



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

MODELLI ARCHITETTURALI PER LO SVILUPPO SOFTWARE: L'APPROCCIO BASATO SUI MICROSERVIZI

RELATORE

Prof. Rita Francese

CANDIDATO

Lorenzo Paolo Cocchinone

Matricola: 0512105326

Università degli Studi di Salerno

Anno Accademico 2021-2022

Try again, I'm wishin' you well!
But you will never roast me better than I roast myself
Mori Calliope.

Il tradizionale approccio allo sviluppo software prevede che tutte le componenti del sistema siano integrate in un'unica grande unità, tale metodologia di sviluppo è conosciuta col nome di architettura monolitica. Un software monolitico è logicamente autonomo, in quanto i suoi componenti sono fortemente accoppiati e interdipendenti.

Negli ultimi anni si è cercato di realizzare un approccio architetturale che non rendesse i componenti di un sistema fortemente accoppiati, un vantaggio molto prezioso considerando che le applicazioni di oggi sono sempre più vaste e ricche di funzionalità, l'architettura a microservizi prevede tale vantaggio e non solo.

Un sistema costruito con l'approccio ai microservizi è suddiviso in piccole componenti indipendenti che lavorano insieme per fornire funzionalità complesse. Ciascun microservizio è responsabile di una specifica parte dell'applicazione e può essere sviluppato, testato e distribuito in modo indipendente dal resto dell'applicativo.

In contrasto con l'architettura monolitica, l'utilizzo dei microservizi permette di avere maggiore flessibilità e scalabilità nello sviluppo delle applicazioni. I microservizi rendono più facile la manutenzione dell'applicazione, poiché è possibile modificare, sostituire o aggiungere un singolo microservizio senza dover modificare più parti dell'applicazione, riducendo in modo significativo il rischio di creare regressione all'interno del codice.

Lo scopo di questo lavoro di tesi è quello di esplorare e descrivere diverse tecnologie al fine di riuscire a produrre un'applicazione basata sui microservizi.

Elenco delle figure	iv
Elenco delle tabelle	v
Elenco delle abbreviazioni	vi
1 Introduzione	1
1.1 Modelli architetturali	1
1.1.1 Architettura Monolitica	2
1.1.2 Architettura su più livelli	3
1.1.3 Architettura guidata dagli eventi	3
1.1.4 Architettura basata sui Microservizi	4
1.2 Obiettivi	5
1.3 Struttura della Tesi	5
2 Concetti Teorici	6
2.1 Introduzione	6
2.2 DevOps	6
2.2.1 DevOps e Microservizi	6
2.3 Container Linux	7
2.3.1 Macchine Virtuali	8
2.3.2 Immagine di un container	9
2.3.3 Orchestrazione dei Container	10

2.4	Lato backend di un applicativo	11
2.4.1	Gestione delle dipendenze	11
3	Tecnologie nel dettaglio	12
3.1	Introduzione	12
3.2	Gestione dei container: Docker	12
3.2.1	L'architettura di Docker	13
3.3	Orchestrazione dei Container: Kubernetes	15
3.3.1	Orchestrazione in locale: minikube	17
3.4	Gestione del Progetto: Apache Maven	17
3.4.1	Un esempio di POM	18
3.5	Framework: Spring Boot	19
4	Hello World	22
4.1	Introduzione	22
4.1.1	Pom di Spring Boot	22
4.1.2	Struttura dei servizi	23
4.2	Architettura	23
4.2.1	Eureka Discovery Service	24
4.2.2	Gateway Service	26
4.2.3	Avvio dei servizi	27
4.3	Aggiunta di un nuovo servizio	28
4.3.1	World Service	28
4.3.2	Modifiche	29
5	Conclusioni	31
	Ringraziamenti	32

Elenco delle figure

1.1	Architettura Monolitica	2
1.2	Pattern Model-View-Controll	3
1.3	Architettura guidata dagli eventi	4
1.4	Architettura a microservizi	4
2.1	Architettura per la gestione dei container	8
2.2	Architettura per la gestione di macchine virtuali	9
2.3	Architettura con software per l'orchestrazione dei container	10
3.1	Architettura di Docker	13
3.2	Persistenza dei dati in Docker	14
3.3	Cluster Kubernetes	15
4.1	Struttura per i progetti in Spring Boot	23
4.2	Architettura del software	24
4.3	Informazioni generali sul servizio Eureka	25
4.4	Informazioni sui microservizi sottoscritti ad Eureka	26
4.5	Funzionamento di Spring Cloud Gateway	26
4.6	Architettura modificata con l'aggiunta di World Service	28
4.7	Diagramma del Database World	29
4.8	World Service è registrato su Eureka	30

Elenco delle tabelle

2.1	Strumenti per l'automazione e CI/CD	7
2.2	Strumenti per la configurazione, testing e monitoraggio	7
2.3	Framework associato al proprio linguaggio di programmazione	11

Elenco delle Abbreviazioni

MVC Model-View-Control	3
IOT Internet of Things	3
JSON JavaScript Object Notation	17
POM Project Object Model	18
XML eXtensible Markup Language	18
JWT JSON Web Token	20
WAR Web Application Archive	21
URL Uniform Resource Locator	24

In questo capitolo affronteremo l'evoluzione dei modelli architetturali utilizzati nello sviluppo di software per dare al lettore un'idea generale sull'argomento trattato in questo lavoro di tesi.

1.1 Modelli architetturali

Nel campo dell'ingegneria del software sono descritte moltissime pratiche da seguire per poter realizzare un software nel miglior modo possibile, ottimizzando costi, tempo e risorse. Col tempo i software sono diventati sempre più massivi e complessi da sviluppare.

L'evoluzione tecnologica, in particolare il passaggio al World Wide Web 2.0 nei primi anni del 2000, ha reso possibile creare e rendere disponibili applicativi software ad una platea di persone sempre maggiore. Le nuove tecnologie per lo sviluppo di applicativi web hanno fatto sì che la creazione di software complessi diventi più gestibile, a discapito di una curva di apprendimento degli strumenti utilizzati più ripida, basti pensare che ormai JavaScript nella sua forma base viene utilizzato molto poco, gli sviluppatori tendono ad utilizzare un framework per agevolare il lavoro.

In tutto questo anche i modelli utilizzati in precedenza come riferimento per la creazione di software dovevano evolversi e adeguarsi ai nuovi strumenti di sviluppo. Il vecchio e caro modello monolitico ha iniziato a dare problemi, poi con l'esplosione

degli ultimi anni del cloud computing si è osservato che tale modello ormai fosse diventato inutilizzabile.

1.1.1 Architettura Monolitica

L'architettura monolitica [1] è stata uno dei primi approcci alla creazione di applicazioni. Tale architettura non solo è semplice da applicare, ma rende lo sviluppo anche più lineare. Un problema di tale approccio risiede nella fase di manutenzione del prodotto, aggiornare per esempio un servizio richiede di cambiare intere porzioni di codice sorgente all'interno dell'applicazione. Per questo motivo software basati su questo modello vengono aggiornati un massimo di due o tre volte l'anno.

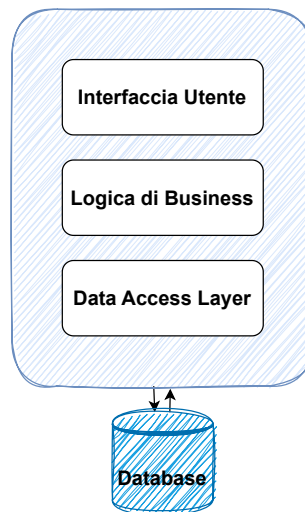


Figura 1.1: Architettura Monolitica

Tale approccio nell'epoca moderna è diventato obsoleto, non è più possibile rilasciare pochi aggiornamenti annuali, che spesso risultano essere solo lavori di manutenzione, con la crescente competizione nel settore, non sviluppare ed integrare nuove funzionalità tende a far perdere interesse verso l'applicativo. L'idea di avere un'applicazione coesa in alcuni casi impone delle limitazioni non indifferenti, per esempio se il servizio di registrazione di un e-commerce non è disponibile per qualche motivo, il resto dell'applicativo deve continuare a funzionare, gli utenti devono poter cercare prodotti da acquistare e se già registrati, anche di poterli acquistare.

