Creazione di una serie di webserver con Load Balancer e log centralizzato tramite Docker

Progetto per il corso Laboratorio Amministrazione Sistemi — [CT0157]
Docente *Fabrizio Romano*

Alessandro Benetton
[874886]

Anno Accademico 2020/2021

Indice

Elenco delle figure

Listings

1 Introduzione al progetto

L'obiettivo del progetto è creare una serie di webserver con accesso regolato tramite load balancer e un server di log centralizzato, il tutto sfruttando il sistema di containerizzazione **Docker**.

1.1 Repo Github

L'intero progetto si può trovare su Github all'indirizzo https://github.com/ale-ben/LAS2021-NGINX-LoadBalancer

1.2 Struttura del sistema

Il sistema è composto da 3 componenti:

- Webserver
  - Elabora e risponde a richieste dei browser
- Load Balancer
  - Reindirizza le richieste dei browser ai webserver
- Log Server
  - Raccoglie i log di tutti i servizi in un unico posto

1.3 Struttura documentazione

Nella sezione 2 verrà presentato il progetto completo e verranno analizzati i file nell'insieme.
Dalla sezione 3 alla sezione 5 verranno analizzati i vari servizi nel dettaglio, con le relative configurazioni e immagini docker.
Nella sezione sezione 6 verrà fatto un approfondimento su un sistema alternativo di load balancing: Docker Swarm

2 Il progetto completo

2.1 Docker compose

Verranno analizzate due varianti del docker compose, la prima relativa al progetto, la seconda relativa ad un ambiente reale.

2.1.1 Versione simulata

Questo è il docker compose usato nel progetto, in un ambiente reale una configurazione simile avrebbe poco senso poichè prevede l'esecuzione di molteplici webhost sulla stessa macchina fisica.

```
version: "3"

services:
  WebServer1:
    build: Dockerfiles/webserver/.
    image: webserver
    container_name: WS1
    networks:
      - Internal
    depends_on:
      - Syslogserver
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://localhost:514"
        tag: "WS1"
    restart: unless-stopped
  WebServer2:
    build: Dockerfiles/webserver/.
    image: webserver
    container_name: WS2
    networks:
      - Internal
    depends_on:
      - Syslogserver
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://localhost:514"
        tag: "WS2"
    restart: unless-stopped
  WebServer3:
    build: Dockerfiles/webserver/.
    image: webserver
    container_name: WS3
    networks:
      - Internal
    depends_on:
      - Syslogserver
    logging:
      driver: syslog
```

```
      options:
        syslog-address: "tcp://localhost:514"
        tag: "WS3"
    restart: unless-stopped
  WebServer4:
    build: Dockerfiles/webserver/.
    image: webserver
    container_name: WS4
    networks:
      - Internal
    depends_on:
      - Syslogserver
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://localhost:514"
        tag: "WS4"
    restart: unless-stopped
  LoadBalancer:
    build: Dockerfiles/load-balancer/.
    image: loadbalancer
    container_name: LB
    networks:
      - Internal
      - External
    ports:
      - "80:80"
    depends_on:
      - WebServer1
      - WebServer2
      - WebServer3
      - WebServer4
      - Syslogserver
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://localhost:514"
        tag: "LB"
    restart: unless-stopped
  Syslogserver:
    build: Dockerfiles/rsyslog/.
    image: syslogserver
    container_name: Syslog
    volumes:
      - "[CARTELLA LOCALE LOG]:/var/log"
    ports:
      - 514:514
      - 514:514/udp
    cap_add:
      - SYSLOG
    restart: unless-stopped

networks:
```

```
95   Internal:
96     internal: true
97   External:
```

Listing 1: Docker Compose Progetto

2.1.2 Analisi Docker Compose

Nel file sono presenti 6 servizi, di cui: 4 web server, 1 load balancer e 1 server log.
In questa configurazione l'unica porta da esporre verso l'esterno è la porta 80 indirizzata all'ip della macchina su cui girano tutti i servizi.
Verrà esposta anche la 514 ma NON è necessario esporla fuori dalla lan locale.
I dettagli di configurazione dei vari servizi verranno analizzati in seguito nelle relative sezioni, si noti però che tutti i servizi hanno delle caratteristiche comuni:

- *build*: Eventuale percorso al Dockerfile per costruire l'immagine
- *image*: Nome dell'immagine da usare, oppure nome dell'immagine costruita tramite build
- *container_name*: Nome univoco del container nel sistema
- *networks*: Lista di network a cui collegare il sistema
- *depends_on*: Lista di servizi da avviare prima del servizio in questione
- *logging*
  - *syslog-address*: Indirizzo del server di log, in questo caso *localhost*
  - *tag*: Nome del servizio con cui identificarsi nel log server
- *restart*: azione da intraprendere in caso di errore o arresto anomalo

2.1.3 Versione effettiva

Versione utile in un ambiente reale.
In questo caso il load balancer e i webserver vengono eseguiti su macchine diverse.

```
1  version: "3"
2
3  services:
4    LoadBalancer:
5      build: Dockerfiles/load-balancer/.
6      image: loadbalancer
7      container_name: LB
8      ports:
9        - "80:80"
10     depends_on:
11       - Syslogserver
12     logging:
13       driver: syslog
```

```
      options:
        syslog-address: "tcp://localhost:514"
        tag: "LB"
    restart: unless-stopped
  Syslogserver:
    build: Dockerfiles/rsyslog/.
    image: syslogserver
    container_name: Syslog
    volumes:
      - "[CARTELLA LOCALE LOG]:/var/log"
    ports:
      - 514:514
      - 514:514/udp
    cap_add:
      - SYSLOG
    restart: unless-stopped
```

Listing 2: Docker Compose Reale Load Balancer

Nel primo file troviamo il load balancer e il server log.
La porta 80 del load balancer deve essere esposta al di fuori della lan locale, la 514 del server log deve essere esposta SOLO SE le macchine su cui gireranno i webserver saranno al di fuori della rete locale.
È inoltre importante ricordarsi di aggiornare gli ip nel config del load balancer per puntare agli ip delle macchine effettive su cui girano i webhost (sottosezione 4.3).

```
version: "3"

services:
  WebServer:
    build: Dockerfiles/webserver/.
    image: webserver
    container_name: WS
    depends_on:
      - Syslogserver
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://[IP Syslogserver]:514"
        tag: "WS-N"
    restart: unless-stopped
```

Listing 3: Docker Compose Reale Load Balancer

Nel secondo file troviamo la configurazione di un webserver.
Questo dockerfile verrà distribuito a tutte le macchine che dovranno ospitare un webserver.
È necessario aggiornare l'indirizzo del server di log in *syslog-server* con l'indirizzo della macchina su cui gira il servizio Syslogserver, è inoltre necessario inserire un id univoco nella sezione *tag* del logging al fine di consentire al server log di distinguere tra le varie istanze del webserver.
Dato che non abbiamo più un network docker, è necessario esporre le porte 80 di tutti gli host alla rete interna, al fine di consentire al load balancer reindirizzare le richieste.

2.2 Come avviare un docker compose

Per avviare un docker compose è sufficiente, una volta avviato il daemon docker, posizionarsi nella cartella contenente il file docker compose ed eseguire il comando *docker compose up*, questo comando creerà e avvierà tutti i servizi descritti nel file.

È inoltre possibile avviare solo una parte dei servizi specificando dopo up il nome del servizio come definito nel docker compose.

Per terminare tutti i servizi si usa il comando *docker compose down*.

Per comandi più avanzati visitare la documentazione ufficiale docker compose cli.

