Go exercise - Chat

Revisions

Audience

This document is designed for developers to test their capabilities in the GO language and Javascript. The exercise also uses NSQ Queue to test the developers capabilities to work with new technologies.

Preface

In this exercise developers will build a chat application using 3 microservices and a front end application using NSQ queue.

Exercise

Build a chat cloud application using 3 microservices written in GO, a front end in Javascript and to work with NSQ queue.
The exercise comes to show the ability to handle applications written as microservices working on a cloud environment environment, with a frontend.

The application will be comprised of the following microservices:

<u>Authentication service:</u>
This microservice will hold users information and will give token if the user is signed in. Also it will provide users information based on the token given for user validation.

The user information (including password and username) can be hardcoded into the application for the sake of this drill.

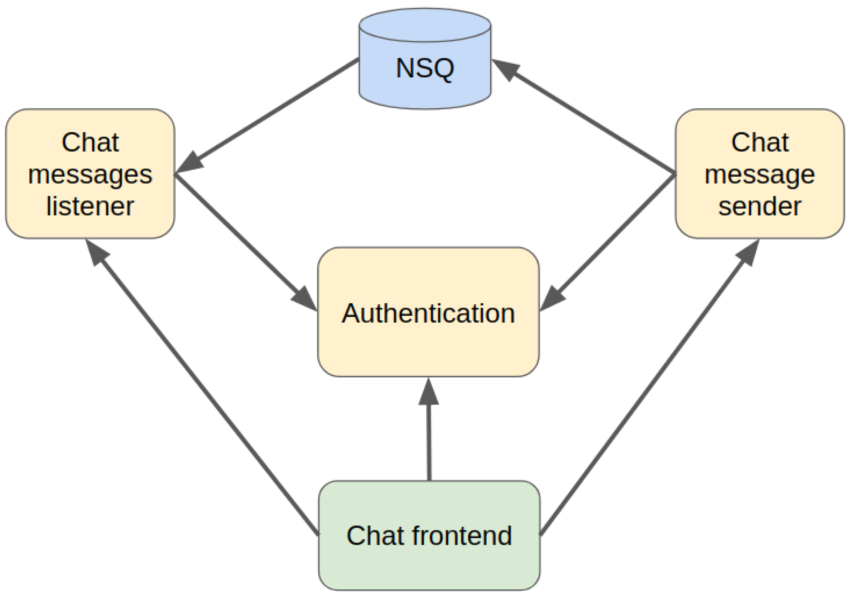
<u>Chat Writer:</u>
This microservice will recieve messages sent by the frontend using an API and send it to the NSQ queue. All message requests will need to be authenticated through the Authentication service to make sure the user indeed logged in the system.

<u>Chat listener:</u>
This microservice will listen to a NSQ queue for new messages. The service will have an API to retrieve all the messages of the chat by the frontend.

<u>Frontend:</u>
The frontend should be written in VueJS, ReactJS or AngularJS.

The chat need to have a login screen. After authentiacation it should give a clear view of the chat messages, refresh automaticly and be able to send messages.

The following is a schematic diagram of the application architecture:



All Microservices should save their port number and connection to other microservices and environment variables (Required). Docker files will be appriciated to each microservice.

Guidelines

The following are the goals of this excercise:

- **A working code.**
  A code that runs is preferred than complicated code with good intentions but without execution.
- **Clear writing**.
  We look for clean clear coding to be easily red. While comments is appriciated, it is prefered that the code would be readable by itself.
- minimum use of heavy packages and frameworks for go microservices.
- **Cloud oriented**.
  All configuration should be red externally preferably from environment variables.

- **Simple installation instructions**
  The application should run on a different environment than your computer, create idempotent microservices.

Please save all your code in a git repository of your choice and leave a documentation if there are any special instructions.