Web Application deployment for production environment

LAS 2022/2023

Michael Basta - 887533

Indice

1 Overview del Progetto

1.1 Scopo

Lo scopo del Progetto è quello di creare un ambiente di produzione pronto per il rilascio al pubblico di un'applicazione web, sfruttando la portabilità di docker, la potenza di nginx e l'affidabilità di alcune delle più conosciute utility Linux.
Viene dimostrato che grazie a questa configurazione, risulta indifferente il tipo di tecnologia usata per costruire l'applicazione web, il sistema operativo dei server e la distanza fisica tra questi ultimi.

1.2 Risorse utilizzate

Le risorse che sono state utilizzate in questo progettosono le seguenti:

- Docker
- Docker compose
- NFS
- Mdadm
- Django
- K6.js
- Nginx
- Rsyslog
- Fstab
- Ajenti
- Virtualbox

Nella sezione allegati sono disponibili i link alle pagine di riferimento di ogni elemento sopra scritto.

1.3 Funzionalità

Il progetto descrive ed implementa un sistema di deployment di un'applicazione web per la produzione. Il sistema sviluppato è pensato per situazioni in cui la richiesta è alta ed è molto importante la privacy e la persistenza dei dati. Infatti sono implementati tramite docker-compose un numero arbitrario di nodi che servono l'applicazione, bilanciati da nginx. Le cartelle statiche dei media e degli altri file sono montate su un

fileserver che si occupa di gestire e fare backup di questi ultimi. Sul fileserver inoltre è instalato il tool di monitoraggio e amministrazione open source Ajenti[1].

Rsyslog è implementato in ogni istanza del docker-compose e si occupa di gestire e organizzare i log di tutti i container docker, inviandoli al server remoto.

Dopo il deployment è possibile testare i tempi di risposta e la distribuzione delle richieste facendo uso degli script disponibili nella cartella automation/ del progetto.

[1] Vedere allegati -Ajenti

2 Componenti

In questa sezione vengono spiegati brevemente i componenti utilizzati, le configurazioni saranno prese in visione nel prossimo capitolo.

2.1 Django

Django è un framework web ad alta produttività scritto in Python. È progettato per semplificare lo sviluppo di applicazioni web complesse, fornendo strumenti e librerie per gestire aspetti comuni come il routing delle URL, l'interazione con il database, la gestione delle sessioni degli utenti, l'autenticazione e l'autorizzazione, la generazione dinamica di contenuti HTML e molto altro.

Django utilizza il pattern architetturale MVC (Model-View-Controller), anche se in realtà segue un'implementazione leggermente diversa chiamata MTV (Model-Template-View).

Questo modello divide l'applicazione in tre componenti principali: il modello, che rappresenta la struttura dei dati e fornisce l'accesso al database; il template, che definisce come il contenuto viene presentato all'utente; e la view, che contiene la logica di business e gestisce le richieste degli utenti.

Django promuove l'uso di pratiche di sviluppo pulito e offre una struttura coerente e organizzata per l'applicazione.

2.2 Nginx

Nginx è un server web leggero, ad alte prestazioni e adatto per l'uso come proxy inverso e server proxy per applicazioni web. È stato creato per gestire un elevato numero di connessioni simultanee in modo efficiente, offrendo una distribuzione affidabile del carico di lavoro e un'elevata velocità di risposta, anche su hardware meno potente.

Originariamente sviluppato da Igor Sysoev nel 2004, Nginx è diventato estremamente popolare negli ultimi anni ed è ampiamente utilizzato come alternativa a server web più tradizionali come Apache, per questo motivo viene impiegato anche in questo progetto.

Un'altra caratteristica chiave di Nginx è la sua capacità di agire come proxy inverso. Questo significa che può accettare le richieste dei client e inoltrarle a un backend

specifico, come un'applicazione web o un server di applicazioni, che può essere eseguito su una macchina diversa. Questa funzionalità è particolarmente utile per la scalabilità e la gestione del carico, in quanto Nginx può distribuire le richieste in modo equo tra diversi server backend, migliorando le prestazioni complessive dell'applicazione.
Qui è utilizzato con funzione di proxy inverso + HTTPS e nel Docker come proxy inverso + load balacer.

