

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

1	Introduzione al progetto	1
1.1	Struttura del sistema	1
1.2	Struttura documentazione	1
2	Il progetto completo	2
2.1	Docker compose	2
2.1.1	Versione simulata	2
2.1.2	Analisi Docker Compose	4
2.1.3	Versione effettiva	4
2.2	Screenshot funzionamento	6
3	I webserver NGINX	7
3.1	Dockerfile	7
3.1.1	Analisi Dockerfile	7
3.2	Servizio docker compose	7
3.3	Configurazione	8
4	Il Load Balancer NGINX	10
4.1	Dockerfile	10
4.1.1	Analisi Dockerfile	10
4.2	Servizio docker compose	10
4.3	Configurazione	11
4.3.1	Modalità di load balancing	11
5	Il server log	13
5.1	Dockerfile	13
5.1.1	Analisi Dockerfile	13
5.2	Servizio docker compose	13
5.3	Configurazione	14
5.3.1	Template nome file	14
5.4	Ricerca di un file di log	16
6	Docker Swarm	17

Elenco delle figure

Listings

1	Docker Compose Progetto	2
2	Docker Compose Reale Load Balancer	4
3	Docker Compose Reale Load Balancer	5
4	Dockerfile Webserver NGINX	7
5	Webserver Docker Compose	7
6	Webserver Docker Compose in multi host	8
7	File di configurazione webserver NGINX	8
8	Dockerfile Load Balancer NGINX	10
9	Load Balancer Docker Compose	10
10	File di configurazione Load Balancer NGINX	11
11	Round Robin pesato	12
12	Least Connected	12
13	Session Persistence	12
14	Dockerfile Rsyslog	13
15	Rsyslog Docker Compose	13
16	File di configurazione Rsyslog	14
17	Script per ricercare log dati specifici parametri	16

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 Struttura del sistema

Il sistema è composto da 3 componenti:

- Webserver
 - Elabora e risponde alle 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

1.2 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 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.

```
1 version: "3"
2
3 services:
4   WebServer1:
5     build: Dockerfiles/webserver/.
6     image: webserver
7     container_name: WS1
8     networks:
9       - Internal
10    depends_on:
11      - Syslogserver
12    logging:
13      driver: syslog
14      options:
15        syslog-address: "tcp://localhost:514"
16        tag: "WS1"
17    restart: unless-stopped
18  WebServer2:
19    build: Dockerfiles/webserver/.
20    image: webserver
21    container_name: WS2
22    networks:
23      - Internal
24    depends_on:
25      - Syslogserver
26    logging:
27      driver: syslog
28      options:
29        syslog-address: "tcp://localhost:514"
30        tag: "WS2"
31    restart: unless-stopped
32  WebServer3:
33    build: Dockerfiles/webserver/.
34    image: webserver
35    container_name: WS3
36    networks:
37      - Internal
38    depends_on:
39      - Syslogserver
40    logging:
41      driver: syslog
```

```
42     options:
43       syslog-address: "tcp://localhost:514"
44       tag: "WS3"
45     restart: unless-stopped
46 WebServer4:
47   build: Dockerfiles/webserver/.
48   image: webserver
49   container_name: WS4
50   networks:
51     - Internal
52   depends_on:
53     - Syslogserver
54   logging:
55     driver: syslog
56     options:
57       syslog-address: "tcp://localhost:514"
58       tag: "WS4"
59   restart: unless-stopped
60 LoadBalancer:
61   build: Dockerfiles/load-balancer/.
62   image: loadbalancer
63   container_name: LB
64   networks:
65     - Internal
66     - External
67   ports:
68     - "80:80"
69   depends_on:
70     - WebServer1
71     - WebServer2
72     - WebServer3
73     - WebServer4
74     - Syslogserver
75   logging:
76     driver: syslog
77     options:
78       syslog-address: "tcp://localhost:514"
79       tag: "LB"
80   restart: unless-stopped
81 Syslogserver:
82   build: Dockerfiles/rsyslog/.
83   image: syslogserver
84   container_name: Syslog
85   volumes:
86     - "[CARTELLA LOCALE LOG]:/var/log"
87   ports:
88     - 514:514
89     - 514:514/udp
90   cap_add:
91     - SYSLOG
92   restart: unless-stopped
93
94 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 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 O nome dell'immagine costruita tramite build
- *container_name*: Nome univoco nel sistema del container
- *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
```



```
14     options:
15       syslog-address: "tcp://localhost:514"
16       tag: "LB"
17     restart: unless-stopped
18 Syslogserver:
19   build: Dockerfiles/rsyslog/.
20   image: syslogserver
21   container_name: Syslog
22   volumes:
23     - "[CARTELLA LOCALE LOG]:/var/log"
24   ports:
25     - 514:514
26     - 514:514/udp
27   cap_add:
28     - SYSLOG
29   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.

