Instructions

- Solve the problems on the following pages.
- There is no time limit or necessary right answer.
- We are looking at the quality of your solution and the way in which the problem has been solved. Try to make it as easy as possible to understand - feel free to add comments, etc.
- We will evaluate this test with the following parameters:
  - **Quality of the code: you should be able to follow good practices.**
  - Communication and thought process! Write a few paragraphs explaining your thought process at the end of the project and include it in your README.
  - Simplicity of the solution: don't add features that have not been requested and don't over-engineer the solution.
  - Modernity of the code: we would like to see usage of modern paradigms, like async, f-strings, etc. (as relevant)
- It is recommended that you work through the parts sequentially, and document your progress and thought process at the end of each part. You do not need to get through all the parts - as long as we can see the details of how you work on the parts you have done, it is perfectly fine!
- Create a repository for your submission and add us to it before you begin:
  - Github: fabiotamagno / nisargatman
  - Gitlab: fabio24 / nisargatman
- Most importantly, enjoy it!

Problem

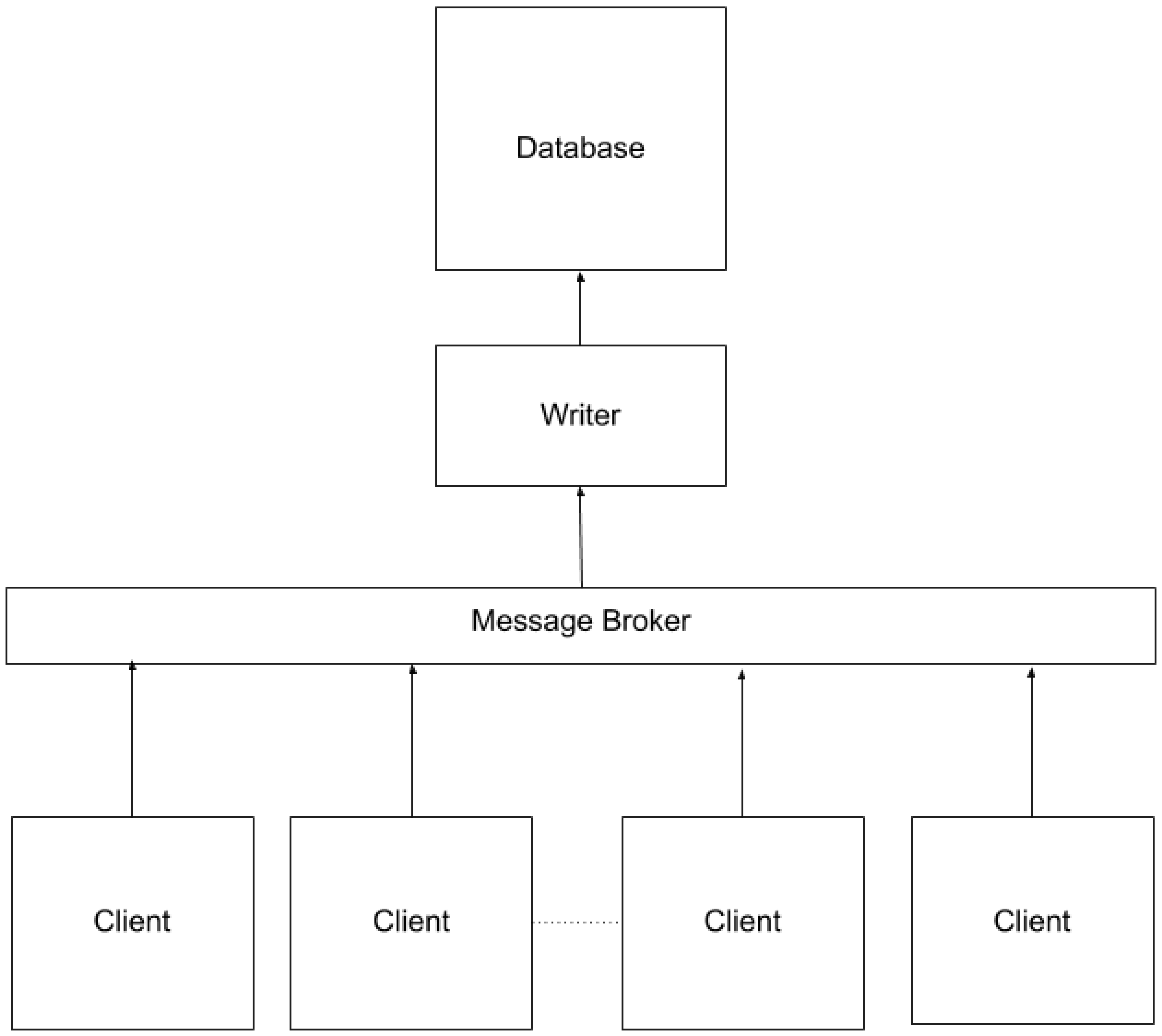
- We want to implement the entire stack of a very simple backend application
- The application we will work on is a simpler variant of Wikipedia that only holds structured information about geographical entities. People around the world can access and update the information.

Part 1

- Create a series of tables in an `SQL database` table to hold the following information:
    - Continent Name (eg. Asia, Europe, etc.)
    - Country Name (eg. UK, USA, Germany, Australia, etc.)
    - City Name (eg. London, Singapore, etc.)
    - Continent Population
    - Country Population
    - City Population
    - Continent Area (sq. meter)
    - Country Area (sq. meter)
    - City Area (sq. meter)
    - City Number of roads
    - City Number of trees
    - City Number of Shops
    - City Number of Schools
    - Country Number of Hospitals
    - Country Number of National Parks
    - Country Number of Rivers
    - Country Number of Schools
- The application should also hold some data validation logic. For example, if there are 2 cities in one country, the sum of the population of the 2 cities cannot be greater than the population of the country. Amend the database to hold this kind of information for a few simple validation rules.
- Write scripts in Python to create, update, delete specific values

Part 2

- Given the volume of reads, writes and updates, the requests have to be managed via a message broker. In this case, we are going to use Kafka.
- The architecture we want for our application is something like this



- You should already have written the code for the database and the writer. Implement the code for the client and amend the code for the writer to allow the information to flow through.

Part 3

- We want to implement logging in the application using Logstash (from ELK). Add this into all parts of the application as you deem necessary.
- Write the logging as a modular service that can be amended / extended easily.
- Containerize the application with docker and docker-compose.

Part 4

- Explain briefly how you designed the system for the validation checks and added that into the database.
- Explain the choices you made in designing the code for the client and message broker related services.
- How would you deploy this into a cloud environment in practice? Would you change any of the components to improve stability / robustness?