# Abstract

This report outlines the implementation and outcomes of a cloud system management project, divided into three assignments. The task involves creating a Cluster Manager (CM) using Python to manage Docker containers. The CM is designed to handle various functionalities, including creating, listing, running commands, stopping, and deleting containers within a cluster. The project also explores data processing and machine learning tasks, demonstrating the versatility of the implemented solution. The whole project can be fetched from [EXID-G/PBL-AI (github.com)](https://github.com/EXID-G/PBL-AI)

# Overview of the Task

The primary objective is to create a Cluster Manager capable of managing Docker containers within a cloud-like environment. The system should efficiently handle the creation, listing, execution of commands, stopping, and deletion of containers. Additionally, the project includes tasks related to data processing and machine learning using the CM.

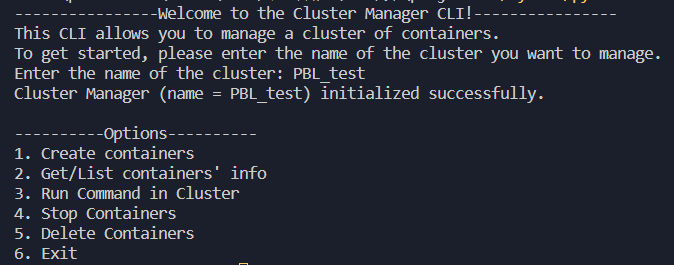
# Assignment 1 - Create Cluster Manager

A Python class named `*ClusterManager*` in `*ClusterManager.py`* is developed to interact with Docker containers. The CM supports creating a specified number of containers, listing running containers, executing commands in the entire cluster, stopping all containers, and deleting all containers. Besides, Extensive logging is implemented to track container events and statuses. In addition to logging specific events, the status of all containers in the cluster is also recorded every minute

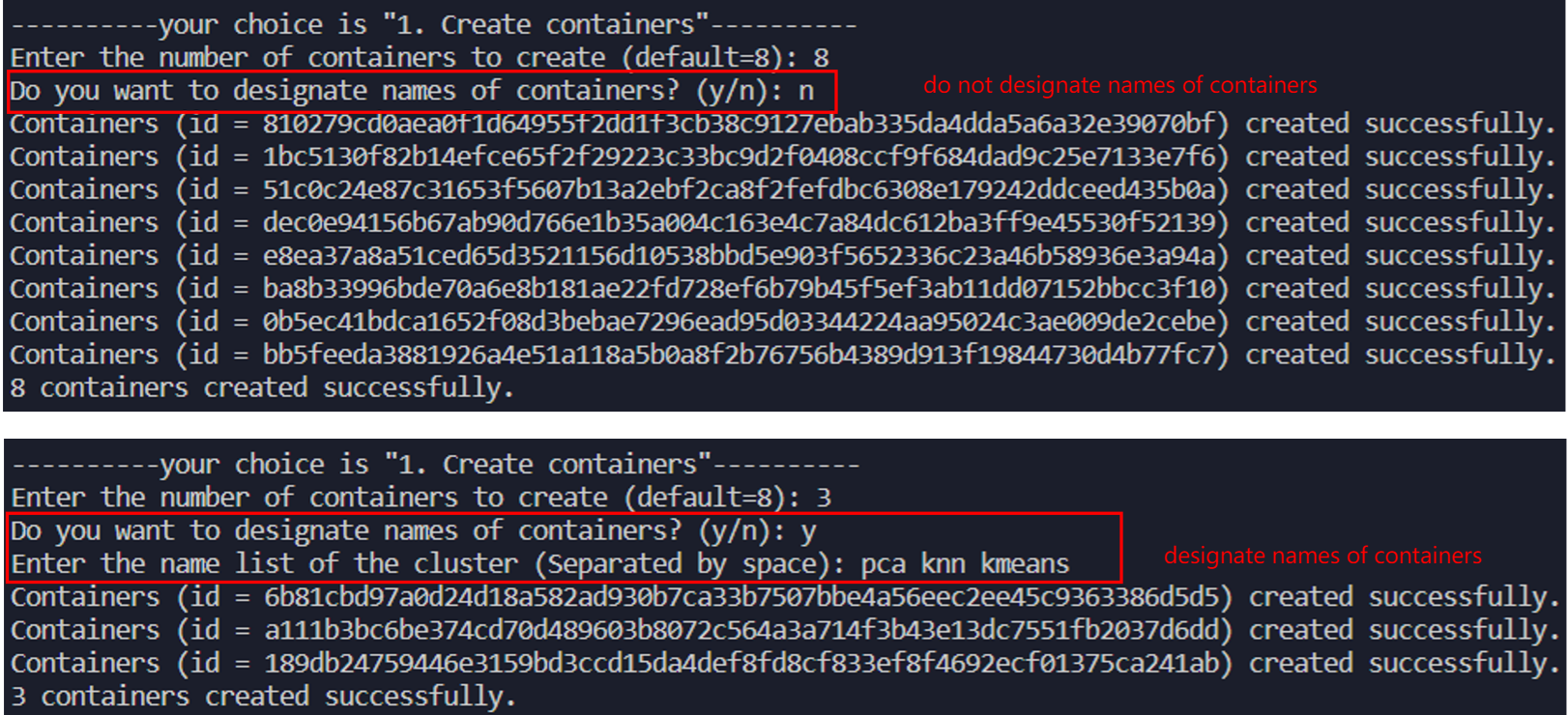
The CM is accessible through a Command-Line Interface (CLI), making it user-friendly. To start the CLI, the client can input the command `python Assignment1-CLI.py` in the terminal.