2.3 NFS

NFS, acronimo di Network File System, è un protocollo di rete che consente la condivisione di file e directory tra sistemi operativi attraverso una rete. È stato sviluppato da Sun Microsystems negli anni '80 ed è diventato uno standard ampiamente utilizzato per l'accesso remoto ai file.
Opera su un modello client-server, in cui un server NFS mette a disposizione file e directory che possono essere montati e accessibili da client NFS sulla rete. I client NFS possono leggere, scrivere e eseguire operazioni sui file condivisi come se fossero presenti localmente sul proprio sistema.
NFS supporta sia la condivisione di sola lettura che la condivisione di lettura/scrittura dei file. Inoltre, offre funzionalità avanzate come la gestione dei permessi di accesso, il caching dei dati per migliorare le prestazioni e la gestione dei lock dei file per garantire la coerenza nell'accesso concorrente. È stato ampiamente adottato in ambienti di rete eterogenei, in particolare nei sistemi Unix e Linux, come l'ambiente di progetto sviluppato.
È importante scrivere una configurazione adeguata per garantire la sicurezza e l'affidabilità delle condivisioni di file.

2.4 Mdadm

Mdadm è un'utilità di gestione dei dispositivi multi-disco (MD) in ambienti Linux. Viene comunemente utilizzata per configurare e gestire array di dischi RAID (Redundant Array of Independent Disks) software. È possibile creare diversi livelli di RAID, come RAID 0, RAID 1, RAID 5, RAID 6 e RAID 10, tra gli altri. L'utilizzo del RAID consente di combinare più dischi fisici in un'unica unità logica, fornendo vantaggi come ridondanza dei dati, prestazioni migliorate o entrambi.

In questo progetto è stato creato un'array di tipo RAID 1, per assicurare la ridondanza dei dati assumendo di non avere restrizioni sulla memoria disponibile.

2.5 Rsyslog

Rsyslog è un demone di sistema per il logging su sistemi Linux. È una versione avanzata e migliorata di syslog, offrendo funzionalità aggiuntive e maggiore flessibilità nella gestione dei log di sistema.

Rsyslog consente di ricevere, elaborare e instradare i log generati da diverse fonti, inclusi servizi di sistema, applicazioni e dispositivi di rete, verso i file di log appropriati o destinazioni remote come server syslog centralizzati. Supporta diversi formati di log e offre la possibilità di filtrare, manipolare e correlare i messaggi di log in base a vari criteri. Offre anche funzionalità di sicurezza avanzate, come la crittografia dei log tramite TLS/SSL e l'autenticazione dei client syslog, che contribuiscono a garantire l'integrità e la riservatezza dei log di sistema.

2.6 Fstab

Fstab è un file di configurazione su sistemi operativi Unix-like, incluso Linux, che definisce le informazioni sul montaggio dei dispositivi di archiviazione (come dischi rigidi, partizioni o dispositivi di rete) durante l'avvio del sistema. Consente di specificare il percorso di montaggio, il tipo di filesystem, le opzioni di montaggio e altre impostazioni pertinenti. In questo caso è stata aggiunta la configurazione dell'array RAID 1 in modo tale da essere montato al boot del fileserver.

2.7 Docker

2.7.1 Docker engine

Docker è una piattaforma open source che semplifica la creazione, la distribuzione e l'esecuzione di applicazioni utilizzando i container. I container Docker consentono di impacchettare un'applicazione insieme alle sue dipendenze in un ambiente isolato, garantendo la portabilità e la riproducibilità dell'applicazione su diversi ambienti di esecuzione.

2.7.2 Docker-compose

Docker-compose è uno strumento che consente di definire e gestire applicazioni multi-container con Docker. Attraverso un file di configurazione YAML, Docker Compose permette di specificare i servizi, le immagini Docker, le reti e i volumi necessari per l'applicazione. Questo semplifica la creazione e

l'orchestrazione di ambienti complessi, consentendo di avviare e interconnettere facilmente più container con una singola istruzione, rendendo il processo di sviluppo e distribuzione delle applicazioni più efficiente e riproducibile.

