# CLOUD ARCHITECTURE

**Introduction**

In today’s digital-first world, public institutions are increasingly adopting cloud-based solutions to enhance service delivery, scalability, and operational efficiency. The Oshawa Public Library (OPL), a key community resource center, has recognized the need to modernize its infrastructure by transitioning to a scalable and resilient cloud environment. To support this transformation, a secure and highly available microservice-based web application has been proposed to improve user access and backend data processing.

This project presents the design and implementation of comprehensive cloud architecture on **Amazon Web Services (AWS)** tailored specifically to OPL’s needs. The solution leverages industry best practices in networking, compute, storage, security, and monitoring to meet the library’s critical requirements: high availability, scalability, security, data durability, and ease of maintenance.

The architecture includes containerized frontend and backend services, a managed PostgreSQL database, object storage for library media, and built-in logging and monitoring through AWS-native tools. This document outlines the design rationale, technical components, and architectural decisions made to ensure OPL’s system is robust, future-proof, and aligned with modern cloud standards.

**Background**

The Oshawa Public Library (OPL) serves as an essential pillar of the local community, providing public access to educational and informational resources. With increasing demand for digital services, OPL identified the need to improve and modernize its IT infrastructure. Currently operating without any cloud integration, the library faces limitations in scalability, availability, and user experience, particularly as user traffic and online service demands continue to grow.

To address this, the library plans to deploy a **microservice-based web application** that separates the frontend user interface from the backend API logic. The front end will allow users to log in, view their accounts, and access library services, while the backend will handle data processing and communication with the library’s centralized database.

The application team has already developed and containerized both services using Docker. Additionally, a **PostgreSQL relational database** will be used to manage library data, while **object storage** is required to hold media assets such as images and documents. With no prior cloud infrastructure in place, the project team has the freedom to architect a solution using any major cloud provider.

For this project, **Amazon Web Services (AWS)** was chosen due to its global presence, mature ecosystem, managed services, and alignment with cloud architecture best practices. This background sets the stage for the architectural design that follows, ensuring OPL’s transition to the cloud is both secure and scalable.

**Rationale**

The decision to design the Oshawa Public Library’s infrastructure using **Amazon Web Services (AWS)** was guided by several key considerations: scalability, security, high availability, cost-effectiveness, and ease of integration with containerized applications. AWS offers a broad range of managed services that align with the project's goals and allow the team to focus on application logic rather than infrastructure maintenance.

**Cloud Provider Selection**

AWS was chosen over other providers due to its:

* Proven track record in public-sector cloud adoption
* Rich suite of services for compute, storage, networking, and security
* Native support for **Docker containers** via **Amazon ECS with Fargate**
* Integrated tools like **CloudWatch** for monitoring and **IAM** for fine-grained access control

**Microservices & Container Strategy**

Given the application’s microservice architecture, **Amazon ECS (Elastic Container Service)** using the **Fargate** launch type was selected. This serverless compute engine for containers eliminates the need to manage EC2 instances while providing flexibility and scalability.

* The **frontend container** was deployed in public subnets to handle web traffic.
* The **backend container** was deployed in private subnets to securely process data and communicate with internal services like the database.

**Storage & Data Management**

To manage user data and transactional records, **Amazon RDS for PostgreSQL** was implemented in a **Multi-AZ** configuration to ensure fault tolerance and automatic failover.  
For storing images and media, **Amazon S3** was used due to its durability, scalability, and support for **versioning** and **lifecycle policies**, which help reduce long-term storage costs.

**Network Design & Security**

A custom **Virtual Private Cloud (VPC)** was designed with subnet segregation across two availability zones to support **high availability**.

* **Public subnets** host internet-facing components such as the Application Load Balancer (ALB) and NAT Gateways.
* **Private subnets** contain sensitive services like the backend API and RDS instance.

**Security Groups**, **IAM Roles**, **S3 Bucket Policies**, and **encryption** at rest/in-transit were applied to protect data and resources at every layer.

**Scalability, Monitoring & Recovery**

* **Auto Scaling Groups** were used to dynamically adjust ECS task counts based on CPU thresholds.
* **Amazon CloudWatch** was configured to monitor metrics and trigger alarms.
* **RDS backups** and **Point-In-Time Recovery (PITR)** provide strong disaster recovery options.
* **S3 versioning** ensures accidental deletions or overwrites are reversible.

This architecture was designed based on AWS best practices, guided by the **Well-Architected Framework**, ensuring that the solution is reliable, secure, cost-optimized, operationally efficient, and performance-efficient.

