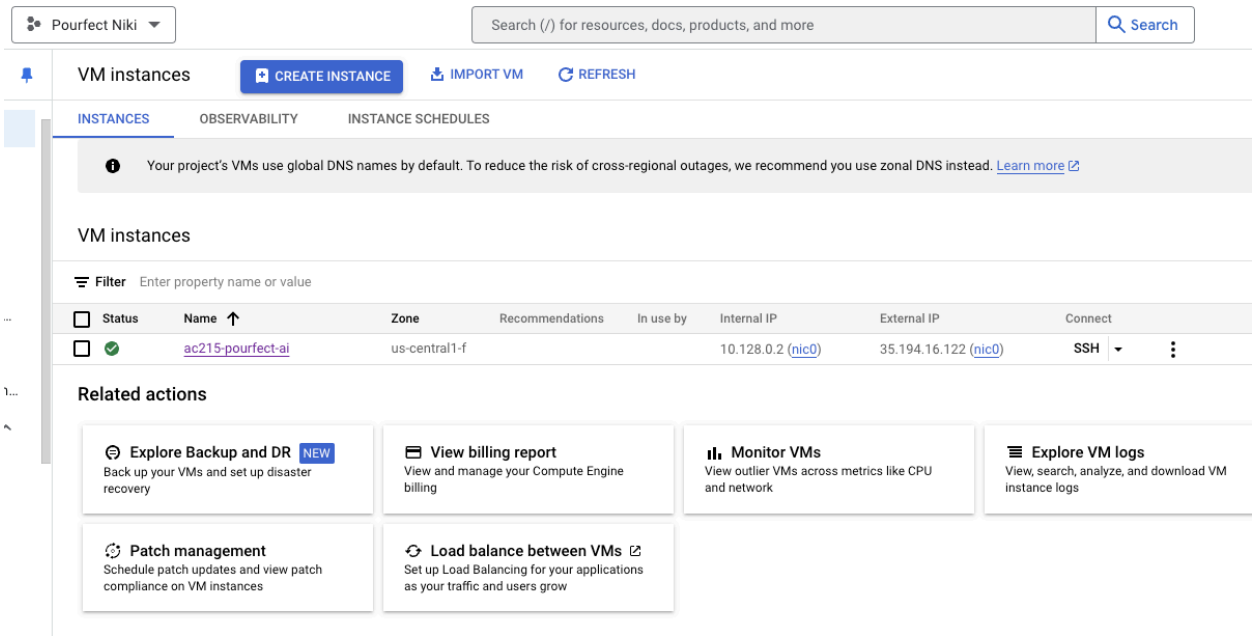
Pourfect-AI Milestone 2 Documentation

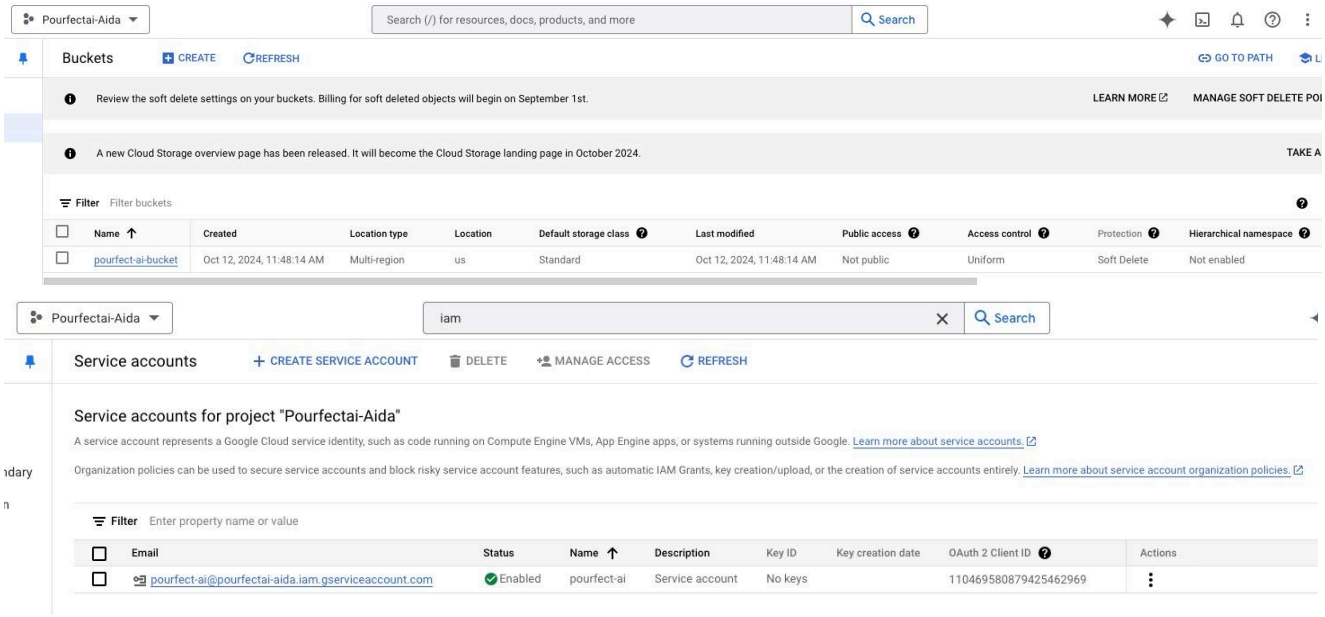
10/10/2024 – Virtual Machine

- Every person on the team created their own project called “Pourfect <NAME>”
- One person of the team, Niki Ekstrom, set up an instance of the virtual machine on GCP with the region us-central1 (Iowa) and e2-medium (2 vCPU, 1 core, 4 GB memory)
    - Monthly estimate of $25.46
    - Did not change service account
    - Named the instance “ac215-pourfect-ai”
- All other team members were given “Compute Admin” and “Service Account User” roles, so that they could SSH into the instance



10/12/2024 – GCP Bucket

- One team member, Aida York, created a GCP bucket called pourfect-ai-bucket with the standard settings in the project called “Pourfectai-Aida”
- We then created a service account with “Storage Object Admin” role
- Aida then granted access to the GCP buckets to all other team members



10/12/2024 – Docker Containers

- We are creating 2 primary docker containers for the project:
    - datapipeline - will handle all data loading and pre-processing
        - Generated the Dockerfile, Pipfile, and Pipefile.lock
            - Dockerfile outline came from tutorials we completed during class
            - We added required packages such as pandas, sci-kit-learn, and google cloud packages to the Pipfile
            - Created the Pipfile.lock from running pipenv lock
            - Made sure we were able to build the container running:
                - docker build -t datapipeline .
        - Created a docker-shell.sh to automatically run the docker image
    - models - will handle running our LLM and RAG pipeline
        - Created Dockerfile based on the LLM tutorial
            - In the Dockerfile, copied the Pipfile and Pipfile.lock from the data pipeline directory
        - Created Dockerfile.sh
            - Ensure that the Docker container can run using just this file

