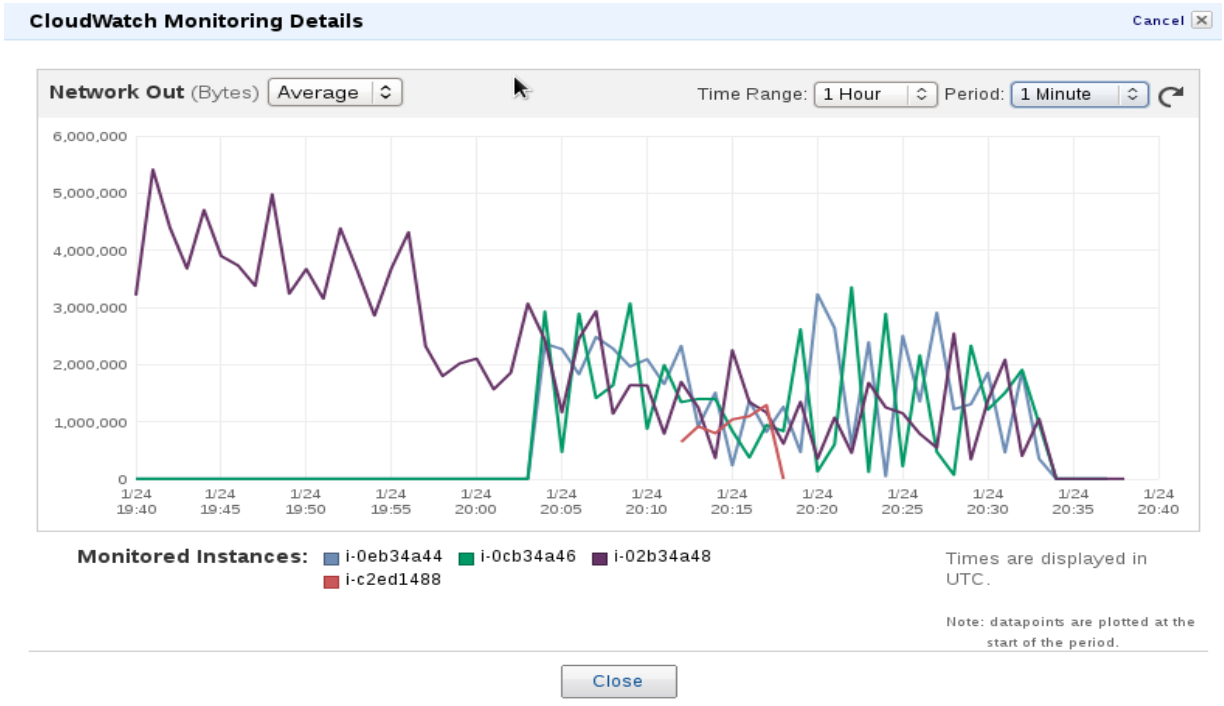

Fig. 12 Metrica NetworkOut delle istanze nel gruppo si autoscaling

La figura 13 fornisce la motivazione sotto forma di grafico che ha portato gli sviluppatori alla scelta di escludere il valore della latenza registrato dal load balancer come metrica su cui impostare un'allarme per una politica di autoscaling. Come si può notare i valori sono diversi ordini di grandezza inferiori al secondo, mentre il valore minimo di soglia dell'allarme che può essere impostato è pari ad 1 secondo. E' ovvio quindi che in questa applicazione un tale allarme non avrebbe trovato la sua efficacia, motivo che ha portato alla sua esclusione. Nella figura 14 è possibile invece osservare in maniera più dettagliata, i valori delle latenze delle singole availability zone, ponendo soprattutto l'attenzione sul lavoro di distribuzione del carico da parte del load balancer.

