Relazione Progetto
Sistemi Cloud

Product Catalogue

Giovanni Imbesi

Matricola: 1000006253

Prof. Giuseppe Pappalardo
Prof. Andrea Francesco Fornaia

17 giugno 2024

Indice

Capitolo 1

Introduzione

In questo elaborato verrà illustrata l'implementazione e il deploy di un'applicazione full-stack su AWS, tramite l'utilizzo di Kubernetes. Questo progetto è stato sviluppato con l'obiettivo di creare un sistema efficiente e scalabile per la gestione di un catalogo di prodotti.
Di seguito un grafico che ne rappresenta la struttura.

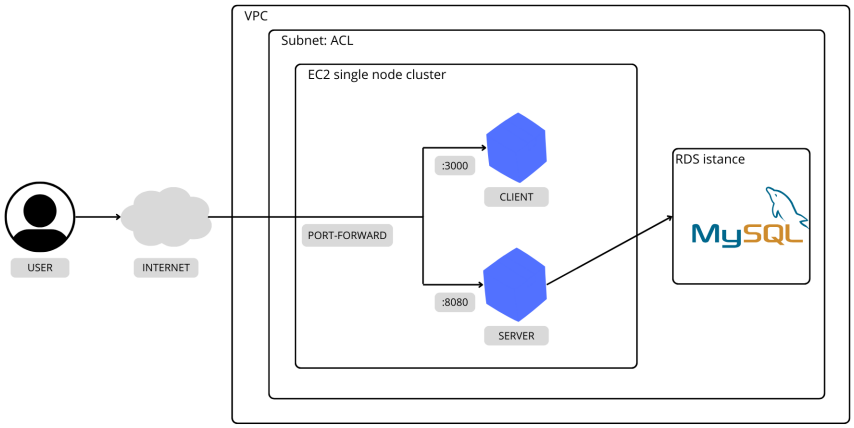

Figura 1.1: Diagramma architetturale

1.1 Motivazioni

Nel contesto attuale, l'implementazione di soluzioni informatiche scalabili e resilienti è essenziale per garantire il successo e la competitività di un'azienda. L'adozione di architetture basate su microservizi e l'utilizzo di strumenti di orchestrazione come Kubernetes sono diventati pilastri fondamentali per la realizzazione di sistemi robusti e performanti.

La scelta di Amazon Web Services (AWS) come infrastruttura cloud è motivata dalla sua affidabilità, scalabilità e vasta gamma di servizi offerti. AWS fornisce un ecosistema completo per lo sviluppo, il deploy e la gestione di applicazioni, consentendo agli sviluppatori di concentrarsi sullo sviluppo del prodotto anziché sull'infrastruttura sottostante.

Nelle sezioni successive di questo elaborato, verranno esaminati nel dettaglio i componenti dell'applicazione, l'architettura di deployment su Kubernetes, le scelte di design e le tecnologie utilizzate per la realizzazione del sistema.

Capitolo 2

Applicativo Client-Server

L'applicativo sviluppato per la gestione del catalogo di prodotti è composto da due componenti principali: il frontend implementato in React e il backend realizzato con Spring Boot. Questi due componenti operano sinergicamente per consentire all'utente di visualizzare e gestire i prodotti nel catalogo in modo intuitivo ed efficiente.

2.1 Frontend

Il frontend dell'applicazione è stato sviluppato utilizzando React, un framework JavaScript ampiamente adottato per la creazione di interfacce utente moderne e reattive. La pagina principale dell'applicazione mostra una selezione dei prodotti presenti nel catalogo, ciascuno con le sue informazioni principali come nome, descrizione e prezzo. L'utente ha la possibilità di eseguire diverse azioni:

- **Inserimento Prodotto**: L'utente può aggiungere un nuovo prodotto al catalogo, inserendone il nome, la descrizione ed il prezzo.
- **Dettaglio Prodotto**: Selezionando un prodotto dalla lista, l'utente può accedere alla pagina di dettaglio del prodotto. Qui, ha la possibilità di visualizzare e modificare le informazioni del prodotto, oltre a poterne effettuare la rimozione.
- **Ricerca**: L'utente può effettuare una ricerca dei prodotti per nome utilizzando la funzionalità di ricerca integrata. Questo consente di individuare rapidamente i prodotti di interesse senza dover scorrere manualmente l'intera lista.

