19/12/2023

ECAMazon

Scalable architecture Project 2023

Dubois Hugo - 195347  
Zhao Yiming - 195187

Table of contents

[Introduction 2](#_Toc153764781)

[Materials 3](#_Toc153764782)

[Methods 4](#_Toc153764783)

[Result 6](#_Toc153764784)

[Discussion 9](#_Toc153764785)

# Introduction

The microservices architecture is a model that structures an application as a collection of small services, each responsible for a specific business capability and deployable independently. Each microservice operates as an autonomous entity, enabling development teams to work independently on different services. This decoupling of services promotes parallel development, allowing for faster release cycles and the ability to respond quickly to changing requirements or customer feedback.

The concept of scalability is positioned as the ability of an IT infrastructure to scale efficiently and effectively. It ensures that the platform can not only meet current demands but also grow and adapt to future needs seamlessly.

The project's goal is to create an e-commerce application. This project allows a company to redefine how people shop, interact with products, and experience the digital marketplace. We are tasked with implementing the 'shop' section of the project. This section is primarily responsible for product classification and price management for each product.

We will need to communicate with the 'stock' section to retrieve all currently available products, obtain data from the 'user' section to gather information about the user, and finally, communicate with 'payment' to send an order placed by the user.

Une image contenant texte, diagramme, écriture manuscrite, capture d’écran

Description générée automatiquement

Figure 1:process of ECAMazon

# Materials

* React: open-source JavaScript library for building user interfaces or UI components, widely used for creating single-page applications where data can be changed without reloading the page and allowing developers to create reusable UI components and manage the state of the application efficiently.
* Docker: allow automate deployment, scaling, and management of applications. It uses containerization technology to package an application and its dependencies into a standardized unit called a container.
* Cassandra DB: Cassandra is a NoSQL database designed to handle large amounts of data across server clusters and is commonly used in applications requiring horizontal scalability, high availability, and efficient management of large data volumes.
* Node server: A Node.js server refers to a web server that is built using Node.js, a runtime environment that allows to execute JavaScript code on the server side. Node.js is well-suited for building scalable and high-performance server-side applications.
* Kubernetes: this is a container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. Kubernetes provides a robust and flexible framework for container orchestration, allowing developers and operators to manage containerized applications seamlessly.

# Methods

We started by creating a Node server to facilitate communication with the servers of other microservices (user, payment, stock) and transmit data related to user-placed orders.

Additionally, we set up a database for the 'shop' section, allowing us to store information retrieved by the backend and establish connections between the data.

The frontend application is developed using React, enabling communication between users and the server.

Une image contenant diagramme, ligne, croquis, Dessin technique

Description générée automatiquement

Figure 2: communication diagram for shop

To retrieve data from multiple servers, we designed our database by creating relationships between the following tables:

* Stocks: Retrieves all available products via the 'stock' microservice. This table records the retrieved data to avoid querying other servers repeatedly.
* Products: Associates different stocks with prices.
* Promo: Records promotion codes.
* ProductPromo: Associates multiple promotion codes with a single product.
* Category: Records all product categories.
* CategoryProducts: Associates multiple products with a category.
* Order: Records past orders.
* Recommendations: Provides product recommendations to customers based on their previous orders.
* User: Records data retrieved via the 'Users' microservice.

Une image contenant texte, diagramme, Plan, Parallèle

Description générée automatiquement

Figure 3: database diagram

# Result

By logging into the site, you are redirected to the 'Categories' tab. You can view all product categories sold in our store. Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Figure 4: categories tab

You can select one of the categories and browse all the products belonging to the 'jeux' category. Here, you can add the second product to the cart; this information will be temporarily stored in the user's session. Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Figure 5: promo tab

By checking the 'basket' tab, you retrieve all the items saved in the session. You can see that the product added earlier is in the cart. You can also edit the quantity of this product by adjusting the '-' or '+' button.  
We can also send all the items in the cart to confirm the order. The information related to this order will be sent to the backend to be recorded in the database, and the corresponding quantity will be deducted from the stock. Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Figure 6: basket tab

As an administrator, you can also delete or add a category. By making changes on this page, you send this modification to the backend. Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

Figure 7: admin tab

By selecting a category on the administrator page, you can also add or remove a product from the database. To add a new product, you need to specify its name, price, and quantity. Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Figure 8: edit product of a category.

Once our application is ready to be used locally, we can add a Dockerfile to build a Docker image, which is a snapshot of a file system and application code forming the basis for a container. Images are built from Dockerfiles and can be shared and reused across different environments.

docker build -t shop-front .

docker run -p 3000:3000 -d shop-front

By creating a Docker Compose file, we can manage multi-container Docker applications. It uses a YAML file to configure application services, networks, and volumes, allowing developers to define complex application stacks.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 9: Example of YAML file

Kubernetes automates the deployment, scaling, and operation of application containers. It abstracts away the underlying infrastructure, making it easier to manage and scale containerized applications.

kubectl apply -f frontend-deployment.yaml

kubectl apply -f frontend-service.yaml

In the context of Kubernetes, a resource refers to an object that defines the desired state for an application. This includes specifications such as the number of replicas, the container image to be used, and various configuration parameters.

# Discussion

Resource efficiency is a notable feature of Docker that contributes to scalability. Containers share the host operating system (OS) kernel, resulting in a more lightweight infrastructure compared to traditional virtual machines. This efficiency allows for faster startup times and optimal utilization of system resources, making Docker an ideal choice for scaling applications.

Scalability in a Dockerized environment is streamlined using container orchestration tools, such as Kubernetes or Docker Swarm. These tools automate the deployment, scaling, and management of containerized applications. By defining a desired state for the application, including the number of replicas, container images, and configuration parameters, Docker orchestration tools enable seamless scaling of applications to meet varying workloads.

Kubernetes is a widely adopted container orchestration tool that simplifies the management of containers across diverse environments, ranging from development and testing to production. Its primary role is to abstract the intricacies of handling containers at scale, presenting a standardized method for deploying and scaling applications in a cloud-native setting.

The platform's dynamic resource allocation enhances overall scalability and resilience. By abstracting away, the complexities of underlying infrastructure, Kubernetes provides a unified and consistent platform for deploying containerized applications. This abstraction is pivotal in achieving seamless scalability, enabling Kubernetes to dynamically adjust the number of container instances based on real-time demand.

The scalability feature of Kubernetes is especially vital for applications facing variable traffic levels or those structured with a microservices architecture. In the context of microservices, where applications are broken down into smaller, independently deployable units, Kubernetes excels in independently scaling each service based on demand.

In essence, Docker and Kubernetes together provide a powerful solution for containerization within a scalable architecture. Docker's attributes of consistency, isolation, resource efficiency, and scalability are complemented by Kubernetes' role as a container orchestration tool. The combined impact of these technologies creates a robust and automated environment for deploying and managing scalable applications, making them integral tools in the DevOps and containerization space.