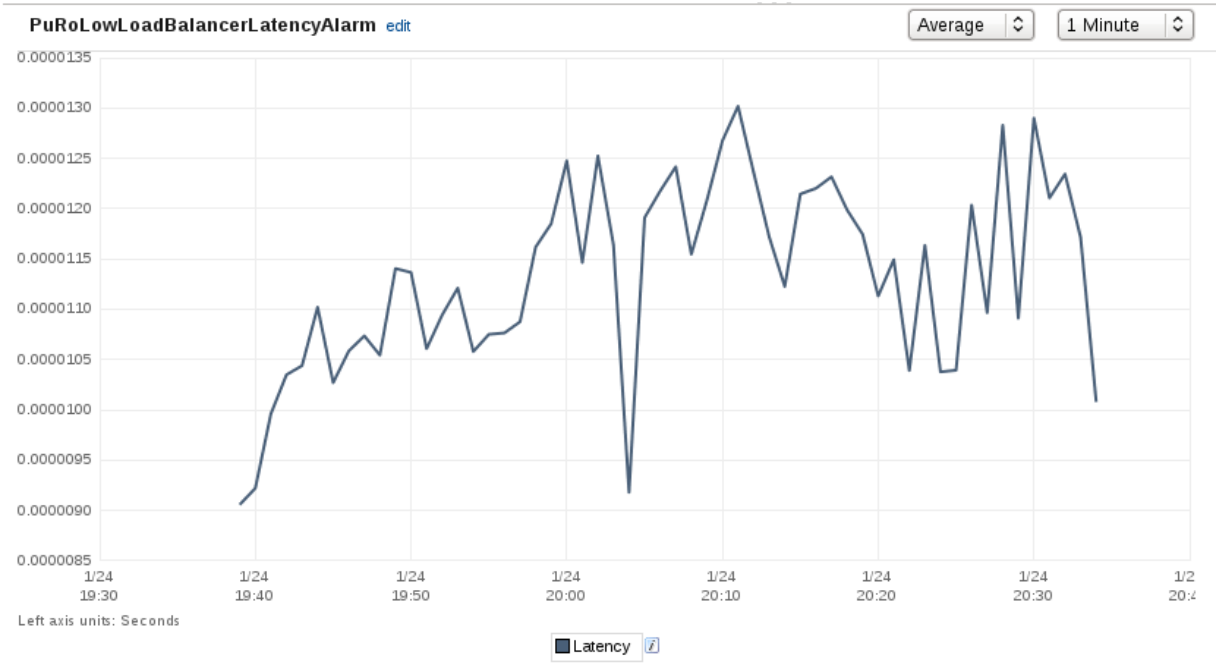

Fig.13 Allarme sulla valore minimo di latenza del load balancer (2 sec)

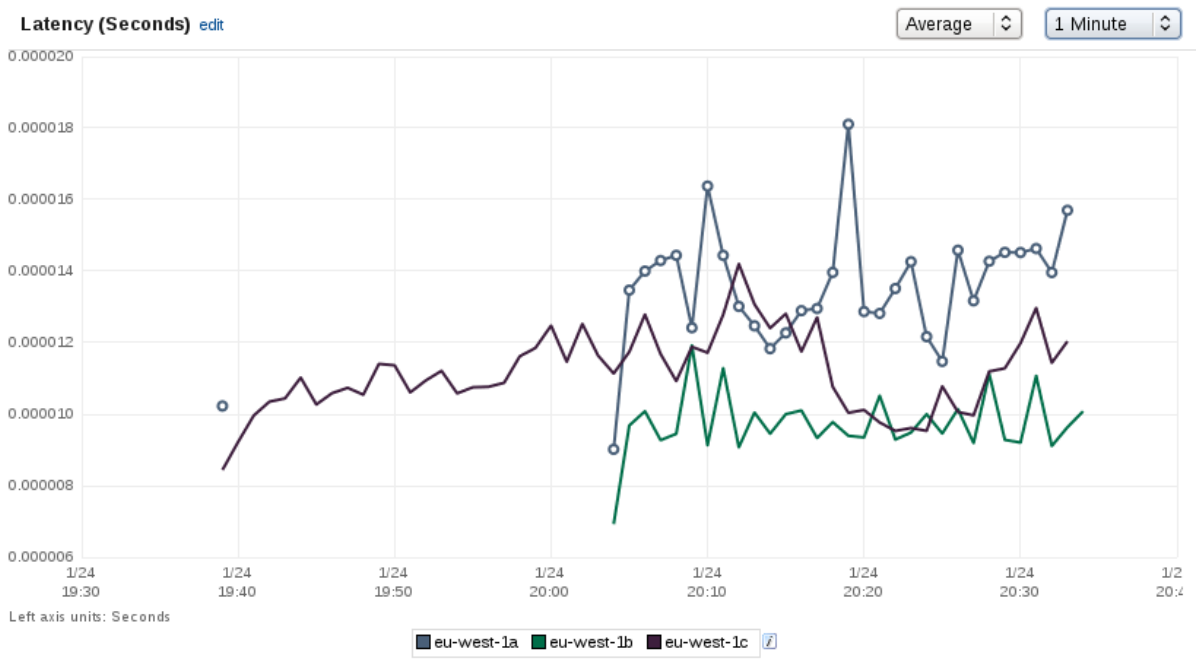

Fig.14 Latenza singole availability zone

Rabid

Rabid è risultato essere un software poco utile nella fase di testing, in quanto, a differenza di Postal, non riporta nessun dato statistico di campionamento e questo, a nostro avviso, lascia allo sviluppatore poca possibilità di analisi dei dati post-testing. Un nota a favore di Rabid, è stata l'assenza di un meccanismo di filtraggio dei messagi di errore generati dal server, che ha dato quindi la possibilità allo sviluppatore di accorgersi dello stato in cui si trovava il server durante il test.

