Progetto Sistemi Centrali di Gianluca Giardina, Aprile 2023

Confronto tra Docker e VirtualBox per la distribuzione di un'applicazione Spring Boot: un'analisi delle prestazioni e dell'utilizzo delle risorse

Abstract

Questo progetto si propone di confrontare l'utilizzo di Docker e VirtualBox per la distribuzione di un'applicazione di prenotazione tavoli al ristorante, sviluppata tramite il framework Spring Boot e utilizzando un server MySQL in locale. L'obiettivo è di analizzare le differenze tra le due tecnologie in termini di tempo di start e stop, utilizzo delle risorse, prestazioni sotto carichi di lavoro intensi e facilità di utilizzo e manutenzione. I risultati saranno analizzati al fine di determinare il miglior approccio per la distribuzione di applicazioni Spring Boot in diversi ambienti.

Introduzione

Con l'aumentare del numero di applicazioni, diventa spesso necessario distribuirle su una singola piattaforma in modo isolato.

Questo obiettivo può essere raggiunto tramite due approcci principali: la virtualizzazione e la containerizzazione.

La virtualizzazione permette di eseguire più sistemi operativi eterogenei sull'hardware di un singolo server fisico, condividendo agevolmente le risorse di CPU e memoria.

Questo possibile grazia all'hypervisor[1]:, il livello software che coordina le VM. Funge da interfaccia tra la VM e l'hardware fisico sottostante, garantendo che ciascuna abbia accesso alle risorse fisiche di cui ha bisogno per essere eseguita. Garantisce inoltre che le VM non interferiscano l'una con l'altra incidendo sullo spazio di memoria o sui cicli di calcolo reciproci.

Esistono due tipi di hypervisor

- **Hypervisor di tipo 1 o "bare-metal"** - interagiscono con le risorse fisiche sottostanti, sostituendo del tutto il sistema operativo tradizionale. Si presentano più comunemente in scenari di server virtuali.
- **Hypervisor di tipo 2** - vengono eseguiti come un'applicazione su un sistema operativo esistente. Più comunemente utilizzati sui dispositivi endpoint per eseguire sistemi operativi alternativi, comportano un sovraccarico delle prestazioni perché devono utilizzare il sistema operativo host per accedere alle risorse hardware sottostanti e coordinarle.

In questi ultimi anni, la containerizzazione ha acquisito sempre più importanza come tecnologia di virtualizzazione alternativa alla virtualizzazione tradizionale basata sull'hypervisor.

Un container è simile ad un'applicazione, che viene eseguita come un processo in cima al sistema operativo (OS) e si isola dalle altre eseguendo nello spazio degli indirizzi proprio. Tuttavia, rispetto ad un normale processo, un container non include solo l'eseguibile dell'applicazione, ma raggruppa anche tutti i software necessari per far funzionare l'applicazione, come librerie e altre dipendenze.

Cgroups e namespaces [2] sono due caratteristiche particolari che permettono a diversi container di eseguirsi sullo stesso server fisico. Cgroups, anche chiamati gruppi di controllo, Sono dei meccanismi del kernel di Linux per controllare l'allocazione delle risorse. Consente agli amministratori di sistema di allocare le risorse, come CPU, memoria, rete o qualsiasi loro combinazione, ai container in esecuzione. Le risorse assegnate ad ogni container possono essere regolate dinamicamente in modo da non poter utilizzare più risorse di quelle specificate nei cgroups.

Il namespace fornisce un'astrazione delle risorse del kernel, come gli ID dei processi, le interfacce di rete, i nomi host, in modo che ogni container sembri avere le proprie risorse isolate. Con i namespace, ad esempio, i processi in esecuzione in diversi container non entreranno mai in conflitto tra di loro, anche quando hanno lo stesso ID del processo.

Entrambe le tecnologie consentono in genere di eseguire più macchine su un singolo server, creando un'illusione di virtualizzazione.

L'obiettivo è quello di isolare le macchine tra loro e dal sistema host, la differenza sta nel metodo con cui ottenere questo isolamento.

La Figura 1 illustra la differenza tra container e macchine virtuali: i container vengono eseguiti sul sistema operativo principale, mentre le macchine virtuali vengono eseguite sull'hypervisor. Il motore di container è in genere combinato con il kernel del sistema operativo principale.

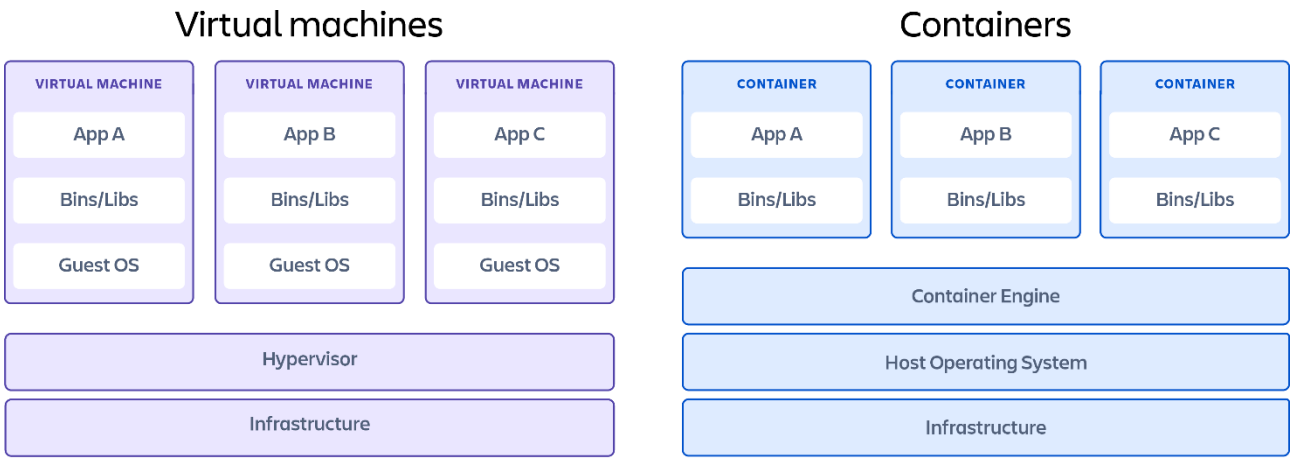


Figura. 1. Architecture comparison virtual machine v.s. Container

Inoltre, le due tecnologie possono essere confrontate in base a diversi fattori, come ad esempio:

Architettura host/guest: L'architettura host/guest delle macchine virtuali e dei contenitori presenta alcune differenze importanti. In particolare, le macchine virtuali consentono l'esecuzione di un kernel guest differente da quello dell'host. Ciò significa che l'utente può utilizzare un sistema operativo guest completamente differente da quello dell'host.

Al contrario, con i contenitori, il kernel è condiviso. Ciò significa che non è possibile eseguire un kernel guest diverso da quello dell'host.