2.8 Ajenti

Ajenti è un pannello di controllo open source che offre un'interfaccia web intuitiva per la gestione di server Linux. Con Ajenti, gli amministratori di sistema possono monitorare le risorse di sistema, gestire servizi, configurare firewall, gestire utenti e molto altro ancora, il tutto attraverso una piattaforma centralizzata e accessibile tramite browser.

2.9 Virtualbox

VirtualBox è un software di virtualizzazione open source che consente di creare e gestire macchine virtuali su diverse piattaforme, tra cui Windows, macOS e Linux. Con VirtualBox, è possibile eseguire più sistemi operativi contemporaneamente sullo stesso computer, fornendo un ambiente isolato e indipendente per l'esecuzione di applicazioni o lo sviluppo software. In questo progetto si utilizzano due o più macchine virtuali a causa della mancanza di server su cui testare. Nonostante ciò tutto è riproducibile sia su server virtuali che su server o computer fisici.

2.10 K6

K6 è un framework open source per il testing del carico e delle prestazioni delle applicazioni. Fornisce un'ampia gamma di funzionalità per la creazione, l'esecuzione e l'analisi dei test di carico, consentendo agli sviluppatori di valutare le prestazioni delle loro applicazioni e identificare eventuali problemi o degrado delle prestazioni.

3 Files di configurazione

In questa sezione vengono commentati e spiegati ad uno ad uno i file di configurazione più rilevanti dell'intero progetto.

3.1 Configurazione di Nginx all'interno dei container docker[2]

La configurazione di Nginx fornita indica un server che ascolta sulla porta 8000. La prima direttiva "location /" inoltra le richieste a un server di backend denominato "web", che ascolta sulla porta 8000. In questo caso "web" viene risolto con l'indirizzo ip del container omonimo, fornito dal sistema dns di Docker. Ciò significa che Nginx fungerà da proxy inverso, inoltrando le richieste ricevute nella posizione di root ("/") al server di backend.

Le direttive "location /static" e "location /media" indicano due percorsi per i file statici e i file multimediali (ad esempio immagini, video, ecc.). Utilizzando l'istruzione "alias", Nginx verrà configurato per servire i file direttamente dalla directory specificata ("/static" e "/media") senza inoltrarli al server di backend. Infatti i volumi remoti con tutti i file statici vengono montati sul container di Docker e non sulle istanze dei server. Nginx utilizza il sistema di load balancing predefinito (Round Robin), in quanto non ne è specificato nessuno particolare. È possibile aggiungere il metodo di load balancing preferito seguendo le istruzioni disponibili sulla documentazione[3].

3.2 Configurazione Rsyslog Docker[4]

La configurazione di rsyslog è commentata per disabilitare il logging UDP, mentre il modulo imtcp viene caricato per consentire il logging TCP sulla porta 514.

Successivamente, viene definito un template chiamato "RemoteDirTemplate" che specifica il percorso e il formato del nome del file di log per il logging remoto. Il template utilizza variabili come %$year%, %$Month%, %$Day% e %$Hour% per organizzare i file di log in una struttura di directory gerarchica basata sulla data e sull'ora. L'applicazione che genera il log viene identificata con %APP-NAME%.

[2] Vedere allegati – Configurazione Nginx Docker
[3] https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/
[4] Vedere allegati – Configurazione Rsyslog Docker

Infine, viene impostata una regola di logging che applica il template "RemoteDirTemplate" solo quando la fonte del log non è "localhost". Quindi, i log provenienti da localhost non verranno scritti nel file specificato dal template, ma verranno invece ignorati.

L'azione "omfile" viene utilizzata per scrivere i log su file utilizzando il template dinamico "RemoteDirTemplate".

3.3 Dockerfile del container rsyslog[5]

Invece di fare una pull del container docker ufficiale di rsyslog si è optato per installarlo in un container Debian, per dimostrare il caso in cui il server di logging non è un container ma un server dedicato. Quindi viene eseguito un aggiornamento del sistema operativo e successivamente viene installato il pacchetto rsyslog. Vengono puliti i file di cache dell'apt e infine viene copiato il file di configurazione rsyslog.conf nella directory appropriata.