2.3 Screenshot funzionamento

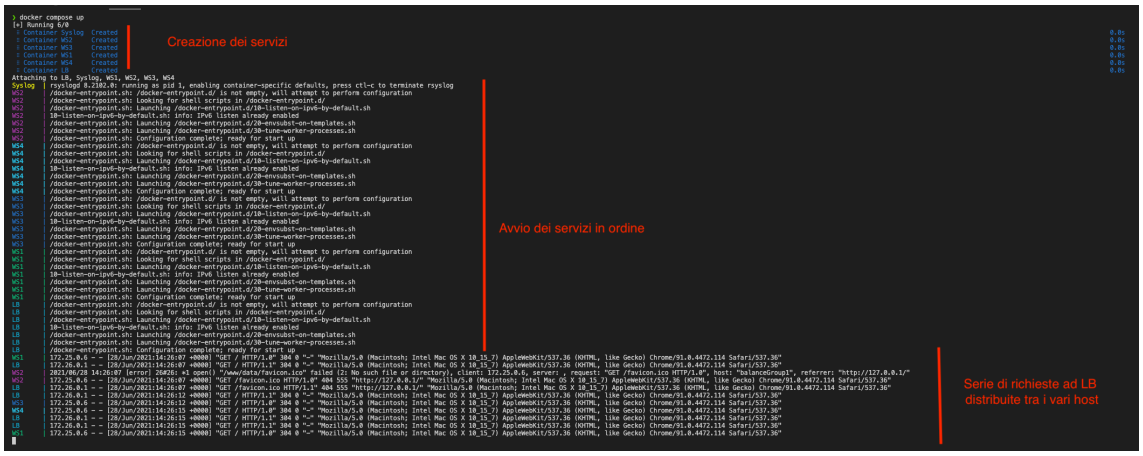

Figura 1: Log di avvio servizi e richieste distribuite

Come si vede dalla Figura 1, il primo servizio avviato è il server log, seguito dai 4 webserver in parallelo e, per ultimo, dal load balancer.
Una volta avviato, il load balancer risponde alle richieste dei client distribuendole tra i server definiti nel config, come visibile nelle ultime righe della Figura 1.

Nella Figura 2 si può notare il template del server di log in azione, i file vengono suddivisi secondo il template e, come previsto, ogni ora viene generato un nuovo file di log.

La Figura 3 mostra il contenuto di uno dei file di log (sezione 5).
Ogni riga contiene:

1. Un timestamp di ricezione del log
2. Indirizzo ip del mittente
3. Tag del servizio
4. Pid del servizio
5. Messaggio

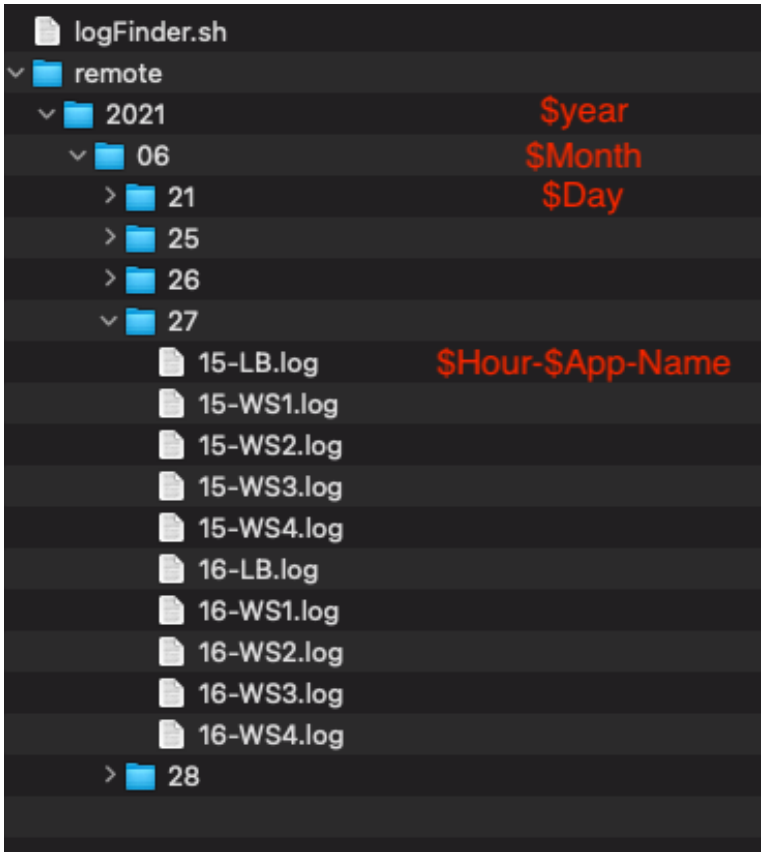

Figura 2: Cartella di salvataggio server di log (sezione 5)

Come previsto, i messaggi mostrati in Figura 3 corrispondono con i log del servizio LB visti in Figura 1.

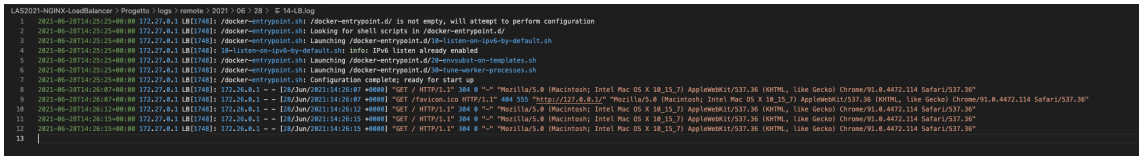

Figura 3: Contenuto del file di log *2021/06/28/14-LB.log*

3 I webserver NGINX

Per l'implementazione del webserver vero e proprio useremo NGINX.
La scelta di usare NGINX è arbitraria, si sarebbe tranquillamente potuto usare un qualunque server web
Partiremo da un immagine docker con NGINX pre installato e la integreremo nel nostro insieme di servizi docker.

Nella sezione *Struttura del sistema* si è detto che vi saranno molteplici webserver tra cui dividere le richieste, nella realtà questi webserver si dovrebbero trovare su macchine differenti al fine di dividere il carico ma, dato che questo progetto è puramente dimostrativo, qui verranno implementate nello stesso host come istanze della stessa immagine docker (verranno comunque evidenziati i cambiamenti necessari per implementare la soluzione multi host).

3.1 Dockerfile

Di seguito il dockerfile per la creazione di un singolo webhost NGINX.

```
1 FROM nginx:1.20.0
2 RUN echo -e "\t\tCopying Config"
3 COPY Contents/nginx.conf /etc/nginx/nginx.conf
4 RUN echo -e "\t\tCopying WebServer"
5 COPY Contents/website /www/data
6 RUN echo -e "\t\Setting Permissions"
7 RUN chmod -R 555 /www/data/.
8 RUN chmod 555 /etc/nginx/nginx.conf
```

Listing 4: Dockerfile Webserver NGINX

3.1.1 Analisi Dockerfile

Scegliamo di utilizzare l'immagine *nginx* alla versione *1.21.0* presente nella repository di immagini ufficiale docker.
Si può facilmente notare che questa è una versione ufficiale poichè non è preceduta dal nome dell'utente che la gestisce (altrimenti sarebbe *utente/nginx*).
La prima modifica che facciamo all'immagine è caricare il nostro file *nginx.conf* nella cartella */etc/nginx/*. Andremo ad analizzare il file di config nella sottosezione 3.3.
Successivamente ci assicuriamo che il config e i file del webserver abbiano i giusti permessi, assegnando *rx* ad utente, gruppo e altri.