Alcuni degli approcci moderni allo sviluppo di applicazioni sono i seguenti:

1. Architettura su più livelli;
2. Architettura guidata dagli eventi;

3. Architettura basata sui Microservizi;

1.1.2 Architettura su più livelli

L'architettura su più livelli ha una struttura tradizionale che permette di dividere l'applicazione in diversi livelli che comunicano tra di loro. Un esempio in letteratura è il pattern Model-View-Control (MVC), basato su tre livelli ed utilizzato spesso per lo sviluppo di applicazioni per il web.

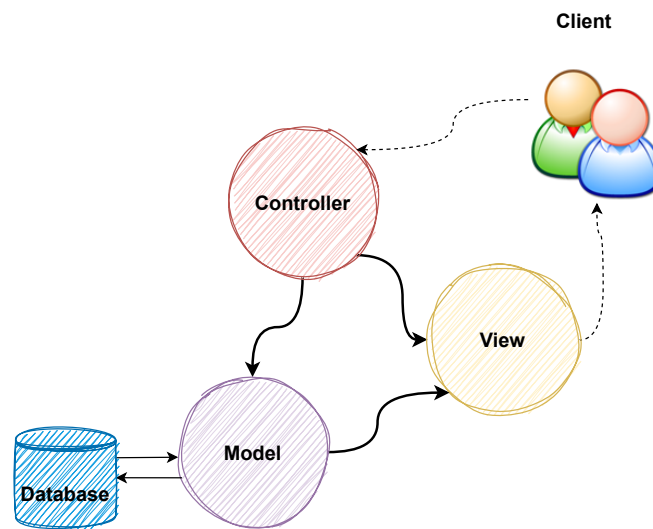


Figura 1.2: Pattern Model-View-Controll

La suddivisione in livelli permette di semplificare la gestione delle dipendenze e l'esecuzione di funzioni logiche.

1.1.3 Architettura guidata dagli eventi

L'architettura guidata dagli eventi [2] è uno stile architetturale dove l'acquisizione, la comunicazione, l'elaborazione e la persistenza degli eventi costituiscono i pilastri portanti della soluzione.

Il fulcro di tale architettura sono gli eventi, per evento si intende qualsiasi avvenimento che porta ad un cambiamento di stato del software o dell'hardware.

L'adozione di tale architettura per la propria applicazione permette di creare un sistema più flessibile che può trasformarsi in base agli eventi, anche in tempo reale. Questo tipo di architettura è il cuore pulsante delle applicazioni sviluppate per l'Internet of Things (IOT) che basa le proprie operazioni sull'acquisizione di dati del

mondo reale, come per esempio la temperatura di una stanza o dell'umidità nell'aria, il cambiamento di un parametro che viene monitorato, porta il sistema a reagire di conseguenza.

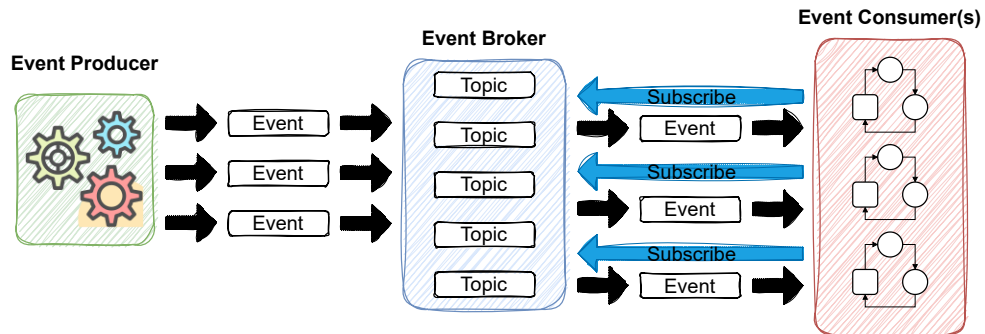


Figura 1.3: Architettura guidata dagli eventi

1.1.4 Architettura basata sui Microservizi

L'architettura a microservizi descrive un approccio allo sviluppo di software. Tale modello permette di scomporre le applicazioni in diversi componenti, chiamati appunto microservizi, che risultano essere indipendenti l'uno dall'altro. Ogni componente del sistema si occuperà di servire una singola funzionalità.

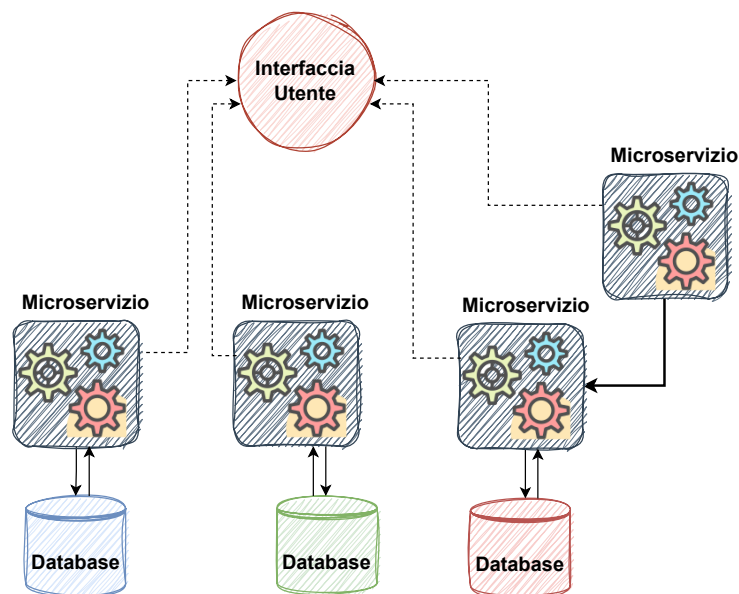


Figura 1.4: Architettura a microservizi

Creando un sistema che si basa sull'approccio ai microservizi, rende la fase di manutenzione dell'applicativo più semplice, permettendo di aggiornare servizi già

presenti o di sviluppare e rilasciare nuove funzionalità in un breve lasso di tempo rispetto, per esempio, all'approccio monolitico, che ricordiamo prevede pochi aggiornamenti annuali. Quello di cui abbiamo appena discusso non è l'unico vantaggio, la scalabilità dell'applicazione diventa dinamica, permettendo di gestire in autonomia il carico di lavoro. Il problema del singolo punto di vulnerabilità (Single Point of Failure) viene risolto dalla indipendenza dei microservizi, quando un componente non è disponibile per manutenzione, anomalie o altro, il sistema continuerà ad essere raggiungibile ed utilizzabile dagli utenti.

1.2 Obiettivi

L'obiettivo di questo lavoro di tesi è quello di esplorare diverse tecnologie che negli ultimi anni si stanno diffondendo, per sviluppare una semplicissima applicazione basata sull'architettura a microservizi, una sorta di Hello World che rappresenta i canoni del modello.

1.3 Struttura della Tesi

La tesi è divisa in quattro capitoli, quello appena concluso ha permesso al lettore di avere una panoramica sull'evoluzione delle architetture utilizzate nello sviluppo di software e del perché è stato necessario allontanarsi dal classico approccio monolitico. Nel prossimo capitolo andremo ad analizzare e ad esplorare alcuni concetti teorici su cui le tecnologie utilizzate nello sviluppo odierno si basano. Il capitolo tre presenterà le tecnologie che sono state utilizzate per la creazione del software e infine, nel capitolo quattro ci concentreremo sul software sviluppato e sui punti di forza.

Il codice sorgente dell'applicazione è disponibile al seguente link: <https://github.com/laurus96/Microservices-Example>

2.1 Introduzione

In questo capitolo vengono illustrati i concetti teorici su cui alcune tecnologie, utilizzate per la realizzazione dell'hello world, si basano.

2.2 DevOps

2.2.1 DevOps e Microservizi

Come discusso nel precedente capitolo, i microservizi sono un approccio architetturale che permette di frammentare il software in sviluppo, in piccole componenti che hanno il compito di fornire una singola attività. L'approccio DevOps [3] fornisce gli strumenti necessari per la gestione, lo sviluppo e la distribuzione delle applicazioni basate sui microservizi.

