

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | SpendSense | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Research Document |  | | | |
|  | | | | May, 2025—Viktoria Todorova—Final 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.

# Research Questions Overview

This research focuses on how to build a cloud-native architecture that is secure, scalable, and reliable for a financial web application like SpendSense. To explore this, the study is guided by four main questions: What cloud services can help the app stay fast and reliable when many users are active? How can strong security measures be added to protect financial data? How can DevOps and CI/CD pipelines be used to support smooth and frequent updates? And finally, what are the best ways to store and manage financial data in the cloud while following GDPR rules? These questions help shape the design choices and guide the overall development approach discussed in this research.

# Architectural Foundations

**Cloud-native architecture**[1] uses microservices[2], containers, and orchestration tools to fully leverage the flexibility of cloud platforms. This design supports scalability, modular development, and simplified maintenance.

**Kubernetes**[3] is widely adopted for orchestrating containerized applications due to its auto-scaling, self-healing, and resource management capabilities. 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**[4] like **RabbitMQ**[5] 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**[6]. 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[7] 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**[8] 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?*

**Research Methods**

To investigate cloud services and scalability for SpendSense, I combined several research methods from the DOT framework[9]. A **literature study** was conducted to understand current best practices in using containerization and serverless computing within scalable cloud-native systems. I also performed a **competitive analysis** to examine how similar financial applications use technologies like Kubernetes, AWS Lambda, and managed databases.

In addition, I carried out a **problem analysis** to identify potential challenges SpendSense might face during high traffic periods. To support architectural decisions, I used **IT architecture sketching** to visually map out and test different infrastructure setups. Together, these methods helped determine which cloud services would best support performance, reliability, and cost-efficiency during varying user loads.

**Overview**

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 Azure Functions for Serverless Tasks**

The Report Generation service will run as an Azure Function. This means it only runs when needed and doesn’t take up resources when it's idle. Azure Functions can quickly handle many requests at once, which is great for short tasks that happen occasionally.

**Managing Traffic**

Traefik[10] 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 will have its own database hosted on AWS RDS[11]. 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.

# Security Best Practices in a Cloud-Native Finance Web App

*How can security best practices be implemented in a cloud-native finance web app?*

**Research Methods**

To research security best practices for SpendSense, I used a mix of methods from the DOT framework. I started with a **literature study** to understand the latest security guidelines for cloud-native apps, including OWASP and GDPR. I then did a **best, good, and bad practices analysis** (by researching real-world projects, but also speaking to colleagues facing similar challenges) to see what works well (or doesn’t) in other financial applications.

To get expert advice, I conducted **expert interviews** with my teachers. I also used **problem analysis** to map out possible security risks in SpendSense’s architecture, you can read more in my OWASP report. To test the system, I applied **security testing** to check for weak points in data protection and service communication.

Finally, I performed **a guideline conformity analysis** to make sure the security design follows important rules and standards such as OWASP and GDPR (you can refer to the relative reports). This mix of methods helped create a well-rounded and reliable security approach.

**Overview**

Security is a crucial aspect to consider when developing an application, especially when it is one that works with sensitive data such as personal finances information. SpendSense employs a reliable security strategy that aligns with modern best practices and regulatory requirements, ensuring data protection, system integrity, and user trust.

**Authentication and Authorization**

SpendSense uses Spring Security along with JWT (JSON Web Tokens) and OAuth2[12] to control who can log in and what they can access. Users must log in securely, and only people with the right permissions can use certain parts of the app. OAuth2 can also connect to trusted login systems like Google or company accounts.

**Data Encryption**

Data in transit (data moving between services or users) is protected using HTTPS, so it can’t be seen or changed during transfer. Data at rest (data stored in databases) will be encrypted using tools provided by AWS RDS and AWS Key Management Service (KMS). This means even if someone gets access to the storage, they can’t read the data without permission.

**Secure Microservices Communication**

Service-to-service communication is facilitated through RabbitMQ, which supports TLS and authentication mechanisms, ensuring that only authorized services exchange messages. Internal APIs can use mutual TLS (mTLS) for added verification and trust between microservices.

**Logging and Monitoring**

All user actions and system events in SpendSense are recorded using logging and monitoring tools like Prometheus[13] and Graphana[14]. These logs can help understand what happened if something goes wrong, such as a system error or failure. They also make it easier to detect suspicious activity or possible security attacks by highlighting unusual behavior. In addition, keeping detailed logs supports compliance with data protection laws like GDPR by providing a clear record of how data is accessed and used.

**DevSecOps and CI/CD Integration**

SpendSense will follow a DevSecOps approach, which means security is built into every step of the development and deployment process. As part of the CI/CD[15] pipeline—an automated system that tests and delivers new code—security checks run automatically whenever a developer makes a change. These checks look for common coding errors, scan software libraries for known vulnerabilities, and inspect container images for potential security risks. This helps catch and fix issues early, before the code reaches production.