3.2 Servizio docker compose

Aggiungiamo molteplici istanze del webserver a docker compose

```
1 WebServer1:
2   build: Dockerfiles/webserver/.
3   image: webserver
4   container_name: WS1
5   networks:
6     - Internal
```

Listing 5: Webserver Docker Compose

La prima riga indica il nome univoco del servizio, ogni istanza del webserver deve avere il proprio nome univoco.
Riga 2 è opzionale e indica il percorso in cui effettuare la build dell'immagine se questa non è presente.
Riga 3 indica il nome dell'immagine. Se non è presente in locale verrà o presa dalla repo remota o buildata (se è presente l'istruzione build).
Riga 4 indica un nickname per il servizio.
Riga 6 impone alla macchina di collegarsi al network *Internal*, il quale non ha accesso alla rete esterna.
È possibile mappare la cartella */www/data* dell'immagine ad una cartella fisica, al fine di poter aggiornare i file del webserver senza dover ricostruire l'immagine completa. Per fare ciò basta aggiungere la voce

```
1     volumes:
2             - "[CARTELLA LOCALE WEBSERVER]:/www/data"
```

al servizio webserver su dockercompose MA bisogna prestare attenzione che l'immagine abbia permessi di lettura sulla cartella condivisa.

Per implementare un sistema multi host le righe 5 e 6 vanno sostituite con la porta su cui esporre il servizio nel formato *Porta fisica:Porta Virtuale*, dove:

- *Porta Fisica* è la porta dell'host su cui esporre il servizio
- *Porta Virtuale* è la porta del container a cui collegare la porta fisica

```
1 WebServer:
2   build: Dockerfiles/webserver/.
3   image: webserver
4   container_name: WS
5   ports:
6     - "80:80"
```

Listing 6: Webserver Docker Compose in multi host

Da notare che, dato che in questo caso i webserver vengono eseguiti su macchine diverse, non è più necessario distinguere i nomi dei servizi.

3.3 Configurazione

```
events {}
http {
  server {
    listen      80;
    location /images/ {
      root /www/static/images;
    }
    location / {
      root /www/data;
    }
  }
}
```

Listing 7: File di configurazione webserver NGINX

Nella configurazione del webserver viene definita la porta su cui ascoltare e le posizioni in cui trovare i file, in questo caso *"/www/static/images"* per le pagine all'indirizzo *"/images"*, *"/www/data"* per tutte le altre.
Nuovi path per il webserver verrano aggiunti qui, rispettando le linee guida della documentazione NGINX.

4 Il Load Balancer NGINX

A questo servizio si collegheranno tutti i client che necessitano di accedere alla risorsa.

4.1 Dockerfile

```
1  FROM nginx:1.20.0
2  RUN echo -e "\t\tCopying Config"
3  COPY Contents/nginx.conf /etc/nginx/nginx.conf
4  RUN echo -e "\t\Setting Permissions"
5  RUN chmod 555 /etc/nginx/nginx.conf
```

Listing 8: Dockerfile Load Balancer NGINX

4.1.1 Analisi Dockerfile

Il file è uguale a quello visto nella sottosezione 3.1 tranne che per il fatto che non carichiamo i file del webserver.

4.2 Servizio docker compose

```
1 LoadBalancer:
2   build: Dockerfiles/load-balancer/.
3   image: loadbalancer
4   container_name: LB
5   networks:
6     - Internal
7     - External
8   ports:
9     - "80:80"
```

Listing 9: Load Balancer Docker Compose

La prima riga indica il nome univoco del servizio.
Riga 2 è opzionale e indica il percorso in cui effettuare la build dell'immagine se questa non è presente.
Riga 3 indica il nome dell'immagine. Se non è presente in locale verrà o presa dalla repo remota o buildata (se è presente l'istruzione build).
Riga 4 indica un nickname per il servizio.
Riga 6 impone al container di collegarsi al network *Internal*, il quale non ha accesso alla rete esterna.
Riga 7 Consente al container di collegari al network *External*, il quale ha accesso alla rete esterna.
Riga 9 Specifica il mapping delle porte in ingresso, la porta del container deve coincidere con quella specificata nel config del Load Balancer.

Per implementare un sistema multi host La riga 6 non è necessaria, dato che le macchine saranno collegate tramite rete esterna.

