

**Progetto Sistemi Cloud**

# **DOCKER SHOWCASE**

**Marco Bongiovanni - W82000189**  
marco.bongiovanni94@gmail.com

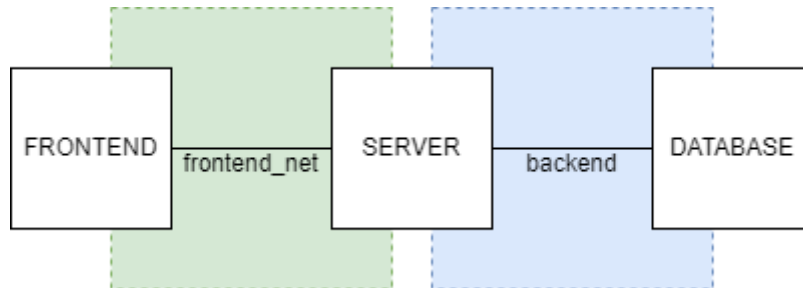
---

## SOMMARIO

<b>Introduzione .....</b>	<b>2</b>
<b>Installazione software aggiuntivo .....</b>	<b>4</b>
<i>Installazione Microk8s.....</i>	<i>4</i>
<i>Installazione Kompose .....</i>	<i>5</i>
<i>Configurazione AWS CLI v2 .....</i>	<i>6</i>
<i>Installazione eksctl .....</i>	<i>7</i>
<b>Struttura del progetto.....</b>	<b>8</b>
<b>Frontend.....</b>	<b>9</b>
<b>Server .....</b>	<b>10</b>
<b>Dockerfile .....</b>	<b>11</b>
<i>Frontend develop .....</i>	<i>11</i>
<i>Frontend release.....</i>	<i>11</i>
<i>Server.....</i>	<i>12</i>
<b>In dettaglio: connessione frontend-server .....</b>	<b>12</b>
<b>Docker Compose .....</b>	<b>13</b>
<b>Docker Swarm .....</b>	<b>15</b>
<b>Configurazione ambiente di deploy Swarm .....</b>	<b>17</b>
<b>Configurazione ambiente di deploy Kubernetes .....</b>	<b>18</b>
<b>EKS.....</b>	<b>20</b>

## INTRODUZIONE

Questo progetto è uno showcase delle tecnologie Docker, la utility Docker Compose, il tool di orchestrazione integrato, ovvero Swarm, Kubernetes e della sua implementazione attraverso i sistemi AWS, EKS.



Il progetto è così strutturato:

- **Database:** realizzato con mongoDB, utilizzato per lo storage di semplici liste di dati;
- **Server:** realizzato con Node.js Express. Questo comunica con il database attraverso una sottorete dedicata, ed è in ascolto sui seguenti endpoint:
  - o **/ping:** fornendo come output un JSON nel seguente formato:
    - **\_id:** identificatore auto generato creato dal DB mongo;
    - **hostname:** identificatore del container creato da Compose, Swarm o K8s;
    - **counter:** numero delle richieste sull'endpoint ricevute dal container individuato con

```
{ "_id": "620d6587b37bed06dc3e67ea",  
  "hostname": "8f2dc94f0504", "counter": 3 }
```

hostname;

Se il container non ha ancora effettuato una connessione al DB, solo **hostname** è valorizzato e una nuova proprietà, **status**, è presente con il valore **pending**.

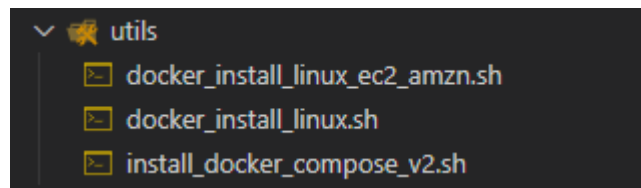
- o **/kill:** che porta all'esecuzione di una chiamata al metodo **NodeJS.Process.kill** che comporta l'arresto del container, la risposta a questa chiamata ha la seguente struttura:

```
{"hostname": "8f2dc94f0504", "status": "killed"}
```

- **frontend:** Una pagina creata con Vue.JS e il relativo plugin Vuetify e servita da un server Apache. Questa comunica con il webserver Express tramite una sottorete dedicata. Tale pagina è costruita attraverso l'uso del comando **build** offerto dalla CLI di Vue.JS, Vue-CLI. L'interfaccia ottenuta permette di effettuare delle richieste agli endpoint descritti precedentemente;

È possibile accedere al progetto al seguente [repository git](#).

La cartella `utils` contiene degli script che possono essere utilizzati per installare Docker e Docker Compose su macchine Linux Ubuntu o istanze EC2 Amazon;



Tra i tool utilizzati vi sono:

- [Multipass](#): utile per la rapida creazione di VM Linux (Ubuntu Server);
- [Microk8s](#): implementazione a singolo nodo di un cluster K8s (con supporto a multinodo, al seguente [link](#) è presente una utile guida su come implementarlo utilizzando [WSL2](#));
- [Kompose](#): utilizzato per convertire velocemente gran parte di una configurazione Docker compose nelle corrispondenti configurazioni per cluster K8s;
- [eksctl](#): semplice CLI utilizzato per creare e mantenere semplici cluster su EKS, servizio Kubernetes Amazon;

## INSTALLAZIONE SOFTWARE AGGIUNTIVO

Di seguito alcuni passi guida seguiti per installare i componenti software utilizzati. Per l'installazione di Multipass e Docker si rimanda alle guide contenute nei relativi siti.