- To build and run the docker container for the data-pipeline:
    - cd into the data-pipeline folder: run ./docker-shell.sh which will allow you to automatically build and run the docker image.
- To get the data from our gcp bucket:
    - Run python dataloader.py
    - Once you see the message : “Downloaded raw_data.csv to app/raw_data.csv”

10/15/2024 – Data Versioning

- As part of our data versioning strategy, we will create folders inside the GCP bucket that are labeled as V1, V2, etc. so we can keep track of all of the data and how it has changed
- Our GCP bucket is structures as follows:
    - Pourfect-ai-bucket
        - Clean_data
            - V1
                - Finetuned - 4,096 max token size, 20 training examples
            - V2
                - Finetuned - 8,192 max token size, 20 training examples
            - V3
                - Finetuned - 8,192 max token size, 50 training examples
        - Raw_data
            - V1
                - Text_data
- Within the clean and raw data folders we will have sub-folders according to the version of the data
- Raw Data
    - In raw data GCP folder (inside the project GCP bucket) we have a folder for the raw tabular data which is the data that was directly downloaded from https://huggingface.co/datasets/erwanlc/cocktails_recipe_no_brand
        - This data is a tabular dataset with a list of cocktail recipes as well as the ingredients that go into making them
    - Text Data
        - Inside the raw data and V1 folder we compiled text files for a set of documents (articles and books) related to cocktail making
        - The following links provide the documentation for where we got these sources from:
            - https://www.themixer.com/en-us/learn/cocktail-making/
            - CocktailsForNewbies.txt
        - https://downloads.ctfassets.net/b0q5etab7zkl/5LMMmjaDUeMrFGYRaY1OlL/4aa3f1b47aed4386c8cfbf5a01066c68/Seedlip_CocktailsAtHome_Ebook__1_.pdf
            - cocktails.txt

