Progetto di Laboratorio di Amministrazione di Sistema 2022/23
Università Ca' Foscari Venezia

Implementazione di una Applicazione Web tramite Docker

1.0

Document Informations

Informazioni	
Deliverable	Progetto di Laboratorio di Amministrazione di Sistema
Data di Consegna	27/06/2023
Team members	STEFANO MASIERO,MAKSYM NOVYTSKYY

Indice

1. Introduzione

Il presente rapporto di laboratorio presenta i risultati e le considerazioni relative al progetto di amministrazione di sistema, focalizzato sulla realizzazione di un rilascio (deployment) di un'applicazione web basata su Angular/NodeJS con un database MongoDB e sull'implementazione dell'infrastruttura necessaria per il suo corretto funzionamento.

Come primo passo, è stato selezionato Docker come base del progetto per la creazione dell'infrastruttura. Questa scelta si basa sulla sua capacità di fornire un ambiente isolato e replicabile per l'esecuzione delle diverse componenti dell'applicazione in tempi ridotti.

Diversi servizi lavorano insieme per comporre l'infrastruttura. Il reverse proxy è stato configurato come punto di ingresso, consentendo il reindirizzamento del traffico verso i web server che ospitano l'applicazione. Inoltre, è stato implementato un replica set MongoDB per ospitare il database utilizzato dall'applicazione, garantendo una gestione efficiente e scalabile dei dati. Infine, sono stati configurati servizi di monitoraggio per monitorare lo stato e le prestazioni dell'infrastruttura.

L'obiettivo finale del progetto è fornire una solida base di supporto all'applicazione, garantendo un ambiente stabile e sicuro per gli utenti e consentendo un facile sviluppo e gestione futura. Nei paragrafi successivi, è presente la rappresentazione grafica del progetto con le istanze e connessioni, la struttura dei file necessari per l'avvio e infine la repository github con il codice.