- **Ordinamento**: La lista dei prodotti può essere ordinata in base al prezzo o al nome.

2.2 Backend

Il backend dell'applicazione è stato implementato utilizzando Spring Boot, un framework Java per lo sviluppo di applicazioni basate su microservizi. Il backend fornisce un'interfaccia per le operazioni CRUD (Create, Read, Update, Delete) sui dati dei prodotti nel catalogo. Le funzionalità principali del backend includono:

- **Gestione dei Prodotti**: Il backend gestisce la logica di business relativa ai prodotti nel catalogo. Questo include la creazione di nuovi prodotti, la visualizzazione dei dettagli dei prodotti esistenti, la modifica delle informazioni dei prodotti e la rimozione dei prodotti dal catalogo.

- **Persistenza dei Dati**: Il backend utilizza un servizio di storage come Amazon RDS su AWS per conservare in modo persistente i dati dei prodotti. Ciò assicura che le informazioni sui prodotti siano mantenute anche in caso di riavvii o scalabilità dei servizi.

- **API RESTful**: Il backend espone un'interfaccia API RESTful che consente al frontend di comunicare in modo efficiente con il backend. Questo permette una separazione chiara tra il frontend e il backend, consentendo loro di essere sviluppati, testati e scalati indipendentemente l'uno dall'altro.

Capitolo 3

Architettura Kubernetes

L'architettura Kubernetes adottata per l'applicazione di gestione del catalogo dei prodotti è progettata per offrire un ambiente scalabile e gestibile per i microservizi frontend e backend. Il cluster Kubernetes è stato creato mediante l'utilizzo di Minikube, una soluzione leggera per eseguire un singolo cluster Kubernetes su una macchina locale, e viene gestito attraverso l'interfaccia della riga di comando kubectl. Di seguito vengono presentate le componenti principali.

3.1 Server Deployment e Service

Il backend dell'applicazione è stato distribuito all'interno del cluster Kubernetes mediante un Deployment. Il server è esposto all'interno del cluster Kubernetes attraverso un Service di tipo NodePort, che reindirizza il traffico in ingresso alla porta 8080 del pod server. Sono inoltre definite le seguenti variabili d'ambiente:

- **SPRING_DATASOURCE_URL**: Specifica l'URL del database RDS su AWS dove sono conservati i dati dei prodotti.
- **SPRING_DATASOURCE_USERNAME**: Fornisce il nome utente necessario per l'accesso al database RDS.
- **SPRING_DATASOURCE_PASSWORD**: Fornisce la password necessaria per l'accesso al database RDS.

3.2 Client Deployment e Service

Analogamente al backend, il frontend dell'applicazione è stato distribuito all'interno del cluster Kubernetes mediante un Deployment.

Il client è esposto all'interno del cluster Kubernetes attraverso un altro Service di tipo NodePort. Questo, reindirizza il traffico HTTP in ingresso dalla porta 80 verso la porta 3000 del pod frontend.

Nel pod è definita una variabile d'ambiente chiamata
REACT_APP_URL, che specifica l'URL del backend al quale il frontend deve inviare le richieste API. Ciò consente al frontend di comunicare con il backend all'interno del cluster Kubernetes.

Quando i deployments vengono eseguiti, il sistema recupera l'immagine Docker del microservizio corrispondente da DockerHub. Questo processo garantisce che il microservizio backend sia facilmente aggiornabile e riproducibile.

Capitolo 4

Amazon Web Services

Il progetto utilizza diversi servizi offerti da Amazon Web Services (AWS) per fornire un ambiente scalabile e affidabile per l'esecuzione e la gestione dei suoi componenti.

4.1 Amazon EC2

Per ospitare il cluster Kubernetes, è stata utilizzata una macchina virtuale EC2 di tipo t3.medium chiamata **ProductEC2**. Questa istanza è stata avviata con un'immagine Ubuntu e configurata manualmente per installare Docker, Minikube e kubectl. Questi strumenti sono stati utilizzati per creare e gestire il cluster Kubernetes all'interno dell'istanza.

È importante sottolineare che sulla macchina EC2 non è presente il codice sorgente della repository, ma solo i file di deployment necessari per avviare i servizi.