In particolare per l'ambiente di sviluppo utilizzato (Windows 10) è stato installato Multipass in versione 1.8.0+ e l'utility Docker Desktop. Su ambiente Linux sono stati utilizzati gli script citati precedentemente per l'installazione rapida di Docker. Per le restanti componenti software sono state seguite le procedure riportate in seguito.

### INSTALLAZIONE MICROK8S

Su Ubuntu 20.04 (macchina virtuale creata con Multipass, dettagli nei prossimi paragrafi) l'installazione è eseguita come segue:

```
# installazione tramite bundle snap

sudo snap install microk8s --classic --channel=1.23/stable

# aggiungere l'utente al gruppo microk8s

sudo usermod -a -G microk8s $USER

sudo chown -f -R $USER ~/.kube

newgrp microk8s


# controllo stato installazione

microk8s status --wait-ready

# add alias (ATTENZIONE solo se kubectl non è già installato!)

nano ~/.bash_aliases

# incollare e salvare il file così modificato (perché questo abbia effetto è
necessario riavviare la shell)

alias k='microk8s kubectl'

# attivare addons utili

microk8s enable dns metallb dashboard

# una volta installato è possibile avviare o fermare microk8s con

# microk8s start o microk8s stop
```

## INSTALLAZIONE KOMPOSE

Sulla stessa macchina virtuale viene installata anche la utility Kompose, il processo segue i seguenti passi:

```
# installare kompose

curl -L
https://github.com/kubernetes/kompose/releases/download/v1.26.1/kompose-linux-
amd64 -o kompose

chmod +x kompose
```

Il progetto contiene già all'interno della cartella *kompose* l'output dell'utilizzo di questa utility, il comando eseguito per ottenere questo risultato è il seguente:

```
# convertire docker-stack.yml per l'utilizzo con k8s (creare prima la cartella
kompose!)

kompose convert -f docker-stack.yml -o kompose
```

**Nota:** a causa dell'incompatibilità con alcune caratteristiche dei file docker compose (eg. i secrets), il risultato ottenuto non sarà corretto «out of the box», tuttavia rappresenta ugualmente un ottimo punto di partenza.

Il file database-secret.yaml è stato creato manualmente in un secondo momento, vista l'incompatibilità di Kompose di convertire external secrets, che sono usati all'interno della configurazione dello stack in docker-stack.yaml, maggiori dettagli a riguardo verranno presentati nel capitolo relativo alla descrizione di tale configurazione.

Le configurazioni finali, presenti nella cartella */k8s/*, sono stati generati a partire da questi, in particolare è stato deciso di usare i valori di default come base per i servizi frontend e server forniti da kubectl con i seguenti comandi

```
k create deployment server --image=mbongiovanni94/stresstestexample:1.00.000 -o
yaml --port=3000

k expose deployment server --type=ClusterIP --port=3000 --name=server -o yaml

k create deployment frontend --image=mbongiovanni94/stresstestfrontend:1.00.001
-o yaml

k expose deployment frontend --type=NodePort --port=80 --name=frontend -o yaml
```

Le configurazioni così ottenute sono state allineate con le personalizzazioni applicate da Kompose per i relativi deploy/service utilizzati per il deploy in locale usando Microk8s.

## CONFIGURAZIONE AWS CLI V2

L'installazione della AWS CLI v2 è necessaria per l'utilizzo della utility eksctl, deve essere per tanto installata in una fase precedente.

Le procedure di installazione seguite (su una macchina virtuale Ubuntu 20.04) sono presenti al seguente [link](#).

Una volta installato si procede con i seguenti step per fornire accesso ad una utenza IAM precedentemente creata:

- Accedere alla propria console AWS (utente root), accedere alla console del servizio IAM e selezionare l'utente che si intende utilizzare;
- Nella tab "Credenziali di sicurezza" creare una nuova chiave di accesso attraverso l'apposito pulsante;

Utenti > mbongiovanni

### Riepilogo

Elimina utente



ARN utente: [redacted] mbongiovanni

Percorso: /

Data di creazione: 2021-11-25 15:12 UTC+0100

1

Autorizzazioni Gruppi (1) Tag Credenziali di sicurezza Consulente accessi

#### Credenziali di accesso

**Riepilogo**

- Collegamento di accesso alla console: [https://\[redacted\].console](#)

**Password console** Abilitata (ultimo accesso ieri) | [Gestione](#)

**Dispositivo MFA assegnato** Non assegnato | [Gestione](#)

**Certificati di firma** Nessuna

#### Chiavi di accesso

Utilizza le chiavi di accesso per effettuare chiamate programmatiche ad AWS dall'interfaccia a riga di comando AWS (CLI), strumenti per PowerShell, SDK di AWS o chiamate API AWS dirette. Puoi avere un massimo di due chiavi di accesso (attive o inattive) alla volta.

Per motivi di sicurezza, non condividere mai le chiavi segrete con altri. Come best practice, si suggerisce di effettuare una rotazione frequente delle chiavi.

La chiave segreta può essere visualizzata o scaricata solo al momento della creazione. Crea una nuova chiave d'accesso se hai perso quella esistente. [Ulteriori informazioni](#)

Crea chiave di accesso

2

ID chiave di accesso	Data creazione	Ultimo utilizzo	Stato
[redacted]	2022-01-15 14:48 UTC+0100	2022-03-07 12:24 UTC+0100 con cloudformation in us-west-2	Attivo   <a href="#">Rendi inattivo</a> <a href="#">Elimina</a>

- Salvare il file .csv generato in quanto non sarà più possibile accedervi in futuro;
- Annotare **Access Key ID** e **Access Key**;
- Effettuare il login sulla CLI AWS con

```
aws configure
```

ed inserire i dati copiati in precedenza.