L'entrypoint dell'immagine viene impostato come "rsyslogd -n", che avvia il demone rsyslog in modalità foreground all'avvio del container. Questo assicura che il servizio rsyslog sia in esecuzione e pronto per ricevere e gestire i log.

3.4 Dockerfile applicazione Django[6]

Questa configurazione Docker utilizza un'immagine di Python versione 3.11 come base per la costruzione del container. La directory di lavoro viene impostata su "/event", che è il nome del sito, ma potrebbe chiamarsi in qualsiasi modo.

Vengono impostate due variabili d'ambiente: "PYTHONDONTWRITEBYTECODE" viene impostato su 1 per evitare la scrittura di file bytecode Python, mentre "PYTHONUNBUFFERED" viene impostato su 1 per l'output senza buffer di Python, migliorando la gestione dei log e la risposta del programma. Viene eseguita una serie di copie dei file nella directory corrente. I file presenti nella directory "event" vengono copiati nella directory di lavoro del container e vengono copiati anche i file "requirements.txt" e "gunicorn_start". Successivamente, vengono eseguiti i comandi "pip install" per aggiornare pip e installare le dipendenze elencate nel file "requirements.txt" senza memorizzare la cache, come se fosse all'interno di un normale

[5] Vedere allegati – Dockerfile Rsyslog
[6] Vedere allegati – Dockerfile applicazione Django

computer. Infine, viene eseguito il comando "python3 manage.py collectstatic --noinput" per raccogliere i file statici del progetto (ad esempio, file CSS o JavaScript) senza richiedere l'input dell'utente. Questi file vengono scritti in una cartella specificata nel progetto, che corrisponde al volume remoto[7] montato nel container docker.

3.5 Docker-compose.yml[8]

La configurazione sembra definisce un ambiente di sviluppo composto da tre servizi chiamati: "web", "nginx" e "rsyslog".

Il servizio "web" viene costruito a partire dal Dockerfile presente nella directory corrente. Viene avviato tramite il comando "sh gunicorn_start" e dipende dal servizio "rsyslog". Tramite il driver di logging "syslog" invia i log a "localhost:514" (cioè il server rsyslog) con il tag "WS1". Inoltre, viene specificato un file di configurazione delle variabili d'ambiente necessarie al funzionamento dell'applicazione Django tramite il file ".env" nella directory "./event".

Il servizio "nginx" utilizza l'immagine "nginx:latest", monta in la cartella locale contenente la configurazione di Nginx e i volumi del fileserver con i file statici e i file multimediali. Dipende sia dal servizio "web" che da "rsyslog" ed espone la porta 8000. Anche questo servizio utilizza il driver di logging "syslog" e invia i log a "localhost:514" con il tag "WS1".

Infine, il servizio "rsyslog" viene installato direttamente a partire da un'immagine di linux Debian. Utilizza l'immagine "syslogserver" e monta il volume per i log remoti. Espone la porta 514 sia in TCP che in UDP e richiede il privilegio di SYSLOG per pote rraccogliere i log di tutti i container. Viene specificato il riavvio del servizio a meno che non sia esplicitamente stoppato.

La configurazione definisce anche tre volumi, "remote_static", "remote_media" e "remote_logs", ciascuno utilizzando il driver NFS con l'indirizzo IP del fileserver (in

[7] Vedere allegati – Docker-compose.yml, dove viene definito il servizio rsyslog
[8] Vedere allegati – Docker-compose.yml

questo caso 192.168.100.228) e il protocollo NFS versione 4. I volumi vengono montati rispettivamente nelle directory "/event/static", "/event/media" e "/event/logs".

3.6 automation/deploy.sh

Lo script inizia con la definizione dell'interprete di script (“#!/usr/bin/bash”).
Successivamente, viene eseguito il comando “sudo docker compose down -v”, che ferma e rimuove i container Docker definiti nel file di composizione docker-compose.yml, insieme ai relativi volumi (“-v”).