        - https://www.infobooks.org/pdfview/mocktails-mastery-drug-education-network-34/
            - Mocktails.txt
        - https://www.infobooks.org/pdfview/bartenders-holy-bible-unknown-34/
            - Bartender's Holy Bible.txt
        - https://abarabove.com/mocktails-beginners-guide/
            - Mocktails101.txt
        - https://makecocktailsathome.com/wp-content/uploads/2023/11/MakeCocktailsAtHome-Printable-Master-v4.pdf
            - Home_cocktails.txt
        - https://cooking.nytimes.com/guides/26-how-to-make-cocktails-and-mixed-drinks
            - CocktailsHowTo.txt
        - https://mixologo.net/wp-content/uploads/2020/07/The-Nomad-Cocktail-Book.pdf
            - NoMadCocktailBook.txt
- Clean Data
    - In clean data GCP folder (inside the project GCP bucket) we have a folder for the cleaned versions of the dataset
    - Processed_data.csv contains the tabular data with preprocessing to include dropping unnecessary columns, drop duplicates, and fixed column formatting, and dropped null values
    - Vectorized_data.csv contains the vectorized and chunked data that is ready to go into the RAG model
- Data Processing Files (Non-RAG)
    - dataloader.py
        - This file loads the raw tabular data from GCP
    - preprocess_data.py
        - This file loads the raw tabular data and pre-processes it by dropping columns, deleting duplicates, and formatting the columns

10/15/2024 – RAG Pipeline

- Data Pre-processing
    - The preprocess_rag.py file handles the data preprocessing to go from the raw text data to chunked and vectorized data that can be fed into the RAG model
        - This file loads the text data and chunks / vectorizes it
        - We use the Langchain library for text processing and chunking. We use the Document class as a way to store the text content and metadata from each text file. In order to chunk the data we then use the Recursive Character Splitter class from Langchain using a chunk size of 100 and chunk overlap of 20
        - We use the VertexAI TextEmbeddingModel to generate the text embeddings after the text data is chunked, and these are stored locally as a CSV file

        - The vectorized data is uploaded back to the GCP Storage Bucket for future access
        - We create connect to a **ChromaDB** instance and create a vector database where we insert the original documents with their metadata, as well as the generated text embeddings
- Generating response with rag
    - The model_rag.py file takes in a user query, and generates an LLM response using RAG
        - First, it embeds the query
        - Then, it retrieves the most relevant documents from the vector database (context)
        - It uses the gemini-1.5-flash-001 model with custom instructions that decide the LLM's tone and behavior
        - Using the context and query, the script uses this generative model to provide a response

10/17/2024 – Finetuning Pipeline

- Creating finetuning data
    - The finetuning_data.py file generates, prepares, and uploads cocktail-related question-answer datasets using Vertex AI's generative models
        - Sets safety settings to prevent model from generating harmful content
        - Generates 20 Q&A pairs about cocktails using the Gemini-1.5 model from Vertex AI
        - It consolidates the data and splits it into train and test data sets (jsonl files) that are uploaded to the GCS bucket
- Finetune the Gemini-1.5 model
    - The train_model.py file finetunes a generative model on Vertex AI
        - It finetunes the gemini-1.5-flash-002 model using the train and test datasets created in finetuning_data.py
        - The finetuned model performance can be viewed on Vertex AI
        - Also includes a chat function (not yet implemented) that connects to the deployed finetuned model endpoint on Vertex AI
        - It sends a user query to the finetuned model and generates the response