The implementations and snapshots of some functions are shown below :

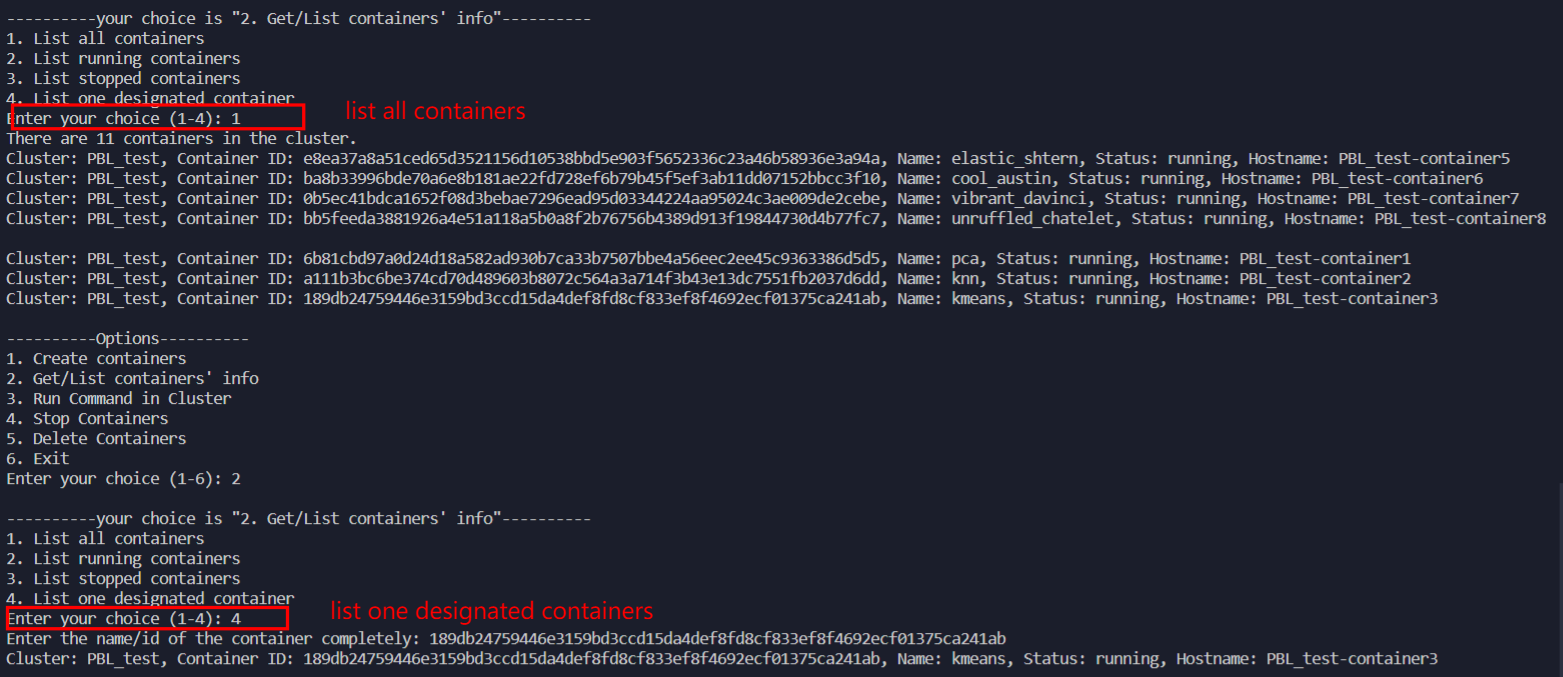
1. Initialization (*\_\_init\_\_*):
   1. Initializes the ClusterManager with a specified `*cluster\_name`*.
   2. Establishes a connection to the Docker daemon using the docker Python library.
   3. Set up logging to record events and container statuses.

