

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | SpendSense | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Research Document |  | | | |
|  | | | | May, 2025—Viktoria Todorova—First Draft |  | | | |
|  | | |  | | |  | | |

# Introduction

The aim of this research is to design and develop a secure and scalable cloud-native architecture for SpendSense, a financial web application. The research investigates how cloud-native technologies, such as Kubernetes, AWS Lambda, and RabbitMQ, can provide the necessary scalability, security, and resilience required for modern financial services.

The architecture is evaluated based on industry best practices and aligned with the GDPR compliance standards for handling sensitive financial data. A key focus of the research is on selecting the right cloud services (containerization, serverless computing) and implementing effective CI/CD pipelines and DevOps practices for seamless deployment.

Through this study, I explored the integration of secure microservices, efficient data storage solutions, and reliable communication channels while ensuring minimal downtime and optimal performance. The research concludes with a set of best practices and design recommendations for building secure, scalable, and GDPR-compliant financial applications in a cloud-native environment.

# Architectural Foundations

**Cloud-native architecture** uses microservices, containers, and orchestration tools to fully leverage the flexibility of cloud platforms. This design supports scalability, modular development, and simplified maintenance.

**Kubernetes** is widely adopted for orchestrating containerized applications due to its auto-scaling, self-healing, and resource management capabilities (Burns et al., 2016). Combined with serverless computing tools like AWS Lambda, it helps optimize costs and allows execution of event-based tasks without persistent infrastructure

**Microservices** divide a system into small, focused services that can be built and deployed independently. This fits well with financial platforms, which often require separation of concerns across domains such as budgeting, reporting, and user accounts. Spring Boot is commonly used for this model due to its simplicity and integrated security features.

To manage service-to-service communication, **message brokers** like **RabbitMQ** are preferred for asynchronous messaging. RabbitMQ supports reliable delivery and decoupling of services, making it ideal for handling tasks like transaction processing and event propagation.

Security is critical in financial systems. Modern applications must address common risks outlined in the **OWASP Top 10** (OWASP, 2023). Secure identity management, encryption, access control, and hardened container configurations are essential. Spring Security, JWT, and OAuth2 provide robust options for enforcing access rules.

Applications handling personal financial data must comply with regulations such as the **GDPR**. This includes securing data at rest and in transit, offering data access controls, and supporting features like data erasure and audit logging. While cloud platforms like AWS offer compliance tools, secure data architecture must be part of the application design itself.

Finally, **DevOps** practices and CI/CD pipelines ensure continuous delivery and stability in production. These pipelines automate testing, security checks, and deployments—essential for financial systems where availability and accuracy are non-negotiable.

# Cloud Services and Scalability

*What cloud services (serverless, containerization) can be used to ensure application efficiency and reliability during increased user traffic?*

SpendSense uses cloud technologies to make sure the app can handle more users, stay reliable, and be cost-effective. The main tools used are containerization with Kubernetes and serverless computing with AWS Lambda.

**Using Kubernetes for Containers**

The Spendings Upload and Budgets services run inside containers managed by Kubernetes. This allows the app to automatically start more service instances when needed—like during busy times. If something goes wrong, Kubernetes can also restart services on its own, helping the system stay online and stable.

**Using AWS Lambda for Serverless Tasks**

The Report Generation service runs as a Lambda function on AWS. This means it only runs when needed and doesn’t take up resources when it's idle. AWS Lambda can quickly handle many requests at once, which is great for short tasks that happen occasionally.

**Managing Traffic**

Traefik is used as the API Gateway. It decides where each user request should go and spreads the traffic across services evenly. This helps keep performance smooth and prevents any one service from getting overloaded.

**Scalable Databases**

Each service has its own database hosted on AWS RDS. RDS can grow when needed by adding more storage or computing power. It also provides backups and can be set up to handle more read requests through replicas.

**Summary**

By using Kubernetes, AWS Lambda, and RDS, SpendSense is built to scale easily. This setup keeps the system fast, reliable, and able to grow as more people use it.

# Research Questions

In this section you can find the research questions that will serve as a structure of this research. I will provide one main question, that summarizes the topic of this study. Furthermore, I will list a few sub-questions that divide the research into smaller issues, which would be easier to research.

##### Main Question

How can a cloud-native architecture be designed to ensure scalability, security, and reliability for a financial web application?

##### Sub-Questions

1. What cloud services (serverless, containerization) can be used to ensure application efficiency and reliability during increased user traffic?
2. How can security best practices be implemented in a cloud-native finance web app?
3. How can DevOps principles and CI/CD pipelines be integrated into the cloud infrastructure for seamless deployment?
4. What are the best ways to handle and store financial data in a cloud environment while ensuring compliance with GDPR regulations?

# Approach and Planning

In this part of the document, I will describe the methodologies I intend on using to answer the above-mentioned questions. I will also present an initial planning and estimation of the required time for completing each part of this study.

##### Methodologies

To complete this research, I will rely on the DOT framework. I will determine the most suitable methodologies for investigating each topic to ensure a structured workflow through the semester. You can see the estimated approach per each question on *Fig. 1.*

//add prototyping and field research

*Fig. 1 – Table that explains the methodologies per question*

##### Planning

I am also providing a table with the planning and the estimated time for each task (*Fig. 2*). This schedule is not concrete, so some minor changes are expected throughout the semester.

*Fig. 2 – The estimated time for each activity necessary to complete the research*

##### Expected Deliverables

At the end of this research, I should have several key deliverables:

* Research Report
* Cloud Native Architecture for a Finance App
* Recommendations and Best Practices for Scalability and Security