1.1. Struttura Docker Network

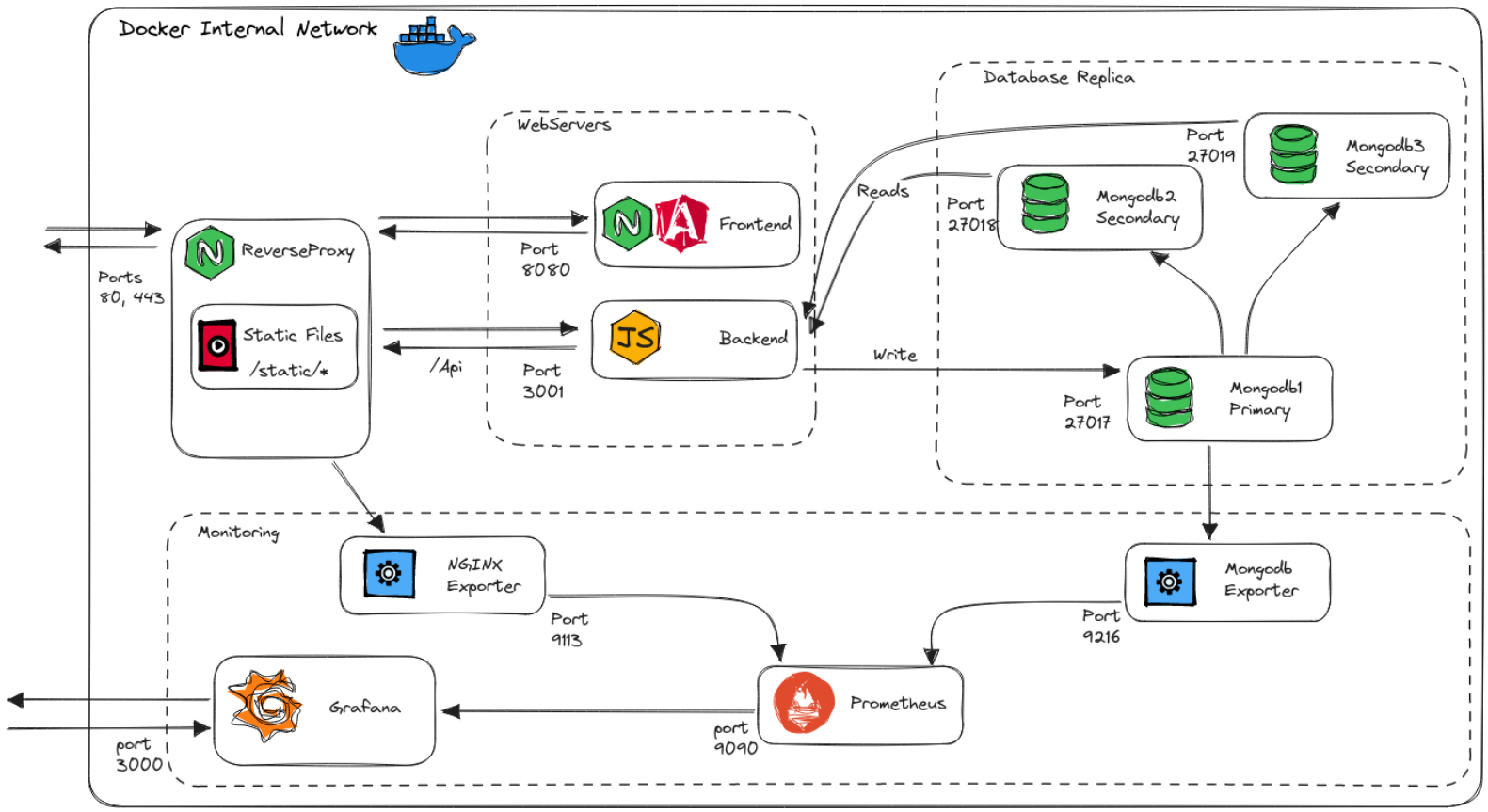

Immagine 1: Schema della rete interna di Docker con servizi attivi e connessioni interne ed esterne.

1.2. Struttura File

```
.
└── Progetto_LAS
    ├── docker-compose.yml
    ├── LICENSE
    ├── README.md
    ├── mongodb/*
    ├── monitoring
    │   ├── grafana/*
    │   └── promethus
    │       └── prometheus.yml
    ├── reverseproxy
    │   ├── contents
    │   │   ├── certificates
    │   │   │   ├── nginx.crt
    │   │   │   └── nginx.key
    │   │   ├── nginx.conf
    │   │   └── static
    │   │       └── images
    │   │           └── cat.jpg
    │   └── Dockerfile
    └── webserver
        ├── backend
        │   ├── Dockerfile
        │   └── expressjs/*
        └── frontend
            ├── angular/*
            ├── Dockerfile
            └── nginx
                └── nginx.conf
```

Blocco 1: Struttura ridotta del progetto rappresentata tramite il comando "tree". I file sono evidenziati in verde, mentre le cartelle sono evidenziate in blu.

1.3. Github

Durante la fase di sviluppo del progetto è stato utilizzato Github come software di versioneamento, per tener traccia delle modifiche effettuate da ciascun membro del gruppo e poter ripristinare vecchie versioni in caso di gravi errori.
Per agevolare la ricostruzione del progetto, viene fornita la repository utilizzata:
https://github.com/St3f4n0Masi3r0/Progetto_LAS

1.4. Osservazioni tecniche e partizionamento

Essendo un sistema progettato per gestire una applicazione web di media dimensione, e la struttura del progetto non si può definire leggera, per questo il sistema richiede una macchina di media/elevata potenza per avere un'esperienza fluida.
Si può immaginare una possibile soluzione hardware e il seguente partizionamento delle risorse:
Si ipotizza di avere un server con una CPU Xeon, con 128GB di RAM e 20TB di storage.
Possibile partizionamento:

- swap: Calcolata col metodo classico. Quindi 128GB * 1.5 = 192 Arrotondato a 200
- /var: Conoscendo che docker salva i container dentro la cartella /var/lib/docker Si utilizza questa directory. Assegnando lo spazio in base a vari container:
    - Mongodb: 1000GB * 3(Replica) = 3000GB
    - Web servers: 500 * 2(Frontend, Backend) = 1000GB
    - ReverseProxy(contiene file statici): 1000GB
    - Grafana: 1000GB
    - Resto(Prometheous e Workers): 500GB * 3 = 1500GB
- /var/log: Per ogni container docker si assegna 100GB per i log:100GB*10(container) = 1000GB
- /: per sicurezza si tiene 800GB

In totale vengono utilizzati 9.5TB dal server. Quindi rimangono ancora a disposizione 10.5TB di spazio. Questo spazio sarà possibile utilizzarlo con LVM in base alle necessità future.