**The main menu**

1. Cluster Creation (*create\_containers* and *\_create\_container*):
   1. Creates Docker containers with specified hostnames.
   2. Utilizes the docker.containers.run method from the Docker SDK to start TensorFlow containers. By default, the image used to create containers is `tensorflow/tensorflow`, which is sufficient for the whole project.
   3. Logs container creation events.

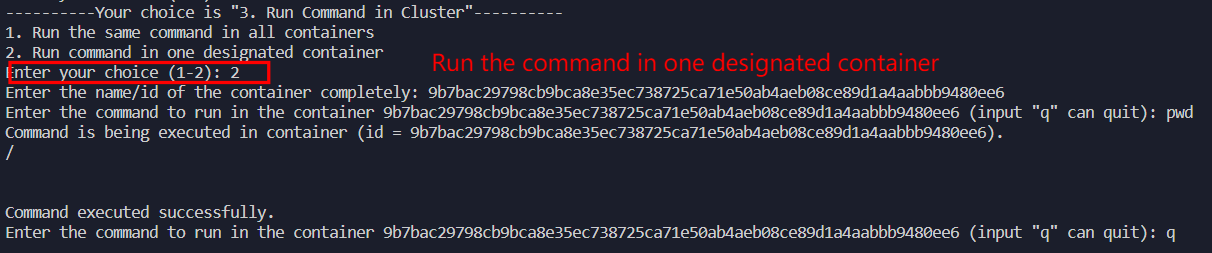


**create containers by not designating names and designating names**

1. Listing Containers (*list\_all\_containers, list\_running\_containers, list\_stopped\_containers*):
   1. Retrieves and displays information about all containers in the cluster.
   2. Differentiate between running and stopped containers.
   3. ****Provides details such as container ID, name, status, and hostname.