6. Costi

Considerando uno scenario in cui il sistema realizzato venga effettivamente utilizzato al livello commerciale, sono stati analizzati i costi di tutti i servizi Amazon utilizzati, per poter ottenere una stima sul costo totale finale.
Le considerazioni svolte sono le seguenti:

- L'account AWS è fuori dal Free Tier, ossia il servizio gratuito offerto il primo anno di iscrizione
- Non viene utilizzato il servizio Simple Notification Service, utile per segnalare via mail all'utente proprietario dell'account AWS cambiamenti importanti nello stato dei singoli servizi
- Per i servizi basati su regione, la regione utilizzata è l'Irlanda

Costo singoli servizi

Route 53

Hosted Zones

$0.50 per hosted zone / month for the first 25 hosted zones
$0.10 per hosted zone / month for additional hosted zones

Niente pagamenti parziali, allo scattare del nuovo mese viene addebitato il mese intero.

Standard Queries

$0.500 per million queries – first 1 Billion queries / month
$0.250 per million queries – over 1 Billion queries / month

Latency Based Routing Queries

$0.750 per million queries – first 1 Billion queries / month
$0.375 per million queries – over 1 Billion queries / month

Ammessi pagamenti parziali - 100000 query standard comportano un costo di (0.500/10^6)*100000$ = 0,05$

Elastic Load Balancing

$0.025 per hour
$0.008 per GB

Elastic Compute Cloud

Facendo le seguenti considerazioni:

- Istanze On Demand
- Micro Istanze T1
- Istanze Linux/Unix

$0.020 per Hour

DynamoDB

Provisioned Throughput Capacity

Write Throughput	$0.0113 per hour for every 10 units of Write Capacity

Read Throughput	$0.0113 per hour for every 50 units of Read Capacity

Indexed Data Storage

First 100 MB stored per month is free.
$1.13 per GB-month thereafter

Data Transfer IN

All data transfer in	$0.000 per GB

Data Transfer OUT

First 1 GB / month	$0.000 per GB
Up to 10 TB / month	$0.120 per GB
Next 40 TB / month	$0.090 per GB
Next 100 TB / month	$0.070 per GB
Next 350 TB / month	$0.050 per GB

Amazon Simple Storage Service

Pricing

	Standard Storage	Reduced Redundancy Storage	Glacier Storage
First 1 TB / month	$0.095 per GB	$0.076 per GB	$0.011 per GB
Next 49 TB / month	$0.080 per GB	$0.064 per GB	$0.011 per GB

Request Pricing

Glacier Archive and Restore Requests	$0.055 per 1,000 requests
Delete Requests	Free
GET and all other Requests	$0.01 per 10,000 requests
Glacier Data Restores	Free

Data Transfer IN

All data transfer in	$0.000 per GB

Data Transfer OUT

First 1 GB / month	$0.000 per GB

Up to 10 TB / month	$0.120 per GB
Next 40 TB / month	$0.090 per GB
Next 100 TB / month	$0.070 per GB
Next 350 TB / month	$0.050 per GB

CloudWatch

Amazon CloudWatch Detailed Monitoring for Amazon EC2 instances (at one-minute frequency)	$3.50 per instance per month (the per metric price below x 7 pre-defined metrics per instance)
Amazon CloudWatch Custom Metrics	$0.50 per metric per month
Amazon CloudWatch Alarms	$0.10 per alarm per month
Amazon CloudWatch API Requests	$0.01 per 1,000 Get, List, or Put requests

Stima costo finale

Per poter stimare il costo finale è necessario fare delle supposizioni relativamente all'utilizzo di ogni servizio. In particolare si suppone:

- Un carico giornaliero di 100000 utenti al giorno
- Ogni utente invia 10 mail e riceve 10 mail al giorno
- Ogni mail pesa 0.5 MB

Route 53:

- 1 Hosted Zone: 0.50$/month
- 20 Standard Query al giorno per utente (ogni utente si connette al servizio 20 volte al giorno, e si trascurano eventuali caching intermedi DNS): (0.500/10^6 * 20 * 30 * 100000)$/month = 30 $/month
- 0 Latency Based Routing Queries: 0 $/month

