https://www.proconference.org/index.php/gec/article/view/gec34-00-008
DOI: 10.30890/2709-1783.2024-34-00-008

UDC 004.2

BOOK LIBRARY MANAGEMENT SYSTEM BASED ON CLOUD MICROSERVICES AWS

СИСТЕМА МЕНЕДЖМЕНТУ БІБЛІОТЕКИ КНИГ НА ОСНОВІ ХМАРНИХ МІКРОСЕРВІСІВ AWS

Smetanenko A.V. / Сметаненко А.В.
bachelor / бакалавр
ORCID: 0009-0002-8224-4086
Petro Mohyla Black Sea National University, Mykolayiv, UA
Чорноморський національний університет імені Петра Могили, м Миколаїв, Україна
Kulakovska I.V. / Кулаковська І.В.
c.f.m.s., as.prof. / к.ф.м.н., доц.
ORCID: 0000-0002-8432-1850
Petro Mohyla Black Sea National University, Mykolayiv, UA
Чорноморський національний університет імені Петра Могили, м Миколаїв, Україна

Abstract. *The developed system demonstrates how microservices can be integrated into a single backend service running on the AWS cloud platform. This ensures high reliability, scalability and security, and allows for flexible adaptation to the changing needs of users and organisations. This system has been successfully deployed on AWS using the relevant services of the cloud platform. The implemented code allows us to create an architecture that is easily maintained, scalable and reusable in other projects. The use of MVC, clean architecture, and DDD ensures modularity, clear separation of responsibilities, and ease of testing and development of the system. Based on this project, approaches to the implementation of distributed systems can be explored.*

Keywords: *distribution system, AWS, Node.js, Bun, TypeScipt, Fastify, Elysiajs, Docker, Makefile, JWT.*

Анотація. *Розроблена система демонструє, як мікросервіси можуть бути інтегровані у єдиний бекенд–сервіс, що функціонує на хмарній платформі AWS. Це забезпечує високу надійність, масштабованість та безпеку, а також дозволяє гнучко адаптуватися до змінних потреб користувачів і організацій. Розподілена система менеджменту бібліотеки книг користувача складається з чотирьох сервісів, які працюють у єдиній екосистемі, але реалізовані як мікросервіси. Ця система була успішно розгорнута на AWS, використовуючи відповідні сервіси хмарної платформи. Реалізований код дозволяє створювати архітектуру, яка легко підтримується, масштабована та може бути перевикористана в інших проектах. Використання MVC, чистої архітектури та DDD забезпечує модульність, чітке розділення відповідальностей та легкість у тестуванні та розвитку системи. На основі цього проекту можна дослідити підходи до реалізації розподілених систем.*

Ключові слова: *розподільна система, AWS, Node.js, Bun, TypeScipt, Fastify, Elysiajs, Docker, Makefile, JWT.*

Introduction.

In today's world, the rapid development of technology necessitates the implementation of innovative solutions to improve the efficiency of business processes. Distributed systems based on cloud services are becoming key elements for ensuring reliability, scalability and security in organisations. Thanks to cloud technologies, companies can adapt their resources to their own needs, which significantly increases their competitiveness and the efficiency of information flow management.

Cloud technologies provide centralised resource management, which allows organisations to optimise infrastructure and support costs. In addition, they provide an opportunity to quickly scale systems in response to growing needs, which is a key factor for successful business development in today's competitive environment. Thus, the implementation of distributed systems based on AWS (Amazon Web Services) cloud services is becoming important for increasing the efficiency, reliability and security of organisations.

Main text

The goal of building this system was to create a distributed architecture consisting of three separate microservices that interact with each other through different protocols. Each microservice is responsible for a specific part of the system's functionality, which ensures flexibility, scalability and reliability.

The overall architecture of the system includes an authentication service that is private and provides user authentication and authorisation, a user management service that is responsible for managing user information and uses the authentication service to verify authentication, and a book management service that is responsible for managing book information and interacts with other services via HTTP and Amazon [2].

To organise the code in the project, we created our own structure and architecture supporting the MVC (Model-View-Controller) pattern, Clean Architecture [3] and Domain-Driven Design (DDD) [2]. These approaches were chosen to create an architecture that can be easily reused across projects, ensuring modularity, scalability and maintainability.