## INSTALLAZIONE EKSCCTL

L'installazione di questa CLI (sulla stessa distribuzione Linux su cui è installata la CLI AWS v2) ha seguito le istruzioni contenute nel seguente [link](#).

È necessario applicare delle policy all'utente IAM legato alla CLI AWS.

L'insieme minimo di permessi utilizzato per questo progetto è disponibile al seguente [link](#).

The screenshot displays the AWS IAM console interface. At the top, there is a table with columns: Nome della policy, Tipo, Utilizzato come, and Descrizione. Below this, two policy details are shown.

**EksAllAccess**  
eksctl minimum IAM policies - EksAllAccess

Policy di autorizzazione (1)

```
1- [{"Version": "2012-10-17",
2-   "Statement": [
3-     {
4-       "Effect": "Allow",
5-       "Action": "eks:*",
6-       "Resource": "*"
7-     },
8-     {
9-       "Action": [
10-        "ssm:GetParameter",
11-        "ssm:GetParameters"
12-      ],
13-       "Resource": [
14-        "arn:aws:ssm:*:934840329821:parameter/aws/*",
15-        "arn:aws:ssm:*:parameter/aws/*"
16-      ],
17-       "Effect": "Allow"
18-     },
19-     {
20-       "Action": [
21-        "kms:CreateGrant",
22-        "kms:DescribeKey"
23-      ],
24-       "Resource": "*",
25-       "Effect": "Allow"
26-     },
27-     {
28-       "Action": [
29-        "kms:CreateGrant",
30-        "kms:DescribeKey"
31-      ],
32-       "Resource": "*",
33-       "Effect": "Allow"
34-     }
35-   ]
36- }]
```

**iamLimitedAccess**  
eksctl minimum IAM policies

Policy di autorizzazione (1)

```
1- [{"Version": "2012-10-17",
2-   "Statement": [
3-     {
4-       "Effect": "Allow",
5-       "Action": "iam:*",
6-       "Resource": "*"
7-     },
8-     {
9-       "Action": [
10-        "ssm:GetParameter",
11-        "ssm:GetParameters"
12-      ],
13-       "Resource": [
14-        "arn:aws:ssm:*:934840329821:parameter/aws/*",
15-        "arn:aws:ssm:*:parameter/aws/*"
16-      ],
17-       "Effect": "Allow"
18-     },
19-     {
20-       "Action": [
21-        "kms:CreateGrant",
22-        "kms:DescribeKey"
23-      ],
24-       "Resource": "*",
25-       "Effect": "Allow"
26-     },
27-     {
28-       "Action": [
29-        "kms:CreateGrant",
30-        "kms:DescribeKey"
31-      ],
32-       "Resource": "*",
33-       "Effect": "Allow"
34-     }
35-   ]
36- }]
```

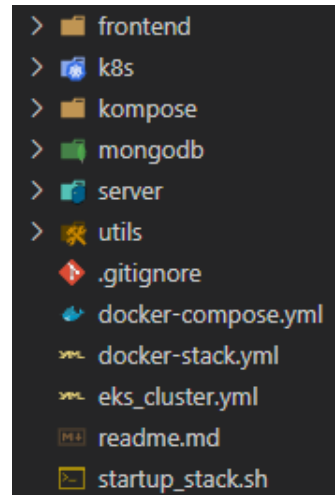
**NOTA:** È stata utilizzata una VM separata rispetto a quella creata per il deploy su un cluster a singolo nodo con Microk8s, descritto successivamente, in modo da evitare eventuali conflitti con la utility kubectl necessaria per il corretto funzionamento della CLI. Le istruzioni per l'installazione della CLI kubectl possono essere trovate al seguente [link](#).



## STRUTTURA DEL PROGETTO

Il progetto è composto dalle seguenti sezioni:

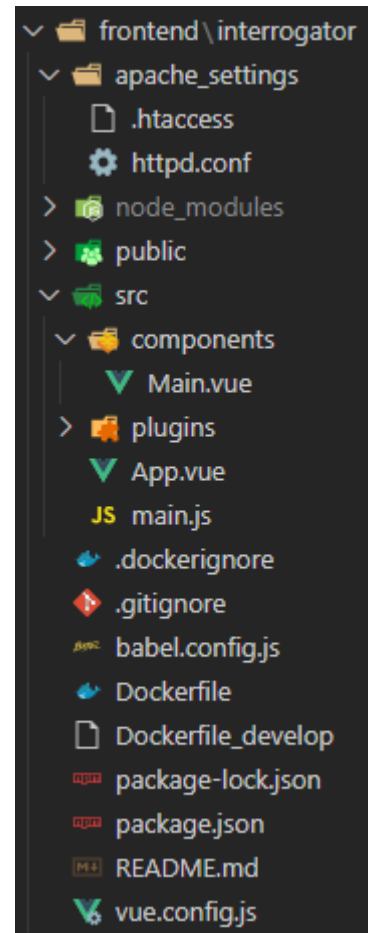
- **frontend:** cartella che ospita il sotto-progetto generato con Vue-CLI ed utilizzato per creare la pagina di frontend;
- **server:** cartella che ospita il sotto-progetto Node.JS che consiste in un server Express.JS;
- **mongodb:** contiene un unico file, `seed.js`, mantenuto per poter inserire dei dati *seed* all'interno del database in fase di creazione del relativo container, utile in fase di development;
- **utils:** contiene gli script di installazione per Docker e Docker Compose citati precedentemente;
- **kompose:** contiene i file `.yaml` di configurazione per il deploy dell'applicazione su un cluster Kubernetes generati dalla utility Kompose;
- **k8s:** contiene le configurazioni per un deploy su un cluster k8s, ottenuto partendo dalle configurazioni presenti della cartella `/kompose`;
- **docker-compose.yml:** file di configurazione per la utility di sviluppo Docker Compose;
- **docker-stack.yml:** file di configurazione per l'orchestrator Swarm;
- **eks\_cluster.yml:** file di configurazione per il deploy dell'applicazione sul servizio EKS;
- **readme.md:** readme con un brief del progetto;
- **startup\_stack.sh:** script di utility per l'avvio di un cluster Swarm, si occupa della creazione dei secret e delle sottoreti necessarie;