2. Guida Installazione

2.1. Simulazione

Prima di far partire tutto bisogna assicurarsi che docker engine e docker-compose siano stati installati sulla macchina target. Per maggiori informazioni consultare il seguente link: Guida.
Successivamente per avviare il progetto e verificarne il funzionamento è necessario scaricare la repository Github(leggere attentamente il file readme), con tutti i file contenuti, spostarsi al suo interno e far partire il docker-compose.yml file tramite il semplice comando:

```
docker-compose up #con -d se non si vuole visualizzare i log
```

Qui viene fornita l'immagine di funzionamento dei container docker tramite l'immagine di docker desktop (non è stato possibile fare lo screenshot dei log a causa di MongoDB replica set necessita di un tempo notevole per partire e i log vengono riempiti da esso) :

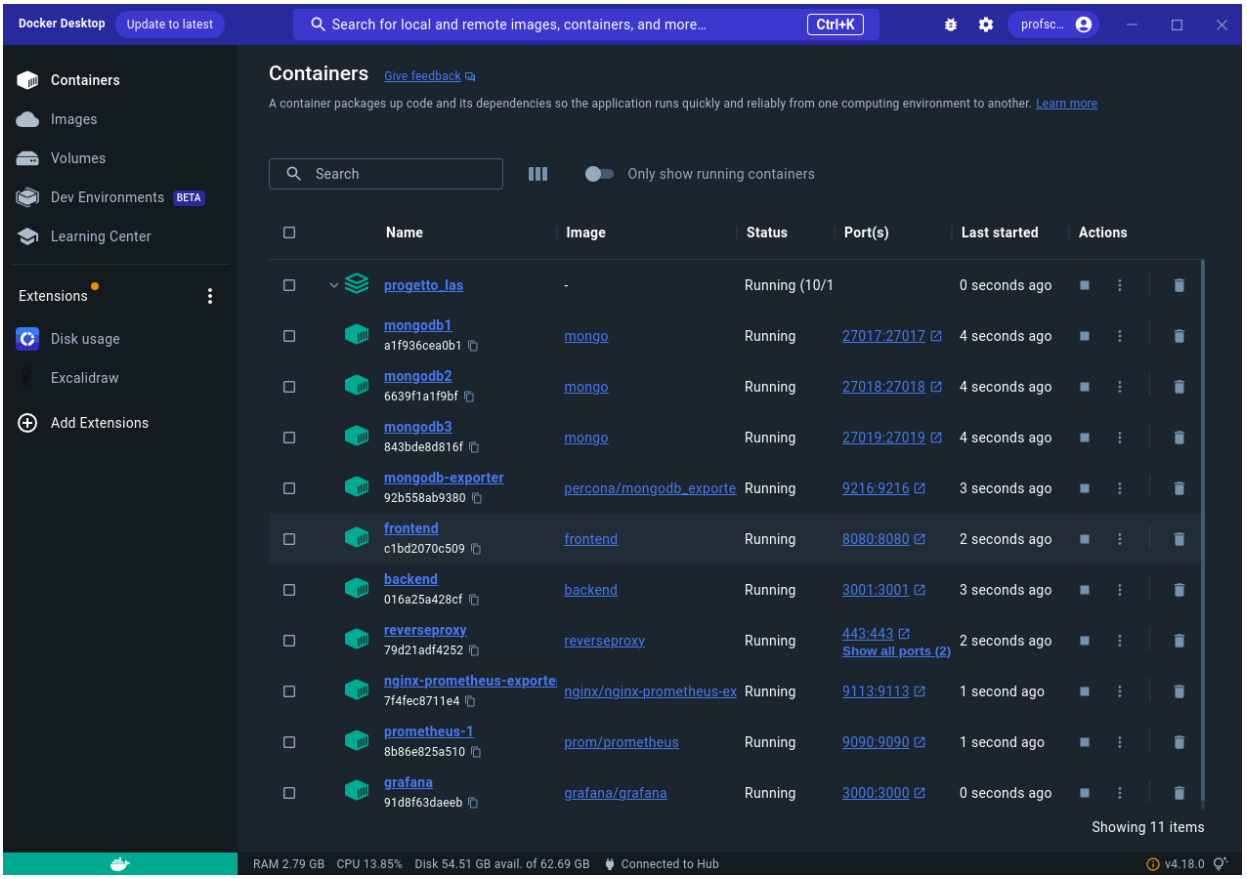

Immagine 2: Screenshot di Docker Desktop con i container del progetto funzionanti

3. Servizi

Nel seguente capitolo sarà fatto un approfondimento sui singoli componenti del progetto. Verrà spiegata la loro funzione e le varie interazioni con altri container.

3.1. Reverse Proxy

Il reverse proxy è un servizio che agisce come punto d'ingresso per il traffico di rete, reindirizzando le richieste verso i server corrispondenti, gestisce i file statici e fornisce le metriche per il suo monitoraggio ad altri servizi.

```
    reverseproxy:
        build: reverseproxy/
        image: reverseproxy
        container_name: reverseproxy
        networks:
            - Internal
            - External
        ports:
            - "80:80"
            - "443:443"
        volumes:
        - ./reverseproxy/contents/certificates:/etc/nginx/ssl
        depends_on:
            - frontend
        restart: unless-stopped