4.3 Configurazione

```
1  events {}
2  http {
3    upstream balanceGroup1 {
4      server WebServer1:80;
5      server WebServer2:80;
6      server WebServer3:80;
7      server WebServer4:80;
8    }
9
10   server {
11     listen 80;
12
13     location / {
14       proxy_pass http://balanceGroup1;
15     }
16   }
17 }
```

Listing 10: File di configurazione Load Balancer NGINX

Alla riga 3 specifichiamo un gruppo di server di nome *balanceGroup1*.
Dalla riga 4 alla riga 7 specifichiamo gli indirizzi dei server appartenenti a *balanceGroup1*, in questo caso vengono utilizzati degli indirizzi appartenenti alla rete *Internal* ma possono essere sostituiti con normalissimi indirizzi HTTP(S).
Alla riga 11 specifichiamo su che porta ascoltare.

Per implementare un sistema multi host Sostituire gli indirizzi alle righe 4-7 con gli indirizzi effettivi dei propri host su cui gira webserver (sezione 3), prestare particolare attenzione alle porte su cui rispondono i webserver (sottosezione 3.3).

4.3.1 Modalità di load balancing

NGINX supporta 3 modalità di load balancing:

1. Round Robin
   - Round Robin standard (Default)
   - Round Robin pesato
2. Least Connected
3. Session Persistence (ip-hash)

Round Robin La tecnica Round Robin consiste nel dividere il carico proporzionalmente tra i vari host.

Round Robin standard (Default) In questo caso tutti gli host hanno lo stesso peso → il carico viene distribuito equamente tra tutti.

Round Robin pesato In questo caso vengono specificati dei pesi per alcuni o tutti i server, il round robin distribuisce le richieste tenendo conto dei pesi specificati.
In caso di omissione del peso questo viene considerato pari a 1.

```
upstream balanceGroup1 {
  server WebServer1:80 weight=3;
  server WebServer2:80;
  server WebServer3:80 weight=2;
  server WebServer4:80;
}
```

Listing 11: Round Robin pesato

In questo caso specifico i server 1 e 3 riceveranno rispettivamente il triplo e il doppio delle richieste rispetto ai server 2 e 4.

Least Connected La tecnica Least Connected tiene traccia del carico di ogni server e reindirizza al server con meno carico al momento della richiesta.
Questa tecnica è molto utile nel caso in cui il tempo di risposta ad una richiesta vari significativamente in base alla richiesta, in questo caso con Least Connected evitiamo di caricare un server con molte richieste in elaborazione.

```
upstream balanceGroup1 {
  least_conn;
  server WebServer1:80;
  server WebServer2:80;
  server WebServer3:80;
  server WebServer4:80;
}
```

Listing 12: Least Connected

Session Persistence (ip-hash) Questa tecnica riassegna ad ogni ip sempre lo stesso server usando una funzione di hash per mappare ad ogni ip un server.

```
upstream balanceGroup1 {
  ip_hash;
  server WebServer1:80;
  server WebServer2:80;
  server WebServer3:80;
  server WebServer4:80;
}
```

Listing 13: Session Persistence

Informazioni più dettagliate sulle tipologie di load balancing di NGINX si possono trovare nella documentazione di NGINX.

5 Il server log

Lo scopo di un server di log è quello di raccogliere log da altre macchine e raggrupparli in un unico posto.

5.1 Dockerfile

Il server log rsyslog verrà implementato senza usare un immagine docker pre-compilata, installando le componenti manualmente.

```
1  FROM ubuntu:21.10
2
3  RUN echo -e "\t\Updating system and installing rsyslog" \
4  && apt-get update \
5  && apt-get install --no-install-recommends -y rsyslog \
6  && apt-get clean \
7  && rm -rf /var/lib/apt/lists/*
8
9  RUN echo -e "\t\tCopying Config"
10 COPY Contents/rsyslog.conf /etc/rsyslog.conf
11
12 ENTRYPOINT ["rsyslogd", "-n"]
```

Listing 14: Dockerfile Rsyslog

5.1.1 Analisi Dockerfile

Partiamo caricando un immagine di *ubuntu:21.10* da *Docker Hub*, su questa immagine, dopo aver aggiornato le sorgenti, installiamo il server *rsyslog*.
Una volta installato il server facciamo pulizia del garbage creato dall'installazione, carichiamo il config di *rsyslog* e impostiamo come punto di partenza il comando *rsyslogd -n*.