## FRONTEND

La cartella *frontend/interrogator* contiene diverse sottocartelle, tra i file maggiormente rilevanti si trovano:

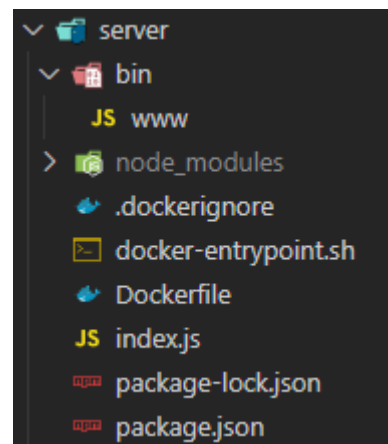
- */src/components/Main.vue*: file Vue che contiene il codice utilizzato per la creazione della landing page. Come già citato al suo interno viene utilizzato il plugin Vuetify per la gestione dei componenti grafici e Axios per effettuare le richieste al server;
- */apache\_setting/\**: file di configurazione del webserver apache contenenti delle personalizzazioni rispetto le configurazioni di default. In particolare *.htaccess* gestisce le CORS policy mentre *httpd.conf* un reverse proxy per gestire le connessioni verso il server;
- *Dockerfile*: *dockerfile* in modalità *release*, descritto successivamente;
- *Dockerfile\_develop*: *dockerfile* in modalità *develop*, descritto successivamente;



## SERVER

Tra i file principali presenti all'interno della cartella `/server` si hanno:

- `/bin/www`: file JS che gestisce l'avvio del server e definisce alcuni listener per intercettare i segnali di sistema come quello di terminazione;
- `/index.js`: contenente il codice che definisce il comportamento del webserver per i relativi endpoint e la connessione dal database;
- `/docker-entrypoint.sh`: utilizzato dal Dockerfile, è uno script altamente diffuso utilizzato come base per gli entrypoint di Dockerfile che andranno usati per creare container sia standalone o con strumenti di sviluppo come Compose sia di produzione, come Swarm. Il suo scopo è convertire i secrets distribuiti multinodo (di default contenuti in un file all'interno del path `/run/secrets/<secret_name>` per i container Linux) in variabili d'ambiente valorizzati con il valore contenuto in tali secrets. Ad esempio il secret `XYZ_DB_PASSWORD_FILE` verrà utilizzato dallo script per leggere il relativo file e valorizzare la variabile d'ambiente `XYZ_DB_PASSWORD`, garantendo la compatibilità di un unico dockerfile ai vincoli espressi da multipli tool che lo utilizzano;



## DOCKERFILE

### FRONTEND DEVELOP

Il file `/frontend/interrogator/Dockerfile_develop.yml`, utilizzato per il deploy dell'applicazione con la utility Compose, rappresenta un dockerfile che segue [le best practices](#) indicate sulla guida relativa all'immagine Node.JS ospitata su Docker Hub. Il Dockerfile esegue le seguenti azioni:

- Proceda alla creazione delle cartelle necessarie;
- Espone la porta indicata via argomento di build, default a 4000;
- Copia i file `package.json` e `package-lock.json` usati per installare le dipendenze necessarie.
- Esegue il comando `"npm run serve"` che avvia un server di sviluppo WebPack (anch'esso utilizza Express.JS), che garantisce diverse feature tra cui le capacità di *hot reloading* che permettono di apprezzare le modifiche al codice in tempo reale senza effettuare nuovamente build o riavviare la pagina lato browser;

### FRONTEND RELEASE

Il file `/frontend/interrogator/Dockerfile.yml` è utilizzato per il deploy del servizio frontend su Docker Swarm e cluster Kubernetes.

Rappresenta una estensione del file `Dockerfile_develop.yml` descritto precedentemente, utilizza il meccanismo di build multi step offerto da Docker, sostituendo il comando `"npm run serve"` con il comando `"npm run build"` che effettua la build del progetto Vue.js all'interno della cartella `/dist/` (creata all'interno del container nel primo step). Questo output viene trasferito all'interno di una immagine httpd, ovvero un server Apache, nella relativa cartella `htdocs`, rendendo di fatto disponibile il contenuto ad essere fornito dal webserver. Di seguito un dettaglio di questa struttura:

```
RUN npm run build
# CMD [ "npm", "run", "build" ]
# the build comand create a /dist folder containing the website builded and ready for prod.

FROM httpd:2
# the --from=0 copy just builder source into the apache webroot
# (instead of using webpack as for the default development env)
COPY --from=0 /opt/node_app/app/dist/ /usr/local/apache2/htdocs/
COPY --from=0 --chown=root:root /opt/node_app/app/apache_settings/.htaccess /usr/local/apache2/conf/.htaccess
COPY --from=0 --chown=root:root /opt/node_app/app/apache_settings/httpd.conf /usr/local/apache2/conf/httpd.conf
EXPOSE $PORT
```

Oltre un guadagno prestazionale, questo consente di creare una immagine con una impronta più piccola in termini di dimensioni.