Basic principles:

- MVC (Model-View-Controller): provides a clear separation of the application

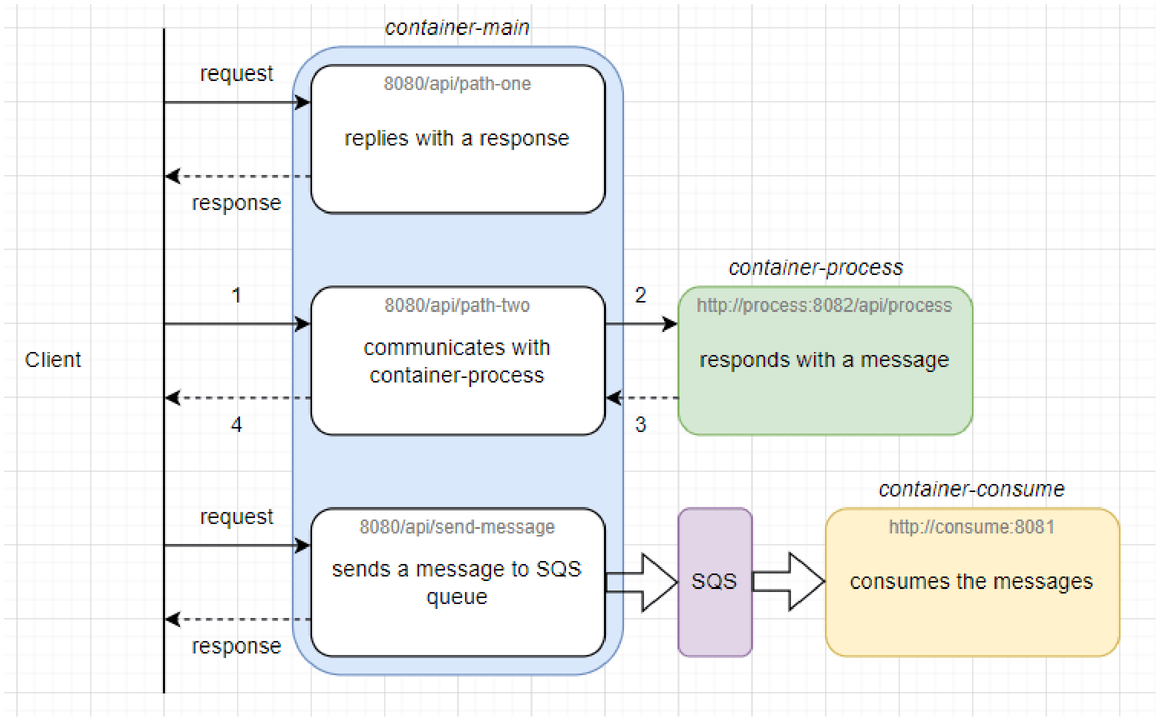
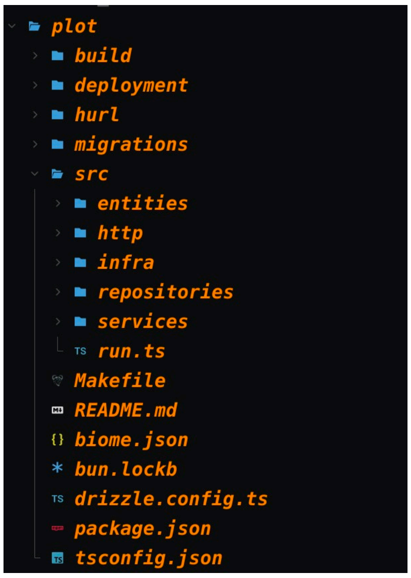


Figure 1 - Example of microservices organisation



```
plot
  build
  deployment
  hurl
  migrations
  src
    entities
    http
    infra
    repositories
    services
    run.ts
  Makefile
  README.md
  biome.json
  bun.lockb
  drizzle.config.ts
  package.json
  tsconfig.json
```