Data la natura dell'architettura che stiamo presentando non tutte le componenti devono essere sviluppate o integrate da noi, è possibile fare affidamento ai servizi cloud, più in generale si tende a non sviluppare componenti che magari sono già disponibili e pronti all'uso, sia per minimizzare il tempo di sviluppo, sia per concentrare tutte le forze del team sulle componenti principali del software. Fare affidamento ad un servizio cloud permette anche di semplificare la manutenzione della componen-

te che vogliamo integrare nel nostro sistema, perché non è più compito nostro curare il servizio.

Ogni servizio però, mantiene le proprie dipendenze e questo crea un problema di integrazione, grazie alla metodologia DevOps sono stati resi disponibili diversi software per far fronte a questa problematica. Oltre al consueto utilizzo di Git per il controllo della versione, di seguito sono riportati alcuni software più diffusi e più utilizzati in questo ambito [4]:

Automazione	Continuous Integration/Continuous Delivery (CI/CD)
Jenkins	CircleCI
Docker	Bamboo
Puppet	TeamCity
Apache Maven	Travis CI
Gradle	Buddy

Tabella 2.1: Strumenti per l'automazione e CI/CD

Configurazione	Testing	Monitoraggio
Chef	Selenium	Nagios
Kubernetes	Tricentis Tosca	Prometheus
Ansible	TestSigma	New Relic
Vagrant	IBM Rational Functional	PagerDuty
Consul	SoapUI	Sensu
Terraform		Splunk
		ELK Stack

Tabella 2.2: Strumenti per la configurazione, testing e monitoraggio

2.3 Container Linux

I Container [5] sono una tecnologia già apparsa nei primi anni 2000. Nel sistema operativo FreeBSD era presente una funzione chiamata jail, prigione, ed era uno strumento per isolare una porzione del sistema operativo. Tale tecnologia è stata

implementata in GNU/Linux dal 2001 e con le varie implementazioni e le varie migliorie sono nati i container, degli spazi del sistema operativo sicuri e controllati.

In Linux il loro funzionamento viene gestito dai cgroups¹, strumenti che operano a livello kernel e che permettono di limitare le risorse utilizzate da un processo o da un gruppo di processi. I gruppi di controllo usano systemd che permette di inizializzare i processi ed isolarli dal resto del sistema operativo.

Attualmente i container sono un approccio utilizzato sia in ambito di sviluppo che in ambito di rilascio e sono nati diversi software per la loro gestione come Linux Container Daemon (comunemente chiamato LXD), Podman e Docker.

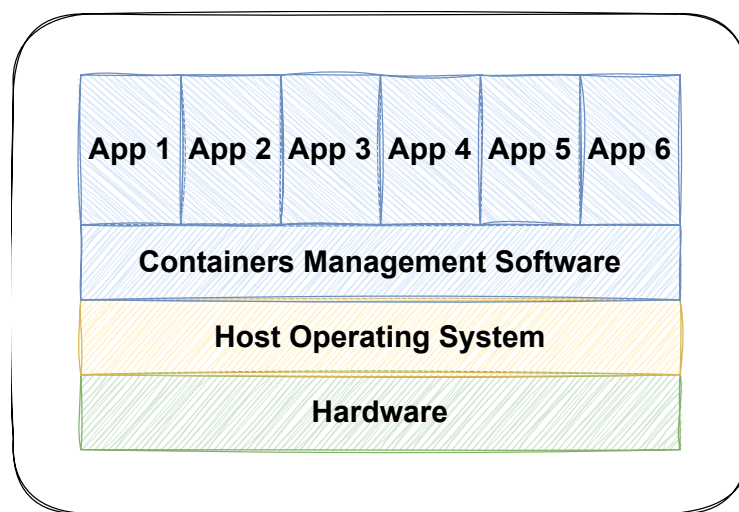


Figura 2.1: Architettura per la gestione dei container

2.3.1 Macchine Virtuali

I container vengono molto spesso scambiati per una tipologia di macchina virtuale, questo avviene in maniera errata, perché come abbiamo visto nel paragrafo precedente, i container rappresentano una porzione del sistema operativo creata e gestita da strumenti che lavorano a livello kernel del sistema operativo GNU/Linux.

Per definizione invece una macchina virtuale crea un ambiente virtuale dove si cerca di emulare il comportamento di una macchina fisica, quindi si cerca non solo di emulare il funzionamento del sistema operativo, ma anche dell'hardware sottostante.

Container e macchine virtuali, seppur diversi per definizione, vengono usati molto spesso per raggiungere lo stesso scopo, quello di rilasciare un software già configurato

¹Gruppi di Controllo

e pronto all'uso. Per questo motivo ci sono molti lavori che mettono a confronto le due tecnologie come nel caso del lavoro [6] di Seo, Kyoung-Taek and Hwang, Hyun-Seo and Moon, Il-Young and Kwon, Oh-Young and Kim, Byeong-Jun che ci permette di osservare come l'utilizzo dei container fornisca delle performance migliori (in questo particolare caso) rispetto all'utilizzo di una macchina virtuale. Questo però non significa che le macchine virtuali ad oggi sono uno strumento datato o superato.

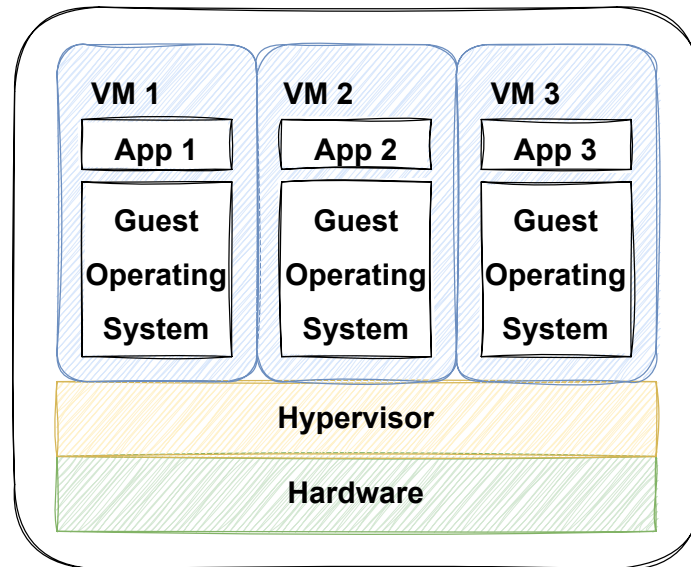


Figura 2.2: Architettura per la gestione di macchine virtuali

2.3.2 Immagine di un container

Abbiamo già visto come è definito un container, ora ci chiediamo come sia possibile usare tale tecnologia. I software per la gestione dei container supportano un particolare tipo di file che fornisce agli strumenti che si occupano della creazione del container, importanti schemi di configurazione. Tale file è chiamato immagine di container e rappresenta, in via astratta, quella porzione del sistema operativo che contiene la nostra applicazione con tutte le dipendenze soddisfatte, tutte le configurazioni, gli script e altro.

Le immagini dei container ci permettono di condividere tale applicazioni con altri utenti e di poterle installare con facilità. Ogni programma di gestione dei container definisce il proprio standard di creazione delle immagini.

Di solito le immagini vengono caricate in un registro, chiamato comunemente container registry, uno spazio dedicato all'archiviazione dell'immagine dove è possibile

gestirle ed avere strumenti utili per l'analisi, per esempio alcuni container registry offrono il servizio di analisi delle vulnerabilità.

2.3.3 Orchestrazione dei Container

Ripensando alla definizione data in precedenza per l'architettura a microservizi e applicando a tale definizione la tecnologia appena presentata, possiamo osservare come ogni componente del nostro sistema può essere considerato come un container, in fondo, abbiamo ripetuto più volte che un microservizio è un'entità autonoma e indipendente, e un container, per definizione, è una porzione di sistema operativo isolato (attenzione questo non implica che i container non possano dialogare tra di loro). La nostra applicazione sarà costituita alla fine da tanti container. Data la necessità di dover gestire più container alla volta, sono [7] nati software che si occupano proprio di questo aspetto.

