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 Webserver

> Un webserver è un'applicazione software che, in esecuzione su un (host) server, è in grado di gestire le richieste di trasferimento di pagine web verso un client, di solito un web browser. (F. Romano - *web.pdf*)

1.1.1 Webserver Distribuito

Content Here

1.2 Load Balancer

> Il load balancing è una tecnica utilizzata nell'ambito dei sistemi informatici che consiste nel distribuire il carico di elaborazione di uno specifico servizio tra più server, aumentando in questo modo scalabilità e affidabilità dell'architettura nel suo complesso. (Wilipedia - *Bilanciamento del carico*)

Un **load balancer** è un software atto ad implementare una tecnica di load balancing.

1.3 Docker

> Docker è un progetto opensource nato con lo scopo di automatizzare e semplificare la distribuzione di applicazioni. (F. Romano - *docker.pdf*)

Docker si basa sul concetto di *Container*

1.3.1 Docker compose

Content Here

1.4 Log Centralizzato

Content Here

2 Struttura del sistema

3 I webserver NGINX

Pe l'implementazione del webserver vero e proprio useremo NGINX.
Partiremo da un immagine docker con NGINX pre installato e la integreremo nel nostro insieme di servizi docker.

Nella sezione *Webserver Distribuito* 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.

```
FROM nginx:1.21.0
RUN echo -e "\t\tCopying Config"
COPY Contents/nginx.conf /etc/nginx/nginx.conf
RUN echo -e "\t\tCopying WebServer"
COPY Contents/website /www/data
```

Listing 1: Dockerfile Webserver NGINX

Questo dockerfile sarà lo stesso per il webserver e per il loadbalancer, entrambi usano NGINX, le differenze si vedranno nella configurazione del servizio e del container (sottosezione 3.3 e sottosezione 3.2).

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*).

L'unica 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.

3.2 Servizio docker compose

Aggiungiamo molteplici istanze del webserver web a docker compose

```
WebServer1:
  build: Dockerfiles/webserver/.
  image: webserver
  container_name: WS1
  networks:
    - Internal
```

Listing 2: 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.

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 3: 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 4: 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

Dato che il servizio usato è lo stesso, il dockerfile è lo stesso visto nella sottosezione 3.1.

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 5: 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 6: 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 7: 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 8: 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 9: 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 10: Dockerfile Rsyslog

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 11: 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 12: 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 dispositivo remoto da cui sono originati i log

Se ad esempio la macchina *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

6 Docker Compose

7 Il progetto finito

8 Docker Swarm