**Summary**

SpendSense applies strong security practices to protect user data in its cloud-native financial application. It uses secure login and access control with Spring Security, JWT, and OAuth2. Data is encrypted during transfer using HTTPS. Services communicate securely through RabbitMQ and mTLS. Logging tools like Prometheus and Grafana help track activity, detect issues, and support GDPR compliance. Security will also be built into the development process using DevSecOps, with automated checks in the CI/CD pipeline to catch problems early.

# DevOps into Cloud Infrastructure

*How can DevOps principles and CI/CD pipelines be integrated into the cloud infrastructure for seamless deployment?*

**Research Methods**

To explore how DevOps and CI/CD could be applied in SpendSense, I used a mix of practical and analytical research methods. From the Field category, I applied **Problem Analysis** to understand the limitations of manual deployment and identify key areas where automation was needed.

In the Lab phase, I used **Component Testing** to check that each pipeline step worked correctly—such as running tests, scanning code with SonarQube, and deploying to AKS. I also **used Static Program Analysis** through Snyk to catch bugs and security issues early.

From the Workshop category, I used **IT Architecture Sketching** to plan the CI/CD pipeline structure, and **Code Review** to make sure the scripts and configurations followed best practices.

These combined methods helped me build a working, automated pipeline that supports reliable and secure deployment in the cloud.

**Overview**

Using DevOps principles and CI/CD pipelines in a cloud-based environment like SpendSense helps make software updates fast, stable, and secure. For a financial web app—where reliability, security, and accuracy are very important—DevOps provides a practical way to manage updates with fewer errors and more control.

In the SpendSense project, DevOps was put into practice through a fully automated CI/CD pipeline built with GitHub Actions. This means that every time developers push new code to GitHub, an automated process starts. This pipeline includes steps for testing the code, checking its quality, building the app, and deploying it to the cloud.

**Build and Test**

The first step of the pipeline builds application and runs tests to make sure the new code works as expected. These tests are important for catching problems early—especially in financial features like budget calculations or data validation. If tests fail, the pipeline stops and sends feedback so issues can be fixed before moving forward.

**SonarQube**

Next, the pipeline checks the quality and security of the code using SonarQube[16][17]. This tool scans the code to find bugs, security risks, and bad practices. It gives a quality score and suggestions for improvement. This step follows the DevSecOps approach, which means adding security checks early in the development process—not just at the end.

**DockerHub Images**

After passing all tests and checks, the code is packaged using Docker. This creates a container image of the app that includes everything it needs to run. The image is then uploaded to Docker Hub, a public registry where it can be accessed for deployment. This ensures consistency across different environments (like development, test, and production).

**Azure Kubernetes Services**

Deployment is handled through Azure Kubernetes Service (AKS). The pipeline updates the Kubernetes cluster by pulling the latest Docker image and rolling out changes gradually, so the app stays online during the update. Kubernetes also automatically replaces failed containers, keeping the system stable.

**Summary**

This setup—GitHub Actions for automation, SonarQube for quality checks, Docker for packaging, and AKS for deployment—makes it easy to update SpendSense often and safely. It reduces manual steps, lowers the chance of mistakes, and ensures high availability. The system also tracks every update, which helps with accountability and future debugging.

Although the current pipeline is simple, it provides a solid foundation. In the future, it could be improved by adding integration tests, Docker image security scans, or monitoring tools like Prometheus and Grafana. Tools like Terraform could also be used to manage the cloud setup as code, making it easier to repeat or change environments.

By following DevOps practices and building an automated CI/CD pipeline, SpendSense can grow and improve continuously—while keeping user data safe and the app running smoothly.

# GDPR Regulations in Cloud Environment

*What are the best ways to handle and store financial data in a cloud environment while ensuring compliance with GDPR regulations?*

**Research Methods**

To understand how to handle financial data securely and meet GDPR requirements, I used a combination of methods from the Library, Field, and Lab categories of the DOT framework. I began with a **Literature Study** to explore GDPR principles and best practices for secure data storage and deletion in cloud environments. This helped shape the technical and legal requirements for SpendSense.

In the Field phase, I used **Problem Analysis** to identify the key risks—such as unauthorized access, insecure communication, and non-compliant data storage. I also performed a **Stakeholder Analysis** to consider the needs of end users, system administrators, and legal teams, especially in areas like data access and the right to be forgotten.

From the Lab category, I tested and validated the security setup using **System Testing**, which included deploying the services with TLS, checking Keycloak authentication, and running deletion flows through the message bus and Saga pattern. I also used **Security Testing** to verify that HTTPS was correctly enforced and access to databases was properly restricted within AKS.

This mix of methods helped ensure that SpendSense's cloud architecture supports both GDPR compliance and secure financial data management.

**Overview**

In financial applications like SpendSense, handling and storing personal data responsibly is a legal and technical priority. The General Data Protection Regulation (GDPR) sets strict rules around how personal data is used, accessed, stored, and deleted. To meet these requirements, SpendSense uses a combination of secure cloud technologies and design patterns that support both compliance and performance.

**HTTPS Protocol**

To protect user data during communication, SpendSense uses TLS certificates to enforce HTTPS across all services. This ensures that any data sent between users and services is encrypted and secure from outside interference. Internal service communication is also encrypted, creating an end-to-end secure data flow.