- Immagine Docker Frontend Release -> 139 MB;
- Immagine Docker Frontend Develop -> 349 MB;

## SERVER

Il file `/server/Dockerfile` è simile a quello progettato per il Frontend develop, con la unica differenza che viene inserito un `entrypoint` che utilizza lo script `docker-entrypoint.sh` come filtro che precede l'esecuzione di eventuali comandi richiesti al container, permettendo la corretta interpretazione dei secrets esterni definiti ad esempio nei deploy Swarm.

```
ENTRYPOINT ["docker-entrypoint.sh"]  
  
CMD [ "node", "./bin/www" ]
```

## IN DETTAGLIO: CONNESSIONE FRONTEND-SERVER

La connessione tra frontend e server è gestita impostando un reverse proxy all'interno delle configurazioni del webserver di development e sulle configurazioni del server Apache in modalità release relative al servizio frontend. Questa operazione è necessaria poiché la rete che collega frontend e server non è accessibile dall'esterno del cluster (specialmente in deploy su Docker in modalità Swarm e Kubernetes).

```
540 <Location "/server/ping">  
541 | ProxyPass "http://server:3000/ping"  
542 </Location>  
543  
544 <Location "/server/kill">  
545 | ProxyPass "http://server:3000/kill"  
546 </Location>  
547
```

Dettaglio `/frontend/interrogator/httpd.conf` per la gestione del reverse proxy in modalità production

```
8 proxy: {  
9   "/server/kill": {  
10    target: 'http://server:3000/kill',  
11    pathRewrite: {'^/server/kill' : ''},  
12    changeOrigin: true  
13  },  
14  "/server/ping": {  
15    target: 'http://server:3000/ping',  
16    pathRewrite: {'^/server/ping' : ''},  
17    changeOrigin: true  
18  },  
19 },
```

Dettaglio `/frontend/interrogator/vue.config.js` per la gestione del reverse proxy in modalità development

Da frontend le chiamate vengono quindi effettuate utilizzando come endpoint lo stesso server che si occupa di servire la pagina all'utente finale, internamente questa verrà poi reindirizzata sulla sottorete appropriata

```
123 axios.get("/server/ping").then((response) => {  
124   if (response.data?.status) {  
125     this.addContainer({hostname: response.data.hostname, status: response.data.status})  
126   } else {  
127     this.addContainer(response?.data[0]);  
128   }  
129 });  
130 }, this.pingIntervalDuration);
```

## DOCKER COMPOSE

Il file `docker-compose.yml` definisce i servizi relativi alle componenti descritte precedentemente, in particolare la componente database sfrutta la sola immagine già fornita sul [Docker Hub](#) ufficiale, definisce le variabili d'ambiente necessarie, ovvero database di default, user e password, con dei dati espressi in chiaro, vista la natura incentrata all'ambiente di sviluppo della utility. Effettua il binding della cartella contenente il file `seed.js` descritto precedentemente e definisce le impostazioni necessarie a stabilire lo stato di "salute" del container nella sezione `healthcheck`. Le stesse variabili d'ambiente sono trasferite al servizio relativo al server, in modo da essere utilizzate per la connessione al database.

```
environment:
  - NODE_ENV=development
  - MONGO_USERNAME=root
  - MONGO_PASSWORD=example
  - MONGO_HOSTNAME=database
  - MONGO_PORT=27017
  - MONGO_DATABASE_NAME=example-database
```

Esempio di utilizzo della variabile `NODE_ENV`, definita come "development" nella configurazione di Compose, che va sovrascrivere la variabile definita nel `dockerfile` con valore di default "production".

```
3 ARG NODE_ENV=production
4 ENV NODE_ENV $NODE_ENV
```

Queste variabili d'ambiente sono utilizzate nel codice per definire diversi parametri necessari al corretto mapping e funzionamento dell'applicazione ad es la definizione dell'URL di connessione al database nel file `/server/index.js`:

```
// Connection URL
const url = `mongodb://${MONGO_USERNAME}:${MONGO_PASSWORD}@${MONGO_HOSTNAME}:${MONGO_PORT}`;
```

Sono definiti due volumi:

```
volumes:
  notused:
  notused_interrogator:
```

Il loro scopo è quello di intercettare i `node_modules` dei due progetti Node.JS, in modo da evitare sovrapposizioni con eventuali cartelle già presenti nell'ambiente host (all'interno delle cartelle del progetto).

Sono inoltre definite le sottoreti utilizzate nel progetto:

```
You, 3 weeks ago | 1 author (You)
networks:
  You, 3 weeks ago | 1 author (You)
  backend:
    driver: bridge
  You, 3 weeks ago | 1 author (You)
  frontend_net:
    driver: bridge
```

È possibile avviare il progetto sfruttando la utility Docker Compose attraverso il comando lanciato sulla root

```
docker compose -f .\docker-compose.yml -p nodemongo up -d --build
```

È possibile omettere il parametro “-d” in modo ottenere uno stream di log di debug.

La pagina dell'applicazione sarà quindi disponibile all'indirizzo <http://localhost:80/>

È quindi possibile distruggere i container e i volumi creati con il comando

```
docker compose -f .\docker-compose.yml -p nodemongo down
```

#### **Nota**

Docker Compose non si occupa di replicazione e resistenza ad eventuali crash dei container. Questi quindi non verranno replicati o riavviati. La chiamata all'endpoint */kill* da parte del frontend fermerà in maniera definitiva l'unico container server presente.

