



Web Application deployment for production environment

LAS 2022/2023

Michael Basta - 887533



Indice

1	Overview del Progetto	5
1.1	Scopo	5
1.2	Risorse utilizzate	5
1.3	Funzionalità	5
2	Componenti	7
2.1	Django	7
2.2	Nginx	7
2.3	NFS	8
2.4	Mdadm	8
2.5	Rsyslog	9
2.6	Fstab	9
2.7	Docker	9
2.7.1	Docker engine	9
2.7.2	Docker-compose	10
2.8	Agenti	10
2.9	Virtualbox	10
2.10	K6	10
2.11	Duplicati	11
3	Files di configurazione	11
3.1	Configurazione di Nginx all'interno dei container docker	11
3.2	Configurazione Rsyslog Docker	12
3.3	Dockerfile del container rsyslog	12
3.4	Dockerfile applicazione Django	13
3.5	Docker-compose.yml	13
3.6	automation/deploy.sh	14



3.7	Automation/testload.sh, testload.js, testload.py	15
3.7.1	/testload.sh.....	15
3.7.2	/testload.js	15
3.7.3	/testload.py	16
3.8	Configurazione RAID1 sul filesaver.....	16
3.9	Configurazione NFS sul filesaver	16
3.10	Configurazione fstab del filesaver.....	16
3.11	Configurazione nginx sul master server	17
3.12	Ajenti dashboard su filesaver	17
3.13	Duplicati backup e dashboard.....	17
4	Testing.....	19
4.1	Version control con GitHub.....	19
4.2	Django	20
4.3	Nginx.....	20
4.4	NFS.....	21
4.5	Mdadm.....	22
4.6	Rsyslog.....	22
4.7	Fstab	23
4.8	Docker	23
4.8.1	Docker engine	23
4.8.2	Docker-compose	23
4.9	Ajenti.....	23
4.10	Virtualbox	24
4.11	K6.....	24
4.12	Duplicati	24
5	Allegati.....	25



5.1	Riferimenti.....	25
5.2	Configurazione Nginx Docker	26
5.3	Configurazione Rsyslog Docker	26
5.4	Dockerfile rsyslog	27
5.5	Dockerfile applicazione Django	27
5.6	Docker-compose.yml	28
5.7	automation/deploy.sh.....	30
5.8	automation/testload.js.....	31
5.9	automation/testload.sh.....	31
5.10	automation/testload.py.....	32
5.11	Configurazione RAID 1 su filesaver con mdadm.....	32
5.12	Configurazione nfs filesaver	32
5.13	configurazione fstab filesaver	33
5.14	config file nginx su master server.....	34
5.15	Ajenti dashboard su filesaver	35
5.16	Duplicati dashboard su filesaver	36



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 progetto sono le seguenti:

- Docker
- Docker compose
- NFS
- Mdaemon
- Django
- K6.js
- Nginx
- Rsyslog
- Fstab
- Agenti
- 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 installato il tool di monitoraggio e amministrazione open source Ajenti¹.

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.

Il fileserver è dotato di configurazione RAID 1 per la massima ridondanza dei file (si potrebbe anche implementare un RAID 5). Dato che un array RAID non sostituisce un backup, sul fileserver è presente anche Duplicati, uno strumento dotato di interfaccia grafica web per eseguire backup criptati e schedulati.

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 Django²

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 Nginx³

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.

² <https://www.djangoproject.com/>

³ <https://www.nginx.com/>



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 balancer.

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 Mdaemon⁴

Mdaemon 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

⁴ <https://linux.die.net/man/8/mdadm>



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 Rsyslog⁵

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 Fstab⁶

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 Docker⁷

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.

⁵ <https://www.rsyslog.com/>

⁶ <https://man7.org/linux/man-pages/man5/fstab.5.html>

⁷ <https://www.docker.com/>

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 Ajenti⁸

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 Virtualbox⁹

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 K6¹⁰

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.