```

Blocco 2: Definizione del reverse proxy nel docker compose, inclusa la configurazione delle porte di ingresso.

Innanzitutto, il reverse proxy si occupa di reindirizzare il traffico verso i web server. Quindi tutte le richieste HTTP(porta 80) o HTTPS(porta 443) vengono gestite dal reverse proxy che utilizza i certificati SSL/TLS per avere una connessione sicura. I certificati(self signed) sono stati specificati tramite i file "nginx.crt" e "nginx.key" presenti nella cartella "etc/nginx/ssl".

```
openssl req -newkey rsa:2048 -nodes -keyout nginx.key -x509 -days 365
-out nginx.crt
```

Blocco 3: Comando utilizzato per la generazione dei certificati self signed

Nella configurazione è stata definita la struttura per gestire i contenuti statici dell'applicazione come file CSS, Javascript e Immagini. Questi contenuti si troveranno nella cartella "var/www/example.com/" e sarà possibile avere accesso tramite la route "/static/"

Inoltre il reverse proxy gestisce le API dell'applicazione reindirizzando le richieste al server backend(su porta 3001) usando la route "/api/". Le specifiche "proxy_set_header" impostano gli header delle richieste per consentire al server backend di ottenere informazioni sul client originale, come l'indirizzo IP.
Nota: il file di configurazione del reverse proxy si può trovare nella cartella "/reverseproxy/contents/nginx.conf"

3.2. Web Servers

In questa sezione viene riportata la vera e propria applicazione. Nel caso specifico è stata costruita una breve demo per gestire una semplice lista di persone (tramite operazioni di CRUD) e visualizzare un immagine. Permettendo di testare le funzionalità del database e dei file statici. In uno scenario vero è proprio qui sarebbe stata sostituita con la vera applicazione complessa.