Costo totale Route 53: 30.5 $/month
Elastic Load Balancing:

- 1 Load Balancer attivo: (0.025 * 24 * 30)$/month = 18 $/month
- Ogni utente invia 10 mail e riceve 10 mail al giorno. Ogni mail pesa 0.5 MB: (0.008/1000 * 0.5 * 20 * 30 * 100000) $/month = 240 $/month

Costo totale Load Elastic Balancing: 258 $/month

Elastic Compute Cloud:

- 10 istanze T1.micro attive in media: (0.020 * 24 * 30 * 10)$/month = 144 $/month

Costo totale Elastic Compute Cloud: 144 $/month

DynamoDB:

- 50 unità di Scrittura: (0.0113 * 5 * 24 * 30)$/month = 40,68 $/month
- 50 unità di Lettura: (0.0113 * 1 * 24 * 30)$/month = 8,13 $/month
- E' stato sopra supposto un carico di 20 mail al giorno da 0.5 MB ciascuna. Si suppone che le mail non vengano cancellate dopo esser state scaricate. Poichè in Dynamo vengono salvate unicamente metadati, si suppone che ogni item occupi 0.5 KB: (20 * 0.5 * 100000 * 30 * 1.13/1000000)$/month = 33,9 $/month (avendo trascurato i primi 100 MB free)
- Nella tabella User e nella tabella Folder sono salvate 100000 voci ciascuna, che avendo poche informazioni sono quantificabili nell'ordine di 0.2 KB: (2 * 0.2 * 100000 * 1.13/1000000) $/month = 0,04 $/month
- Poichè i trasferimenti fra EC2 e Dynamo avvengono all'interno della stessa regione, non ci sono costi di trasferimento: 0$/month

Costo totale DynamoDB: 82,75 $ / month

Simple Storage Service:

- Avendo supposto una dimensione per mail pari a 0.5 MB, per un totale di 20 mail al giorno per utente, e poichè S3 memorizza tutta la mail: (0.5 * 20 * 100000 * 30) MB = 30 TB. Quindi (0.095 * 30) $ / month= 2,85 $/month
- 10 PUT al giorno per utente: (10 * 100000 * 30 * 0.01/1000) $/month= 300 $/month
- 10 GET al giorno per utente: (10 * 100000 * 30 * 0.01/10000) $/month = 30 $/month
- 0 DELETE al giorno per utente: 0 $/month

Costo totale S3: 332.85 $ / month

CloudWatch:

- Detailed Monitoring: (3.50 * 10) $ /month = 35 $ /month
- Supponiamo per ogni metrica di base (7 metriche) = 14 x 0,10 $/month = 1,4 $ /month
- Trascuriamo i costi legati a Get, List e Put

Costo totale CloudWatch: 36,4 $ / month

Stima finale

Il costo finale del progetto in termini di servizi Amazon è il seguente: <u>884,5 $/month</u>

7. Sviluppi futuri

IMAP

Come più volte sottolineato in diverse sezioni, il servizio è stato progettato tenendo a mente un'eventuale successiva implementazione di IMAP. Il protocollo con i relativi comandi è stato studiato a fondo. Le principali problematiche nell'affrontare l'implementazione del protocollo sono sorte una volta analizzate alcune sue feature peculiari, non presenti in POP3, in particolare:

- Possibilità a più utenti di connettersi in parallelo con lo stesso account
- Condivisione della stessa cartella fra account diversi

Entrambe le funzionalità in realtà comportano la stessa problematica, ossia il dover in qualche modo notificare l'utente delle modifiche effettuate sulle risorse condivise. DynamoDB di base non offre meccanismi di sincronizzazione di accesso, che sono quindi lasciati al servizio chiamante. La soluzione escogitata consiste nell'aggiungere alle tre tabelle di DynamoDB un campo contentente la data di ultima modifica; in questo modo ad ogni accesso dell'utente agli item della tabella, la prima operazione eseguita rappresenta il confronto della data in memoria, con quella su database; una eventuale discrepanza comporta un aggiornamento delle informazioni in memoria. Il campo in questione risulta essere **lastupdate**, figurante appunto nelle descrizioni di tutte e tre le tabelle.
Un problema importante di questa soluzione consiste nel tradeoff fra consistenza e prestazioni. Le cartelle sono rappresentate in memoria come un albero di padri e figli; un aggiornamento di una cartella in memoria, per mantenere la consistenza delle informazioni, deve richiedere l'aggiornamento di tutto il ramo dell'albero a cui appartiene la cartella; non solo le informazioni sulla cartella, ma anche le mail stesse sarebbero da ricaricare, con conseguente appesantimento del sistema e **degrado delle prestazioni**.
Per evidenti limiti strutturali della soluzione si è deciso di porre da parte la stessa.