⁸ <https://ajenti.org/>

⁹ <https://www.virtualbox.org/>

¹⁰ <https://k6.io/>

2.11 Duplicati¹¹

Duplicati è un programma per il backup che consente di proteggere i dati importanti facendo copie di sicurezza su dispositivi di archiviazione esterni, come hard disk o server cloud, oppure su cartelle locali. Utilizzando una combinazione di algoritmi di crittografia (AES, etc), sistemi compressione, controllo dell'integrità e di autenticazione, garantisce la sicurezza dei file. Offre anche funzionalità avanzate come il backup incrementale, la pianificazione automatica e la possibilità di ripristinare i dati in modo selettivo.

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 docker¹²

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 documentazione¹³.

¹¹ <https://www.duplicati.com/>

¹² Vedere allegati – Configurazione Nginx Docker

¹³ <https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>

3.2 Configurazione Rsyslog Docker¹⁴

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%.

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 rsyslog¹⁵

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.

¹⁴ Vedere allegati – Configurazione Rsyslog Docker

¹⁵ Vedere allegati – Dockerfile Rsyslog

3.4 Dockerfile applicazione Django¹⁶

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 `"unicorn_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 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 remoto¹⁷ montato nel container docker.

3.5 Docker-compose.yml¹⁸

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 unicorn_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"`.

¹⁶ Vedere allegati – Dockerfile applicazione Django

¹⁷ Vedere allegati – Docker-compose.yml, dove viene definito il servizio rsyslog

¹⁸ Vedere allegati – Docker-compose.yml

Il servizio "nginx" utilizza l'immagine "nginx:latest", monta in la cartella locale contenente la configurazione di Nginx e i volumi del filesaver 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 poter raccogliere 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 filesaver (in 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.sh¹⁹

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.js²⁰

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.

¹⁹ Vedere allegati – automation/testload.sh

²⁰ Vedere allegati – automation/testload.js

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.py²¹

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”.

3.8 Configurazione RAID1 sul fileserver²²

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 fileserver²³

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 fileserver²⁴

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

²¹ Vedere allegati – automation/testload.py

²² Vedere allegati – Configurazione RAID 1 su fileserver con mdadm

²³ Vedere allegati – Configurazione NFS fileserver

²⁴ Vedere allegati – Configurazione fstab fileserver



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`.

3.12 Agenti 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.

3.13 Duplicati backup e dashboard

Dalle immagini allegate è possibile dedurre la configurazione del backup principale del fileserver. L'operazione è resa semplicissima dalla dashboard grafica fornita alla porta 8200 del fileserver, dove è possibile configurare:

1. Il nome del backup
2. Il tipo di crittazione e autenticazione per il backup
3. Le cartelle e i file di cui fare le copie
4. La destinazione del backup che spazia da una cartella locale a una remota o addirittura l'integrazione con Amazon S3
5. Lo scheduling del backup
6. La ritenzione delle copie del backup, da tutte alla ritenzione smart
7. (bonus) Sono disponibili moltissime impostazioni avanzate di basso livello tra cui anche la dimensione dei volumi singoli in cui viene scomposto ogni backup.



Sempre la dashboard offre un'interfaccia semplice per ripristinare i file, controllare i log, verificare l'integrità dei backup, del database e nel caso ripararli.

Un'altra opzione interessante è la possibilità di compattare i backup:

Dalla documentazione ufficiale:

“Il processo di compattazione si occupa di questo. Quando una percentuale predefinita di un volume viene utilizzata da backup obsoleti, il volume viene scaricato, i blocchi vecchi vengono rimossi e i blocchi ancora in uso vengono ricompresi e ri-cifrati. Il volume più piccolo senza contenuti obsoleti viene caricato e il volume originale viene eliminato, liberando capacità di archiviazione nel backend. La compattazione può risultare in molti volumi piccoli nel backend. Se esistono abbastanza file piccoli che possono essere combinati in uno o più volumi della dimensione definita (predefinita 50 MB), questi volumi piccoli vengono scaricati, riaggregati e caricati nel backend, sostituendo questi file piccoli.”