In generale questo tipo di software creano uno strato aggiuntivo tra l'applicativo che si occupa dei container e i container veri e propri:

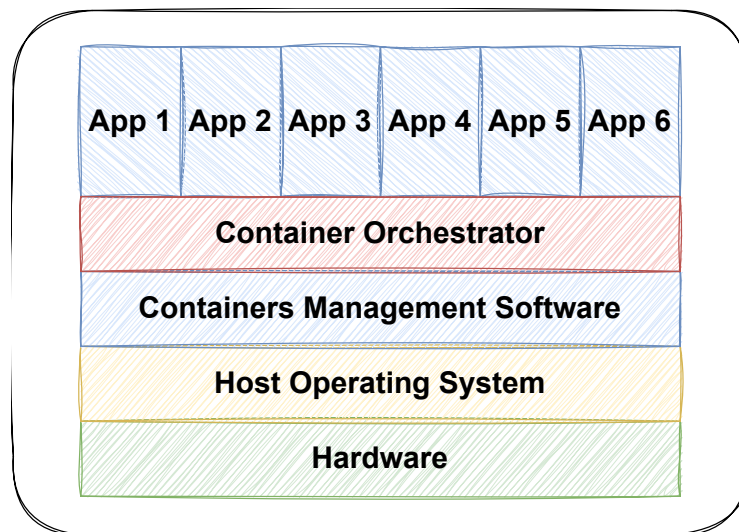


Figura 2.3: Architettura con software per l'orchestrazione dei container

Software per l'orchestrazione dei container [8] mettono a disposizione strumenti in grado di eseguire diverse operazioni di gestione, come il monitoraggio dello stato, il carico di lavoro attuale e molto altro.

2.4 Lato backend di un applicativo

Di solito possiamo dividere le nostre applicazioni in due grandi blocchi chiamati rispettivamente backend e frontend. La parte backend di un'applicazione implementa la cosiddetta logica di business, mentre la parte di frontend rappresenta ciò che è visibile all'utente finale.

Nel corso degli anni sono nate diverse tecnologie per lo sviluppo e la gestione di questi macro blocchi e con la crescente complessità delle applicazioni sono nati anche dei software utili per gestire e risolvere diversi problemi che appaiono nei grandi progetti, come la gestione delle dipendenze.

Secondo le principali piattaforme di ricerca per il lavoro i linguaggi più popolari per lo sviluppo back end sono: Java, Python, C# e JavaScript e stanno crescendo l'uso di Rust e Golang. Di seguito è riportata una tabella con i principali framework assegnati al linguaggio di programmazione:

Linguaggio di Programmazione	Framework
Java	Springboot
C#	ASP.NET
JavaScript	Node.js
Python	Django

Tabella 2.3: Framework associato al proprio linguaggio di programmazione

2.4.1 Gestione delle dipendenze

Una delle problematiche che affligge i grandi progetti è la gestione delle dipendenze. Molto spesso in applicativi reali si tende ad utilizzare diverse tecnologie per rispettare standard o per utilizzare funzioni che il linguaggio di programmazione scelto non rende disponibili tramite le librerie di base. Negli anni sono nati degli strumenti che permettono di automatizzare la gestione delle dipendenze delegando ad un software tutto il lavoro necessario per installare, aggiornare e gestire nuove dipendenze.

3.1 Introduzione

In questo capitolo andremo ad introdurre e analizzare alcune tecnologie che implementano i concetti riportanti nel capitolo precedente. Lo scopo di questo capitolo è quello di permettere al lettore di familiarizzare con alcuni concetti che poi saranno trattati nel prossimo capitolo.

3.2 Gestione dei container: Docker

Uno dei software disponibili per la gestione dei container, nonché uno dei più famosi, è Docker. Disponibile in forma gratuita, Docker è un progetto open source scritto interamente in Go e supportato da numerose aziende, come Google.

Abbiamo già discusso riguardo alla funzionalità e alla potenzialità dei container nel precedente capitolo. Grazie a Docker è possibile utilizzare i container anche su macchine che non utilizzano il kernel di GNU/Linux, in queste macchine viene creato uno strato aggiuntivo di virtualizzazione che permette all'applicativo di creare e gestire i container. Docker fornisce agli sviluppatori gli strumenti necessari per creare, gestire e configurare i container, tramite linea di comando o tramite un'interfaccia grafica.

3.2.1 L'architettura di Docker

L'architettura di Docker è basata sul tradizionale modello client-server.

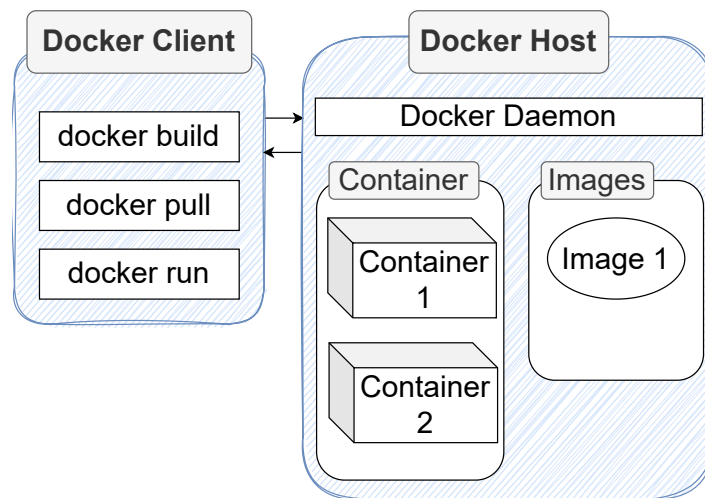


Figura 3.1: Architettura di Docker

Docker Client Il docker client permette all'utente di interagire tramite interfaccia grafica o tramite riga di comando con il docker host. L'interfaccia grafica implementata in Docker però non permette di gestire tutte le componenti, l'utilizzo di Docker tramite riga di comando è consigliato anche dalla documentazione ufficiale. Di seguito sono riportati alcuni dei comandi più frequentemente usati:

1. `docker build -t <image_name>;`
2. `docker pull <image_name>;`
3. `docker run --name <container_name> <image_name>;`

Il primo comando permette di creare un'immagine da un dockerfile ¹, il secondo di scaricare un'immagine da Docker Hub ². Il terzo comando permette la creazione di un container da un'immagine presente in locale, nel caso l'immagine non sia presente, Docker avviserà l'utente che dovrà eseguire il comando due per scaricare l'immagine richiesta.

¹Un file di configurazione usato da Docker per la creazione di immagini.

²Il container register ufficiale di Docker <https://hub.docker.com/>.

Docker Host Il docker host rappresenta il cuore di Docker. Questo componente ha il compito di controllare i container. Per la definizione data in precedenza sappiamo che un container rappresenta una porzione del sistema operativo completamente isolata, allora ci chiediamo come sia possibile che i container, nella realtà, riescano a connettersi e scambiarsi informazioni tra di loro. Il kernel GNU/Linux non fornisce nessun meccanismo di dialogo tra container, ma Docker implementa una logica di networking per eliminare questo limite. L'implementazione delle regole di networking però rende le applicazioni che utilizzano i container meno sicure, durante la fase di sviluppo bisogna tener conto delle vulnerabilità scoperte sulle tecnologie utilizzate e correggerle. Per questo alcuni registri delle immagini permettono di effettuare delle scansioni sulle immagini e in caso di vulnerabilità notificare l'utente.

Di base tutti i file generati all'interno di un container vengono archiviati momentaneamente in un layer di persistenza dei dati, una volta che il container verrà distrutto (o terminato), i dati creati andranno persi. Docker implementa un meccanismo per rendere la persistenza dei dati definitiva, andando ad affidarsi alla macchina su cui docker host è in esecuzione.