Utenti multiregione

Un'altra possibile estensione, consiste nell'effettuare il deployment dei server mail in più di una regione, in modo da poter fornire anche **Scalabilità Geografica**. E' possibile realizzare più versioni di tale funzionalità, ognuna caratterizzata da una propria complessità e da specifiche problematiche.

Prima versione

Ogni regione deve esser dotata del proprio load balancer a cui andranno assegnate diverse istanze in diverse Availability Zone (per **Tolleranza ai Guasti)** come già ampliamente descritto.
Per poter utilizzare tale funzionalità, giunge in supporto Route 53, il quale mette a disposizione la possibilità di effettuare risoluzioni di indirizzo in base alla latenza, tramite **Latency Based Routing Query**.
Si prenda in considerazione un utente residente in Cina, che provi a contattare il dominio **puro-mail.tk**. Route 53 tramite Latency Based Routing Query invia la richiesta ai server posti nella regione Cina, piuttosto che nelle altre regioni utilizzate dal servizio.
Una tale implementazione soffre della seguente problematica:

- Poichè Route 53 effettua la traduzione dell'indirizzo scegliendo quello posto più vicino (in termini di latenza) all'utente, se per questioni di ingombro di rete, l'utente in Cina venisse reindirizzato verso la regione Irlanda, priva di riferimenti all'utente, la relativa richieste diverrebbe insoddisfabile. La situazione esposta potrebbe verificarsi con molta frequenza per utenti posti al confine fra due regioni.

Seconda Versione

Una possibile soluzione al problema della prima versione, consiste nel porre le informazioni dell'utente in ogni regione su cui è stato effettuato il deployment dei server mail. Un utente in fase di registrazione al

servizio, sceglie una regione di appartenenza. In tal modo qualora le richieste dell'utente della Cina venissero inviate ad una regione diversa, per esempio l'Irlanda, i server interrogando la tabella User di DynamoDB dell'Irlanda, otterrebbero l'esatta locazione delle informazioni dell'utente, andando quindi poi ad interrogare i servizi della regione Cina.
Le Problematiche della seconda versione sono le seguenti:

- Le informazioni dell'utente, a seguito di modifiche, vanno sincronizzate fra le varie regioni
- Per gli utenti viaggiatori, una singola regione di appartenenza risulterebbe una restrizione troppo limitante.

Terza Versione

La prima problematica della seconda versione non è insormontabile, ma richiede la conoscenza da parte dei server di tutte le regioni utilizzate dal servizio, per poter effettuare l'aggiornamento dei dati appena modificati della tabella utente.
Relativamente alla seconda problematica si può pensare di utilizzare una qualche metrica che tenga conto degli accessi dell'utente; se l'utente della Cina effettuasse molto più accessi in Irlanda - ad esempio in seguito ad un trasferimento in tale regione - per diminuire la latenza si potrebbe copiare le informazioni da DynamoDB e S3 presenti nella regione originale, e porle in quella di nuova residenza.
Le nuove problematiche introdotte sono le seguenti:

- La metrica dovrebbe essere modellata bene per evitare continue creazione di copie delle informazioni dell'utente
- Dopo aver verificato la cessazione totale degli accessi ad una regione, per un importante periodo di tempo, la replica nella suddetta regione dovrebbe venir cancellata
- Problema cruciale: necessario mantenere sincronizzate le repliche di DynamoDB e S3 fra le varie regioni per motivi di consistenza.

Le problematiche elencate hanno comportato la decisione di porre in secondo piano questa funzionalità, per quanto piuttosto avanzata e sicuramente utile.

8. Conclusioni

Il progetto realizzato ha permesso di approfondire lo studio sui sistemi distribuiti e sulle relative problematiche. Per poter procedere nella realizzazione, è stato necessario analizzare in maniera approfondita i protocolli mail e sono state apprezzate le potenzialità degli Amazon Web Services.
Il risultato finale, sicuramente non privo di lacune, risulta carente specialmente nella parte del supporto alla comunicazione mail; non è supportato IMAP, ed in SMTP risultano mancanti molti meccanismi di autenticazione; la mancanza maggiore è sicuramente da ricercarsi nell'impossibilità di inviare mail ad utenti esterni al dominio.
Analizzando invece il risultato finale in ottica delle peculiarità di un sistema distribuito, a parer nostro, si sono ottenuti ottimi risultati; il sistema risulta scalabile al fronte di un carico variabile, altamente distribuito sembra grazie alla sua scalabilità, tollerante ai guasti grazie alle accortezze utilizzati durante le chiamate ai servizi DynamoDB ed S3.
E' sicuramente possibile espandere il servizio realizzato, muovendosi nella direzione indicata nel paragrafo degli sviluppi futuri, creando un sistema ancora più scalabile - grazie agli utenti multiregione - e più completo - tramite l'implementazione del protocollo IMAP.