4 Testing

Quasi la totalità delle componenti ha delle funzioni di testing, utili per capire e scovare bug prima di rovinare tutto pushandoli in produzione. Se qualcosa non ha tali funzionalità è sempre possibile scrivere uno script apposito in un qualsiasi linguaggio applicabile.

4.1 Version control con GitHub

GitHub Actions è una funzionalità di GitHub che consente di automatizzare i processi di sviluppo e distribuzione del software. Per quanto riguarda Django, GitHub Actions può essere utilizzato per automatizzare il processo di test, integrazione continua e distribuzione (in questo caso dell'applicazione Django).

- **Test automatici:** È possibile configurare GitHub Actions per eseguire automaticamente i test dell'applicazione ad ogni push o pull request. Questo assicura che il codice sia sempre verificato prima di essere pushato nel repository principale.
- **Integrazione continua:** GitHub Actions permette di configurare una pipeline di integrazione continua specifica per Django. Questo significa che ogni volta che vengono apportate modifiche al codice, è possibile eseguire automaticamente i test, generare report di copertura del codice, effettuare il controllo di qualità del codice e altre attività di controllo.
- **Deployment automatizzato:** Utilizzando GitHub Actions, è possibile automatizzare il processo di deployment. È possibile configurare la pipeline di distribuzione in modo che ogni volta che il codice viene pushato su un determinato branch, l'applicazione venga automaticamente distribuita su un server di produzione (non utilizzato in questo progetto, perchè implementato con Docker).
- **Notifiche e avvisi:** GitHub Actions offre funzionalità di notifica che consentono di ricevere avvisi quando si verificano errori o fallimenti nei processi di test e distribuzione. Questo aiuta a individuare rapidamente eventuali problemi e risolverli tempestivamente.

4.2 Django

Django fornisce una serie di funzioni di test che semplificano il processo di creazione e l'esecuzione dei test per le applicazioni web. Alcune delle principali funzioni di test includono:

- **Test di unità:** Django supporta i test di unità che consentono di verificare il corretto funzionamento delle singole unità di codice, come funzioni e classi. È possibile eseguire asserzioni per verificare che i risultati siano conformi alle aspettative.
- **Test di integrazione:** I test di integrazione consentono di verificare l'interazione tra diverse parti del sistema. Django fornisce strumenti per creare test che simulano richieste HTTP, testare le view, i modelli, i form e le operazioni di database.
- **Test funzionali:** Django supporta i test funzionali, che consentono di verificare il comportamento dell'applicazione nel suo insieme. È possibile simulare le interazioni degli utenti, come l'invio di moduli o il clic su link, e verificare che le risposte siano corrette.
- **Test di sicurezza:** Django offre strumenti per eseguire test di sicurezza per individuare vulnerabilità comuni come le vulnerabilità di Cross-Site Scripting (XSS) e di Cross-Site Request Forgery (CSRF). Questi test aiutano a garantire che l'applicazione sia protetta da possibili attacchi.
- **Test di performance:** Django consente di creare test per valutare le prestazioni dell'applicazione. È possibile misurare i tempi di risposta, il carico massimo che l'applicazione può gestire e l'utilizzo delle risorse del sistema.
- **Test di regressione:** I test di regressione sono utilizzati per verificare che le modifiche apportate al codice non abbiano introdotto nuovi bug o rotto funzionalità esistenti. Django fornisce strumenti per automatizzare i test di regressione e facilitare la verifica continua dell'integrità del codice.

4.3 Nginx

Di seguito sono riportate alcune strategie comuni per testare Nginx:

- **Verifica dello stato di avvio:** È possibile eseguire un test per verificare che Nginx si avvii correttamente senza errori. Questo può essere fatto eseguendo il

comando di avvio di Nginx e controllando se viene visualizzato un output senza errori o messaggi di avviso.