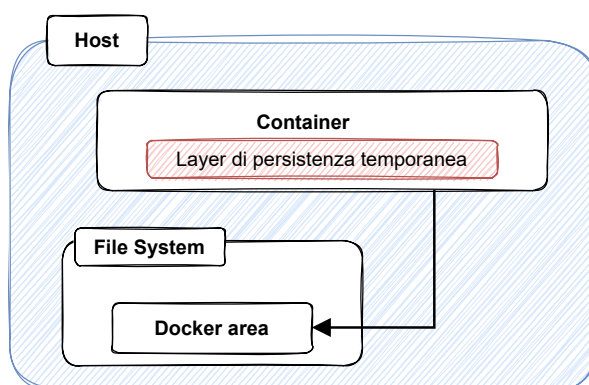


Figura 3.2: Persistenza dei dati in Docker

Per semplicità consideriamo che Docker sia installato su una distribuzione del sistema operativo GNU/Linux, come possiamo osservare dalla figura sopra riportata, Docker implementa un meccanismo che permette ai container di dialogare col file system del computer host per accedere ad un'area di memoria, denominata docker area e creata durante l'installazione di Docker, e rendere persistenti i file creati durante l'esecuzione del container ³.

³In realtà Docker implementa altri due sistemi per la persistenza dei dati, il primo salva i dati generati in una qualsiasi posizione libera della memoria del computer host, il secondo metodo invece

3.3 Orchestrazione dei Container: Kubernetes

Kubernetes [9] è un software per l'orchestrazione dei container open source che fornisce gli strumenti necessari per gestire i sistemi distribuiti in modo resiliente. Uno degli strumenti più importanti che Kubernetes implementa è il load balancer, alcune volte può presentarsi un eccessivo carico di lavoro su uno dei servizi, il load balancer permette una gestione automatica del carico di lavoro, in modo da non causare anomalie nel sistema.

Kubernetes è un software che generalmente è disponibile tramite un servizio cloud, quando richiediamo un servizio del genere ciò che ci viene assegnato è chiamato un cluster. Di seguito possiamo osservare come funziona un cluster di Kubernetes:

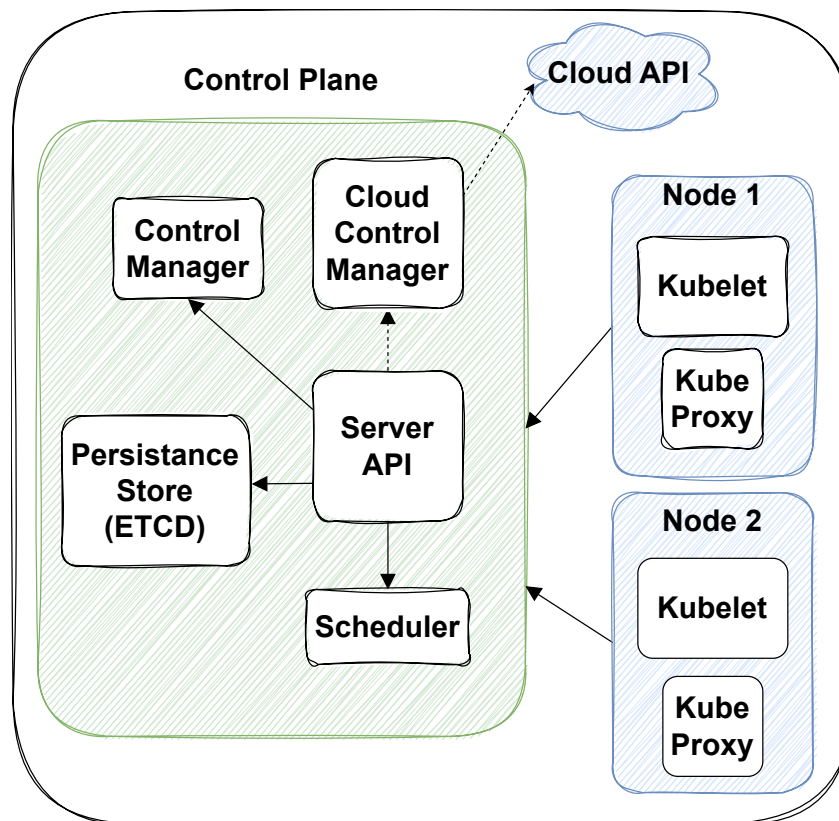


Figura 3.3: Cluster Kubernetes

Control Plane Il control plane è il componente principale di ogni cluster, mette a disposizione dell'utente tutte le funzionalità principali di Kubernetes. Il control plane può anche essere distribuito su più macchine.

permette di depositare i dati nella memoria centrale, di fatto rendendoli temporanei.

Server API La prima componente che approfondiamo è il Server API. Il compito di questo componente è quello di esporre le API per accedere al control plane e poter controllare lo stato l'intero cluster.

Persistence Store ETCD⁴ è un software open source che si basa su concetto di rendere distribuita la cartella /etc presente nei sistemi operativi basati su Unix. In Unix la cartella etc contiene diversi file di configurazione. La componente integrata nel cluster di Kubernetes si occupa di rendere persistenti i file di configurazione utilizzando un sistema di chiave-valore e creando automaticamente dei backup sui dati ogni 30 minuti.

Scheduler Lo scheduler è quel componente del cluster che si occupa di gestire i nuovi pods assegnandoli ad uno dei nodi disponibile. Per pod si intende un gruppo di uno o più container che condividono le stesse informazioni di networking e di persistenza dei dati. Un pod rappresenta la più piccola unità che può essere creata e gestita da Kubernetes.

Controller Manager Il controller manager è formato da un insieme di processi che si occupano di gestire diverse parti del cluster. Un controller è un componente che monitora costantemente lo stato del cluster tramite il server api e se lo stato desiderato non è quello attuale cerca di riportare il sistema ad uno stato precedente.

Cloud Controller Manager Esattamente come il controller manager, il cloud controller manager rappresenta un insieme di processi controller che si occupano di monitorare lo stato del cluster, ma solo di quelle componenti che gestiscono i processi cloud. Il cloud controller manager crea un collegamento tra il server API e la piattaforma di cloud scelta. Se per esempio si sceglie di utilizzare Kubernetes senza fare affidamento ad un servizio di cloud esterno, questa componente non farà parte del cluster.

Node Un nodo rappresenta un ambiente di runtime dove i container sono creati e gestiti.

⁴Letteralmente /etc distributed.

Kubelet Processo che gestisce l'esecuzione dei container nei pod. Tale componente prevede l'esecuzione di diverse istruzioni che monitorano lo stato dei pod.

Kube-Proxy La rete condivisa tra container dello stesso pod viene gestita dalla componente nota come kube-proxy, inoltre permette ad un nodo di poter dialogare con gli altri nodi del cluster. L'implementazione base prevede l'utilizzo delle regole di networking del sistema operativo, ma è anche possibile che sia stesso la componente a gestire le comunicazioni.

3.3.1 Orchestrazione in locale: minikube

Kubernetes è un servizio accessibile principalmente via un servizio cloud. Generalmente tutti i più grandi cloud provider permettono di istanziare un cluster di Kubernetes. Essendo Kubernetes un software open source, la community ha reso disponibile un'implementazione in locale chiamata minikube. Tale software utilizza proprio Docker per emulare il funzionamento di Kubernetes.

3.4 Gestione del Progetto: Apache Maven

Una delle problematiche che affligge lo sviluppo moderno è la gestione delle dipendenze. Durante lo sviluppo di applicazioni è necessario fare affidamento su librerie esterne, per sopperire a mancate implementazione del linguaggio utilizzato. Una delle librerie esterne più utilizzate per lavorare con i file JavaScript Object Notation (JSON) in Java è la libreria sviluppata da Google e mantenuta dalla comunità open source GSON. Java permette l'uso di librerie esterne a patto che il loro file sia incluso nel progetto. L'inclusione manuale delle librerie in Java però porta a diverse problematiche, sia con le librerie stesse, sia col coordinamento con il team di sviluppo, se il nostro progetto necessitasse di più librerie esterne dovremmo scaricare da Internet il file e poi includerlo nel nostro progetto e per ogni libreria dovremmo soddisfare, se presente, la catena di dipendenze, nel fare questo dovremmo anche informare il resto del team di sviluppo.

Fortunatamente sono stati sviluppati software che si occupano di questo particolare aspetto (e non solo) della gestione del progetto. Apache Maven è uno di questi software ed è utilizzato principalmente in applicativi software basati su Java. Grazie a Maven la gestione delle dipendenze di un progetto diventa molto più semplice.

