Manic Compression: A Music Compression Service

Introduction

This project focuses on creating a cloud-based music file compression service. Users can upload music files, which are then compressed using Azure Functions. The process is coordinated by Azure Service Bus, ensuring efficient and scalable handling of file processing tasks. The compressed files are made available for download or streaming.

The idea is to be able to quickly compress share music files with other users, perhaps with a "live" emulation of a listening party.

Objectives

1. Allow users to upload music files to Azure Blob Storage.
2. Use Azure Service Bus to coordinate the compression process.
3. Implement Azure Functions to handle file compression and/or format conversion.
4. Provide users with the ability to download or stream the compressed music files.
5. Implement additional features (stretch goals) to enhance user experience.

Architecture

Components

- **Azure Blob Storage**: For storing audio files
- **Azure Service Bus**: To manage the workflow of aduio file processing (event driven architecture)
- **Azure Functions**: To perform the compression (and possibly other fun algorithms) of audio files.
- **Azure Kubernetes Or Azure Apps Service**: For scalabale application deployment (server deployment, frontend deployment, database deployment and user session as stretch goal)

User Workflow

1. User uploads a music file to Azure Blob Storage.
2. A message is sent to Azure Service Bus, triggering an Azure Function.
3. The Azure Function retrieves the file, runs the compression algorithm, and re-uploads the compressed file to Blob Storage.
4. Azure Service Bus is notified upon completion, which then triggers a notification to the user.
5. The user can download or stream the compressed music file.

Stretch Goals

User Authentication and Data Persistence

- **Azure SQL Database / CosmoDB / Container running PG?**

- idea is to have user auth and persist user data across sessions

Chat Feature for User Interaction

- **Azure Communication Services**: chat feature possibly w/ video?

Additional Algorithms That Can Be Run As Azure Functions

- Automated Mastering (EQ, Compress, Normalize, Limit)
- Reverb / Echo / Delay
- Paulstretch / silly effects

Additional Features

- AI audio analyzer (https://learn.microsoft.com/en-us/azure/media-services/latest/analyze-video-audio-files-concept)
- Spectral Analysis / Visualization of Wave Form
- Batch processing (upload multiple files at once)
- Audio adjustment presets / user-defined presets
- etc etc etc

Azure Service Descriptions

Azure Functions

Azure functions is a service that offers serverless computing for several programming languages, deploying code in the form of functions without managing the underlying hardware. These functions are designed to respond to events such as HTTP requests, message queues, etc. They are stateless, meaning that they are independent and can easily scale. The cost is based on the usage, which means that it can be cost-effective for sporadically executed functions. In summary, Azure Functions allows developers to write business logic code, without being concerned about managing infrastructure.

Azure Service Bus

Azure service bus provides a communication channel for distributed applications. It enables asynchronous communication between application components. The communication is done through publish/subscribe topics and message queues. It provides some reliability mechanisms, such as dealing with network failures and retry policies. It also offers the possibility to group messages into a single session, ensuring the order of the messages. In summary, Azure Service Bus allows developers to decouple the system components by using a reliable asynchronous communication channel.

Azure Database for Postgres

Azure Database for Postgres is a relational database service in the Microsoft cloud that is based on the open source PG database. It delivers high availability in the cloud, data protection w/ automatic backups and pointin-time restore, elastic scaling, monitoring, automation, and pay-as-you-go pricing. These are several of the benefits of using a managed DB service vs. running PG in a virtual machine.

Cost Estimation

Developing this as a proof of concept, and with only two users, is going to be very cheap. Looking at the cost breakdown using the pricing calculator, we're looking at a conservative estimate of \$13.16 for a month of uptime.

Note that this number supposes that we are using the free tier of Azure App Service or keeping the AKS Kubernetes cluster in a down state except when demonstrating. This number also supposes we *aren't* using a self managed Postgres service, which would make a lot more sense given the scope of this project (if we did, our total cost would be less than a dollar). Add \$3 to the total if we use the communications service as part of our stretch goals.

So to summarize:

- \$13.16 total if we use managed RDMS
- < \$1.00 if we self-manage RDMS
-
    - \$3.00 or so if we include our stretch goal of using Azure Communications service

Here's how we broke down these costs:

- Azure Functions: \$0.00
    - The first 400,000 GB/s of execution and 1,000,000 executions are free.
    - 512MB memory x 2 minutes execution time x 666 executions per month = \$0.00
- Azure Service Bus: \$0.05
    - 1 million operations per month = \$0.05
- Azure Storage: \$0.59
    - 10 GB capacity
    - 10,000 write operations
    - 10,000 read operations
    - 10 GB index
- Azure Postgres = \$12.41
    - 1 server x 1 month
    - may not be necessary (why not use a container running PG?)
- Stretch Goal Service: Azure Communications
    - 50 VOIP calls w/ 2 participantsd, 15 minutes each = \$3.00