Figure 2 - Code structure and architecture

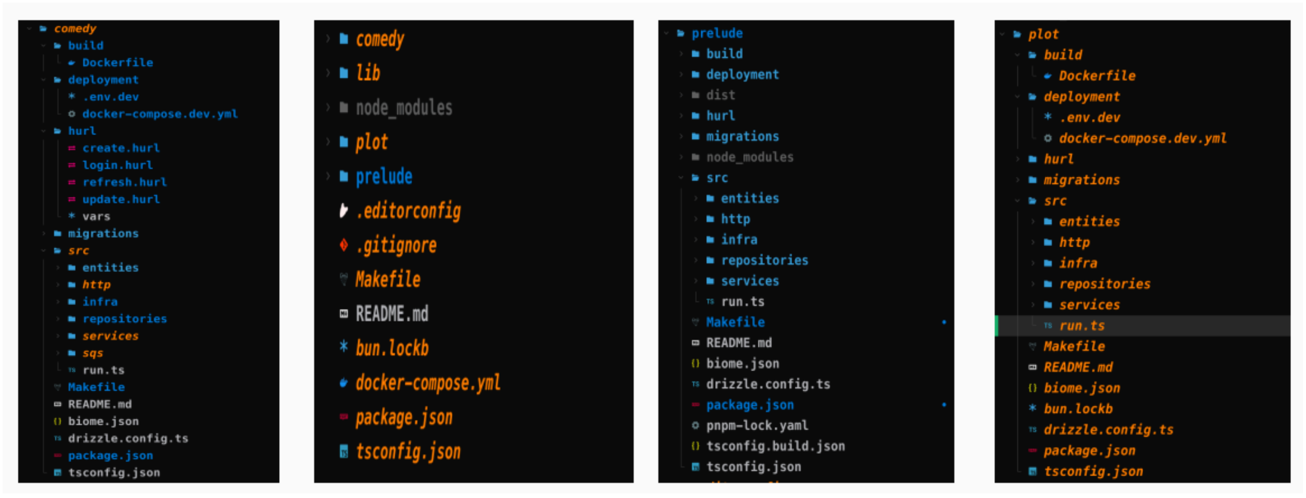
logic into three main components: model, view, and controller. This helps to separate responsibilities and simplify the maintenance and development of the application;

- Clean Architecture: aims to ensure independence from frameworks, easy testing, and increased system flexibility. It involves dividing the system into several layers, each of which is responsible for its own part of the functionality;

- DDD (Domain-Driven Design): Domain-driven design focuses on creating a system that reflects business logic and requirements. This is achieved through the use of domain models, aggregates, repositories and other DDD concepts.

This code organisation allows you to create an architecture that is easy to maintain, scalable and reusable in other projects. The use of MVC, clean architecture and DDD ensures modularity, clear separation of responsibilities and ease of testing and development of the system.

The main technologies chosen to implement the microservice architecture are: Node.js for its asynchronous nature, high performance and a large ecosystem of modules; Bun for its high speed of JavaScript code execution and built-in tools for testing and packaging; TypeScript for static typing, which allows detecting errors at the compilation stage and supports modern JavaScript standards; Docker for environment isolation, which makes applications portable and independent of the environment, and simplifies the process of deploying and updating applications.



```
comedy
  build
    Dockerfile
  deployment
    .env.dev
    docker-compose.dev.yml
  hurl
    create.hurl
    login.hurl
    refresh.hurl
    update.hurl
    vars
  migrations
  src
    entities
    http
    infra
    repositories
    services
    sqs
    run.ts
  Makefile
  README.md
  biome.json
  drizzle.config.ts
  package.json
  tsconfig.json
```