**Authentication and Authorization**

User identity and access are managed with OAuth2, using Keycloak as the identity provider. Keycloak enables role-based access control, secure login flows, and integration with external systems if needed (e.g., Google login). OAuth2 tokens, such as JWTs, ensure that only authorized users can access certain parts of the application, and that each request is verifiable.

**Data at Rest**

All data is stored in MySQL databases hosted within Azure Kubernetes Service (AKS)[18]. These databases are protected by the platform’s built-in security (like network isolation and access policies) which ensures stored financial data cannot be accessed without proper authorization. Each microservice manages its own database, applying data minimization by only storing what is necessary for its function.

**Right To Be Forgotten**

To comply with GDPR’s “right to be forgotten”[19], SpendSense uses a message bus and Saga pattern. When a user requests account deletion, the request is passed through the system as an event. Each service that stores user data listens to this event and deletes its own relevant data. This event-driven design ensures that personal data is erased across all services in a coordinated and reliable way, even in distributed systems.

**Azure**

Finally, hosting services on Azure AKS gives access to a secure, compliant cloud infrastructure. Azure provides tools for managing secrets, monitoring database activity, and enforcing regional data storage policies—important for GDPR’s data residency rules.

**Summary**

In conclusion, SpendSense achieves GDPR compliance through secure communication (HTTPS/TLS), strong access control (OAuth2 + Keycloak), cloud databases on Azure AKS, and a Saga-based deletion process for user rights. This cloud-native approach balances technical efficiency with the legal and ethical responsibilities of managing financial data.

# Conclusion

This research set out to answer four core questions to guide the design of a secure, scalable, and GDPR-compliant cloud-native financial application—SpendSense. First, to support scalability and reliability during increased user traffic, the system uses Kubernetes for container orchestration and Azure Functions for serverless functions, allowing it to grow and adapt as needed. Second, security best practices are implemented through Spring Security, OAuth2, HTTPS encryption, and DevSecOps pipelines, protecting user data and maintaining system integrity. Third, the integration of DevOps principles using GitHub Actions, SonarQube, Docker, and Azure Kubernetes Services ensures fast, automated, and stable deployments. Finally, GDPR compliance is achieved by combining secure data storage in AKS-hosted MySQL databases, strong authentication with Keycloak, and a Saga pattern for the right to be forgotten.

Together, these solutions demonstrate how cloud-native technologies and modern software practices can work in harmony to build financial applications that are both technically strong and legally responsible.

# REFERANCES

1. <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition>
2. <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>
3. <https://kubernetes.io/>
4. <https://en.wikipedia.org/wiki/Message_broker#:~:text=A%20message%20broker%20is%20an,exchange%20messages%2C%20effectively%20implementing%20decoupling.>
5. <https://www.rabbitmq.com/>
6. <https://owasp.org/www-project-top-ten/>
7. <https://usercentrics.com/resources/gdpr-checklist/?utm_source=google&utm_medium=cpc&utm_campaign=extension-apps&utm_content=sitelink&utm_term=privacy%20policy%20for%20mobile%20app&utm_source=google&utm_medium=cpc&utm_term=privacy%20policy%20for%20mobile%20app&utm_campaign=uc_src_eu_en_generic_bof_apps&utm_ad_group=en-phrase-apps_privacypolicy&gad_source=1&gad_campaignid=21828247342&gbraid=0AAAAABYXPmVveVxo3LmTwdQBbiOHi-09W&gclid=Cj0KCQjw0qTCBhCmARIsAAj8C4Ym94W64uxE5ojh1g84FV42M-UXdrLjmd9qeFvLwNZ_-caXV2gKSwoaAqJoEALw_wcB>
8. <https://www.atlassian.com/devops#:~:text=DevOps%20is%20a%20set%20of,and%20collaboration%2C%20and%20technology%20automation.>
9. <https://ictresearchmethods.nl/>
10. <https://traefik.io/traefik/>
11. <https://aws.amazon.com/rds/#:~:text=Amazon%20Relational%20Database%20Service%20(Amazon,%2C%20backing%20up%2C%20and%20patching.>
12. <https://auth0.com/intro-to-iam/what-is-oauth-2>
13. <https://www.tigera.io/learn/guides/prometheus-monitoring/#:~:text=Prometheus%20is%20typically%20used%20to,separated%20with%20line%20feed%20characters.>
14. <https://medium.com/@MetricFire/what-is-grafana-8de44d241765>
15. <https://www.redhat.com/en/topics/devops/what-is-ci-cd#:~:text=CI%2FCD%2C%20which%20stands%20for,accelerate%20the%20software%20development%20lifecycle.>
16. <https://www.sonarsource.com/products/sonarqube/>
17. <https://medium.com/@piyushkashyap045/comprehensive-guide-to-sonarqube-understanding-benefits-setup-and-code-quality-analysis-with-caffbc8afa0f>
18. <https://learn.microsoft.com/en-us/azure/aks/what-is-aks>
19. <https://en.wikipedia.org/wiki/Right_to_be_forgotten>