CSC11006 - INTRODUCTION TO CLOUD COMPUTING SERVICES PROJECT

PROJECT1: Build a Scalable Web Application with Cloud Backend

I. General information

ID:	LAB1
Period:	3 weeks
Deadline:	
Team	Group of 3 students

II. Outcome:

This lab will adapt to these following course's outcomes:

- G2.1, G2.2, G2.3, G3.1, G6.1

III. Describe:

Develop and deploy a simple, scalable web application using the **free tier of AWS or Azure**. The project will integrate load balancing, auto-scaling, cloud monitoring, and cost management to demonstrate real-world cloud service usage while staying within free-tier constraints.

A. Technical Requirements

1. Application Design

- **Frontend Development**:
  - Create a responsive web application using **HTML**, **CSS**, and **JavaScript**.
  - Application Example: **Task Manager** or **Personal Blog**.
    - Allow users to:
      - Add, edit, and delete tasks or blog posts.
      - View tasks/blogs in a list format.
      - Filter tasks by due date or priority.
- **Backend Development**:
  - Build a backend API using **Node.js** or **Python Flask/Django** to handle CRUD operations for the application.
  - API Endpoints:
    - GET /tasks – Retrieve all tasks.
    - POST /tasks – Add a new task.

        - PUT /tasks/:id – Update a task.
        - DELETE /tasks/:id – Delete a task.

---

2. Cloud Infrastructure Setup

- **Frontend Hosting**:
    - Host the frontend on:
        - **AWS**: S3 Bucket with static website hosting enabled (Free Tier: 5 GB storage, 2,000 PUT, 10,000 GET requests).
        - **Azure**: Blob Storage with static website hosting enabled (Free Tier: 5 GB storage, 20,000 read/write operations).
- **Backend Hosting**:
    - Deploy the backend API on:
        - **AWS**: EC2 (t2.micro instance, Free Tier: 750 hours/month).
        - **Azure**: App Service or Virtual Machines (B1S instance, Free Tier: 750 hours/month).

---

3. Database Integration

- Use a cloud database to store application data.
    - **AWS**:
        - Relational Database: RDS (Free Tier: 20 GB MySQL/PostgreSQL).
        - NoSQL Option: DynamoDB (Free Tier: 25 GB storage, 25 RCU/WCU).
    - **Azure**:
        - Relational Database: SQL Database (Free Tier: 250 GB storage in basic tier).
        - NoSQL Option: Cosmos DB (Free Tier: 400 RU/s and 5 GB storage).

---

4. Load Balancing

- Configure load balancing to distribute traffic across multiple backend instances.
    - **AWS**: Elastic Load Balancer (Free Tier: 15 GB of data processing).
    - **Azure**: Basic Load Balancer within the same Virtual Network (Free).

---

5. Auto-Scaling

- Set up auto-scaling to dynamically adjust the number of backend instances based on traffic.
    - **AWS**: Auto-Scaling Group with scaling policies:
        - Add a new instance when CPU > 70%.
        - Remove an instance when CPU < 30%.

    - **Azure**: Virtual Machine Scale Sets:
        - Scale between 1-2 instances based on CPU utilization.

---

6. Monitoring and Cost Management

- **Cloud Monitoring**:
    - Monitor resource usage (e.g., CPU, memory) and set up alerts.
        - **AWS**: Amazon CloudWatch (Free Tier: 10 custom metrics, 1,000 API requests).
        - **Azure**: Azure Monitor (Free Tier: 5 GB data ingestion/month).
- **Cost Management**:
    - Track and analyze resource usage to ensure it stays within free-tier limits.
        - **AWS**: AWS Cost Explorer for usage alerts.
        - **Azure**: Azure Cost Management to monitor expenditures.

