HW2 Design Document:

Written by Moshiko (Moshe) Hassan

The project is divided into different scale levels, in this implementation we will implement the system to support 1000 concurrent users, our first version is composed by two Lambdas, Elastic Cache, DynamoDB, and S3 bucket.
When scaling up we would transform from lambda to ECS or EKS depending on the scale and we will separate the main code into microservices and jobs that will replace the lambdas. Additionally, we will replace the dynamoDB with RDS or self-managed DB which could scale up at a cheaper price.

Scaling Discussion with Pricing:

Users	Compute	Storage	Total Cost	Approximate Users Served
1000	AWS Lambda (~$0.40)	ElasticCache ($15), DynamoDB ($1500), S3 (~$0.023)	~$1515.40	1000
10,000	ECS (~$365)	ElasticCache ($100), DynamoDB ($15000), S3 (~$0.23)	~$15465.23	10,000
Over 1 Million	EKS (~$9000)	ElasticCache ($1000), RDS ($1050), S3 (~$2.30)	~$16052.30 - $20000	Over 1 Million

1000 Users

1. **Compute:**
    - **2 x AWS Lambda:**
        - Assumption: 1000 requests per minute, 1 million requests per month.
        - Cost: $0.20 per 1 million requests + $0.00001667 per GB-second.
        - Approximate monthly cost: $0.20 + (1 million * 128MB/1024MB * 1ms * $0.00001667) = ~$0.40
2. **Storage:**
    - **ElasticCache:**
        - Assumption: Cache data for active chats.
        - Cost: Approximately $15 per month for cache.t2.micro.
    - **DynamoDB:**
        - Assumption: 1 read and 1 write per user per minute.
        - Cost: $1.25 per WCU and $0.25 per RCU.
        - Approximate monthly cost: $1.25 * 1000 + $0.25 * 1000 = ~$1500.
    - **S3:**
        - Assumption: 1GB data storage.

        - Cost: $0.023 per GB.
        - Approximate monthly cost: ~$0.023.
3. **Total Cost for 1000 Users:**
    - Compute: ~$0.40
    - Storage: ~$1515
    - **Total: ~$1515.40**
4. **Number of Users Served:**
    - AWS Lambda can handle 1000 users with the given configuration.

10,000 Users

1. **Compute:**
    - **ECS Cluster:**
        - Assumption: 10,000 requests per minute.
        - Cost: $0.04 per vCPU-hour and $0.005 per GB-hour.
        - Approximate monthly cost: 10 instances of t2.micro (each with 1 vCPU and 1GB memory).
        - Monthly cost: $0.04 * 730 * 10 + $0.005 * 730 * 10 = ~$365.
2. **Storage:**
    - **ElasticCache:**
        - Assumption: Larger cache size required.
        - Cost: Approximately $100 per month for cache.m3.medium.
    - **DynamoDB:**
        - Assumption: 10 read and 10 write per user per minute.
        - Cost: $1.25 per WCU and $0.25 per RCU.
        - Approximate monthly cost: $1.25 * 10000 + $0.25 * 10000 = ~$15000.
    - **S3:**
        - Assumption: 10GB data storage.
        - Cost: $0.023 per GB.
        - Approximate monthly cost: ~$0.23.
3. **Total Cost for 10,000 Users:**
    - Compute: ~$365
    - Storage: ~$15100.23
    - **Total: ~$15465.23**
4. **Number of Users Served:**
    - ECS Cluster can handle 10,000 users with the given configuration.

Over 1 Million Users

1. **Compute:**
    - **EKS:**
        - Assumption: 1 million requests per minute.
        - Cost: $0.10 per hour per cluster + instance costs.
        - Approximate monthly cost: 100 instances of t3.medium (each with 2 vCPU and 4GB memory).
        - Monthly cost: $0.10 * 730 + ($0.0416 * 730 * 100) + ($0.0059 * 730 * 400) = ~$9000.
2. **Storage:**
    - **ElasticCache:**
        - Assumption: Larger cache size required.
        - Cost: Approximately $1000 per month for cache.r5.large.