Maven non è uno strumento che gestisce solo le dipendenze di un progetto, permette anche di automatizzare la creazione del pacchetto e la possibilità di specificare una determinata struttura di progetto.

Il fulcro del software è un file chiamato Project Object Model (POM), scritto in eXtensible Markup Language (XML), tale file contiene tutte le istruzioni che necessità Maven per una corretta gestione del progetto. Nel pom sono presenti diverse informazioni, quelle più generiche che riguardano la versione, il nome, le informazioni sulla versione di Java utilizzata o altro, a quelle più specifiche come le dipendenze da soddisfare per il corretto funzionamento del software in sviluppo.

Maven permette anche di integrare dei plugin di terze parti, in questo modo è possibile creare degli strumenti per personalizzare il comportamento di Maven. Per esempio il framework Spring Boot integra il proprio plugin all'interno dei suoi progetti che utilizzano Maven.

3.4.1 Un esempio di POM

Introduzione del file Le prime righe di ogni file pom specificano la versione della struttura e delle specifiche da seguire per quanto riguarda la scrittura dell'intero documento.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache
   .org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
```

Listing 3.1: Introduzione del POM

Informazioni del progetto Dopo le informazioni che riguardano il documento xml, abbiamo le informazioni che riguardano il progetto vero e proprio.

```
1 <groupId>com.example.package</groupId>
2 <artifactId>package</artifactId> <!-- Nome del pacchetto JAR/WAR -->
3 <version>0.0.1-SNAPSHOT</version> <!-- Versione del pacchetto JAR/WAR -->
4 <name>Nome del progetto</name>
5 <description>Descrizione</description>
```

Listing 3.2: Informazioni del progetto che Maven deve gestire

Come è possibile leggere in questa parte di xml vengono specificate le informazioni come nome del progetto, versione e descrizione. Queste informazioni sono utili a Maven durante la fase di creazione del pacchetto. Possono anche essere specificate delle proprietà, come per esempio la versione di Java utilizzata:

```
1 <properties>
2   <java.version>11</java.version>
3 </properties>
```

Listing 3.3: Specifica della versione di Java

Dipendenze e Plugin Infine, il cuore del pom contiene tutte le informazioni per la gestione delle dipendenze e dei plugin. In questa parte di codice oltre a specificare quali plugin e quali dipendenze si intende utilizzare è anche possibile specificare ulteriori regole che Maven dovrà eseguire. Per esempio nei progetti in Java viene quasi sempre utilizzato il framework JUnit per effettuare dei test sull'applicativo in fase di sviluppo, questo pacchetto però non sarà più necessario quando l'applicazione sarà pronta per la distribuzione ed è possibile definire delle regole per escludere la dipendenza di JUnit quando Maven si occuperà di creare il pacchetto destinato al rilascio.

```
1 <dependencies>
2   <!-- Sezione dedicata alle dipendenze da soddisfare -->
3 </dependencies>
4
5 <build>
6   <plugins>
7     <!-- Sezione dedicata ai Plugin-->
8   </plugins>
9 </build>
10 </project>
```

Listing 3.4: Dipendenze e Plugin nel POM

3.5 Framework: Spring Boot

Uno dei framework più utilizzati per lo sviluppo back-end con il linguaggio di programmazione Java è sicuramente Spring Boot. La popolarità di questo framework

lo rende una delle scelte migliori quando si decide di lavorare ad un'applicazione che necessita di Java, grazie alla vasta utenza che ha deciso di includerlo nei propri progetti in rete è possibile trovare soluzioni di ogni tipo. Netflix è la principale sostenitrice del progetto Spring Boot, non solo aiuta nello sviluppo del framework, ma lo utilizza anche per il suo sistema basato sull'applicazione di streaming.

Spring Boot è basato sul framework Spring che a sua volta è basato sulle specifiche di Java Enterprise Edition. Il framework che stiamo esplorando ha 3 vantaggi principali:

1. Configurazione automatica;
2. Inversion of Control;
3. Portabilità dell'applicativo;

Configurazione automatica Le funzionalità implementate da Spring Boot sono operative dal momento della loro richiesta. Per esempio, se viene richiesta la componente di Spring Security, che come intuibile dal nome implementa i servizi per rendere sicura l'applicazione, una volta che la dipendenza sarà soddisfatta, Spring Boot assumerà il controllo per applicare a tale componente una configurazione di base, di fatto rendendola subito operativa.

Questo meccanismo non implica però che tale configurazione di base non possa essere modificata per implementare magari dei servizi aggiunti. Facendo riferimento sempre al caso di Spring Security, nessuno ci vieta di implementare un ulteriore sistema di autenticazione come JSON Web Token (JWT).

Spring Boot permette di dover scrivere meno codice grazie alla configurazione automatica delle componenti, ma allo stesso tempo fornisce i meccanismi per configurare manualmente a nostro piacimento tale componente.

Inversion of Control Il framework Spring Boot eredita da Spring due meccanismi: l'inversion of control e la dependency injection.

Per inversion of control si intende quel meccanismo di ingegneria del software che trasferisce il controllo non più agli oggetti del nostro applicativo, ma al framework stesso. Nei linguaggi di programmazione tradizionali, come il C, noi facciamo uso delle librerie, con l'inversion of control il framework prende il controllo del flusso del nostro codice e quando richiesto lo richiama.

I vantaggi dell'inversion of control sono diversi:

1. disaccoppiamento dell'esecuzione di un servizio e la sua implementazione;
2. facilità di implementazione di diverse soluzioni;
3. facilità di testing dell'intero applicativo, grazie alla possibilità di isolare componenti e le loro dipendenza.
4. una maggior modularità del programma;

Per implementare l'inversion of control Spring utilizza il design pattern dependency injection.

Portabilità La portabilità è l'ultimo aspetto principale di Spring Boot. Questa funzionalità permette di creare un unico pacchetto con estensione Web Application Archive (WAR) del nostro applicativo, ma a differenza di un file WAR classico, Spring Boot rende il nostro pacchetto completo, pronto per il deployment.

4.1 Introduzione

In questo capitolo applicheremo i concetti teorici descritti precedentemente per sviluppare un software basato sui microservizi. Lo scopo è quello di verificare come tale approccio permette uno sviluppo più semplice delle applicazioni dividendo l'intera logica di business in piccoli servizi ognuno indipendente dall'altro.

4.1.1 Pom di Spring Boot

L'utilizzo di Spring Boot come framework back-end genera un file pom con delle informazioni di base. Queste informazioni sono condivise tra tutti i progetti che utilizzano Maven e Spring Boot.

```
1  <!-- ... -->
2  <parent>
3      <groupId>org.springframework.boot</groupId>
4      <artifactId>spring-boot-starter-parent</artifactId>
5      <version>2.7.5</version>
6      <relativePath/> <!-- lookup parent from repository -->
7  </parent>
8  <!-- ... -->
9      <plugin>
10         <groupId>org.springframework.boot</groupId>
```

```
11     <artifactId>spring-boot-maven-plugin</artifactId>
12 </plugin>
```

Nel nostro caso l'estratto del pom riportato indica l'aggiunta dell'utilizzo del framework Spring Boot alla versione 2.7.5 e l'aggiunta del plugin ufficiale sviluppato da Spring Boot.

4.1.2 Struttura dei servizi

Per la gestione dei singoli progetti abbiamo utilizzato Apache Maven. Come spiegato nei precedenti capitoli, Maven è un ottimo strumento, non solo permette di soddisfare tutte le dipendenze richieste dall'applicativo, ma permette anche di standardizzare la struttura di un progetto, per Spring Boot la struttura è molto simile ad un semplice progetto in Java.



Figura 4.1: Struttura per i progetti in Spring Boot

4.2 Architettura

L'applicazione sviluppata presenta tutta la logica di business all'interno di due microservizi. Image Service è il primo servizio sviluppato, il suo scopo è quello di recu-

cati parametri che aiutano la configurazione delle applicazioni. Nel caso del nostro servizio il file in questione è scritto in questo modo:

```
1 spring.application.name=eureka-server
2
3 server.port=8761
4
5 eureka.instance.prefer-ip-address=true
6 eureka.client.register-with-eureka=false
7 eureka.client.fetch-registry=false
```

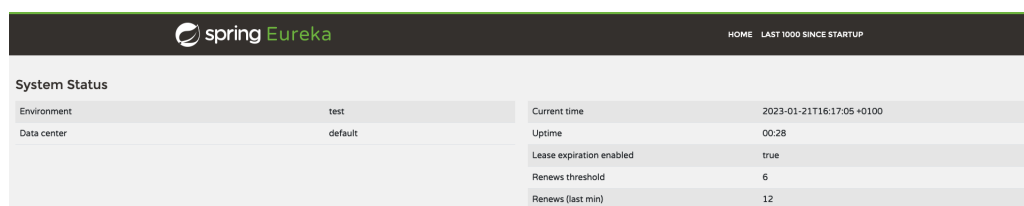
Listing 4.1: Application.properties di Eureka

Nel file sono riportate alcune informazioni base come il nome e la porta su cui verrà esposto il servizio, questo parametro andrà a sovrascrivere la configurazione base del server Tomcat che di default espone gli applicativi sulla porta 8080. dato che la nostra applicazione avrà diverse componenti conviene assegnare ad ognuno una porta diversa.

Ulteriori informazioni impostano dei parametri per il servizio eureka, in questo file specifichiamo che diamo libero arbitrio ad Eureka per quando riguarda la gestione della registrazione dell'indirizzo ip dei vari microservizi.

L'implementazione di Eureka prevede la presenza di due enti, il server e i client. Il servizio di discovery si comporta da server mentre i vari microservizi da client, quest'ultimi una volta effettuato il deploy cercheranno subito di contattare il discovery service (server) per effettuare la registrazione.

Gestione dei servizi Spring Boot espone un endpoint per il servizio Eureka che permette di accedere ad una dashboard per verificare lo stato dell'intero sistema, compresi i servizi che sono riusciti a registrarsi ad Eureka



The screenshot shows the Spring Eureka dashboard. At the top, there's a header with the 'spring Eureka' logo and navigation links 'HOME' and 'LAST 1000 SINCE STARTUP'. Below the header, the 'System Status' section is displayed. It contains two tables. The first table lists 'Environment' as 'test' and 'Data center' as 'default'. The second table lists various system metrics: 'Current time' (2023-01-21T16:17:05 +0100), 'Uptime' (00:28), 'Lease expiration enabled' (true), 'Renews threshold' (6), and 'Renews (last min)' (12).

System Status	
Environment	test
Data center	default
Current time	2023-01-21T16:17:05 +0100
Uptime	00:28
Lease expiration enabled	true
Renews threshold	6
Renews (last min)	12

Figura 4.3: Informazioni generali sul servizio Eureka

Nella stessa pagina possibile non solo verificare lo stato del discovery service, ma anche i servizi registrati, il loro stato e il loro indirizzo di registrazione locale in tempo reale.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
GALLERY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:gallery-service:8100
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:gateway-service:8525
IMAGE-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:image-service:8200

Figura 4.4: Informazioni sui microservizi sottoscritti ad Eureka

4.2.2 Gateway Service

Spring Cloud Gateway [10] ha l'obiettivo di fornire un meccanismo di instradamento ai servizi che permetta di mantenere aspetti come la sicurezza e il monitoraggio delle risorse.

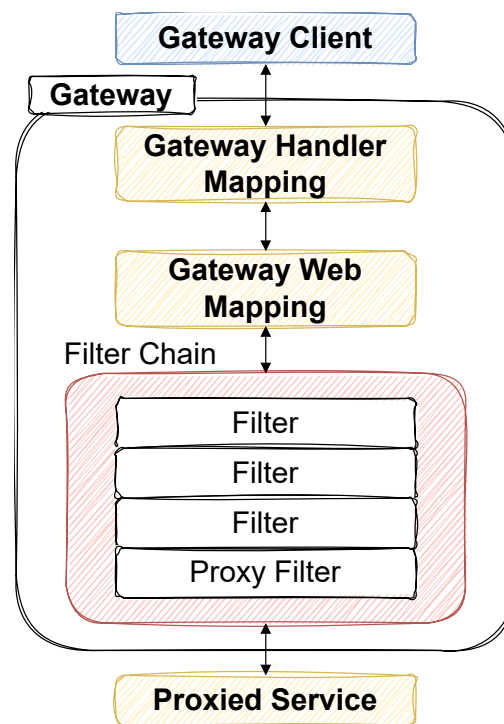


Figura 4.5: Funzionamento di Spring Cloud Gateway

Quando il servizio riceve una richiesta da un client, il componente Gateway Handler Mapping si occupa di riconoscere se il percorso inserito è corretto, in caso affermativo invia la richiesta al Gateway Web Handler che gestisce la richiesta, quest'ultima passa attraverso ad una serie di filtri che implementano logica di sicurezza, ed infine si collega al servizio richiesto dal client.

Durante la configurazione di tale servizio è necessario specificare i vari percorsi dei servizi. Nel nostro caso dobbiamo specificare due servizi, Image Service e Gallery Service, in questo modo il gateway sarà instruito su come instradare il traffico dei dati. Nel file `application.properties` specifichiamo tre attributi principali:

```
1 spring.cloud.gateway.routes[0].id=gallery
2 spring.cloud.gateway.routes[0].uri=lb://GALLERY-SERVICE
3 spring.cloud.gateway.routes[0].predicates[0]=Path=/gallery/**
4
5 spring.cloud.gateway.routes[1].id=image
6 spring.cloud.gateway.routes[1].uri=lb://IMAGE-SERVICE
7 spring.cloud.gateway.routes[1].predicates[0]=Path=/image/**
```

Listing 4.2: `Application.properties` del Gateway

Possiamo notare che gli URI presenti nella configurazione coincidono con i nomi dei servizi che sono registrati sul server eureka 4.3, questo perché il Gateway andrà a ricercare questi servizi proprio nel server di Eureka, nel caso i nomi non coincida quando si andrà a richiedere al Gateway di instradare del traffico verso un servizio quest'ultimo restituirà una pagina di errore.

Sempre nell'URI è possibile osservare che prima di ogni nome di servizio compare "lb", il gateway oltre all'instradamento del traffico si occupa anche di applicare una politica di load balancer, in modo da poter gestire il carico di lavoro in entrata e in uscita.

4.2.3 Avvio dei servizi

L'ordine di deploy dei servizi è importante, il discovery service deve essere il primo servizio ad essere disponibile questo perché i vari servizi appena avviati invieranno una richiesta di registrazione e se Eureka è offline i servizi si arresteranno con un errore. L'ordine utilizzato per l'avvio dell'applicativo è il seguente:

1. Eureka Server Service;
2. Image Service;
3. Gallery Service;
4. Gateway Service;

L'ordine di avvio del gateway non è importante, il servizio in questione effettuerà periodicamente una chiamata ad Eureka per verificare la presenza di nuovi servizi, se presenti verificherà il nome di quest'ultimi con i nomi specificati nel proprio file di configurazione.

4.3 Aggiunta di un nuovo servizio

Abbiamo detto più volte che grazie all'approccio ai microservizi andiamo a creare un software modulare con i servizi che sono indipendenti gli uni dagli altri, questo ci dice che è anche possibile implementare nuove tecnologie molto più facilmente.

4.3.1 World Service

World è il risultato del lavoro svolto durante l'attività di tirocinio presso Kineton. Sviluppato con Spring Boot, tale servizio si basa sul database world, una banca dati messa a disposizione gratuitamente dal sito di MySQL. Per prima cosa andiamo a modificare il diagramma dell'architettura visto precedentemente per implementare questo nuovo servizio:

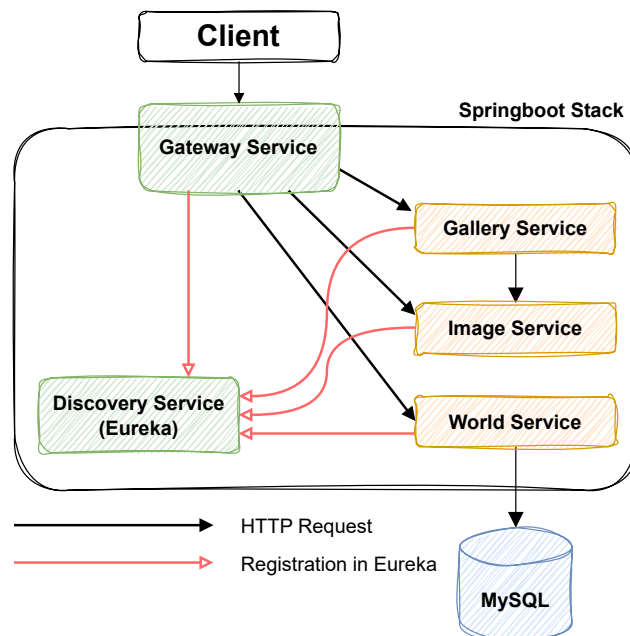


Figura 4.6: Architettura modificata con l'aggiunta di World Service

Oltre ad aver inserito il servizio che si occupa di esporre le API facciamo anche affidamento su un database che contiene i nostri dati.

Diagramma Entità Relazione di World Il diagramma ER del database World è formato da tre tabelle. Le due tabelle `countrylanguage` e `city` sono in relazione con la tabella `country` tramite una relazione uno a molti. Nella tabella `countrylanguage`, l'attributo `Language` è una chiave primaria per la tabella e l'attributo `countryCode` è dichiarata come chiave primaria e come chiave esterna che fa riferimento all'attributo `Code` di `country`. Anche nella tabella `city` l'attributo `CountryCode` è una chiave esterna, che fa riferimento all'attributo `Code` di `country`. Infine in `country` la chiave primaria è `Code`.



Figura 4.7: Diagramma del Database World

4.3.2 Modifiche

Una volta importato il progetto i cambiamenti da effettuare lato codice sono semplici. Prima di tutto dobbiamo dichiarare una nuova dipendenza. L'applicazione era stata creata per esporre delle API, non era previsto il suo ingresso in un'applicazione basata sui microservizi, quindi nel pom andremmo ad aggiungere la dipendenza per poter far sì che il servizio possa essere scoperto da Eureka:

```
1 <dependency>
```

```

2 <groupId>org.springframework.cloud</groupId>
3 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
4 <version>3.1.4</version>
5 </dependency>

```

Listing 4.3: Dipendenza di Eureka Client

Nel file di configurazione dobbiamo dare un nome al nostro servizio per poter essere registrato, una porta diversa dalla configurazione di base di Spring Boot (la porta 8080) e infine dobbiamo specificare l'URL del servizio di discovery Eureka:

```

1 #Eureka Client
2 spring.application.name=WORLD-SERVICE
3
4 server.port=8000
5 eureka.client.service-url.default-zone=http://localhost:8761/eureka

```

Listing 4.4: Configurazione per Eureka

L'ultima modifica che dobbiamo effettuare è al file di configurazione del servizio che offre il Gateway. Questo perché, come visto in precedenza il gateway deve poter riconoscere l'instradamento dei servizi, quindi non ci resta che aggiungere il percorso per il nostro nuovo servizio:

```

1 spring.cloud.gateway.routes[2].id=world
2 spring.cloud.gateway.routes[2].uri=lb://WORLD-SERVICE
3 spring.cloud.gateway.routes[2].predicates[0]=Path=/api/**

```

Listing 4.5: Configurazione per il Gateway

Una volta effettuato il deploy di tutti i servizi, possiamo notare sulla dashboard di Eureka che il nostro servizio è attivo ed è stato registrato:

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
GALLERY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:gallery-service:8100
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:gateway-service:8626
IMAGE-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:image-service:8200
WORLD-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.8:world-service:8000

Figura 4.8: World Service è registrato su Eureka

Lo scopo di questo lavoro era quello di presentare il modello architetturale basato sui microservizi. L'avanzamento delle tecnologie ha portato ad una crescente richiesta di software sempre più impegnativi, l'approccio di sviluppo col modello monolitico inizia a non essere più la scelta più ideale.

Abbiamo discusso di alcune delle tecnologie che implementato concetti teorici di inizi anni 2000 e abbiamo presentato un lavoro che basato sull'approccio ai microservizi ha reso evidente i vantaggi offerti da questa architettura. Le tecnologie che negli anni si sono perfezionate hanno migliorato sempre di più l'architettura ai microservizi, ciò che prima poteva creare dei problemi oggi è stato risolto tramite diverse tecnologie.

Buffo, per la creazione di un applicativo a microservizi abbiamo dovuto far affidamento su diverse tecnologie che offrissero diversi servizi e a sua volta ogni strumento utilizzato è stato creato facendo affidamento ad altri servizi. Questa ridondanza è ciò che rende questo approccio così versatile e potente, non importa come un servizio sia stato scritto o implementato.

Con il sempre più crescente interesse verso il cloud computing, le applicazioni e i software stanno cambiando drasticamente.

Ringraziamenti

Siamo giunti alla fine di un lungo percorso. Prima di tutto vorrei ringraziare tutte le persone che hanno letto e sono riusciti ad arrivare fino a questo punto.

Ringrazio la professoressa **Rita Francese** che mi ha dato l'opportunità di svolgere il mio lavoro di tirocinio esternamente presso Kineton e che col suo corso di Tecnologie e Software per il web oggi sto entrando nel mondo dello sviluppo di applicazioni web.

Tutti le persone e ormai colleghi di Kineton che mi hanno seguito nei mesi precedenti nello studio di queste tecnologie, in particolare il mio tutor **Paolo Nappi** e **Valentino Vivone**.

I miei amici e colleghi universitari, in particolare **Michele Brignola**, **Giuseppe Arienzo**, **Alessandro D'Onofrio**, **Mattia Mascolo** e **Andrea Raiola** che sono riusciti a sopportare il mio caratteraccio durante questi anni universitari e che hanno ritardato il conseguimento di questo titolo.

Mio zio **Rosario** che mi ha sempre motivato e incoraggiato e il mio cane **Ciccia** che mi ha fatto compagnia durante i pisolini pomeridiani di ritorno dall'università

Per finire vorrei ringraziare la persona che mi sta supportando da circa 25 anni, mia **Mamma**, nei momenti buoi della carriera universitaria c'è sempre stata e ha sempre appoggiato il mio percorso. Grazie.



Bibliografia

- [1] RedHat. «Cos'è un'architettura applicativa?» (2020), indirizzo: <https://www.redhat.com/it/topics/cloud-native-apps/what-is-an-application-architecture> (visitato il 2022).
- [2] RedHat. «Cos'è l'architettura guidata dagli eventi?» (2019), indirizzo: <https://www.redhat.com/it/topics/integration/what-is-event-driven-architecture> (visitato il 2022).
- [3] C. Ebert, G. Gallardo, J. Hernantes e N. Serrano, «DevOps,» *IEEE Software*, vol. 33, n. 3, pp. 94–100, 2016. doi: 10.1109/MS.2016.68.
- [4] A. Pathak. «I 30 Migliori Strumenti di DevOps da Tenere d'Occhio nel 2023.» (2022), indirizzo: <https://kinsta.com/it/blog/strumenti-devops/#strumenti-di-automazione-devops> (visitato il 2023).
- [5] RedHat. «Cos'è un container Linux?» (2022), indirizzo: <https://www.redhat.com/it/topics/containers/whats-a-linux-container> (visitato il 2022).
- [6] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon e B.-J. Kim, «Performance comparison analysis of linux container and virtual machine for building cloud,» *Advanced Science and Technology Letters*, vol. 66, n. 105-111, p. 2, 2014.
- [7] B. Burns, J. Beda e K. Hightower, «Orchestrazione di container,» 2022.
- [8] L. Cabibbo, «Orchestrazione di container con Kubernetes,» 2022.
- [9] Kubernetes. «Cos'è Kubernetes?» (2020), indirizzo: <https://kubernetes.io/it/docs/concepts/overview/what-is-kubernetes/> (visitato il 2023).

-
- [10] Springboot. «How Spring Cloud Gateway Work.» (), indirizzo: <https://cloud.spring.io/spring-cloud-gateway/reference/html/#gateway-how-it-works> (visitato il 2023).