Appendice A - Guide varie

Connessione ad istanze mediante ssh

Per collegarsi ad una istanza mediante il protocollo ssh è necessario digitare il seguente comando da terminale:

```
ssh -i  /path-to-key-pair/your-key-pair.pem  nome_utente@nome-istanza-DNS
```

Per copiare un file dal proprio computer verso una istanze lanciare il seguente comando:

```
scp -i  /path-to-key-pair/your-key-pair.pem  /percorso_file/file.ext
ubuntu@nome-istanza-DNS:/home/ubuntu
```

Configurazione istanze

In un gruppo di AutoScaling le istanze vengono create a partire da una AMI (Amazon Machine Image). Affinchè una istanza appena creata e aggiunta in un gruppo di AutoScaling possa esser pronta a servire le richieste dei client, devono essere eseguiti prima diversi passi.
E' necessario creare un file jar eseguibile del progetto, ad esempio dal nome **PuRo.jar**
Bisogna eseguire l'upload di tale file su una istanza tramite il comando:

```
scp -i  /path-to-key-pair/your-key-pair.pem  /percorso_file_jar/PuRo.jar
ubuntu@nome-istanza-DNS:/home/ubuntu
```

L'applicazione deve essere configurata per partire in automatico all'avvio del sistema:

- Aprire il file di configurazione
- */etc/rc.local*
- Aggiungere prima della riga finale "*exit0*" la seguente riga:
  /usr/bin/java -jar /home/ubuntu/PuRo.jar

Affinchè l'istanza venga classificata come "healthy" dal load balancer, è necessario installare sulla stessa Apache o Tomcat. Il load balancer eseguirà un ping su protocollo HTTP sulla porta e il path specificato.
Si è deciso di scegliere il servlet container Tomcat, avendo creato una semplice interfaccia web capace di aggiungere nuovi utenti al servizio. Tomcat è stato messo in ascolto sulla porta 80 (la porta di default è la 8080).
Terminati i passi precedenti è necessario creare una AMI a partire dalla istanza modificata, il cui id sarà utilizzato nella successiva guida all'AutoScaling. Tutte le successive istanze create in automatico, avranno la stessa AMI.

Guida Autoscaling

Qui di seguito viene mostrata una sequenza possibile di comandi per configurare il servizio di autoscaling. Per poter seguire questa guida è necessario installare due programmi:

- CloudWatch Command Line Tool
- AutoScaling Command Line Tool

Il link per scaricarli è indicato nella bibliografia.
Tutti i comandi hanno come destinazione (endpoint) la regione **eu-west-1**, Irlanda.

Crea una configurazione di lancio, specificando l'AMI, il tipo, il security group e la chiave delle istanze che verranno create in automatico:

```
as-create-launch-config PuRo-config --image-id "ami-id" --instance-type
t1.micro --group PuRo-Sec -key PuRo-key --region eu-west-1
```

Creazione di un gruppo di istanze, impostando il nome del load balancer. Il load balancer, creato da interfaccia web, deve supportare tutte le availability zone qui specificate. A seguire viene creato un gruppo di AutoScaling, caratterizzato da un numero minimo di istanze pari a 3 e un numero massimo pari a 6; le istanze vengono equidistribuite fra le tre availability zone:

```
as-create-auto-scaling-group PuRo-group --launch-configuration PuRo-config
--availability-zones eu-west-1a eu-west-1b eu-west-1c --min-size 3 --max-size
6 --load-balancers PuRoLoadBalancer --region eu-west-1
```

Crea una policy da eseguire allo scattare di un certo allarme (definito successivamente). Almeno 300 secondi fra l'esecuzione di una policy e l'altra (dello stesso tipo). L'azione eseguita consiste nell'incrementare di un'unità il numero di istanze. Il comando rilascia in output un arn, Amazon Resource Name, da utilizzare successivamente nella definizione degli allarmi.

```
as-put-scaling-policy PuRoScaleUpPolicy --auto-scaling-group PuRo-group
--adjustment=1 --type ChangeInCapacity  --cooldown 300 --region eu-west-1
```

Crea un allarme alla cui attivazione viene lanciata la policy associata.