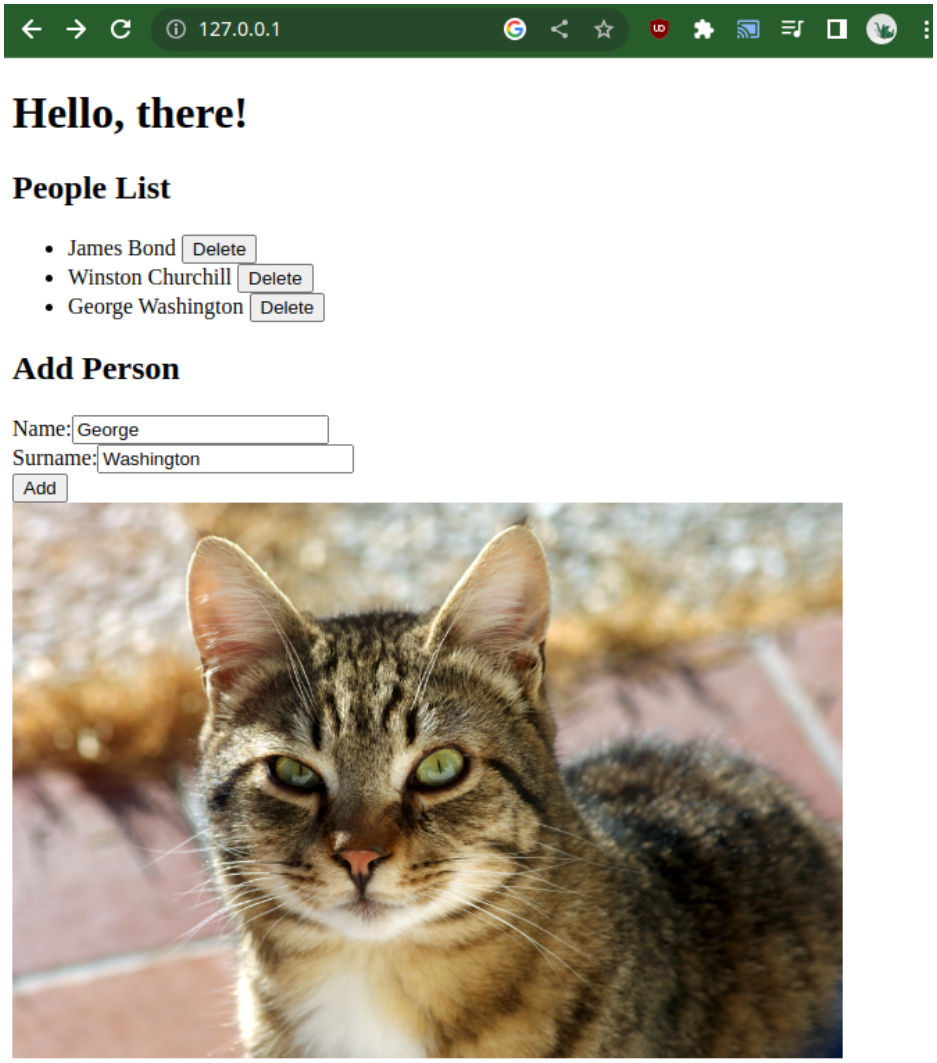

Immagine 3: Screenshot applicazione demo

3.2.1. Backend

Il servizio permette all'interno della applicazione web di fornire i dati attraverso la route "/api"
In base alla richiesta viene fatta l'apposita richiesta al database che a sua volta viene reinviato al client. Sotto viene elencata la lista degli endpoint e che funzione svolgono.

```
'/people'   POST  #utilizzato per visualizzare la lista delle persone nel DB
'/person'   GET   #utilizzato per aggiungere nuova persona nel DB
'/person/:id' DELETE    #utilizzato per eliminare la persona nel DB
```

Blocco 4: Endpoint presenti nel backend attuale.

Il server è realizzato tramite ExpressJS all'interno dell'interprete NodeJS.
Sotto viene riportato il Dockerfile responsabile alla installazione e esecuzione dei servizi sopra elencati.

```
FROM node:latest AS backend
WORKDIR /app/backend
COPY expressjs/ .
RUN npm install
CMD ["node", "server.js"]
```

Blocco 5: Dockerfile del backend presente nella cartella "/webserver/backend/"

Di seguito è riportata la parte del docker-compose, si può osservare che è collegato alla rete interna per non essere esposto direttamente all'esterno.
Inoltre, l'avvio del backend avviene successivamente ai container "mongodb" per garantire la connessione al database.

```
backend:
        container_name: backend
        build: webserver/backend/
        image: backend
        ports:
        - 3001:3001
        depends_on:
            - mongodb1
            - mongodb2
            - mongodb3
        networks:
            - Internal
        restart: unless-stopped
```

Blocco 6: Definizione del backed nel docker compose, specificando le porte, reti e dipendenze

3.2.2. Frontend

Il frontend è costituito da un'applicazione di test, visibile nell'Immagine 3, che viene utilizzata per testare le funzionalità sia dell'applicazione stessa che dell'ambiente Docker complessivo. Questa applicazione è stata sviluppata utilizzando il framework Angular.