Questo può rappresentare una limitazione in alcune situazioni, ad esempio quando si utilizzano applicazioni legacy che richiedono una versione specifica del kernel o quando si utilizzano applicazioni che richiedono funzionalità specifiche del sistema operativo

Avvio: Per avviare una macchina virtuale, è necessario prima accendere il sistema operativo dell'host.

Una volta acceso l'host, si può creare un'istanza della VM, questa istanza avvierà un sistema operativo completo, con tutte le applicazioni necessarie per eseguire l'applicazione desiderata, questo processo può richiedere un consumo di risorse importante anche in termini di tempo.

D'altra parte, l'avvio di un container inizia con la creazione di un'immagine. Questa immagine contiene solo le librerie e i file necessari per l'applicazione desiderata. Successivamente, è possibile eseguire il container su qualsiasi host che abbia un'architettura compatibile con quella dell'immagine.

Questo rende l'avvio di un container molto più veloce rispetto all'avvio di una VM.

S**tandardizzazione**: Le macchine virtuali sono come un sistema operativo standard completo con tutte le funzionalità. D'altra parte, i contenitori sono progettati per l'esecuzione di specifiche applicazioni e includono solo le librerie e i pacchetti necessari per l'applicazione stessa

Portabilità [6]: I container sono ambienti di esecuzione leggeri che consentono di isolare e distribuire applicazioni in maniera semplice ed efficiente. Essi comprendono tutti gli elementi necessari per l'esecuzione dell'applicazione stessa, da librerie a runtime specifici, senza la necessità di replicare l'intero sistema operativo. In questo modo, i container risultano essere molto rapidi e leggeri, ed offrono la possibilità di gestire efficacemente un gran numero di applicazioni simultaneamente.

Le VM, invece, sono degli ambienti completamente isolati all'interno dei quali è possibile eseguire un sistema operativo completo e l'intera gerarchia di applicazioni e servizi. Per questo motivo, le VM sono generalmente più pesanti dei container, ma al contempo offrono una maggiore flessibilità e compatibilità con differenti ambienti di esecuzione.

In definitiva, entrambe le soluzioni offrono un'elevata portabilità, ma la loro scelta dipende dalle esigenze specifiche dell'applicazione da distribuire. Se si cerca massima leggerezza ed efficienza, i container appaiono come la scelta migliore. Se invece si necessità di un ambiente completo e isolato, la VM rappresenta la soluzione più adeguata

I containers Docker

Docker [3] è una tecnologia di containerizzazione open-source che permette di creare e distribuire applicazioni in modo semplice e portabile. A differenza di altre tecnologie di container come LXC (Linux Containers), Docker si basa su una architettura client-server, in cui un server Docker gira su un host e permette di gestire i container.

La chiave per comprendere come Docker funziona è capire la differenza tra immagini e container. Un'immagine Docker è un pacchetto leggero e autonomo che contiene tutti i file necessari per eseguire un'applicazione. Questa immagine può essere costruita partendo da un file di configurazione Dockerfile che specifica le dipendenze dell'applicazione, le librerie e i file necessari per l'esecuzione. Un'immagine Docker può essere condivisa, archiviata e distribuita su un Docker Registry.

Un container Docker è un'istanza eseguibile di un'immagine Docker [4]. Quando viene eseguito, un container è isolato dal resto del sistema operativo, ma condivide il kernel con l'host. Ciò significa che un container ha il proprio sistema file e i propri processi, ma condivide il kernel con il sistema operativo sottostante. Questo rende i container Docker molto leggeri, efficienti e portabili.

Rispetto ad altre tecnologie di containerizzazione [5], Docker offre alcune caratteristiche uniche. Ad esempio, consente di definire le dipendenze dell'applicazione e le librerie richieste in un file di configurazione Dockerfile, semplificando il processo di creazione e distribuzione dell'applicazione.

Inoltre, Docker offre un'ampia gamma di strumenti per gestire i container, come Docker Compose per gestire l'orchestrazione di più container e Kubernetes per la gestione di cluster di container su larga scala.

VirtualBox

VirtualBox [7] è una soluzione di virtualizzazione open-source sviluppato da Oracle che permette di eseguire multiple macchine virtuali su un unico host. Si tratta di una tecnologia di virtualizzazione hardware, che consente l'esecuzione di sistemi operativi guest su una piattaforma host.

Una delle principali differenze di VirtualBox rispetto alle altre tecnologie di virtualizzazione è la sua capacità di supportare numerosi sistemi operativi, tra cui Windows, Linux, macOS, Solaris e altri. Inoltre, VirtualBox è caratterizzato da una grande flessibilità nella configurazione e personalizzazione delle macchine virtuali, consentendo ad ogni utente di impostare le specifiche esigenze di risorse, rete e storage per ogni singolo sistema guest.

Il processo di creazione di una macchina virtuale in VirtualBox inizia con la creazione di un'immagine virtuale, un file contenente tutti i dati necessari per eseguire il sistema operativo all'interno della macchina virtuale.

L'immagine può essere creata da zero, oppure può essere importata da una fonte esterna come una macchina virtuale già esistente. Una volta creata l'immagine, si può procedere alla creazione della macchina virtuale vera e propria, in cui si impostano le specifiche di risorse, memoria, rete, storage e altre opzioni di configurazione.

Una volta creata la macchina virtuale, è possibile eseguirla in VirtualBox, che crea un'istanza del sistema operativo guest all'interno della macchina virtuale, isolandolo dal sistema host. Ciò significa che i programmi e i dati del sistema operativo guest non possono interferire con quelli del sistema host.

Inoltre, la macchina virtuale può essere facilmente clonata o distribuita su altri sistemi, rendendo VirtualBox uno strumento molto utile per lo sviluppo e il testing di applicazioni.

Obiettivi del progetto

Il progetto in questione ha come obiettivo la comparazione tra l'utilizzo di Docker e VirtualBox per la distribuzione di un'applicazione di prenotazione tavoli al ristorante, sviluppata tramite il framework Spring Boot e utilizzando un server MySQL in locale.