- period è la grandezza in secondi del tempo durante il quale viene analizzata la metrica
- evaluation-periods è il numero di periodi dopo i quali vengono effettuate le stime finali
- threshold: soglia di allarme

A seguire vengono definiti, come esempio, quattro allarmi legati ad utilizzo eccessivo della Cpu, della banda in download e upload e del load balancer

```
mon-put-metric-alarm PuRoHighCPUAlarm  --comparison-operator
GreaterThanThreshold  --evaluation-periods  3 --metric-name  CPUUtilization
--namespace  "AWS/EC2"  --period  120  --statistic Average --threshold 80
--alarm-actions arn:"arn creato precedentemente con PuRoScaleUpPolicy"
--dimensions "AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoHighNetworkInAlarm  --comparison-operator
```

```
GreaterThanThreshold  --evaluation-periods  3 --metric-name  NetworkIn
--namespace  "AWS/EC2"  --period  120  --statistic Average --threshold 1500000
--alarm-actions arn:”arn creato precedentemente con PuRoScaleUpPolicy”
--dimensions "AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoHighNetworkOutAlarm --comparison-operator
GreaterThanThreshold  --evaluation-periods  3 --metric-name  NetworkOut
--namespace  "AWS/EC2"  --period  120  --statistic Average --threshold 2000000
--alarm-actions arn:”arn creato precedentemente con PuRoScaleUpPolicy”
--dimensions "AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoHighLoadBalancerLatencyAlarm --comparison-operator
GreaterThanThreshold  --evaluation-periods  3 --metric-name  Latency
--namespace  "AWS/ELB"  --period  120  --statistic Average --threshold 5
--alarm-actions arn:”arn creato precedentemente con PuRoScaleUpPolicy”
--dimensions "LoadBalancerName=PuRoLoadBalancer" --region eu-west-1
```

Dopo aver creato la policy e gli allarmi per un utilizzo eccessivo, è necessario creare le controparti che rilevino un alleggerimento del carico.
Policy LowLoad:

```
as-put-scaling-policy PuRoScaleDownPolicy --auto-scaling-group PuRo-group
--adjustment=-1 --type ChangeInCapacity  --cooldown 300 --region eu-west-1
```

Allarmi LowLoad:

```
mon-put-metric-alarm PuRoLowCPUAlarm  --comparison-operator  LessThanThreshold
--evaluation-periods  3 --metric-name  CPUUtilization --namespace  "AWS/EC2"
--period 120 --statistic Average --threshold  20  --alarm-actions
arn:”arn creato precedentemente con PuRoScaleDownPolicy” --dimensions
"AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoLowNetworkInAlarm  --comparison-operator
LessThanThreshold --evaluation-periods  3 --metric-name  NetworkIn --namespace
"AWS/EC2"  --period 120 --statistic Average --threshold  500000
--alarm-actions arn:”arn creato precedentemente con PuRoScaleDownPolicy”
--dimensions "AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoLowNetworkOutAlarm  --comparison-operator
LessThanThreshold --evaluation-periods  3 --metric-name  NetworkOut
--namespace  "AWS/EC2"  --period 120 --statistic Average --threshold  200000
--alarm-actions arn:”arn creato precedentemente con PuRoScaleDownPolicy”
--dimensions "AutoScalingGroupName=PuRo-group" --region eu-west-1

mon-put-metric-alarm PuRoLowLoadBalancerLatencyAlarm --comparison-operator
LessThanThreshold  --evaluation-periods  3 --metric-name  Latency  --namespace
"AWS/ELB"  --period  120  --statistic Average --threshold 2 --alarm-actions
arn:”arn creato precedentemente con PuRoScaleDownPolicy” --dimensions
"LoadBalancerName=PuRoLoadBalancer" --region eu-west-1
```

Una volta concluso il testing, è possibile eliminare quanto precedentemente definito.

Eliminazione policy:

```
as-delete-policy PuRoScaleDownPolicy --auto-scaling-group PuRo-group --region eu-west-1
as-delete-policy PuRoScaleUpPolicy --auto-scaling-group PuRo-group --region eu-west-1
```

Eliminazione allarmi:

```
mon-delete-alarms PuRoLowCPUAlarm --region eu-west-1
mon-delete-alarms PuRoLowNetworkInAlarm --region eu-west-1
mon-delete-alarms PuRoLowNetworkOutAlarm --region eu-west-1
mon-delete-alarms PuRoLowLoadBalancerLatencyAlarm --region eu-west-1

mon-delete-alarms PuRoHighCPUAlarm --region eu-west-1
mon-delete-alarms PuRoHighNetworkInAlarm --region eu-west-1
mon-delete-alarms PuRoHighNetworkOutAlarm --region eu-west-1
mon-delete-alarms PuRoHighLoadBalancerLatencyAlarm --region eu-west-1
```