## DOCKER SWARM

Il file *docker-stack.yml* definisce la struttura di uno stack di servizi orchestrati attraverso Docker Swarm, l'orchestratore incluso con la suite Docker.

Il servizio relativo al Database utilizza, come su Docker Compose e Kubernetes, l'immagine ufficiale disponibile su Docker hub come definito nelle precedenti sezioni. Per quanto riguarda i servizi relativi a server e frontend vengono utilizzate delle immagini ottenute dalle build dei relativi Dockerfile <sup>1</sup>, queste sono state pubblicate su Docker hub e sono accessibili ai seguenti indirizzi:

1. [frontend](#) v. 1.00.001
2. [server](#) v. 1.00.000

Come descritto precedentemente, lo script *startup\_stack.sh* si occupa dell'esecuzione dei comandi di avvio di un nuovo stack.

```
echo "---- init swarm ----"
docker swarm init

echo "---- init creating secrets ----"
echo "root" | docker secret create mongo_root -
echo "example" | docker secret create mongo_root_password -

echo "---- init creating networks ----"
docker network create -d overlay frontend_net
docker network create -d overlay backend
```

Figura 1 *startup\_stack.sh*

Lo script si prenderà cura di inizializzare uno Swarm nella macchina su cui viene avviato e creare i secrets e le sottoreti necessarie, successivamente verranno esposti ulteriori dettagli sulla procedura seguita.

In questo scenario è possibile apprezzare l'applicazione dello script di entrypoint, */server/docker-entrypoint.sh*:

```
file_env() {
    local var="$1"
    local fileVar="${var}_FILE"
    local def="${2:-}"
    if [ "${!var:-}" ] && [ "${!fileVar:-}" ]; then
        echo >&2 "error: both $var and $fileVar are set (but are exclusive)"
        exit 1
    fi
    local val="$def"
    if [ "${!var:-}" ]; then
        val="${!var}"
    elif [ "${!fileVar:-}" ]; then
        val="$(cat "${!fileVar}")"
    fi
    echo "var" $var
    echo "fileVar" $fileVar
    echo "def" $def
    echo "val" $val

    export "$var"="$val"
    unset "$fileVar"
}

file_env 'MONGO_USERNAME'
file_env 'MONGO_PASSWORD'

echo "MONGO_USERNAME" $MONGO_USERNAME
echo "MONGO_PASSWORD" $MONGO_PASSWORD
```

Figura 2 Contenuto script *docker-entrypoint.sh*

---

<sup>1</sup> Per quanto riguarda il frontend si tratta dell'immagine ottenuta dal Dockerfile di release: */frontend/interrogator/Dockerfile*



```

var MONGO_USERNAME
fileVar MONGO_USERNAME_FILE
def
val root
var MONGO_PASSWORD
fileVar MONGO_PASSWORD_FILE
def
val example
MONGO_USERNAME root
MONGO_PASSWORD example
Webserver is ready and listening on port 3000
Connected successfully to database
mongodb://root:example@database:27017

```

Figura 3 Output esecuzione script in container

Come mostrato nella Fig.3, le variabili *MONGO\_USERNAME\_FILE* e *MONGO\_PASSWORD\_FILE*, definite nella configurazione dello stack e valorizzate con i due secret creati dallo script *startup-stack.sh*, *mongo\_root* e *mongo\_root\_password*, come mostrato in Fig.1, sono esportati come variabili d'ambiente nel formato atteso dal container Docker. Si è così ottenuto il risultato di rendere trasparente il passaggio tra variabile d'ambiente, come avviene per il deploy utilizzando Compose, e secret file come in quest'ultimo caso.

```

node@c392aaca600c:/opt/node_app/app$ echo $MONGO_USERNAME_FILE
/run/secrets/mongo_root
node@c392aaca600c:/opt/node_app/app$ cat $MONGO_USERNAME_FILE
root

```

Figura 4 Valore di uno dei secret creati dallo script *startup-stack.sh* letti da un container

## CONFIGURAZIONE AMBIENTE DI DEPLOY SWARM

L'ambiente di deploy simulato per un deploy dell'applicazione in modalità Swarm è stato costruito come segue:

1. Una volta eseguita l'installazione di Multipass, come indicato precedentemente, è possibile creare delle macchine virtuali con il comando

```
multipass launch -n node1 -d 20G -m 512M && multipass launch -n node2 -d 20G -m 512M
```

Questo comando creerà due macchine virtuali, node1 e node2;

2. Viene successivamente effettuato il mount della cartella contenente il progetto ad un mountpoint del nodo1 con il comando:

```
multipass mount <path_to_project> node1:<mounting_path>
```

3. È possibile avviare una macchina virtuale creata tramite il comando:

```
multipass shell node1
```