```
1 version: "3"
2
3 services:
4   WebServer:
5     build: Dockerfiles/webserver/.
6     image: webserver
7     container_name: WS
8     depends_on:
9       - Syslogserver
10    logging:
11      driver: syslog
12      options:
13        syslog-address: "tcp://[IP Syslogserver]:514"
14        tag: "WS-N"
15    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 Screenshot funzionamento

Content Here

3 I webserver NGINX

Pe 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 ?? si è detto che vi saranno molteplici webserver tra cui dividere le richieste, nella realtà questi webserver si troverebbero 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\tSetting 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 web 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

```
1 events {}
2 http {
3   server {
4     listen      80;
5     location /images/ {
6       root /www/static/images;
```

```
7     }  
8     location / {  
9         root /www/data;  
10    }  
11 }  
12 }
```

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 verranno 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\tSetting 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 collegarsi 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.

```
1 upstream balanceGroup1 {
2     server WebServer1:80 weight=3;
3     server WebServer2:80;
4     server WebServer3:80 weight=2;
5     server WebServer4:80;
6 }
```

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.

```
1 upstream balanceGroup1 {
2     least_conn;
3     server WebServer1:80;
4     server WebServer2:80;
5     server WebServer3:80;
6     server WebServer4:80;
7 }
```

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.

```
1 upstream balanceGroup1 {
2     ip_hash;
3     server WebServer1:80;
4     server WebServer2:80;
5     server WebServer3:80;
6     server WebServer4:80;
7 }
```

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\tUpdating 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 essere 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 (o indirizzo ip de DNS non disponibile) della sorgente.

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 macchine 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:

```
1 #!/bin/sh
2
3 HOST="WS1"
4 YEAR=""
5 MONTH=""
6 DAY=""
7
8 LIMIT="5" # Numero massimo di elementi da visualizzare
9 SEPARATOR="/" # / su sistemi base Unix o Darwin, \ su sistemi base MS-DOS
10 BASE_DIR="./remote" # Directory di partenza
11
12 if [ -z "$HOST" ]; then
13     HOST=".*"
14 fi
15
16 if [ -z "$YEAR" ]; then
17     YEAR="[0-9][0-9][0-9][0-9]"
18 fi
19
20 if [ -z "$MONTH" ]; then
21     MONTH="[0-9][0-9]"
22 fi
23
24 if [ -z "$DAY" ]; then
25     DAY="[0-9][0-9]"
26 fi
27
28 if [ -z "$BASE_DIR" ]; then
29     BASE_DIR="."
30 fi
31
32 REGEX=".*$SEPARATOR$YEAR$SEPARATOR$MONTH$SEPARATOR$DAY$SEPARATOR[0-9][0-9]-$HOST
33     .log"
34
35 if [ -z "$LIMIT" ]; then
36     find $BASE_DIR -regex $REGEX
37 else
38     find $BASE_DIR -regex $REGEX | head -$LIMIT
39 fi
```

Listing 17: Script per ricercare log dati specifici parametri

6 Docker Swarm