Prima di eliminare il gruppo di AutoScaling è necessario settare a zero la relativa dimensione, ossia eliminare tutte le istanze presenti in esso:

```
as-update-auto-scaling-group PuRo-group --min-size 0 --max-size 0 --region eu-west-1
```

Elimina l'auto scaling group. E' necessario controllare la console ed aspettare che il comando precedente abbia effetto, ossia che tutte le istanze vengano terminate..

```
as-delete-auto-scaling-group PuRo-group --region eu-west-1
```

Eliminazione configurazione di lancio:

```
as-delete-launch-config PuRo-config --region eu-west-1
```

Appendice B - HOWTO

In questa sezione verranno descritti i passi per effettuare il test del progetto.
Su ogni istanza è stato configurato il servlet container **Tomcat**, su cui è stato eseguito il deploy di una piccola e rudimentale applicazione web per la creazione di utenti.
Collegarsi all'indirizzo **puro-mail.tk/puro**
Inserirere **username** (l'indirizzo email creato sarà **username@puro-mail.tk**), **password**, **firstname**, **lastname** e cliccare su **Crea Utente**.
Configurare un client di posta elettronica come segue**:**

Server	**Porta**
SMTP	12365 (o 25)
SMTPS	12366
POP3	12450

POP3S	12451

L'indirizzo del server di SMTP e POP3 deve essere impostato come **puro-mail.tk**

A questo punto è possibile utilizzare il servizio mail.
Come già descritto nella relazione, si ricorda nuovamente che è possibile utilizzare il servizio per inviare posta fra utenti interni al dominio **puro-mail.tk**. E' possibile inviare mail da utenti esterni verso utenti interni, grazie al record mx del servizio Route 53. Non è possibile inviare posta verso utenti esterni, per motivi di autenticazione smtp (cfr. funzione *handleExt* in *PuRoSmptManager).*

Per la configurazione del sistema, è stata inserita nella tabella User una voce (la cui hashkey ha valore **puro-config**) scorrelata dal resto del contesto delle informazioni sugli utenti. Le opzioni di configurazione sono:

- Il nome del dominio (di default: **puro-mail.tk**)
- I valori di throughput lettura/scrittura della tabella User
- I valori di throughput lettura/scrittura della tabella Folder
- I valori di throughput lettura/scrittura della tabella MetaMail

Non è stata predisposta alcuna interfaccia grafica, quindi per la configurazione del sistema è necessario agire direttamente dalla console AWS. In caso di necessità di riconfigurare il sistema, per poter propagarre le modifiche è possibile agire in due modi:

- Eseguire il reboot delle istanze attualmente attive
- Tramite il tool di autoscaling (cfr. Bibliografia), impostare la dimensione dell'AutoScaling Group a zero, e quindi reimpostarla ai valori precedenti. Per poterlo fare è necessario lanciare consecutivamente i seguenti due comandi:
    - as-update-auto-scaling-group PuRo-group --min-size 0 --max-size 0 --region eu-west-1
    - as-update-auto-scaling-group PuRo-group --min-size **y** --max-size **y** --region eu-west-1

  dove x e y rappresentato il numero minimo e massimo delle istanze del gruppo di AutoScaling.

BIbliografia

- Route 53: http://aws.amazon.com/route53/
- Elastic Load Balancing: http://aws.amazon.com/elasticloadbalancing/
- Elastic Compute Cloud: http://aws.amazon.com/ec2/
- DynamoDB: http://aws.amazon.com/dynamodb/
- Simple Storage Service: http://aws.amazon.com/s3/
- CloudWatch: http://aws.amazon.com/cloudwatch/
- AutoScaling: http://aws.amazon.com/autoscaling/
- AutoScaling Command Line Tool: http://aws.amazon.com/developertools/2535
- CloudWatch Command Line Tool: http://aws.amazon.com/developertools/2534
- POP3: http://it.wikipedia.org/wiki/Post_Office_Protocol
- SMTP: http://it.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
- IMAP: http://it.wikipedia.org/wiki/Internet_Message_Access_Protocol
- RFC 1939 - Post Office Protocol 3: http://tools.ietf.org/html/rfc1939
- RFC 5321 - Simple Mail Transfer Protocol: http://tools.ietf.org/html/rfc5321
- RFC 3501 - Internet Message Protocol: http://tools.ietf.org/html/rfc3501
- Postal/Rabid - http://doc.coker.com.au/projects/postal/