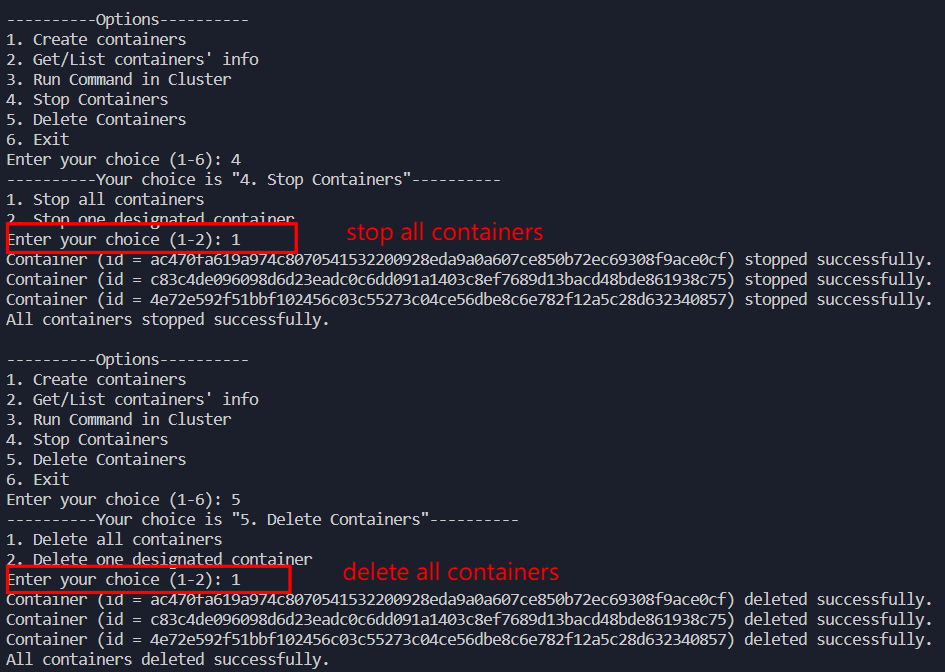
**List information of all containers or one designated container**

1. Executing Commands in Containers (*run\_command\_in\_cluster, \_execute\_command\_in\_container*):
   1. Runs a specified command in all containers or one designated container of the cluster. The client can run commands continuously until the `*q*` is entered.
   2. Handles container status checks, starts containers if necessary, and executes commands.
   3. Logs the output and exit code of the executed command.

****

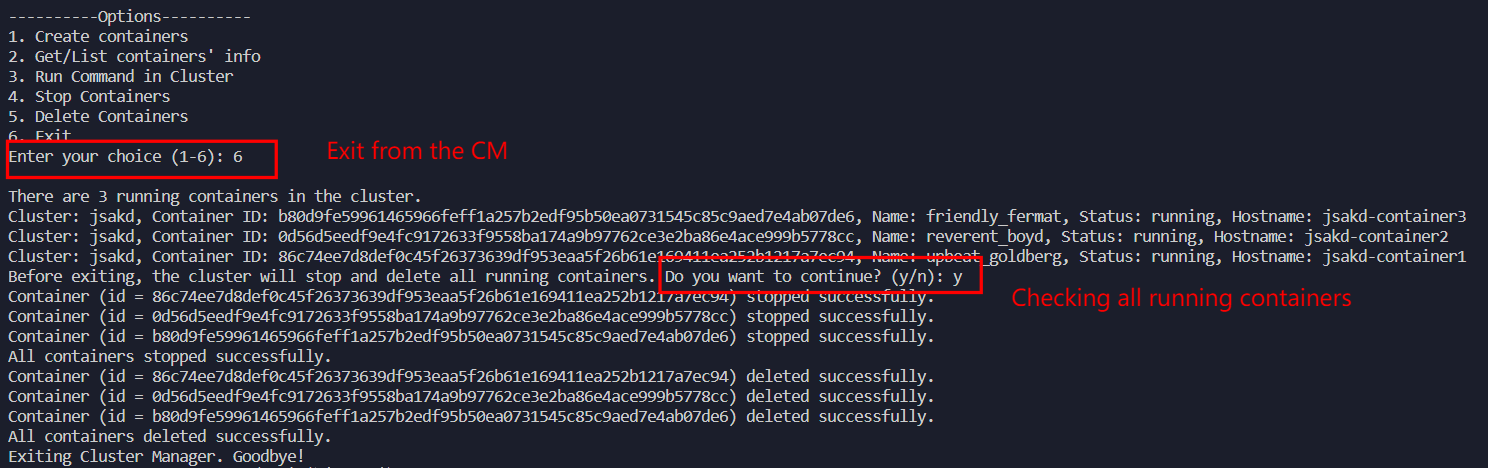
**run the command in one designated container**