Viene eseguito un controllo condizionale utilizzando l'istruzione “if”. Se il primo argomento passato allo script (“$1”) è uguale a "clean", allora vengono eliminati i container e ricostruiti a partire dal sorgente, in modo tale da integrare eventuali cambiamenti e aggiornamenti. Il comando “sudo docker compose up --build -d --scale web=3”, che avvia i container Docker specificati nel file di composizione docker-compose.yml, in modalità detached (“-d”) e specificando che il servizio "web" deve essere scalato a 3 istanze (“--scale web=3”).

Se la condizione nell'istruzione “if” non è verificata, viene eseguito il blocco di codice all'interno del blocco “else”. In questo caso, viene eseguito il comando “sudo docker compose up -d --scale web=3”, che avvia i container Docker come descritto sopra, ma senza ricostruire le immagini.

Infine, viene eseguito il comando “sudo docker exec progetto_las-web-1 python3 manage.py collectstatic –noinput”, che esegue un comando all'interno di un container Docker specificato (“progetto_las-web-1”). Il comando “python3 manage.py collectstatic” viene eseguito all'interno del container per raccogliere i file statici dell'applicazione Django, con l'opzione “--noinput” che esclude la richiesta di conferma per sovrascrivere i file esistenti.

3.7 Automation/testload.sh, testload.js, testload.py

Nella repo è disponibile uno script per testare la distribuzione delle richieste ai node, il carico e i tempi di risposta e caricamento sotto sforzo. Lo script principale chiama due script secondari che svolgono funzioni diverse.

3.7.1 /testload.sh[9]

Questo è lo script principale, esegue test di carico dell'applicazione web utilizzando k6 con un numero specifico di utenti virtuali (10) e una durata predefinita (30s), quindi analizza la distribuzione del carico utilizzando uno script Python. I messaggi di output servono a fornire informazioni sullo stato dei test eseguiti.

Inoltre è possibile eseguire un test fino a fare crashare l'applicazione se si passa l'argomento "crash", questo serve a testare grossolanamente quante connessioni simultanee riesce a sostenere la soluzione adottata di load balancer e istanze dell'app.

Dopodichè chiama uno script python che verifica la distribuzione delle richieste ai nodi e le stampa sotto forma di dizionario.

3.7.2 /testload.js[10]

In questo file è definito un semplice test di carico utilizzando k6.

La prima riga "import http from 'k6/http';" importa il modulo "http" di k6, che fornisce funzionalità per effettuare richieste HTTP.

La riga successiva "import { sleep } from 'k6';" importa la funzione "sleep" dal modulo k6, che viene utilizzata per inserire una pausa nel test di carico.

La funzione principale "export default function () { ... }" definisce il punto di ingresso del test di carico. All'interno della funzione, viene eseguita una richiesta GET all'applicazione web (in questo caso http://192.168.100.201) utilizzando la libreria "http" importata.

Successivamente, viene inserita una pausa di 1 secondo utilizzando "sleep(1);".

3.7.3 /testload.py[11]

Lo script Python utilizza le librerie "requests", "time", "pprint" e "tqdm" per effettuare richieste HTTP, gestire tempi di pausa, stampare i risultati in modo leggibile e user-friendly. Viene eseguito un loop di 100 iterazioni in cui viene effettuata una richiesta GET all'URL specificato, si estrae il valore "name" dal JSON di risposta e si contano le occorrenze di ogni valore "name" nel dizionario "results". Infine, viene stampato il risultato in un formato leggibile utilizzando la funzione "pprint".

[9] Vedere allegati – automation/testload.sh
[10] Vedere allegati – automation/testload.js
[11] Vedere allegati – automation/testload.py

3.8 Configurazione RAID1 sul fileserver[12]

Questa configurazione rappresenta un singolo array RAID denominato "/dev/md/0".
Ha una versione di metadati pari a 1.2. L'UUID (identificativo univoco) specificato per l'array RAID è "3a3bcbf7:69ec5ae9:a97478ed:ab82a47e" e il nome assegnato all'array RAID è "fileserver1:0".