Nel Dockerfile, viene inizialmente copiata l'applicazione dalla cartella "angular/" e installate le dipendenze. Successivamente, l'applicazione viene costruita(build) e infine trasferita sul web server nginx.

```
FROM node:latest AS frontend
WORKDIR /app/frontend
COPY angular/ .
RUN npm install
RUN npm run build

FROM nginx:latest
COPY --from=frontend /app/frontend/dist/las-app /usr/share/nginx/html
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
```

Blocco 7: Dockerfile del frontend presente nella cartella "/webserver/frontend/"

Nel docker compose, come nel server backend, è collegato soltanto alla network interna, e viene eseguito successivamente al backend per garantire la corretta funzionalità del applicazione.

```
frontend:
    container_name: frontend
    build: webserver/frontend/
    image: frontend
    ports:
        - "8080:8080"
    depends_on:
        - backend
    networks:
        - Internal
    restart: unless-stopped
```

Blocco 8: Definizione del frontend nel docker compose, specificando le porte, reti e dipendenze.

3.3. Database

L'applicazione utilizza un replica set di MongoDB, che consiste in un gruppo di istanze MongoDB contenenti gli stessi dati. Il replica set consente la divisione del lavoro, in cui l'istanza primaria gestisce le scritture, mentre quelle secondarie gestiscono le letture. Questo permette un funzionamento senza interruzioni, anche in caso di caduta di una istanza l'applicazione continuerà a funzionare.

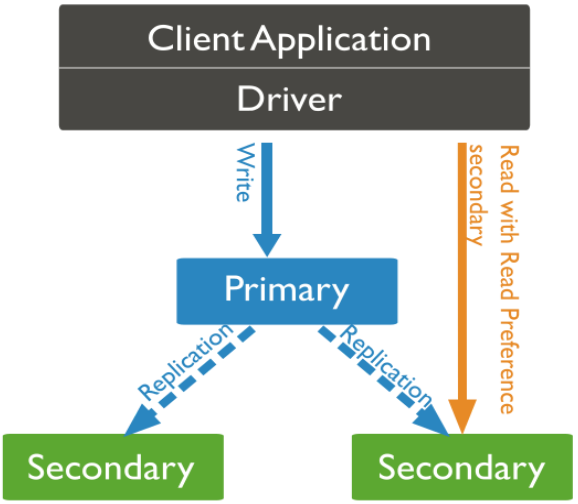

Immagine 4: Rappresentazione funzionamento MongoDB Replica Set

Il MongoDB replica è formato da tre istanze, con "mongodb1" selezionato come l'istanza primaria durante la configurazione iniziale. Questo avviene con l'utilizzo del healthckeck, che presente solo nell'istanza "mongodb1". Il healthckeck eseguirà il comando necessario di inizializzazione e successivamente verifica il corretto funzionamento dell'intero replica set.

```
      healthcheck:
           test: test $$(echo "rs.initiate(
{_id:'rs0',members:        #nome del replica set è "rs0"
[{_id:0,host:\"mongodb1:27017\"},
{_id:1,host:\"mongodb2:27018\"},
{_id:2,host:\"mongodb3:27019\"}]
}).ok || rs.status().ok" | mongosh --port 27017 --quiet) -eq 1
           interval: 10s
           start_period: 30s
           timeout: 60s
```

Blocco 9: HealthCheck usato per inizializzare il replica set "rs0", specificando tutte le istanze del set.

