Design an architecture to provide a membership export

The team discovered that users are interested in **exporting all of their memberships** from the system to run their own analysis once a month as a **CSV file**. Because the creation of the export file would take some seconds, the team decided to go for an **asynchronous process** for creating the file and sending it via email. The process will be triggered by an API call of the user.

Your task is to **map out a diagram** that visualizes the asynchronous process from receiving the request to sending the export file to the user. This diagram should include all software / infrastructure components that will be needed to make the process as stable and scalable as possible.

Implementation

The solution is already implemented in code (all except the email sending part). For this solution I have used NestJS background processors (@nestjs/bull and Redis combo). Redis is important for production as we want to have recovery.

1. Request for export would be received on api side
2. Info about request would be written in db (user id who requested, initial state). This info would be used if I need to get a job
3. The job would be scheduled by putting a new message with versioned data into the appropriate queue.
4. Job Processors will listen for new messages in the queue.
5. On message receival it will update job state to in progress,and send all membership data to the user who scheduled the job.
6. Finally, the job state will be updated to succeeded or failed in db.

System Design

Environment

We will use AWS for our system.

DB

1. For job status maintenance we should use NoSQL preferable DynamoDD as we can have expiration or a record.

2. For the membership data we can go with relation db as it will provide ACID. An integer primary key will be used for ease of internal data management and reference, while a GUID may be used for replication or to ensure uniqueness across distributed systems.
3. For user and auth we can use NoSQL as well for easy extensibility of a profile

API

Setup

Api will be hosted on Kubernetes with redis secrets injection. AWS kubernetes ingress controller will be used for managing elastic load balancer. We will need minimal two pods in Kubernetes deployment for API. For resilience we should implement multiple zones and do geo DNS with Route 53.

Authentication and Authorization

JWT authentication should be implemented with claim based authentication. Access token expiration should be 5min while refresh token 30days. With this approach we will not use session cookies and we can scale horizontally.

Documentation

Open API with Swagger UI is already included into API