Per analizzare le differenze tra le due tecnologie, si terranno in considerazione vari aspetti:

1. Tempo di start e stop: si prenderà in considerazione il tempo necessario per avviare e fermare l'applicazione utilizzando Docker e VirtualBox
2. Utilizzo delle risorse: si analizzerà come le due tecnologie gestiscono le risorse hardware, come CPU, memoria e spazio su disco dell'applicazione in condizioni normali e sotto carico. Si valuterà se ci sono differenze significative nel consumo di risorse tra le due tecnologie e si cercherà di comprendere i fattori che influenzano tale consumo, come il tipo di virtualizzazione utilizzata e il sistema operativo.
3. Performance: Verranno valutate le prestazioni dei sistemi sotto carichi di lavoro intensi (Utilizzando come criterio il numero di fallimenti per richieste inviate). Per testarne le prestazioni, verrà utilizzato uno strumento open source che permette di simulare un gran numero di utenti che interagiscono contemporaneamente con la webapp. I risultati verranno analizzati per determinare l'efficienza del sistema sotto stress e la sua capacità di gestire carichi di lavoro intensi.
4. Facilità di utilizzo e manutenzione: Saranno valutati i vantaggi e gli svantaggi di entrambe le opzioni per quanto riguarda la configurazione dell'ambiente di esecuzione e la distribuzione dell'applicazione su diversi ambienti, come quelli di sviluppo, test e produzione. In particolare, verrà preso in considerazione l'impatto dell'utilizzo di Docker sulla complessità della configurazione dell'ambiente di esecuzione e sui requisiti hardware e software, nonché la necessità di utilizzare una macchina a 64 bit e di prestare attenzione alla sicurezza.

Sviluppo Web Application

La webapp è stata sviluppata in SpringBoot, un framework per lo sviluppo di applicazioni web basato su Java, che permette l'inizializzazione rapida di un progetto con una configurazione preimpostata. [8]

L'obiettivo della webapp è quello di permettere la prenotazione dei tavoli un ristorante inserendo i propri dati personali e le informazioni relative alla prenotazione.

Per la memorizzazione dei dati è stato utilizzato JPA (Java Persistence API) che è una specifica che permette ai programmatori l'accesso e la gestione dei dati memorizzati in un database relazionale tramite oggetti Java.

In particolare il sistema di gestione di database relazionale (RDBMS) è MySQL.

Nel codice Java mostrato nelle seguenti porzioni di codice si utilizza un Repository chiamato ReservationRepository, che estende l'interfaccia JpaRepository.

Questa interfaccia fornisce una serie di metodi comuni per interagire con un database, come ad esempio la ricerca di tutte le prenotazioni presenti nel database tramite il metodo findAll(). Questi metodi permettono di gestire in modo semplice e organizzato i dati del progetto.

Di seguito è presente il codice del model Reservation:

```java
@Entity
@Table(name = "reservations")
public class Reservation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "email")
    private String email;

    @Column(name = "phone")
    private String phone;

    @Column(name = "date")
    private LocalDateTime date;

    @Column(name = "num_guests")
    private int numGuests;
```

```
    // costruttori, getter e setter

    public Reservation() {
    }

    public Reservation(String name, String email, String phone, LocalDateTime date,
int numGuests) {
        this.name = name;
        this.email = email;
        this.phone = phone;
        this.date = date;
        this.numGuests = numGuests;
    }
```

In questo model, ogni prenotazione è associata ad una riga della tabella "reservations" del database MySQL. La classe Reservation è annotata con "@Entity" e "@Table(name = "reservations")" per specificare che rappresenta un'entità del database e che è mappata sulla tabella "reservations" rispettivamente.

La classe ha un campo id generato automaticamente con l'annotazione "@Id" e "@GeneratedValue(strategy = GenerationType.IDENTITY)" che lo definisce come chiave primaria.

Ci sono anche cinque campi aggiuntivi che rappresentano i dati della prenotazione e sono mappati sulle colonne della tabella con l'annotazione "@Column(name = "nome_colonna")".

Di seguito è presente il codice del controller ReservationController:

```
@Controller
public class ReservationController {

    @Autowired
    private ReservationRepository reservationRepository;

    @GetMapping("/")
    public String index(Model model) {
        List<Reservation> reservations = reservationRepository.findAll();
        model.addAttribute("reservations", reservations);
        return "index";
    }

    @GetMapping("/reservation")
    public String showReservationForm(Model model) {
        model.addAttribute("reservation", new Reservation());
        return "reservation";
    }

    @PostMapping("/reservation")
    public String submitReservationForm(@ModelAttribute Reservation reservation,
Model model) {
        reservationRepository.save(reservation);
        model.addAttribute("message", "Reservation submitted successfully!");
        return "success";
    }

}
```

All'interno del controller package, il ReservationController fa uso delle funzionalità di JPA per accedere ai dati presenti nel database.

La funzione index(Model model) permette di visualizzare l'indice di tutte le prenotazioni effettuate tramite il metodo findAll() sulla repository di Reservation.

La funzione showReservationForm(Model model) permette di visualizzare il form per la prenotazione. Attraverso addAttribute viene passato all'html un nuovo oggetto Reservation.

La funzione submitReservationForm(@ModelAttribute Reservation reservation, Model model) viene chiamata una volta che l'utente ha scelto di inviare i dati del form. Anche in questo caso attraverso l'oggetto Model viene passato un messaggio di successo.

Nella figura 2 è riassunto il funzionamento di una prenotazione tramite richiesta POST, Spring Boot riceve la richiesta e mappa i dati in un oggetto Reservation che viene salvato nel database mediante il repository ReservationRepository.

Infine, la webapp restituisce una pagina di successo al browser.

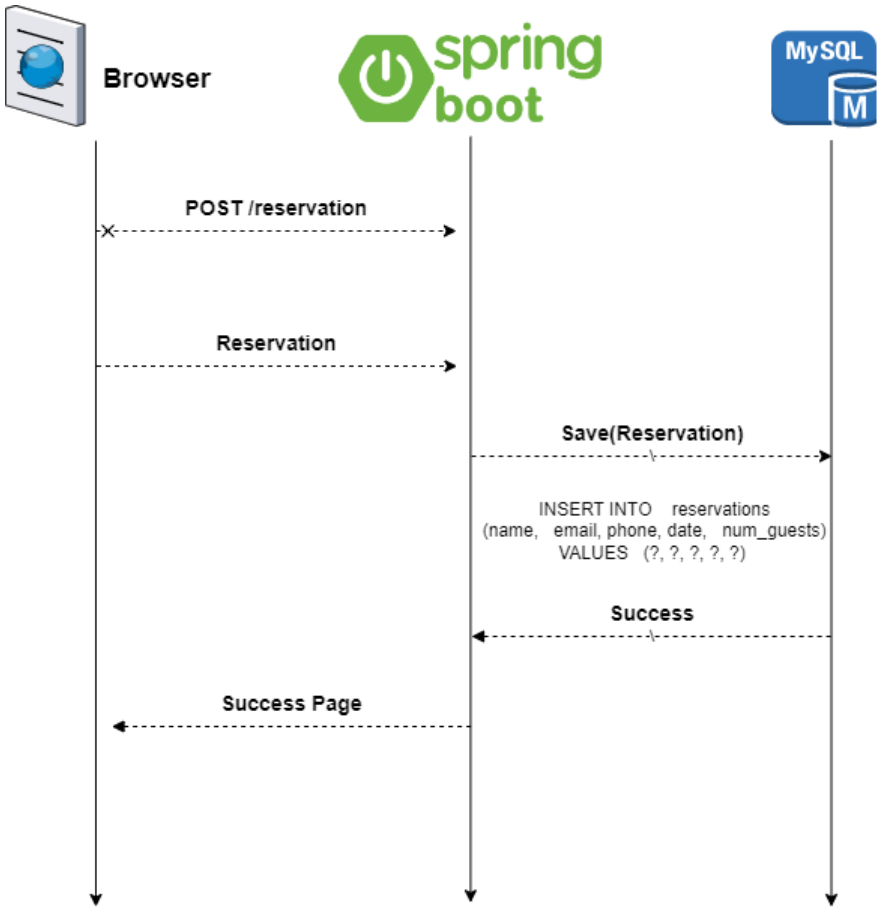


Figura. 2 ***Diagram of the Booking Process via POST Request with Spring Boot and Repository"***