- **Test delle configurazioni:** Nginx utilizza file di configurazione per definire il comportamento del server. È possibile testare la correttezza delle configurazioni utilizzando il comando `nginx -t` per verificare se ci sono errori di sintassi o problemi di configurazione. Questo aiuta a evitare errori di configurazione che potrebbero compromettere il funzionamento del server.
- **Test delle prestazioni:** È possibile utilizzare strumenti di benchmarking, come ApacheBench (ab) o Siege, per testare le prestazioni di Nginx. Questi strumenti consentono di simulare una serie di richieste al server e misurare i tempi di risposta, la velocità di trasferimento e altre metriche di prestazione. È possibile utilizzare diversi scenari di test per valutare le prestazioni del server in diverse condizioni di carico.
- **Test delle configurazioni di sicurezza:** Nginx offre funzionalità di sicurezza avanzate come SSL/TLS e controllo degli accessi. È possibile testare la correttezza e la robustezza delle configurazioni di sicurezza utilizzando strumenti come SSL Labs Server Test, che valuta la qualità della configurazione SSL/TLS, o strumenti di scansione della sicurezza come Nikto o Nmap, che possono rilevare potenziali vulnerabilità o debolezze.
- **Monitoraggio continuo:** È consigliabile configurare il monitoraggio continuo di Nginx utilizzando strumenti come Nagios, Zabbix o Prometheus. Questi strumenti possono monitorare le metriche di prestazione, il carico del server, i tempi di risposta e inviare avvisi in caso di problemi o anomalie.

Testare Nginx è importante per garantire che il server sia configurato correttamente, funzioni in modo affidabile, offra prestazioni ottimali e mantenga un adeguato livello di sicurezza.

4.4 NFS

Per testare NFS su Linux, si hanno le seguenti possibilità:

- **Test prestazioni:** Si può utilizzare strumenti come dd, iotop o nfsstat per misurare la velocità di lettura e scrittura, la latenza e altre metriche di prestazione.

- **Monitoraggio e debugging:** È consigliabile monitorare il server e i client NFS utilizzando strumenti come `nfsstat`, `nfsiostat` e il log di sistema per identificare eventuali errori, ritardi o problemi di connettività.

4.5 Mdadm

Per testare un array RAID creato con `mdadm` ci sono diverse possibilità:

- **Monitoraggio dello stato dell'array:** Si può utilizzare il comando `cat /proc/mdstat` per verificare lo stato dell'array RAID e assicurarti che sia inizializzato correttamente.
- **Test di resilienza:** Per testare la resilienza dell'array RAID, si può simulare un guasto di uno dei dischi nel sistema. Lo si può fare rimuovendo un disco fisicamente (se supportato) o utilizzando il comando `mdadm --fail /dev/md0 /dev/sdX` per contrassegnare un disco come guasto. Successivamente, verificare se l'array continua a funzionare correttamente e se i dati sono ancora accessibili.
- **Ricostruzione dell'array:** Se si è simulato un guasto e contrassegnato un disco come guasto, si può testare la ricostruzione dell'array RAID utilizzando il comando `mdadm --manage /dev/md0 --add /dev/sdX` per aggiungere nuovamente il disco all'array. Monitora lo stato dell'operazione di ricostruzione e verifica se l'array viene ripristinato correttamente.
- **Test di prestazioni:** Se si desidera testare le prestazioni dell'array RAID, si possono utilizzare strumenti come `dd`, `bonnie++` o `fio` per misurare la velocità di lettura e scrittura, la latenza e altre metriche di prestazione.
- **Monitoraggio continuo:** È consigliabile configurare il monitoraggio continuo dell'array RAID utilizzando strumenti come `mdadm --monitor` o strumenti di monitoraggio di sistema per identificare eventuali errori, avvisi o problemi di resilienza.