Nel file docker compose sono presenti tre istanze di MongoDB, dichiarate in modo simile, con la differenza che solo la prima istanza ha il "healthckeck" e le porte utilizzate sono diverse per ciascuna istanza.

```
    mongodb1:
        container_name: mongodb1
        image: mongo
        restart: always
        ports:
        - "27017:27017"      #MongoDB2=27018  e MongoDB3=27019
        command: mongod --replSet rs0 --port 27017 --bind_ip_all
        healthcheck:         #Healthcheck presente solo nel primo
           . . .
        volumes:
         - ./mongodb/db1:/data/db   #Presente per aver accesso ai dati
        networks:                   # e dare la possibilità di fare un backup
            - Internal              # diretto (non incluso nel progetto)
```

Blocco 10: Definizione del mongodb al interno del docker compose.

Nota: I volumi sono stati configurati per dare la possibilità esternamente di eseguire backup dei dati.

3.4. Monitoring

Come tutte le applicazioni robuste, necessitano di un sistema di monitoraggio, nel caso specifico è stato utilizzato Grafana per visualizzare i dati, Prometheus per elaborare i dati, e i vari exporter per raccogliere i dati. Ogni componente svolge un ruolo fondamentale per garantire un monitoraggio preciso e in tempo reale dell'applicazione.

3.4.1. Exporters

L'utilizzo degli exporter è fondamentale per l'estrazione dei dati da diversi servizi, il loro ruolo consiste di interrogare periodicamente i diversi servizi e ottenerne lo stato.
Nel sistema sono stati utilizzati exporter per ottenere dati dal reverse proxy e dal mongodb replica.

```
    nginx-prometheus-exporter:
        image: nginx/nginx-prometheus-exporter
        ports:
        - 9113:9113
        environment:
            SCRAPE_URI: http://reverseproxy/metrics
            NGINX_RETRIES: 10
        networks:
        - Internal
        depends_on:
            - reverseproxy
```

Blocco 11: Definizione del exporter utilizzato per il reverse proxy

```
    mongodb-exporter:
        image: percona/mongodb_exporter:2.37.0
        command:
            - '--mongodb.uri=mongodb://mongodb1:27017'
            - '--collect-all'
        container_name: mongodb-exporter
        restart: always
        ports:
            - 9216:9216
        networks:
            - Internal
        depends_on:
            - mongodb1
            - mongodb2
            - mongodb3
```

Blocco 12: Definizione del exporter utilizzato per il mongodb replica

3.4.2. Prometheus

Per sfruttare i vari dati raccolti dagli exporter è fondamentale il lavoro di Prometheus, un sistema di monitoraggio che permette di aggregare, memorizzare ed elaborare le diverse metriche raccolte dagli exporter. In seguito le informazioni elaborate vengono messe a disposizione per Grafana che si occuperà di visualizzarle.

Viene fornito il file di configurazione del servizio Prometheus, in cui vengono specificati i diversi exporter utilizzati e gli intervalli di tempo tra le richieste.

```
scrape_configs:
  - job_name: 'nginx-prometheus-exporter'
    scrape_interval: 10s
    static_configs:
      - targets: ['nginx-prometheus-exporter:9113']

  - job_name: 'mongodb-prometheus-exporter'
    scrape_interval: 10s
    static_configs:
      - targets: ['mongodb-exporter:9216']
```

Blocco 13: File configurazione Prometheus, presente nella cartella “monitoring/prometheus/”

```
    prometheus:
        image: prom/prometheus
        ports:
        - 9090:9090
        networks:
        - Internal
        volumes:
        - ./monitoring/promethus:/etc/prometheus
        command:
        - '--config.file=/etc/prometheus/prometheus.yml'
        depends_on:
            - nginx-prometheus-exporter
            - mongodb-exporter
```

Blocco 14: Definizione del servizio Prometheus dentro al docker compose.

3.4.3. Grafana

L'ultimo componente del sistema di monitoraggio è Grafana, un'applicazione web che permette la visualizzazione e l'analisi interattiva dei dati. Grafana effettua richieste a Prometheus per ottenere le metriche e le presenta tramite apposite dashboard. È possibile scaricare dashboard predefinite dal sito ufficiale di Grafana, oppure è possibile creare dashboard personalizzate per un'esperienza su misura.