5.2 Servizio docker compose

```
1  Syslogserver:
2    build: Dockerfiles/rsyslog/.
3    image: syslogserver
4    container_name: Syslog
5    volumes:
6      - "[PERCORSO COMPLETO CARTELLA LOG LOCALE]:/var/log"
7    ports:
8      - 514:514
9      - 514:514/udp
10   cap_add:
11     - SYSLOG
```

Listing 15: Rsyslog Docker Compose

La prima riga indica il nome univoco del servizio.
Riga 2 è opzionale e indica il percorso in cui effettuare la build dell'immagine se questa non è

presente.
Riga 3 indica il nome dell'immagine. Se non è presente in locale verrà o presa dalla repo remota o buildata (se è presente l'istruzione build).
Riga 4 indica un nickname per il servizio.
Riga 6 mappa una directory locale in cui salvare i log alla directory remota */var/log*. Su questa cartella locale saranno salvati i log ricevuti dalle macchine
Riga 8 e 9 Aprono la port 514 in TCP e UDP per consentire al server di ricevere i log.
Se si intende usare solo uno dei protocolli (TCP o UDP), la porta relativa all'altro protocollo va eliminata.
Riga 11 specifica che il server ha bisogno di permessi aggiuntivi di tipo *SYSLOG*, per info su questi permessi consultare *man 7 capabilities*.

5.3 Configurazione

```
1  # Commentare per disabilitare UDP logging
2  #module(load="imudp")
3  #input(type="imudp" port="514")
4
5  # Commentare per disabilitare TCP logging
6  module(load="imtcp")
7  input(type="imtcp" port="514")
8
9  # Template nome file log remoto
10 template(name="RemoteDirTemplate" type="string" string="/var/log/remote/%$year
       %/%$Month%/%$Day%/%$Hour%-%APP-NAME%.log")
11 # Regole di log
12 if ($source != "localhost") then {
13   action(type="omfile" dynaFile="RemoteDirTemplate")
14 }
```

Listing 16: File di configurazione Rsyslog

In questa configurazione abilitiamo solo la versione TCP del servizio di log, per abilitare anche UDP è necessario rimuovere il commento dalle righe 2 e 3.
Alla riga 3 e 7 definiamo le porte per il servizio di log rispettivamente UDP e TCP, queste porte possono esssere modificate ma DEVONO corrispondere a quelle definite alle righe 8 e 9 nella sottosezione 5.1.
La riga 10 definisce il template per il nome dei file su cui salvare i log remoti, verrà analizzata a parte nella sottosottosezione 5.3.1.
Le righe 12 e 13 applicano il template definito alla riga 10 solo ai log provenienti da sorgenti esterne, ovvero con l'attributo *source* diverso da *localhost*.

5.3.1 Template nome file

Il template per il nome di file è il seguente:
/var/log/remote/%$year%/%$Month%/%$Day%/%$Hour%-%APP-NAME%.log
Possiamo suddividere il template in 3 parti:

1. */var/log/remote/*
   - Percorso FISSO della cartella root su cui salvare i log.
2. *%$year%/%$Month%/%$Day%/*
   - Percorso VARIABILE della cartella finale su cui salvare i log.
   - Dipende da:
     - *$year*
     - *$Month*
     - *$Day*
3. *%$Hour%-%APP-NAME%.log*
   - Nome del file in cui salvare i log
   - Dipende da:
   - - *$Hour*
     - *$APP-NAME*
       * Identificativo del programma remoto da cui sono originati i log
       * Può essere sostituito con *$fromhost*, l'hostname della sorgente (o indirizzo ip se DNS non disponibile).