4.6 Rsyslog

Si possono eseguire alcune azioni al fine di verificare il corretto funzionamento di `rsyslog`:

- **Generare un log di prova:** si può generare manualmente un log di prova per verificare se `rsyslog` lo acquisisce correttamente. Ad esempio, con il comando

logger per inviare un messaggio di log personalizzato: `logger "Questo è un messaggio di log di prova"`

- **Simulazione di eventi di log:** è possibile simulare eventi di log specifici per testare le regole di log e le azioni definite nella configurazione di rsyslog. Ad esempio, si può creare uno script che generi messaggi di log per un'applicazione specifica e verificare se rsyslog li elabora come previsto.

4.7 Fstab

Per fstab non sono previsti test di routine, generalmente al momento della configurazione si verifica la sintassi e se il volume/disco viene montato correttamente dopo un reboot.

4.8 Docker

Docker va testato principalmente durante l'installazione.

4.8.1 Docker engine

- **Eseguire un container di prova:** Si può testare Docker eseguendo un container di prova. Ad esempio, il comando `docker run hello-world` per scaricare e avviare un semplice container che stampa un messaggio di saluto.
- **Monitorare e registrare i log:** Verificare che i log dei container siano correttamente registrati e disponibili. Si può utilizzare il comando `docker logs nome_container` per visualizzare i log di un container specifico.

4.8.2 Docker-compose

Docker compose dipende strettamente da Docker Engine, quindi il suo corretto funzionamento è legato al corretto funzionamento di quest'ultimo.

4.9 Ajenti

Una volta installato Ajenti e dopo aver avuto accesso alla dashboard, in linea teorica non si dovrebbe avere bisogno di ulteriore testing, fatto salvo il caso in cui si scrive e si salva una nuova configurazione.



4.10 Virtualbox

VirtualBox è sviluppato e mantenuto da Oracle e non necessita di testing, salvo quello delle configurazioni, da parte dell'utente.

4.11 K6

Gli unici test che si possono attuare riguardanti K6 sono quelli circa la correttezza degli script da eseguire. Ci si può riferire alla documentazione ufficiale.

4.12 Duplicati

Duplicati non ha bisogno di testing specifico, ma comunque offre una visualizzazione user-friendly dei logs di ogni backup eseguito.



5 Allegati

5.1 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/>)
- Duplicati – (<https://www.duplicati.com/>)



5.2 Configurazione Nginx Docker

```
server {  
    listen 8000;  
  
    location / {  
        proxy_pass http://web:8000;  
    }  
  
    location /static {  
        alias /static;  
    }  
  
    location /media {  
        alias /media;  
    }  
}
```

5.3 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")  
}
```



5.4 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"]
```

5.5 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
```



5.6 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
```

5.7 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
```



5.8 automation/testload.js

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

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

5.9 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
```



5.10 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)
```

5.11 Configurazione RAID 1 su fileserver con mdadm

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

5.12 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)
```

