MapReduce Implementation on google cloud to find anagrams of books.

This program attempts to use the MapReduce programming model to find out the anagrams in a list of books. The program uses an object orientated model to accomplish this. This allows components to be recyclable so that data can be split into different instances of the steps, book anagrams.py, contains the overarching logic of the program, it calls the relevant class methods, and manages the retrieval and output of the data to buckets.

Main.py defines the entry point of the python program, it sets up the listener in the main function, to listen to requests made. Run() sets up the stop_words list for use by the main program, and calls the main MapReduce program.

Book_anagrams.py, contains 4 constants defined, these contain the buckets of the input and output, as well as the project to be used. This can be changed when running on a different project, or other inputs are to be used. During development the costliest function to run was the mapping function, this is likely as a result of all the iterations of the map and its substitutions. Efforts was made to multi process this function up to n number of cores. By splitting the data to run on each core available. However, upon implementation, this method was slower, this could be because of the implementation, however it seemed likely that the excess in calling all the functions on the array likely contributed to the increase in execution time and overhead. Instead, data is split and ran on its own, resulting in a slight increase in performance. The class also contains functions which use googles api to download data from a bucket into string for processing, and to output a string of data to the bucket.

The mapper.py expects a list of strings, when the map function is called, it cleans each of the lists to extract out only the words, which are then added to a new array for mapping. Once all these words are mapped to a value, the get function can be called to get the mapped words.

The shuffler.py expects a list of mapped values, when the shuffle function is called, it forms a tuple of the hashed value, the key and the original word. Before the word is hashed however, it is sorted alphabetically, this is so that the hash generated can be compared to other anagrams directly, as they are all sorted. Once the tuple has been formed, the list is then sorted by using the first hashed value of each tuple. The get method then returns the shuffled data.

Finally, reducer.py uses a dataclass in group.py, this dataclass (group) stores the number of anagrams in the list, the shuffled list is then iterated through. As it's ordered, the execution time is expected to be minimal, as any matching anagrams, will be stored next to each other, leveraging the design benefit of the shuffler to hash alphabetically ordered words. Finally, once the reducer has finished, a custom function called anagrams, can be called which applies the custom logic, to reduce the list to only include groups, whose list of

anagrams count is greater than one. Finally, the output function can then be called, which will reduce the list so all duplicates can be filtered out, as depending on the success of clean function called in mapper, there could be some residual words.

The program uses several, different serverless cloud technologies, these technologies include, google cloud run, google cloud storage and google cloud repository. The whole program is encapsulated inside of a docker container, this was to provide the same environment for development as the environment that would be deployed to cloud run. Inside of this container all python packages required are installed. Inside of main.py I set up a listener for requests, when a request is received it runs the MapReduce model. This was then deployed to google cloud run, using the google container registry, to allow for immediate deployment whilst developing. Google cloud run was chosen because it is serverless, allows for docker integration and can scale up to as many instances as required by the traffic. This abstracts the scaling away from the program, in addition it provides functionality to run the program using a single curl command instead of a more complicated run process. Google cloud was used as a storage medium, for storing the input text, and bucket for outputting. These buckets are defined as constants in book_anagrams.py, so they can be changed to use different locations.

The main core design idea was to make the code as robust as possible, and as fast as possible, data streaming was explored, however this would've led to increased overhead in joining lists back together, it therefore seemed more efficient and quicker to split inside the costliest function and perform the other steps separately. Another key idea in design, was the ease of setup docker provided an easy platform to ensure robust code, by keeping the execution environment the same. By building the modules object orientated it also allowed development to function independently and provide greater amounts of checks to ensure that they were working as expected.