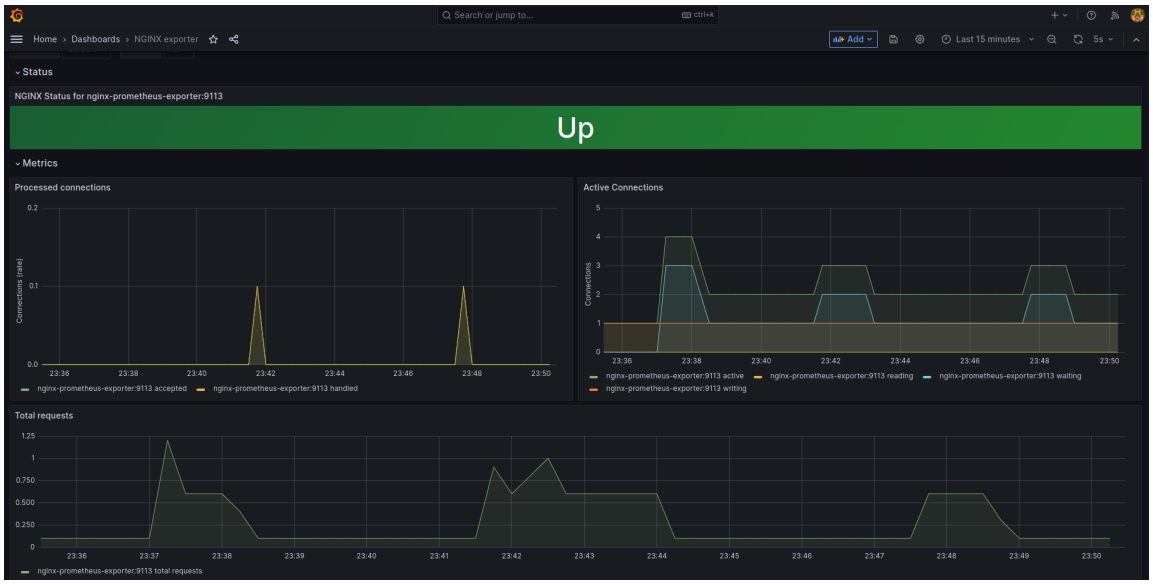

Immagine 5: Dashboard riguardante il Reverse Proxy

Immagine 6: Dashboard riguardante il MongoDB

```
    grafana:
        image: grafana/grafana
        container_name: grafana
        restart: unless-stopped
        ports:
        - 3000:3000
        networks:
        - Internal
        - External
        environment:
        - GF_SECURITY_ADMIN_USER=admin
        - GF_SECURITY_ADMIN_PASSWORD=password
        volumes:
        - ./monitoring/grafana:/var/lib/grafana
        depends_on:
        - prometheus
```

Blocco 15: Definizione di Grafana nel docker compose. Con le dashboard importate da “monitoring/grafana”

4. Conclusioni/Riflessioni

In conclusione, il progetto di amministrazione di sistema ha fornito un'infrastruttura solida per l'applicazione web basata su Angular/NodeJS e il database MongoDB. Per una applicazione di piccole/medie dimensioni l’infrastruttura corrente è più che sufficiente. Tuttavia, nel caso in cui l’applicazione comincia ad avere un elevato traffico sarà necessario espandere l’infrastruttura. Una possibile soluzione consiste nell’implementazione di Kubernetes, consentendo la creazione di molteplici container per i web server e una riorganizzazione del MongoDB replica set con creazione di ulteriori istanze o addirittura implementare un cluster distribuito con il supporto dei replica set. Queste modifiche permetteranno di gestire in modo efficace l'aumento del carico di lavoro e garantire una maggiore scalabilità dell'infrastruttura.

La realizzazione del progetto ha permesso al team l’opportunità di acquisire una profonda comprensione degli elementi chiave dell’amministrazione di sistema, inclusa la progettazione e l'implementazione di un'infrastruttura. Sono stati affrontati, la gestione dei container Docker, la configurazione di vari servizi come il reverse proxy e il monitoraggio con Grafana e Prometheus. Durante l'esperienza, è stata acquisita una conoscenza approfondita del funzionamento dell'infrastruttura, consentendo di esplorare possibili soluzioni per l'espansione e la scalabilità.