B. Deliverables

1. **Web Application**:
    - A fully functional web application with frontend, backend, and database integration.
    - Load balancing and auto-scaling enabled for backend services.
2. **Monitoring Dashboard**:
    - CloudWatch (AWS) or Azure Monitor dashboards with metrics like CPU, memory, and network utilization.
3. **Cost Report**:
    - Detailed cost report showing that the project stayed within free-tier limits.
4. **Documentation**:
    - A step-by-step guide for deploying the application, including:
        - Frontend and backend setup.
        - Database configuration.
        - Load balancing and auto-scaling configuration.
        - Monitoring and cost tracking.
5. **Source Code**:
    - A GitHub repository containing:
        - Frontend code (HTML, CSS, JavaScript).
        - Backend code (API implementation).
        - Deployment scripts or Terraform templates (optional).

C. Evaluation Criteria

1. **Functionality** (40%):
    - Application performs CRUD operations seamlessly.

    - Integration between frontend, backend, and database works correctly.
2. **Cloud Infrastructure Setup** (30%):
    - Load balancing and auto-scaling are correctly configured.
    - Monitoring dashboards and alerts are properly set up.
3. **Cost Efficiency** (20%):
    - Resources stay within free-tier limits without incurring additional costs.
4. **Documentation and Presentation** (10%):
    - Clear and well-structured documentation with visuals (e.g., architecture diagram, screenshots).

Timeline

Week	Tasks
Week 1	Develop frontend and backend APIs locally.
Week 2	Deploy backend to cloud (AWS EC2/Azure App Service). Configure database integration.
	Implement load balancing, auto-scaling, and monitoring. Host the frontend in S3/Blob Storage.
Week 4	Perform testing, finalize documentation, and submit all deliverables.

Free Tier Resource Limits Summary
AWS Free Tier

Service	Free Tier Limit	Purpose in Project
EC2	750 hours/month (t2.micro instance)	Host the backend API.
S3	5 GB storage, 2,000 PUT and 10,000 GET requests/month	Host the static frontend files.
Elastic Load Balancer (ELB)	15 GB data processing/month	Distribute traffic across backend instances.
Auto-Scaling	Free scaling within the total EC2 hours limit	Scale backend instances based on demand.
RDS (Relational Database)	750 hours/month, 20 GB storage	Store application data (MySQL or PostgreSQL).
DynamoDB (NoSQL)	25 GB storage, 25 RCU and WCU	Alternative database for NoSQL requirements.
CloudWatch	10 custom metrics, 1,000 API requests/month	Monitor resource utilization and set alerts.
Cost Explorer	Free	Track usage and ensure project stays on budget.

App Service	1 GB storage, 60 CPU minutes/day	Host the backend API.
Blob Storage	5 GB storage, 20,000 read/write operations/month	Host the static frontend files.

Load Balancer	Free for basic tier within Virtual Network (VNet)	Distribute traffic across backend instances.
Virtual Machine (B1S)	750 hours/month	Host the backend API (alternative to App Service).
Virtual Machine Scale Sets	Free scaling within the free-tier VM hours limit	Scale backend instances based on demand.
SQL Database	Free tier, 250 GB storage in basic tier	Store application data (relational database).
Cosmos DB (NoSQL)	400 RU/s provisioned throughput, 5 GB storage	Alternative database for NoSQL requirements.
Azure Monitor	5 GB data ingestion/month	Monitor resource utilization and set alerts.
Cost Management	Free	Track usage and ensure project stays on budget.

Important Guidelines

1. **Monitor Usage**:
    - Regularly track free-tier usage through **AWS Billing Dashboard** or **Azure Cost Management**.
2. **Efficient Resource Use**:
    - Avoid creating unnecessary resources. Terminate unused instances and services.
3. **Stay Within Limits**:
    - For compute services (EC2 or App Service), ensure usage doesn't exceed 750 hours/month.
    - Use small datasets and limit file sizes to stay within storage and database free-tier limits.
4. **Utilize Local Environments**:
    - Develop and test locally whenever possible to reduce cloud resource usage.

This **Free Tier Resource Limits Summary** ensures students can complete the project without incurring costs, provided they adhere to these guidelines.