- **RDS:**
  - Assumption: High throughput.
  - Cost: $0.048 per hour for db.t3.medium (30 instances).
  - Approximate monthly cost: $0.048 * 730 * 30 = ~$1050.
  - DBA - approx 5-10K $
- **S3:**
  - Assumption: 100GB data storage.
  - Cost: $0.023 per GB.
  - Approximate monthly cost: ~$2.30.

3. **Total Cost for Over 1 Million Users:**
   - Compute: ~$9000
   - Storage: ~$2052.30
   - **Total: ~$11052.30**
4. **Number of Users Served:**
   - EKS Cluster can handle over 1 million users with the given configuration.

Project Design:

Project structure:

1. messaging_system/
   a. app/
      i. __init__.py
      ii. main.py
      iii. backup_lambda_function.py
      iv. lambda_function.py
      v. config.py
      vi. schemas.py
      vii. crud.py
      viii. database.py
      ix. models.py
      x. requirements.txt
   b. packages/python
   c. terraform/
      i. main.tf
      ii. backup.tf
      iii. lambda.tf
      iv. variables.tf
      v. outputs.tf
      vi. provider.tf
   d. .gitignore
   e. README.md

Functionalities

- Register a new user (generating a new id)
- Send a message to a user (via its id):

    - Messages are sent from a user to a user
    - Check if the user is blocked from sending
- Allow a user to block another user from sending a message to them
- Creating a group
- Adding / removing users
- Sending messages to a group
- Users can check for their messages
    - Assume that users will check messages at least once a minute - consider scaling factors here.

Compute:

1. lambda - python to contain the 1000 first users
2. ECS fargate or EKS - not implemented - for more than 1000 users, the application will be compiled into a container image instead.

Storage:

The storage would contain 3 levels of storage:

1. cache - hot \ latest cached highest speed
    a. The app should first ask the cache for the data.
    b. if the data is not loaded then the app will ask the DB.
    c. for every user the cache should contain the following:
        i. only active chats in the last hour if none, or the user is inactive we should keep his latest 5 chats by the date
        ii. every chat object should contain
            1. an index of start and end messages by incremental id to be indexed in the DB
            2. the last X messages - X is configurable in the backend
2. DB - hot \ moderate storage
    a. Everything in the cache should be backed in the DB.
    b. For every user should contain the following static data
        i. list of chats indexed by user id
        ii. list of groups indexed by user id
        iii. for each chat group and private should contain the following:
            1. and index stop and start
            2. the last Y messages - Y is configurable in the backend
3. s3 - cold storage
    a. for each user should contain a backup in the following hierarchy
        i. /{user-id}/{year}/{day_of_year}/{hour}/{chat_id}
    b. The data should have retention for Z days - where Z is in the config
    c. The data will be backed up every hour

The application should be in Python and need to include the following routes:

Functionalities Implementation

1. **Register a New User**
    - Endpoint: `POST /register`
    - Function: `register_user`

    - Description: Generates a new user ID and stores user details in the database.
2. **Send a Message to a User**
    - Endpoint: `POST /send_message`
    - Function: `send_message`
    - Description: Sends a message from one user to another. Checks if the sender is blocked by the receiver.
3. **Block a User**
    - Endpoint: `POST /block_user`
    - Function: `block_user`
    - Description: Allows a user to block another user from sending messages to them.
4. **Create a Group**
    - Endpoint: `POST /create_group`
    - Function: `create_group`
    - Description: Creates a new group.
5. **Add User to a Group**
    - Endpoint: `POST /add_user_to_group`
    - Function: `add_user_to_group`
    - Description: Adds a user to an existing group.
6. **Remove User from a Group**
    - Endpoint: `POST /remove_user_from_group`
    - Function: `remove_user_from_group`
    - Description: Removes a user from a group.
7. **Send Message to a Group**
    - Endpoint: `POST /send_message_to_group`
    - Function: `send_message_to_group`
    - Description: Sends a message to all members of a group.
8. **Get Messages**
    - Endpoint: `POST /get_messages`
    - Function: `get_messages`
    - Description: Retrieves messages for a user. Considers scaling for frequent checks.