LambdaPay

Project overview

Cloud Computing Technologies

Massimiliano Capparuccia (65918A)

November 8, 2025

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Table of Contents

Context & Goals

1 Introduction

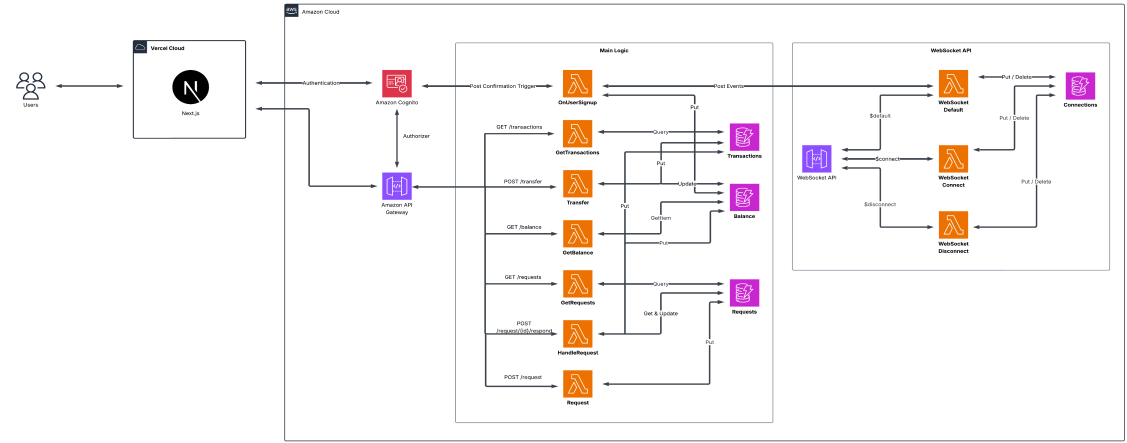
- **What is LambdaPay?** A fully serverless digital payments application built with AWS Lambda that allows users to transfer money, request payments and manage their balance.
- **Why Serverless?**
  - Zero Server Management
  - Automatic Scaling
  - Fast Deployment
  - Seamless Updates & Rollbacks

Table of Contents

2 System Design

Architecture Overview

2 System Design



Components

The components used in the application are:

- **API Gateway:** A fully managed service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale.
- **Lambda Functions:** A serverless compute service that lets you run code without provisioning or managing servers, and scales automatically with incoming traffic.
- **Cognito:** Provides user sign-up, sign-in, and access control for web and mobile apps. Supports multi-factor authentication, social identity providers, and SAML.
- **DynamoDB:** A serverless, NoSQL database service that scales to zero, has no cold starts, no version upgrades, no maintenance windows, no patching, and no downtime maintenance.

WebSockets

2 System Design

The real-time communication in the application is built with:

- **API Gateway WebSocket API:** Using API Gateway as a service for handling WebSocket connections, routing messages and maintaining connection state at scale.
- **Lambda Functions:** Handlers for the ‘$connect$‘, ‘$disconnect$‘ and custom message routes, enabling authentication and broadcast logic.
- **DynamoDB:** To stores active connection IDs and user session metadata.

Frontend

2 System Design

The frontend of the application uses:

- **Next.js 14**: A React framework that use server-side rendering, static site generation, and client-side components.
- **TypeScript** for type safety
- **Tailwind CSS** + **shadcn/ui**: A utility-first CSS framework combined with modern UI components for professional design.
- **AWS Amplify** for integrated authentication with Cognito

Table of Contents

Serverless

3 Deployment

Deployment of the backend is managed by:

- **Serverless Framework**: An open-source CLI toolkit with an approachable YAML syntax for defining and deploying serverless applications (functions and infrastructure) across cloud providers, enabling zero-friction development.
- **serverless.yml**: The central configuration file where you declare AWS Lambda functions and related resources for automatic deployment.

Frontend Deployment (CI/CD Pipeline)

3 Deployment

The frontend is deployed and continuously updated via:

- **Vercel for GitHub**: A Git integration that automatically builds and deploys your application on every push or pull request.
- **Automatic Redeployments**: Each commit to the connected GitHub repository triggers a new build and global deployment via Vercel's CI/CD, ensuring your app is always up-to-date and allowing instant rollbacks to previous deployments if needed.

Table of Contents

Security

4 Demo

To demonstrate security:

- Simple script show that without using the correct cookie for authorization, the response from the lambdas are ”401 - Unauthorized”
- The IAM Roles are tightly scoped using least privilege.

```
iamRoleStatements:
  dynamodb:Query,GetItem,PutItem,UpdateItem,DeleteItem,Scan
  cognito-idp:ListUsers,AdminGetUser
```

Load Balance

4 Demo

To demonstrate load balancing, I used a tool called ***Artillery***, an open-source performance testing toolkit that lets you simulate high loads on HTTP, WebSocket and other protocols to measure and analyze the scalability and responsiveness of your systems.

- Define test scenarios in a simple YAML or JSON format, specifying virtual users, arrival rates, and request flows.
- Gather detailed metrics such as response times, error rates, throughput, and per-request statistics.

Fault Tolerance

4 Demo

To demonstrate the fault tolerance, I simulated a failure in the **Transfer**:

- Despite the error, other services like balance and transactions still worked, highlighting the isolation and resilience of the architecture.
- This was done to show that even if a single component fails, the rest of the system continues to operate normally.

Thank you for listening!

- GitHub repository: https://github.com/MaxKappa/LambdaPay
- Demo: https://lamdbapay.spidernetwork.it