5.13 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
```

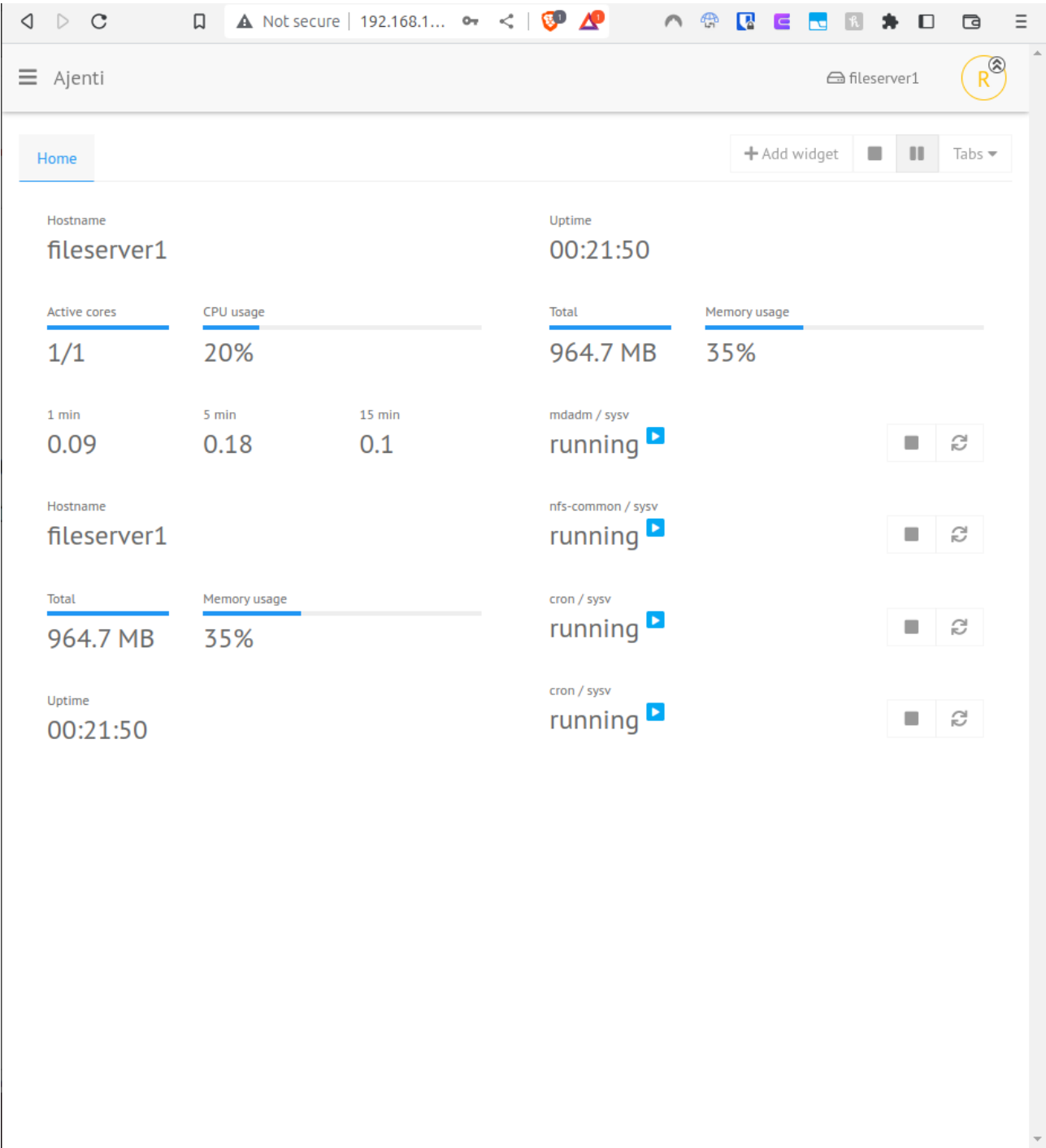


5.14 config file nginx su master server