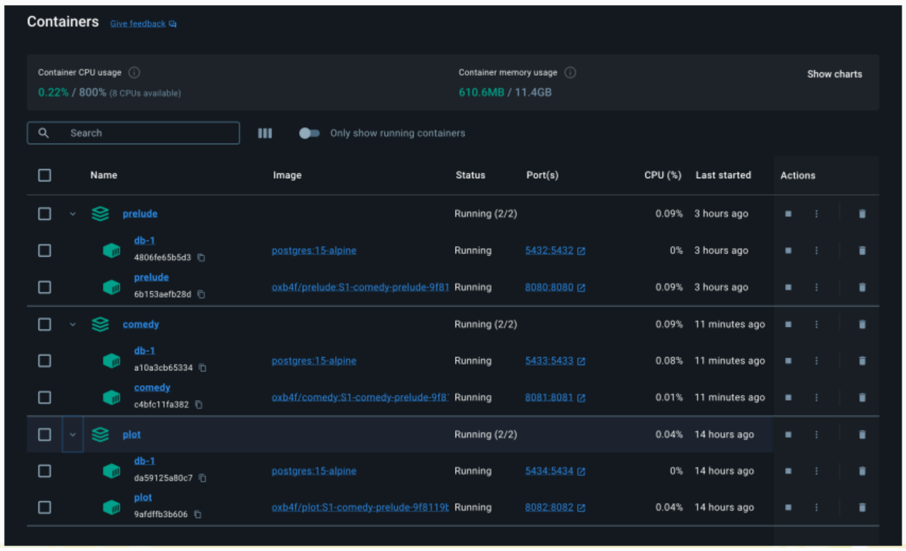
```
comedy
lib
node_modules
plot
prelude
.editorconfig
.gitignore
Makefile
README.md
bun.lockb
docker-compose.yml
package.json
tsconfig.json
```

```
prelude
  build
  deployment
  dist
  hurl
  migrations
  node_modules
  src
    entities
    http
    infra
    repositories
    services
    run.ts
  Makefile
  README.md
  biome.json
  drizzle.config.ts
  package.json
  pnpm-lock.yaml
  tsconfig.build.json
  tsconfig.json
```

```
plot
  build
    Dockerfile
  deployment
    .env.dev
    docker-compose.dev.yml
  hurl
  migrations
  src
    entities
    http
    infra
    repositories
    services
    run.ts
  Makefile
  README.md
  biome.json
  bun.lockb
  drizzle.config.ts
  package.json
  tsconfig.json
```

Figure 3 - Developed application structure and architecture

The result of the study is a designed and implemented distributed system based on AWS cloud services that will take into account the advantages and disadvantages of existing solutions on the market and meet the needs of users, ensuring reliability, scalability and security.



Containers Give feedback

Container CPU usage 0.22% / 800% (8 CPUs available)

Container memory usage 610.6MB / 11.4GB

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
prelude		Running (2/2)		0.09%	3 hours ago	
db-1 4806fe65b5d3	postgres:15-alpine	Running	5432:5432	0%	3 hours ago	
prelude 6b153aefb28d	oxb4f/prelude:S1-comedy-prelude-9f81	Running	8080:8080	0.09%	3 hours ago	
comedy		Running (2/2)		0.09%	11 minutes ago	
db-1 a10a3cb65334	postgres:15-alpine	Running	5433:5433	0.08%	11 minutes ago	
comedy c4bfc11fa382	oxb4f/comedy:S1-comedy-prelude-9f8	Running	8081:8081	0.01%	11 minutes ago	
plot		Running (2/2)		0.04%	14 hours ago	
db-1 da59125a80c7	postgres:15-alpine	Running	5434:5434	0%	14 hours ago	
plot 9afdffb3b606	oxb4f/plot:S1-comedy-prelude-9f8119b	Running	8082:8082	0.04%	14 hours ago	

Figure 4 - An example of a local launch of a distributed system

The result is a distributed system with three microservices for the book application, which includes an authorisation and authentication service, a user management service, and a book management service.

All these technologies ensure the creation of a flexible, scalable, and easily maintained system. To organise the code in the project, we chose the mono-repository approach, which allows all services to be stored in one repository and simplifies the development, maintenance, and deployment processes.

Summary and conclusions.

The processes of designing and implementing a distributed system based on cloud services are considered. The methods and technologies used to create a distributed system based on cloud services ensure reliability, scalability and security. We analysed and selected a technology stack for building a microservices system consisting of three servers that interact via different protocols and are written on different frameworks and runtime environments. All servers are based on AWS, which ensures scalability, security and reliability of the system.

Література:

1. Newman S. Building Microservices. O'Reilly Media, 2015. 280 p. (accessed 01.03.2024).

2. Amazon Web Services (AWS). Official AWS documentation. URL: https://aws.amazon.com (accessed 01.10.2023).

3. Sbarski P. Serverless Architectures on AWS. Manning Publications, 2017. 320 p. (accessed 20.04.2024).

Тези відправлено: 26.08.2024 г.