Se ad esempio la macchina con il programma *pippo* generasse un log il 01/01/1970 alle ore 00:05, il percorso finale verrebbe ad essere:
/var/log/remote/1970/01/01-pippo.log
È stato scelto questo ordine delle variabili arbitrariamente, raccogliere i log per data e ora e, in seguito per macchina, consente di avere una migliore visione di insieme.
Altre alternative valide sarebbero potute essere:

- */var/log/remote/%APP-NAME%-%$year%/%$Month%/%$Day%/%$Hour%.log*
  - Suddivide prima per macchina e, successivamente, per data.
  - Fornisce una migliore visione temporale per le singole macchine ma peggiore visione di insieme sul sistema completo.
- */var/log/remote/%$year%/%$Month%/%$Day%/%$Hour%.log*
  - Ignora l'attributo *APP-NAME*, raccoglie i log di tutte le macchine nello stesso file, suddivisi per data.
  - Visione d'insieme sul sistema completo MA rischio di generare file molto pesanti e di difficile lettura.
- Qualunque altra configurazione con le variabili presenti sopra e altre dalla documentazione ufficiale rsynclog

5.4 Ricerca di un file di log

Usando il template definito sopra, per cercare un file di log si può usare il seguente script bash:

```
#!/bin/sh

HOST="WS1"
YEAR=""
MONTH=""
DAY=""

LIMIT="5" # Numero massimo di elementi da visualizzare
SEPARATOR="/" # / su sistemi base Unix o Darwin, \ su sistemi base MS-DOS
BASE_DIR="./remote" # Directory di partenza

if [ -z "$HOST" ]; then
  HOST=".*"
fi

if [ -z "$YEAR" ]; then
  YEAR="[0-9][0-9][0-9][0-9]"
fi

if [ -z "$MONTH" ]; then
  MONTH="[0-9][0-9]"
fi

if [ -z "$DAY" ]; then
  DAY="[0-9][0-9]"
fi

if [ -z "$BASE_DIR" ]; then
  BASE_DIR="."
fi

REGEX=".*$SEPARATOR$YEAR$SEPARATOR$MONTH$SEPARATOR$DAY$SEPARATOR[0-9][0-9]-$HOST
    .log"

if [ -z "$LIMIT" ]; then
  find $BASE_DIR -regex $REGEX
else
  find $BASE_DIR -regex $REGEX | head -$LIMIT
fi
```

Listing 17: Script per ricercare log dati specifici parametri

6 Docker Swarm

6.1 –scale

Avviando docker compose con il flag *–scale nome_servizio=n* docker crea automaticamente *n* istanze del servizio *nome_servizio* e gestisce automaticamente il load balancing con tecnica Round Robin.
Da notare che per effettuare uno scaling automatico, docker richiede che nel compose del servizio non sia presente l'attributo *container_name*.
Questa era un opzione disponibile anche come flag dentro a docker compose v2 ma è stata deprecata in favore di docker swarm.

Nel caso di questo progetto, per avviare molteplici istanze del webserver il comando sarebbe dovuto essere *docker compose up WebServer1 –scale WebServer1=4*

6.2 Docker Swarm

Docker Swarm è una feature che consente l'unione di più istanze docker su macchine differenti sotto un controllo centralizzato.
Ogni istanza di docker, chiamata *nodo*, può avere uno dei due ruoli:

- Manager
- Worker

Il nodo manager conosce in ogni momento lo stato dei nodi worker e si occupa di dividere le task (e, di conseguenza, il carico) fra questi.
Il nodo manager è anche responsabile di gestire i vari errori dei singoli nodi, reindirizzando le task ad altri nodi worker.

Nel momento in cui si ha un docker swarm funzionante, per replicare il sistema realizzato in questo progetto (ignorando il server log) è sufficiente aggiungere al docker compose del webserver l'attributo *scale: 4*.
Questo attributo, disponibile solo se si usa docker in modalità swarm, crea in automatico 4 istanze del webserver e gestisce il load balancing tra queste.

Per maggiori informazioni su docker swarm visitare la documentazione ufficiale.