Deploying dell'applicazione

Durante la fase di deployment, si deve trasferire il software sviluppato dall'ambiente di sviluppo a quello di produzione. In questa sezione, verra esaminato il procedimento di deploy della Web Application, che è stato eseguito su entrambe le piattaforme, ovvero una virtual machine VirtualBox e un container Docker. Verra illustrato anche le procedure di installazione dell'applicazione, indicando le specifiche tecniche dell'hardware e del software utilizzati per il progetto.

La prima operazione consiste nell'effettuare la build del progetto Spring Boot attraverso il comando "mvn package".

Questo comando del software Maven viene utilizzato per compilare e costruire un progetto Java, che comprende la compilazione del codice del progetto e la creazione di un file JAR. Questo file contiene il codice compilato e tutte le dipendenze necessarie per eseguire il progetto

Virtual Machine

Innanzitutto, l'applicazione viene testata con una macchina virtuale utilizzando nel sistema operativo host la distribuzione Linux Ubuntu. Le specifiche di sistema utilizzate:

- Hypervisor: Oracle VirtualBox versione 6.1.2 , Type-2 Hypervisor
- Sistema operativo guest: Ubuntu 22.10 LTS
- Spring Boot: versione 3.0
- Java: versione 17
- DBMS: mysql Ver 8.0.32

I passaggi necessari per effettuare il deploy dell'applicazione sulla vm sono stati i seguenti:

- Configurazione della VM ed installazione del sistema operativo guest

- Installazione e configurazione del DBMS mysql

- Creazione del database "restaurant" attraverso la query "create database restaurant"

-Clonare da git il repository contenente l'archivo jar

- Esecuzione della webapp attraverso il comando :

```
java -jar RestaurantApplication.jar --server.port=8080
        --spring.datasource.url=jdbc:mysql://localhost3306/restaurant
        --spring.datasource.username=root --spring.datasource.password=1234
```

La *Figura 3 Sintetizza i passaggi seguiti per il deploy*

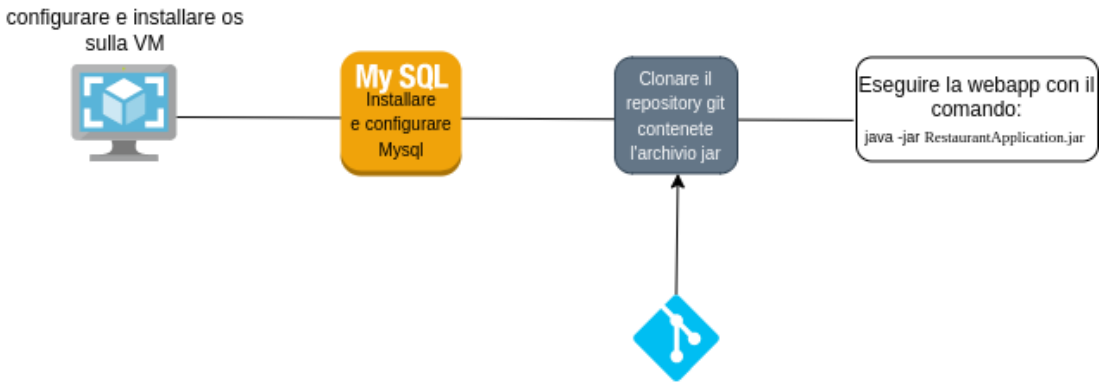


Figura. 3 ***Steps for Deploying the Application on VM.***

Docker

Per effettuare il deploy del sistema utilizzando docker, sono state utilizzate diverse immagini.

La creazione della prima immagine nominata “Backend” è stata eseguita attraverso il seguente DockerFile:

```
FROM openjdk:17-jdk-alpine AS build
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

La prima riga indica l'immagine openjdk:17-jdk-alpine, che verrà utilizzata per creare l'immagine del container.

La seconda istruzione definisce un argomento denominato JAR_FILE, che rappresenta il percorso del file JAR dell'applicazione.

Il comando COPY copia il file JAR specificato dall'argomento JAR_FILE all'interno del contenitore, rinominandolo come app.jar.

L'ultima riga specifica il comando che verrà eseguito quando il contenitore viene avviato. In questo caso, viene eseguita l'applicazione Java specificata tramite l'opzione –jar.

Successivamente è stato necessario creare un file configurazione di docker compose, un tool per definire e avviare differenti container contemporaneamente.

Il file **docker-compose.yml** descrive un ambiente di sviluppo utilizzato per il progetto.

La versione utilizzata è la 3 e il file è composto da due servizi: il server e il database.

Il servizio server utilizza l'immagine docker “**backend**” e viene esposto sulla porta 8080 per consentire la comunicazione con l'esterno. Vengono inoltre specificate delle variabili d'ambiente per collegarsi al database MySQL, come ad esempio l'URL e la porta del server.

Il servizio mysqldb utilizza l'immagine docker di MySQL 5.7 e viene creato automaticamente un database chiamato "restaurantdb". Vengono specificati anche username e password per accedere al database.

Infine, viene specificata una rete docker chiamata "springmysql-net" alla quale i due servizi appartengono e che consente la comunicazione tra di essi.

Questo permette di evitare l'utilizzo di indirizzi IP statici e semplifica la configurazione dell'ambiente.

Docker-compose.yml:

```
version: "3"
services:
  server:
    image: backend
    ports:
      - "8080:8080"
    environment:
      - spring.datasource.url=jdbc:mysql://mysqldb:3306/employeedb?useSSL=false
    networks:
      - springmysql-net
    depends_on:
      - mysqldb
  mysqldb:
    image: mysql:5.7
    networks:
      - springmysql-net
    environment:
      - MYSQL_ROOT_PASSWORD=1234
      - MYSQL_DATABASE=restaurantdb
      - MYSQL_USER=sa
      - MYSQL_PASSWORD=1234
networks:
  springmysql-net:
```

La figura ? Riassume il contenuto del docker-compose file

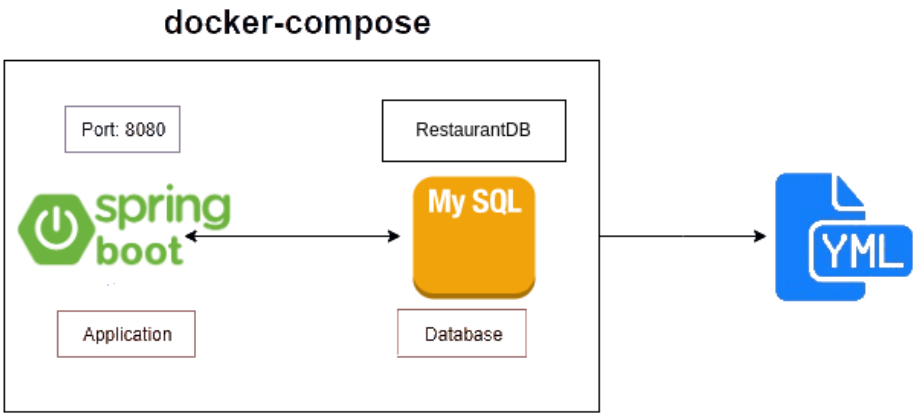


*Figura. 4 "**Summary of Docker-Compose Configuration File"***

Per quanto concerne l'esecuzione dei container, bisogna:

- clonare il repository git contenente l'immagine "backend" ed il file docker-compose.yml

-importare l'immagine "backend" con il comando "docker load < backend.tar"

-eseguire il comando docker-compose up

La figura 5 Schematizza i passaggi per l’esecuzione dei container

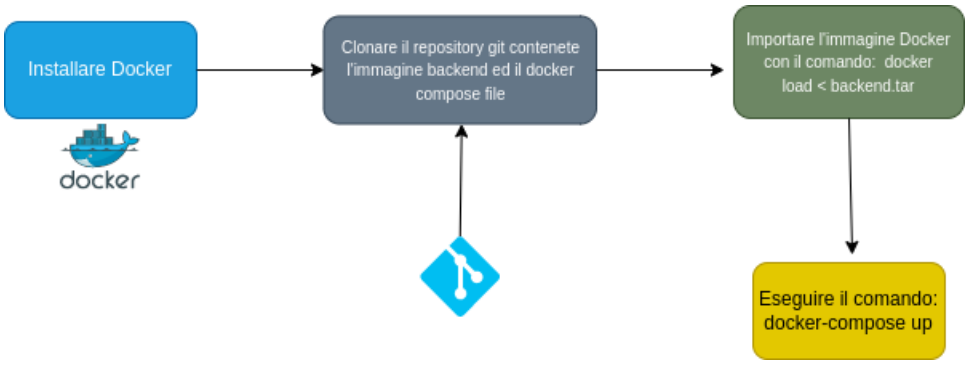


Figura. 5 ***Steps for Deploying the Application on Docker.***

Strumenti Utilizzati per il test

Per confrontare le prestazioni delle due soluzioni, è stata utilizzato una configurazione di test costituita da una macchina con processore AMD ryzen 3300x, 4 GB di RAM e un disco SSD da 240 GB.

Inoltre è stato utilizzato Il framework Locust per valutare le prestazioni delle soluzioni in esame.

Locust è un framework di testing open-source che consente di simulare il carico di lavoro di un grande numero di utenti su un'applicazione Web monitorando le prestazioni dell'applicazione mentre il carico di lavoro aumenta.[9]

Per poter misurare l’utilizzo della CPU da parte dei processi è stato sviluppato il tool **CpuTester** visionabile all’interno del repo GIT.

L'applicazione scritta in Java consiste in un monitor di utilizzo della CPU di un processo specifico.

In particolare, l'utente deve specificare il PID del processo che vuole monitorare all'interno del codice.

Il codice utilizza il comando "top" per recuperare informazioni sull'utilizzo della CPU del processo specificato. L'utilizzo della CPU viene memorizzato come un valore in virgola mobile.

Successivamente il valore viene stampato a video e scritto su un file di output specificato tramite la variabile "outputFilePath".

L'applicazione gira in un loop infinito, che effettua il monitoraggio ogni secondo, per mezzo di una sleep. Il valore di soglia per cui si stampa il valore dell'utilizzo della CPU del processo è rappresentato dall'istruzione "if(cpuUsage>1)".

In sintesi, l'applicazione offre la possibilità di monitorare l'utilizzo della CPU di un processo specifico e tenere traccia della sua evoluzione nel tempo.

Fase di Testing

Nelle seguenti sezioni, quando si farà riferimento alle macchine virtuali, si parlerà di quelle create con VirtualBox.

Tuttavia, è importante notare che i risultati ottenuti possono variare a seconda del software di virtualizzazione utilizzato e della configurazione dell'host. In ogni caso, i principi generali e le conclusioni tratte da questo studio rimangono validi indipendentemente dal software di virtualizzazione utilizzato.

Start and Stop time

In questa analisi, verrà valutato il tempo necessario per avviare e fermare una macchina virtuale contenente la webapp e un database MySQL, e il tempo per avviare e fermare due immagini Docker contenenti rispettivamente il servizio del database MySQL e la Web App Springboot.