3.9 Configurazione NFS sul fileserver[13]

Per ognuna delle tre cartelle da condividere, è stata aggiunta una linea di configurazione NFS che ne consente la condivisione in lettura/scrittura della directory "/event/{nomedirectory}" per l'intervallo di indirizzi IP dei server a disposizione (in questo caso 192.168.100.0/24), con operazioni di scrittura eseguite in modo sincrono e senza la verifica di sottodirectory.

3.10 Configurazione fstab del fileserver[14]

La linea aggiunta di configurazione indica che il dispositivo "/dev/md0" (cioè l'array raid) sarà montato nella directory "/event" utilizzando il sistema di file "ext4" con le opzioni predefinite, senza generare errori in caso di mancato montaggio e abilitando la funzionalità di "discard" per il trim del disco.

3.11 Configurazione nginx sul master server

Questa configurazione è stata scritta in modo da instradare le richieste ai percorsi specificati al server di backend, che è in esecuzione localmente sulla porta 8000.
Si definisce un server che risponde all'indirizzo IP del master server da dove viene servita l'applicazione (in questo caso 192.168.100.125).
La sezione "location /" indica che tutte le richieste saranno inoltrate al server di backend in esecuzione su http://localhost:8000.
La sezione "location /static" specifica che le richieste relative alla directory "static" saranno inoltrate al server di backend su http://localhost:8000/static.
La sezione "location /media" specifica che le richieste relative alla directory "media" saranno inoltrate al server di backend su http://localhost:8000/media.

[12] Vedere allegati – Configurazione RAID 1 su fileserver con mdadm
[13] Vedere allegati – Configurazione NFS fileserver
[14] Vedere allegati – Configurazione fstab fileserver

3.12 Ajenti dashboard su fileserver

Sul server è installato Ajenti, un tool open source che permette di avere una dashboard modulare condivisa in locale per gestire ed amministrare in modo più user-friendly il server e i processi.

4 Allegati

4.1 Riferimenti

- Docker – (https://www.docker.com/)
- Docker compose – (https://docs.docker.com/compose/)
- NFS – (https://linux.die.net/man/5/nfs)
- Mdadm – (https://linux.die.net/man/8/mdadm)
- Django – (https://www.djangoproject.com/)
- K6.js – (https://k6.io/)
- Nginx – (https://www.nginx.com/)
- Rsyslog – (https://www.rsyslog.com/)
- Fstab – (https://man7.org/linux/man-pages/man5/fstab.5.html)
- Ajenti – (https://ajenti.org/)
- Virtualbox – (https://www.virtualbox.org/)

4.2 Configurazione Nginx Docker

```
server {
     listen 8000;

     location / {
          proxy_pass http://web:8000;
     }

     location /static {
          alias /static;
     }

     location /media {
          alias /media;
     }
}
```

4.3 Configurazione Rsyslog Docker

```
# Comment to disable UDP logging
#module(load="imudp")
#input(type="imudp" port="514")

# Comment to disable TCP logging
module(load="imtcp")
input(type="imtcp" port="514")

# Remote logging filename template
template(name="RemoteDirTemplate" type="string"
string="/var/log/remote/%$year%/%$Month%/%$Day%/%$Hour%-%APP-
NAME%.log")

# Logging rule
if ($source != "localhost") then {
     action(type="omfile" dynaFile="RemoteDirTemplate")
}
```

4.4 Dockerfile rsyslog

```
FROM debian:latest

RUN apt-get update
RUN apt-get install --no-install-recommends -y rsyslog
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/*

COPY rsyslog.conf /rsyslog.conf/etc/rsyslog.conf

ENTRYPOINT [“rsyslogd”, “-n”]
```

4.5 Dockerfile applicazione Django

```
FROM python:3.11

WORKDIR /event
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

COPY ./event ./
COPY ./requirements.txt ./
COPY ./gunicorn_start ./

RUN pip install --upgrade pip --no-cache-dir
RUN pip install -r ./requirements.txt --no-cache-dir
RUN python3 manage.py collectstatic –noinput
```