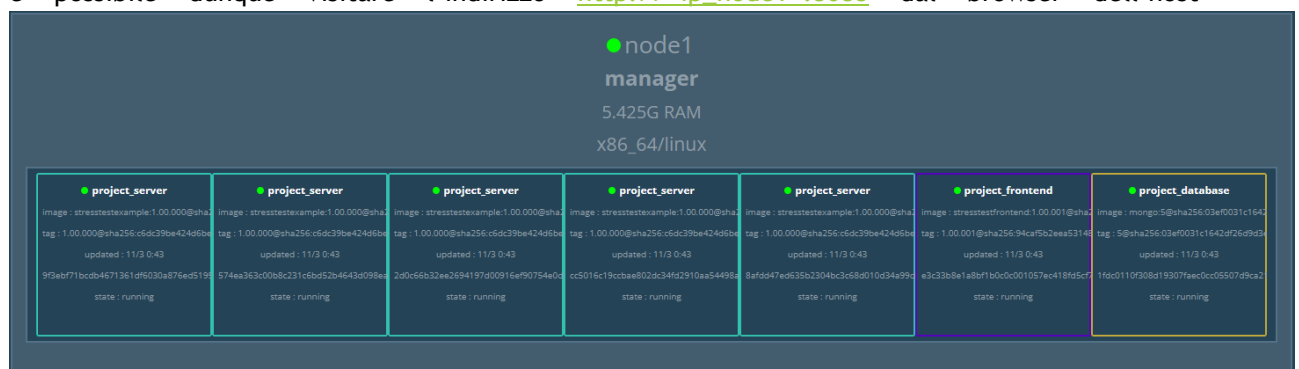
4. Spostandosi sul <mounting\_path><sup>2</sup> si avvia lo script ./utils/docker\_install\_linux.sh per installare Docker e Docker Compose;
5. Si inizializza lo Swarm con l'avvio dello *script\_stack.sh*, verrà stampato in console da un token di collegamento da incollare nella VM node2 creata precedentemente, questa viene avviata e configurata con le stesse modalità descritte nei primi 3 step;
6. Dalla VM node1 è possibile effettuare il deploy dello stack con il comando

```
docker stack deploy -c docker-stack.yml project
```

7. Il frontend sarà accessibile visitando su un browser dell'host uno degli indirizzi IP delle VM alla porta 80;
8. È possibile monitorare inoltre lo stato dello stack utilizzando [Docker Swarm Visualizer](#), che si può avviare utilizzando il seguente comando dal node1

```
docker run -it -d -p 8088:8080 -v /var/run/docker.sock:/var/run/docker.sock dockersamples/visualizer
```

è possibile dunque visitare l'indirizzo [http://<ip\\_node1>:8088](http://<ip_node1>:8088) dal browser dell'host



9. Sarà quindi possibile rimuovere lo stack con il comando

```
docker stack rm project
```

<sup>2</sup> Se si riscontrano problemi di esecuzione degli script è consigliabile copiare la cartella del progetto montata in un altro path ed eseguire un `chmod +x` sui file .sh interessati, questo è probabilmente dovuto ad un problema di ownership tra i file nella cartella montata e la VM

## CONFIGURAZIONE AMBIENTE DI DEPLOY KUBERNETES

La configurazione dell'ambiente di deploy segue si basa sulla creazione di una macchina virtuale, montaggio della cartella di progetto ed installazione di Docker così come illustrato nel capitolo precedente.

È possibile utilizzare il comando<sup>3</sup>

```
k apply -f ./k8s/
```

dalla root del progetto, in questo caso accesso.

È possibile quindi accedere alla pagina del progetto dal browser dell'host seguendo l'indirizzo [http://<multipass\\_VM\\_IP>:30446](http://<multipass_VM_IP>:30446).

È possibile avviare la Dashboard principale di Kubernetes, abilitata in fase di installazione di microk8s eseguendo i seguenti comandi:

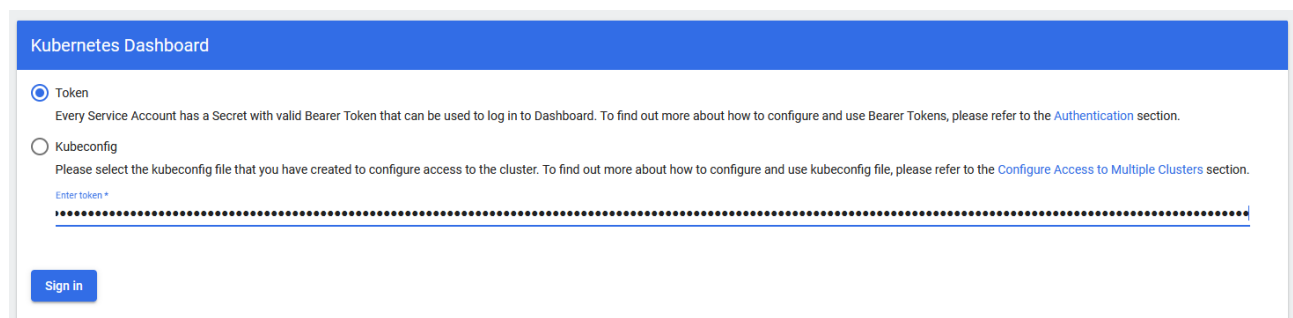
```
# Ottenere la porta su cui è esposta la dashboard con
k get service/kubernetes-dashboard -n kube-system

# Ottenere il token con
token=$(k -n kube-system get secret | grep default-token | cut -d " " -f1)
k -n kube-system describe secret $token

# effettuare il port-forwarding della porta 443 in modo da rendere la dashboard
accessibile anche dall'host

microk8s kubectl port-forward -n kube-system service/kubernetes-dashboard
10443:443 --address 0.0.0.0
```

Visitando il link [https://<multipass\\_VM\\_IP>:10443](https://<multipass_VM_IP>:10443) si potrà accedere alla dashboard utilizzando il token ottenuto in precedenza



Kubernetes Dashboard

☒ Token  
Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

☐ Kubeconfig  
Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

Enter token \*

.....