Come mostrato nei risultati del test nella Figura 6, il tempo richiesto per avviare la macchina virtuale è di circa 30 secondi, mentre per fermarla sono necessari 18 secondi. Questo a causa del fatto che viene sottoposta a tutti i processi di inizializzazione del sistema operativo.

Invece, con l'utilizzo di due immagini Docker (attraverso un docker-compose), il tempo per avviare i servizi è notevolmente inferiore, in media di 45 ms per l'avvio e 32 ms per l'arresto.

In conclusione, è possibile osservare come la tecnologia dei container Docker sia una scelta migliore rispetto alle macchine virtuali, non solo relativamente al tempo di avvio e arresto del sistema, ma anche per la sua struttura leggera e modulare che consente di eseguire applicazioni con tempi di inattività ridotti, garantendo una migliore efficienza del sistema.

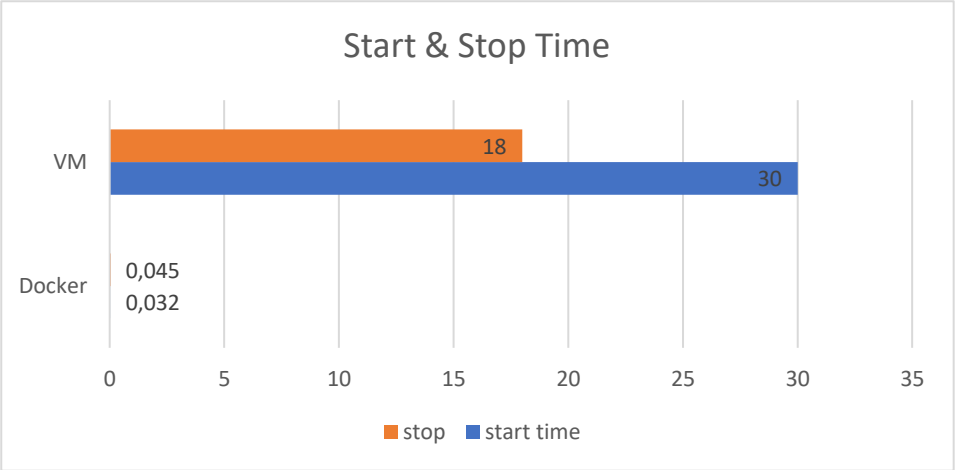


Figura. 6 ***" Analysis of startup and shutdown time for virtual machines and Docker images "***

Image Size

Le immagini delle macchine virtuali (VirtualBox) sono memorizzate come file vmdk. Questi file sono generalmente di grandi dimensioni, poiché contengono tutte le informazioni della macchina virtuale. Invece, le immagini Docker utilizzano il file system Union (noto anche come UnionFS) che consente ai file e alle directory di un sistema di file separato di essere formati in un singolo sistema di file coerente.

Di conseguenza, le immagini Docker occupano meno spazio di archiviazione.

Come mostrato nella *Figura 7,* l'immagine Docker per l'applicazione web SpringBoot presa in considerazione è più leggera dell'90% rispetto all'immagine della macchina virtuale. Pertanto, si può concludere dicendo che la tecnologia dei container Docker è la scelta migliore in termini di dimensioni dell'immagine.

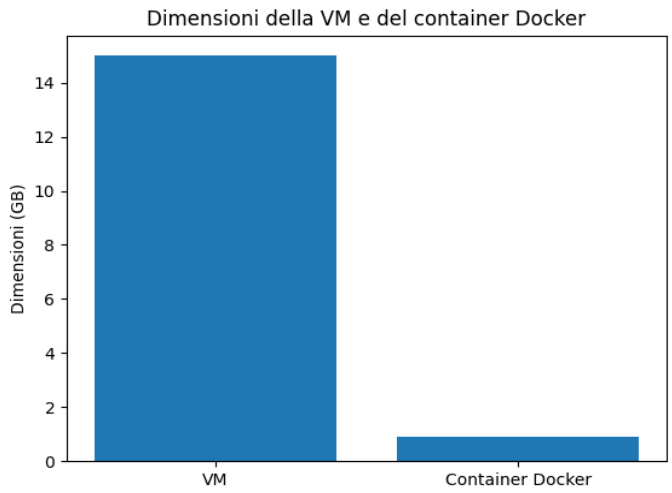


Figura. 7 ***" Comparison of Docker image and VirtualBox image size"***

Performance

In questa fase dello studio, sono state analizzate le prestazioni della webapp utilizzando il framework di testing delle prestazioni "Locust", uno strumento open source scritto in Python che permette di simulare un gran numero di utenti che interagiscono contemporaneamente con una webapp, al fine di testarne prestazioni sotto carichi di lavoro intensi.

La richiesta Post utilizzata per la simulazione è contenuta nello script myrequest.py.

Il carico è stato gradualmente aumentato di 5 richieste alla volta ogni secondo, fino a raggiungere il totale di 1500 richieste inviate simultaneamente alla fine del test al fine di simulare un elevato traffico sul server e testare le prestazioni del sistema sotto carico.

Durante la simulazione, sono state monitorate le prestazioni dei server attraverso i grafici generati da Locust.

In particolare, la *figura 8*, il numero di richieste inviate durante il test, mentre le figure *9 e 10* mostrano il tempo di risposta medio delle richieste (rps) e il numero di fallimenti.

Durante il test, sono stati eseguiti 1500 utenti virtuali che hanno effettuato richieste alla webapp. I risultati sono stati analizzati attraverso i report generati dallo strumento Locust, che mostrano il numero di richieste inviate, il tempo di risposta e il numero di fallimenti.

I risultati del test hanno mostrato che la webapp eseguita su un container Docker ha avuto un tempo di risposta inferiore rispetto a quella eseguita su una macchina virtuale. Inoltre, la webapp su Docker ha gestito con successo un maggior numero di richieste simultanee, rispetto a quella su macchina virtuale, dimostrando di essere più scalabile e in grado di gestire carichi di lavoro intensi, in modo più efficiente.

Figura. 8 ***"Number of users"***