4.6 Docker-compose.yml

```
version: "3.9"

services:
  web:
    build: .
    command: sh gunicorn_start
    # ports: # host:container
    #   - 8000:8000
    env_file:
      - ./event/.env
    restart: unless-stopped
    depends_on:
      - rsyslog
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://localhost:514"
        tag: "WS1"

  nginx:
    image: nginx:latest
    volumes:
      - ./nginx_conf:/etc/nginx/conf.d
      - remote_static:/static
      - remote_media:/media
    depends_on:
      - web
      - rsyslog
    ports:
      - 8000:8000
```

```
    restart: unless-stopped
      logging:
        driver: syslog
        options:
          syslog-address: “tcp://localhost:514”
          tag: “WS1”

    rsyslog:
      build: ./rsyslog/
      image: syslogserver
      volumes:
        - remote_logs:/var/log
      ports:
        - 514:514
        - 514:514/udp
      cap_add:
        - SYSLOG
      restart: unless-stopped

volumes:
  remote_static:
    driver_opts:
      type: nfs
      o: addr=192.168.100.228,nfsvers=4
      device: :/event/static
  remote_media:
    driver_opts:
      type: nfs
      o: addr=192.168.100.228,nfsvers=4
      device: :/event/media
  remote_logs:
    driver_opts:
     type: nfs
```

```
      o: addr=192.168.100.228,nfsvers=4
      device: :/event/logs
```

4.7 automation/deploy.sh

```
#!/usr/bin/bash

sudo docker compose down -v

if [ $1 = “clean” ]
then
  # sudo rm -rf /event/volumes/static/*
  sudo docker compose up --build -d --scale web=3
else
  sudo docker compose up -d --scale web=3
fi

sudo docker exec progetto_las-web-1 python3 manage.py collectstatic
-noinput
```

4.8 automation/testload.js

```
import http from 'k6/http';
import { sleep } from 'k6';

export default function () {
     http.get('http://192.168.100.201');
     sleep(1);
}
```

4.9 automation/testload.sh

```
echo "Running test load times...\n"
echo "\n\n\n-------[ LOAD TIMES ]-------\n"
if [ $1 = "crash" ]
then
     k6 run --vus 10000 --duration 30s automation/testload.js
else
     k6 run --vus 10 --duration 30s automation/testload.js
fi
echo "Running test load distribution...\n"
echo "\n\n\n-------[ LOAD DISTRIBUTION ]-------\n
python3 automation/testload.py
```

4.10 automation/testload.py

```
import requests
from time import sleep
from pprint import pprint
from tqdm import tqdm

url = 'http://192.168.100.201/testload/'
results = {}

for _ in tqdm(range(100)):
     r = requests.get(url)
     name = r.json()['name']
     if name in results:
          results[name] += 1
     else:
     results[name] = 0
pprint(results)
```

4.11 Configurazione RAID 1 su fileserver con mdadm

```
ARRAY /dev/md/0  metadata=1.2
UUID=3a3bcbf7:69ec5ae9:a97478ed:ab82a47e name=fileserver1:0
```

4.12 Configurazione nfs fileserver

```
# /etc/exports: the access control list for filesystems which may
be exported
#         to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
#   /srv/homes                    hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
```

```
#                                                       /srv/nfs4
gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/event/static  192.168.100.0/24(rw,sync,no_subtree_check)
/event/media   192.168.100.0/24(rw,sync,no_subtree_check)
/event/logs   192.168.100.0/24(rw,sync,no_subtree_check)
```

4.13 configurazione fstab fileserver

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# systemd generates mount units based on this file, see systemd.mount(5).
# Please run 'systemctl daemon-reload' after making changes here.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=6713592d-a939-4161-9097-d537c5f07a20 /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=0d9d8622-da42-4ba7-9c26-42789714f985 none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/md0 /event ext4 defaults,nofail,discard 0 0
```

4.14 config file nginx su master server

```
server {
     server_name 192.168.100.125;

     location / {
          proxy_pass http://localhost:8000;
     }

     location /static {
          proxy_pass http://localhost:8000/static;
     }

     location /media {
          proxy_pass http://localhost:8000/media;
     }

}
```

4.15 Ajenti dashboard su fileserver