Per consentire la comunicazione con i pod dall'esterno, i servizi sono stati esposti tramite **port-forwarding**. Ciò significa che le porte dei servizi Kubernetes sono state inoltrate dalla macchina virtuale EC2 a quelle locali, consentendo l'accesso ai servizi dal di fuori del cluster.

Il gruppo di sicurezza associato all'istanza EC2 accetta le richieste SSH da qualsiasi indirizzo e il traffico TCP sulle porte esposte dai pod, garantendo un adeguato livello di sicurezza per l'accesso remoto e la comunicazione con i servizi all'interno del cluster.

4.2 Amazon RDS

Per la gestione dei dati dei prodotti, è stato utilizzato Amazon RDS per configurare un database MySQL. RDS semplifica la gestione del database fornendo funzionalità di backup, ripristino, scalabilità e monitoraggio integrati.

Il database ospita i dati dei prodotti nel catalogo, consentendo al backend dell'applicazione di archiviare e recuperare informazioni sui prodotti in modo affidabile e scalabile. La connessione al database è configurata nel backend utilizzando l'URL, il nome utente e la password forniti da RDS.

Capitolo 5

CI/CD

Il processo di Continuous Integration (CI) e Continuous Deployment (CD) è gestito utilizzando GitHub Actions e Ansible. Questo permette di automatizzare la build delle immagini Docker e il deployment del software all'interno del cluster Kubernetes su AWS.

5.1 GitHub Actions

Il file principale delle GitHub Actions **main.yml** è configurato per eseguire il processo di build e deployment quando avviene un push sul branch **main** del repository. Il processo include le seguenti fasi:

- **Checkout Source**: Effettua il checkout del codice sorgente dal repository.
- **Login to Docker Hub**: Effettua il login su Docker Hub utilizzando le credenziali fornite tramite GitHub Secrets.
- **Build e Push delle Immagini Docker**: Utilizzando Docker Buildx, vengono costruite le immagini Docker per i microservizi frontend e backend, e pushate su Docker Hub.
- **Esecuzione del Playbook Ansible**: Utilizzando l'azione **dawidd6/action-ansible-playbook@v2**, viene eseguito il playbook Ansible per il deployment nel cluster Kubernetes.

5.2 Ansible Playbook

Il playbook Ansible si occupa del deployment dell'applicazione nel cluster Kubernetes. Esso si occupa di stabilire una connessione SSH con la macchina EC2, al fine di eseguire dei comandi predefiniti.
Il playbook contiene le seguenti attività:

- **Deploy del Cluster Kubernetes**: Esegue uno script bash per applicare i file di deployment all'interno del cluster Kubernetes. Questo avvia i pod dei microservizi nel cluster.
- **Port Forwarding**: Avvia il port forwarding per i servizi del backend e del frontend, consentendo loro di essere accessibili esternamente al cluster.

Capitolo 6

Conclusioni

La soluzione presentata rappresenta una versione iniziale funzionante del progetto, ben strutturata grazie all'adozione dell'approccio DevOps e dell'architettura cloud native. Questa base solida offre numerose opportunità per futuri sviluppi, tra cui:

- **Nuove funzionalità**: L'architettura DevOps implementata facilita l'aggiunta di nuove funzionalità sia lato front-end che back-end.
- **Scalabilità con Kubernetes su EC2 o EKS**: Un possibile sviluppo futuro potrebbe essere la migrazione del cluster Kubernetes su EKS (Elastic Kubernetes Service). Questa soluzione permetterebbe una gestione avanzata delle risorse, facilitando l'orchestrazione dei container e la scalabilità orizzontale del backend.
- **Utilizzo di Load Balancer AWS**: Per migliorare la distribuzione del carico e garantire maggiore affidabilità, si potrebbe considerare l'implementazione di un Load Balancer di AWS. Questo aiuterebbe a distribuire uniformemente le richieste tra le istanze del backend, aumentando la resilienza del sistema.
- **Elastic IP per Stabilità degli Indirizzi**: L'utilizzo di Elastic IP (Indirizzi IP Elastici) potrebbe garantire stabilità agli indirizzi del backend, evitando cambiamenti frequenti e fornendo una base più consistente per l'esposizione del servizio.