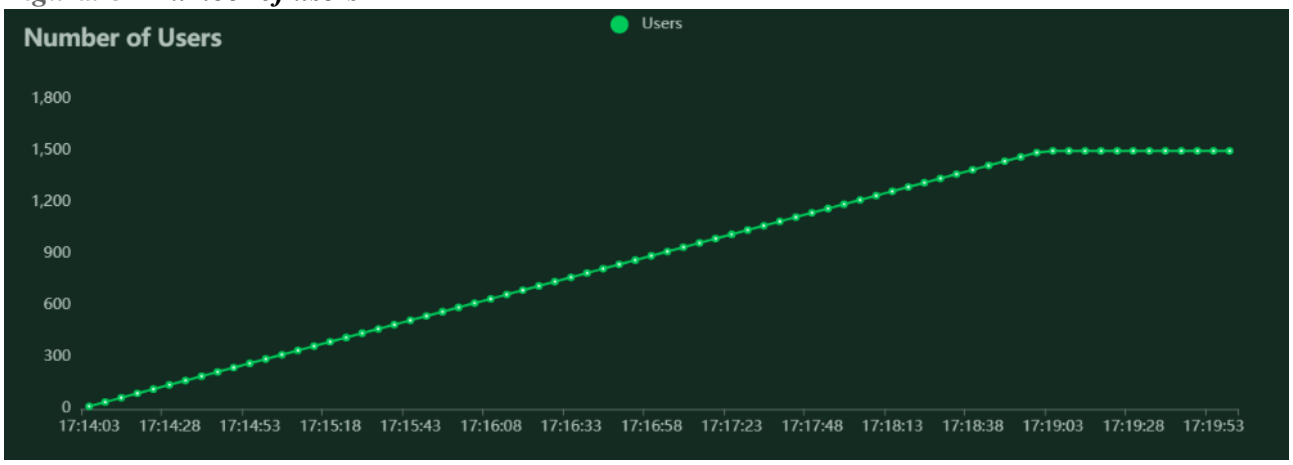


Figura.9 ***"Docker"***

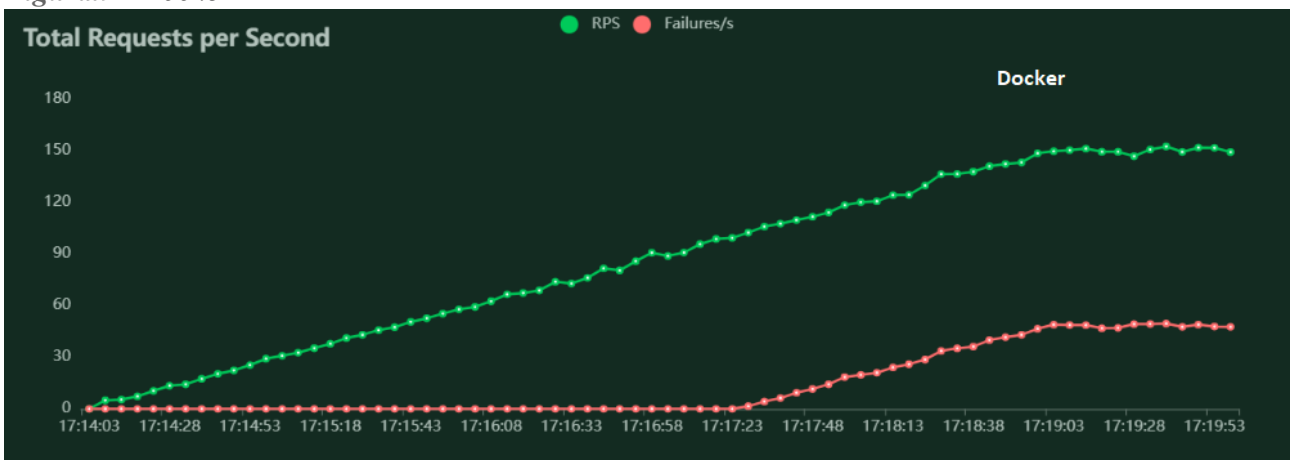
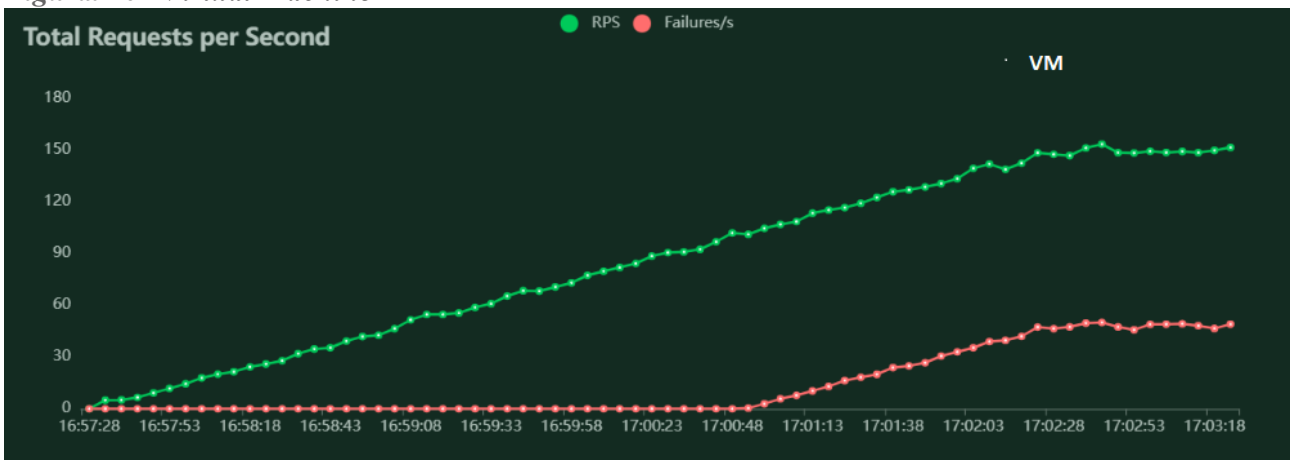


Figura. 10 ***"Virtual Machine"***



Per visionare i report in maniera dinamica e dettagliata, visionare i file html presenti nel repository git: report-docker.html e report-vm.html

CPU Performance

Per quanto concerne l'utilizzo della CPU, è stato utilizzato il tool java CpuTester, attraverso il quale è stato possibile monitorarne l'utilizzo.

Dai risultati ottenuti (*Figura 9 e 10*), è stato osservato che in condizioni normali, senza carichi di lavoro intensi, entrambe le soluzioni hanno un consumo di risorse simile. Tuttavia, quando il sistema è stato sottoposto a un numero crescente di richieste, l'esecuzione dell'applicazione su docker è risultata più efficiente, con un minor utilizzo della CPU, con picchi di utilizzo del 50% per Docker e del 60% per le macchine virtuali.

Ciò significa che se l'applicazione non richiede una gestione di carichi di lavoro intensi, l'utilizzo di container Docker o di macchine virtuali può essere una scelta meno rilevante in termini di prestazioni.

Dimostrando quindi che l'approccio dei container Docker può essere una scelta migliore rispetto alle macchine virtuali per le applicazioni che richiedono una maggiore scalabilità e capacità di gestione di carichi di lavoro intensi.

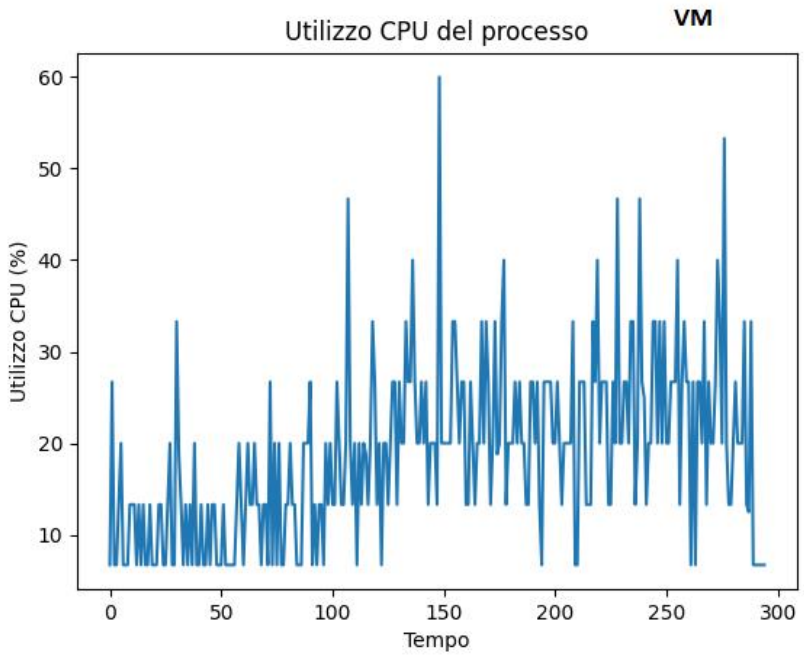


*Figura. 11 "**Cpu usage Virtual Machine"***

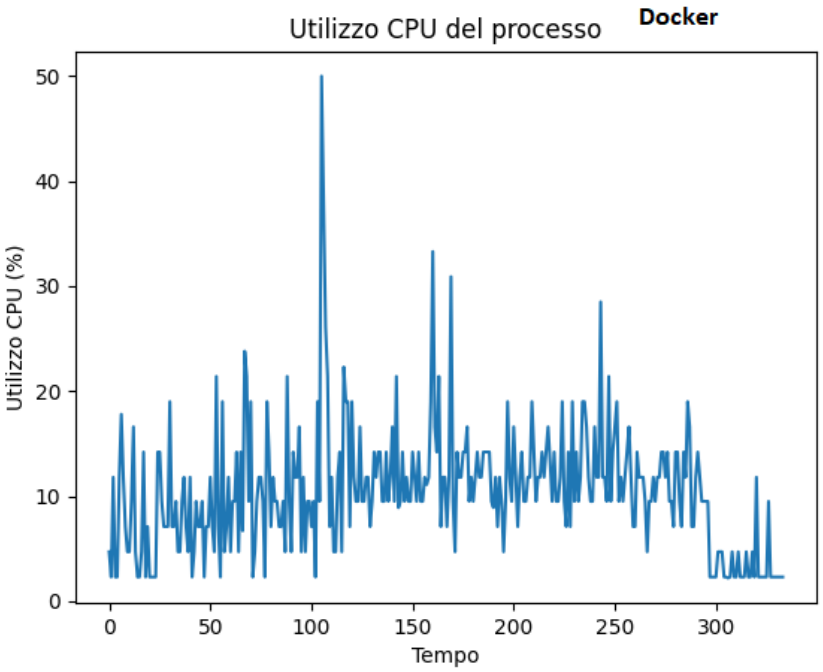


Figura. 12 ***"Cpu Usage Docker"***

Facilità di utilizzo e manutenzione

Inoltre, la distribuzione dell'applicazione su diversi ambienti risulta più agevole, poiché l'immagine Docker contiene tutti i componenti necessari all'applicazione e può essere facilmente spostata da un ambiente all'altro.

Tuttavia, come sottolineato, l'utilizzo di Docker richiede una configurazione più dettagliata rispetto alle macchine virtuali, soprattutto nella definizione delle risorse hardware e software. Inoltre, se si desidera eseguire un contenitore basato su un sistema operativo diverso, è necessario utilizzare una macchina differente.

Infine, come già accennato, è importante adottare pratiche di sicurezza consigliate nella documentazione Docker per minimizzare i rischi di vulnerabilità legati alla condivisione del kernel del sistema operativo dell'host.

In sintesi, l'utilizzo di Docker rappresenta una scelta più facile da gestire rispetto alle macchine virtuali, grazie alla sua flessibilità e capacità di semplificare la distribuzione dell'applicazione su diversi ambienti, a discapito però di una maggiore attenzione nella configurazione dell'ambiente di esecuzione.

Conclusioni

In generale, il confronto tra l'utilizzo di Docker e delle Virtual Machine Virtualbox per distribuire un'applicazione web ha evidenziato alcuni vantaggi e svantaggi in entrambe le soluzioni.

In questa analisi sono stati valutati i tempi necessari per avviare e fermare una macchina virtuale contenente una web app e un database MySQL, e il tempo per avviare e fermare due immagini Docker contenenti rispettivamente il servizio del database MySQL e la web app Springboot.

Dalle rilevazioni è emerso che l'utilizzo dei container Docker è preferibile rispetto alle macchine virtuali, non solo per il tempo di avvio e arresto del sistema, ma anche per la sua struttura leggera e modulare che consente di eseguire applicazioni con tempi di inattività ridotti, garantendo una migliore efficienza del sistema.