```
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;  
    }  
}
```



5.15 Ajenti dashboard su fileserver





5.16 Duplicati dashboard su filesaver

Duplicati

Canary

No scheduled tasks

Donate

Home

Add backup

Restore

Settings

About

Log out

1

2

3

4

5

General

Destination

Source Data

Schedule

Options

General backup settings

Name

event_backup

Description (optional)

backup for event shared folders

Encryption

AES-256 encryption, built in

Passphrase

Repeat Passphrase

Show

Generate

Strength: Useless

Next >

Duplicati

Canary

No scheduled tasks

Donate

Home

Add backup

Restore

Settings

About

Log out

1

2

3

4

5

General

Destination

Source Data

Schedule

Options

Source data

Show hidden folders

User data

Computer

bin

boot

dev

etc

event

home

lib

Add a path directly

Add path

Filters

Exclude

< Previous

Next >



1

General

2

Destination

3

Source Data

4

Schedule

5

Options

No scheduled tasks

||

Donate

Home

+ Add backup

Restore

Settings

About

Log out

Backup destination

Storage Type

Local folder or drive

Folder path

srv

sys

tmp

usr

var

backups

cache

lib

local

Username

Optional authentication username

Password

Optional authentication password

Test connection

Advanced options

< Previous

Next >

1

General

2

Destination

3

Source Data

4

Schedule

5

Options

No scheduled tasks

||

Donate

Home

+ Add backup

Restore

Settings

About

Log out

Schedule

☒ Automatically run backups.

If a date was missed, the job will run as soon as possible.

Next time

01:00 PM

06/12/2023

Run again every

2

Hours

Allowed days

☒ Mon

☒ Tue

☒ Wed

☒ Thu

☒ Fri

☒ Sat


☒ Sun

< Previous



Next >


37 of 40



 **Duplicati**
Canary

No scheduled tasks

Donate 

Home

+ Add backup

Restore

Settings

About

Log out

1 General

2 Destination

3 Source Data

4 Schedule


5 Options

General options

Remote volume size

50

MByte

The backups will be split up into multiple files called volumes. Here you can set the maximum size of the individual volume files. See this page for more information. 

Backup retention




Smart backup retention


Over time backups will be deleted automatically. There will remain one backup for each of the last 7 days, each of the last 4 weeks, each of the last 12 months. There will always be at least one remaining backup.

Advanced options



< Previous


Save

Visit us on   

 **Duplicati**
Canary

Next scheduled task: event_backup Tomorrow at 1:00 PM

Donate 

Home



+ Add backup

Restore

Settings

About

Log out

 **event_backup** 

Description:
backup for event shared folders

Operations:
[Run now](#) [Restore files ...](#)




Configuration:
[Edit ...](#) [Export ...](#) [Delete ...](#)

Advanced:
[Database ...](#) [Verify files](#) [Compact now](#) [Commandline ...](#)

Reporting:
[Show log ...](#) [Create bug report ...](#)

Last successful backup: Never - [Run now](#)

Next scheduled run: Tomorrow at 1:00 PM

Visit us on   

38 of 40



Duplicati
Canary

Next scheduled task: event_backup Tomorrow at 1:00 PM

Donate

Home

Add backup

Restore

Settings

About

Log out

Log data for event_backup

General

Remote

Jun 11, 2023 2:12 PM - Operation: Backup

Time

Start 2023-06-11 14:12:19

End 2023-06-11 14:12:19

Duration 00:00:00

Source Files

Examined 160 (5.01 MB)

Opened 0 (0 bytes)

Added 0 (0 bytes)

Modified 0 (0 bytes)

Deleted 0

Test Phase

Delete Phase (Old Backup Versions)

Warnings 0

Errors 0

Complete log

Jun 11, 2023 2:12 PM - Operation: Backup

Time

Start 2023-06-11 14:12:11

End 2023-06-11 14:12:14

Duration 00:00:03

Source Files

Examined 160 (5.01 MB)

Opened 160 (5.01 MB)

Added 160 (5.01 MB)

Modified 0 (0 bytes)

Deleted 0

Test Phase

Compact Phase

Delete Phase (Old Backup Versions)

Warnings 0

Errors 0

Duplicati
Canary

Next scheduled task: event_backup Tomorrow at 1:00 PM

Donate

Home

Add backup

Restore

Settings

About

Log out

1

2

Select files

Restore options

Restore files from event_backup

Restore from

0: Jun 11, 2023 2:12 PM

Latest

0: Jun 11, 2023 2:12 PM

type:com.duplicati

Search for files

Search

/event/

logs

lost+found

media


static

Continue



Visit us on


39 of 40





 **Duplicati**
Canary


Next scheduled task: event_backup Tomorrow at 1:00 PM


 


[Donate](#) 


 Home

 Add backup

 **Restore**

 Settings

 About

 Log out

1
Select files

2
Restore options

Restore options

Where do you want to restore the files to?

☒ Original location

☐ Pick location

Folder path

[Browse](#)

How do you want to handle existing files?

☒ Overwrite

☐ Save different versions with timestamp in file name

Permissions

☐ Restore read/write permissions

[Back](#)

[Restore](#)