Sign In

<sup>3</sup> NB: “k” è un alias creato in fase di installazione di microk8s, nel caso in cui non sia stato applicato è possibile utilizzare kubectl integrato su microk8s con ‘microk8s.kubectl’

kubernetes

default
Search

+
🔔
👤

☰
Workloads

Workloads N

Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets

Service N

Ingresses
Services

Config and Storage

Config Maps N
Persistent Volume Claims N
Secrets N
Storage Classes

Cluster

Cluster Role Bindings

Deployments

Name	Namespace	Images
<span>●</span> <a href="#">server</a>	default	mbongiovanni94/stresstestexample:1.00.000
<span>●</span> <a href="#">database</a>	default	mongo:5

È infine possibile eliminare le risorse create utilizzando il comando:

```
k delete -f ./k8s/
```

## EKS

Come ultimo scenario presentato in questo progetto, è stato scelto di utilizzare la utility `eksctl` per effettuare un deploy dell'applicazione sul servizio Kubernetes offerto da AWS, tramite il servizio chiamato EKS, Elastic Kubernetes Service. Per questo scopo è stato creato un file di configurazione, `/eks_cluster.yml`, attraverso il comando

```
eksctl create cluster --name project-cluster --region us-west-2 --instance-  
types=m5.large --dry-run
```

È quindi possibile procedere al deploy<sup>4 5</sup>

```
eksctl create cluster -f eks_cluster.yml
```

Una volta completata la creazione del cluster sarà possibile utilizzare gli stessi comandi descritti nel capitolo relativo al deploy dell'applicazione sul cluster a singolo nodo fornito da Microk8s.

```
# ottenere le informazioni relative allo stack creato con  
kubectl get all -o wide  
  
# l'indirizzo pubblico assegnato al load balancer che servirà l'applicazione sarà  
visibile nella tabella relativa al servizio frontend  
  
# è possibile cancellare il cluster creato con il seguente comando  
eksctl delete cluster -f eks_cluster.yml
```

Nota, in questo scenario i comandi fanno riferimento alle configurazioni presenti all'interno della cartella `/kompose`, i comandi di deploy e remove diventano così

```
kubectl apply -f kompose/  
  
kubectl delete -f kompose/
```

---

<sup>4</sup> Questa operazione è relativamente lunga, la durata complessiva nei test sperimentati durante lo sviluppo del progetto arrivano in media tra i 25 - 30 minuti.

<sup>5</sup> Ndr. Questo deploy utilizza due istanze EC2 AWS di tipo `m5.large`, al momento della stesura di questo documento queste istanze **non** sono coperte dal tier di prova per la piattaforma AWS, l'esecuzione dei seguenti comandi potrebbe comportare un consumo di credito.

```
marco@DESKTOP-NFRFSL: /mnt/c/Users/marco/projects/uni/sistemi_cloud/docker_swarm_k8s_showcase$ kubectl get all -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
pod/database-66b7df5994-rrpr2	1/1	Running	0	14s	192.168.19.57	ip-192-168-7-166.us-west-2.compute.internal	<none>	<none>
pod/frontend-58c555c494-xqblm	1/1	Running	0	11s	192.168.16.6	ip-192-168-7-166.us-west-2.compute.internal	<none>	<none>
pod/server-7b7f8584c7-dzhqp	1/1	Running	0	7s	192.168.78.155	ip-192-168-77-49.us-west-2.compute.internal	<none>	<none>
pod/server-7b7f8584c7-hknfg	1/1	Running	0	7s	192.168.31.73	ip-192-168-7-166.us-west-2.compute.internal	<none>	<none>
pod/server-7b7f8584c7-p4xwk	1/1	Running	0	7s	192.168.6.229	ip-192-168-7-166.us-west-2.compute.internal	<none>	<none>
pod/server-7b7f8584c7-vktdc	1/1	Running	0	7s	192.168.6.18	ip-192-168-7-166.us-west-2.compute.internal	<none>	<none>
pod/server-7b7f8584c7-xzhrj	1/1	Running	0	7s	192.168.68.188	ip-192-168-77-49.us-west-2.compute.internal	<none>	<none>

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
service/database	ClusterIP	10.100.230.50	<none>	27017/TCP	15s	io.kompose.service=database
service/frontend	LoadBalancer	10.100.240.88	af9250831787c4749803c750b22509e0-1177198289.us-west-2.elb.amazonaws.com	80:38876/TCP	11s	io.kompose.service=frontend
service/kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	38m	<none>
service/server	ClusterIP	10.100.147.52	<none>	3800/TCP	9s	io.kompose.service=server

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
deployment.apps/database	1/1	1	1	19s	database	mongo:5	io.kompose.service=database
deployment.apps/frontend	1/1	1	1	16s	frontend	mbongiovanni94/stresstestfrontend:1.00.001	io.kompose.service=frontend
deployment.apps/server	5/5	5	5	12s	server	mbongiovanni94/stresstestexample:1.00.000	io.kompose.service=server

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/database-66b7df5994	1	1	1	20s	database	mongo:5	io.kompose.service=database,pod-template-hash=66b7df5994
replicaset.apps/frontend-58c555c494	1	1	1	17s	frontend	mbongiovanni94/stresstestfrontend:1.00.001	io.kompose.service=frontend,pod-template-hash=58c555c494
replicaset.apps/server-7b7f8584c7	5	5	5	13s	server	mbongiovanni94/stresstestexample:1.00.000	io.kompose.service=server,pod-template-hash=7b7f8584c7

Sarà possibile verificare lo stato del cluster dalla console EKS sul portale AWS

Cluster (1) Info					Elimina	Aggiungi cluster ▼
<input type="text" value="Filtra il cluster per nome, stato, versione kubernetes o provider"/>					< 1 >	
Nome del cluster	Stato	Versione Kubernetes	Provider			
<input type="radio"/> stresstest-cluster	Attivo	1.21	EKS			