È stato anche evidenziato come le immagini delle macchine virtuali occupino più spazio di archiviazione rispetto alle immagini Docker, grazie all'utilizzo del file system Union, che consente alle directory di un sistema di file separato di essere formate in un singolo sistema di file coerente. Inoltre, sono stati analizzati i test delle prestazioni della web app utilizzando il framework di testing delle prestazioni "Locust", evidenziando come la web app su Docker abbia gestito con successo un maggior numero di richieste simultanee, rispetto a quella su macchina virtuale, dimostrando di essere più scalabile e in grado di gestire carichi di lavoro intensi.

Per quanto riguarda l'utilizzo della CPU, i risultati hanno dimostrato che l'utilizzo dei container Docker è preferibile rispetto alle macchine virtuali in termini di efficienza.

Di conseguenza sotto tutti i punti analizzati e dettagliati in questo progetto, Docker si è rilevato essere la scelta migliore.

Ma esistono contesti in cui l'uso di VirtualBox risulta essere la scelta preferibile rispetto ai container, si possono prendere in considerazione i seguenti esempi[10]:

- Ospitare workload legacy e monolitici: le applicazioni legacy e monolitiche spesso richiedono un ambiente operativo completo, che le VM possono fornire.
- Isolare cicli di sviluppo rischiosi: se si vuole isolare un ciclo di sviluppo rischioso da altre parti dell'infrastruttura, le VM possono fornire un isolamento completo.
- Fornire risorse infrastrutturali: le VM possono essere utilizzate per fornire risorse infrastrutturali, come reti, server e dati. Ad esempio, permettono di creare un ambiente di sviluppo e test completamente isolato.

- Eseguire un sistema operativo diverso all'interno di un altro sistema operativo: le VM consentono di eseguire un sistema operativo diverso da quello del sistema host, consentendo di eseguire applicazioni che richiedono un sistema operativo specifico. Ad esempio, è possibile eseguire un sistema operativo Unix all'interno di un sistema operativo Linux.

In conclusione, l'utilizzo di Docker è sicuramente più vantaggioso rispetto alle macchine virtuali VirtualBox per l'esecuzione di applicazioni web. Tuttavia, è importante considerare il contesto specifico in cui le VM possono fornire una soluzione più adatta alle esigenze dell'infrastruttura IT.

Il confronto tra queste due soluzioni evidenzia come Docker sia una tecnologia avanzata e moderna che offre una maggiore efficienza, scalabilità, modularità e leggerezza rispetto alle macchine virtuali, che restano un'alternativa valida per situazioni specifiche.

La scelta finale, quindi, deve tener conto di diversi fattori, tra cui la natura dell'applicazione, la complessità dell'infrastruttura e le esigenze specifiche dell'azienda.

Bibliografia

Repository GitHub: https://github.com/GianlucaGiardina/SistemiCentraliProgetto

[1] "Type 1 hypervisor vs. Type 2 hypervisor: What's the difference?", Red Hat, https://www.redhat.com/en/topics/virtualization/type-1-hypervisor-vs-type-2-hypervisor
[2] https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/
https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms
[3]"Docker Overview." Docker. https://www.docker.com/resources/what-container
[4]"Dockerfile reference." Docker. https://docs.docker.com/engine/reference/builder/
[5] "Why Docker?", Docker Documentation, https://docs.docker.com/get-started/overview/#why-docker
[6] https://www.redhat.com/en/topics/containers/containers-vs-vms
[7] Oracle, "VirtualBox Overview", https://www.virtualbox.org/manual/ch01.html
[8] https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started
[9] https://docs.locust.io/en/stable/what-is-locust.html