**Summary of Architecture: Oshawa Public Library AWS Design**

**Overview**

The architecture is a **secure, highly available, and scalable cloud environment** designed on AWS to support OPL’s new microservice-based web application. It includes public and private services, a backend PostgreSQL database, object storage, and end-to-end monitoring.

**1. Network Infrastructure**

* A custom **VPC (10.0.0.0/16)** was created, divided into:
  + **2 Public Subnets** (10.0.1.0/24 and 10.0.2.0/24)
  + **2 Private Subnets** (10.0.3.0/24 and 10.0.4.0/24)
* Public subnets are located in **different availability zones (us-east-1 & us-east-2)** to ensure high availability.
* Internet connectivity is provided via an **Internet Gateway**, and private subnets access the internet through **NAT Gateways** in public subnets.
* Separate **route tables** for public and private subnets were configured and clearly labeled.

**2. Compute Layer**

* **Amazon ECS (Fargate)** was used to deploy Dockerized microservices:
  + **Frontend ECS Services** deployed across both public subnets
  + **Backend API ECS Services** deployed across private subnets
* Services are split across AZs to ensure **failover resilience**.
* An **ECS Task Execution IAM Role** was created and used by both services to securely interact with AWS services like RDS and S3.

**3. Auto Scaling**

* Each ECS service (Frontend and Backend) is connected to an **Auto Scaling Group**:
  + **Min: 2 | Max: 4**
  + **CPU Utilization Threshold: 75%**
* This allows the system to handle traffic spikes while optimizing cost.

**4. Storage Layer**

* **Amazon RDS (PostgreSQL)** was deployed in a **Multi-AZ configuration** to ensure high availability and failover support.
* **Point-In-Time Recovery (PITR)** was enabled, with daily backups to support disaster recovery.
* A central **S3 Bucket** was deployed for storing images and digital assets:
  + **Versioning Enabled**
  + **AES-256 Encryption**
  + **Lifecycle Policy** to archive files after 30 days

**5. Security Features**

* **Security Groups** were created and clearly defined:
  + SG-ALB: Allows inbound HTTP/HTTPS from the internet
  + SG-Frontend: Allows inbound HTTP only from the ALB SG
  + SG-Backend: Allows access only from SG-Frontend
  + SG-RDS: Allows PostgreSQL traffic only from SG-Backend
* IAM Role is scoped with least-privilege permissions:
  + S3:GetObject, S3:PutObject, rds-db:connect
* Subnet isolation is maintained to ensure that backend and database are not publicly accessible.

**6. Monitoring & Logging**

* **Amazon CloudWatch** was integrated into the architecture:
  + Collects ECS metrics and alarms
  + Tracks RDS performance and backup status
  + Triggers scaling actions via alarms
  + Collects logs from ECS tasks
* Arrows from CloudWatch clearly show its connection to ECS services, RDS, and Auto Scaling.

**7. Redundancy & High Availability**

* Every tier of the architecture (frontend, backend, RDS) is spread across **multiple AZs**.
* Load distribution, failover routing, and autoscaling enable high uptime and resiliency.
* **Backup and restore policies** for the database ensure disaster recovery.

**Key Components Implemented**

| **Layer** | **Components** |
| --- | --- |
| **Network** | Custom VPC (10.0.0.0/16), 2 Public Subnets, 2 Private Subnets across 2 Availability Zones, Internet Gateway, NAT Gateways, Route Tables |
| **Compute** | Amazon ECS (Fargate) for containerized services: Frontend (public subnet), Backend (private subnet) |
| **Storage** | Amazon RDS for PostgreSQL (Multi-AZ), Amazon S3 for media asset storage with versioning and lifecycle policy |
| **Security** | Security Groups (ALB, ECS, RDS), IAM Roles for ECS task permissions, S3 Bucket Policies, Encryption enabled for RDS and S3 |
| **Scalability** | ECS Auto Scaling Groups with CPU-based policies (Min: 2, Max: 4), distributed services across AZs |
| **Monitoring** | Amazon CloudWatch for ECS metrics, RDS monitoring, and auto scaling triggers; alarms and logs configured |
| **Backup & Recovery** | RDS Backups with Point-in-Time Recovery (PITR); S3 versioning for object-level recovery |

## Architecture diagram:

A computer screen shot of a diagram

AI-generated content may be incorrect.

## References (APA 7 Format)

* Amazon Web Services. (2024). Amazon ECS Documentation. <https://docs.aws.amazon.com/ecs/>
* Amazon Web Services. (2024). AWS Well-Architected Framework. <https://docs.aws.amazon.com/wellarchitected/>
* Amazon Web Services. (2024). Amazon CloudWatch Documentation. <https://docs.aws.amazon.com/cloudwatch/>