1. Stopping Containers (*stop\_all\_containers, \_stop\_container*):
   1. Stops all containers or one designated container in the cluster.
   2. Logs the event of stopping each container.
2. Deleting Containers (*delete\_all\_containers, \_delete\_container*):
   1. Deletes all containers in the cluster.
   2. Removes container entries from the internal list.
   3. Logs the event of deleting each container.

****

**Stop and delete all containers**

If the client wants to exit from the program, he/she can enter a `6` into the terminal. After stopping and deleting all containers, the cluster manager will terminate.

****

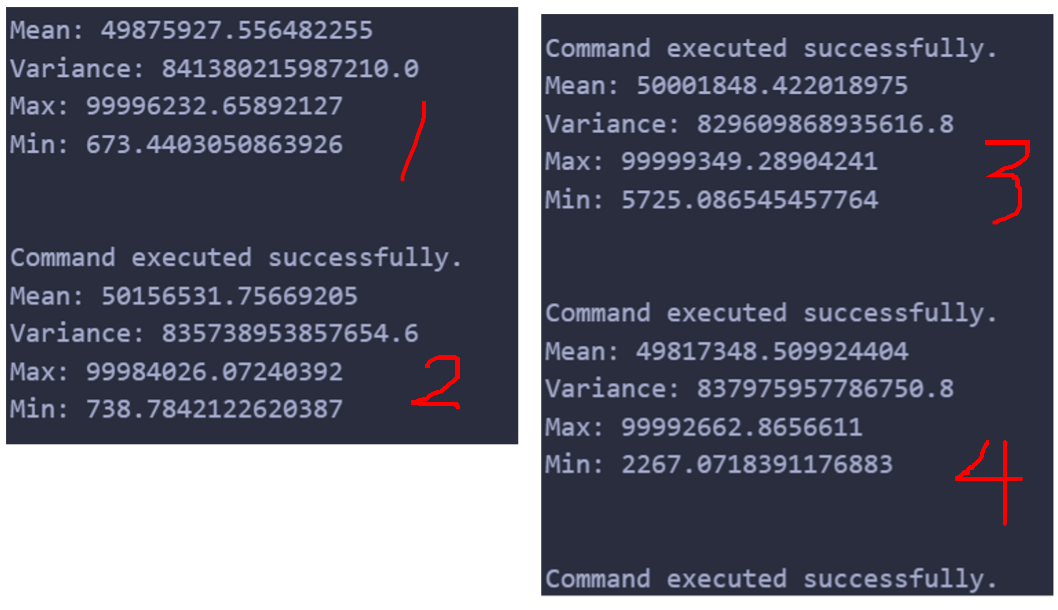
**Exit from the program**

# Assignment 2 - Data Processing in CM

The CM is extended to support data volumes, enabling shared data storage among containers. There are functions in the `*ClusterManager`* Class to implement parallel processing of a large array of data across multiple containers.

Data Processing in Parallel (*distribute\_and\_process\_data\_parallel, \_process\_data\_in\_container*):

1. Distributes a large array of data across containers for parallel processing.
2. Utilizes Python's concurrent.futures.ThreadPoolExecutor for parallel execution.
3. Executes a specified Python file in each container with a specific data chunk.
4. Handles container selection and chunk distribution.
5. Parallel Processing: Optimizes data processing by distributing tasks across multiple containers.



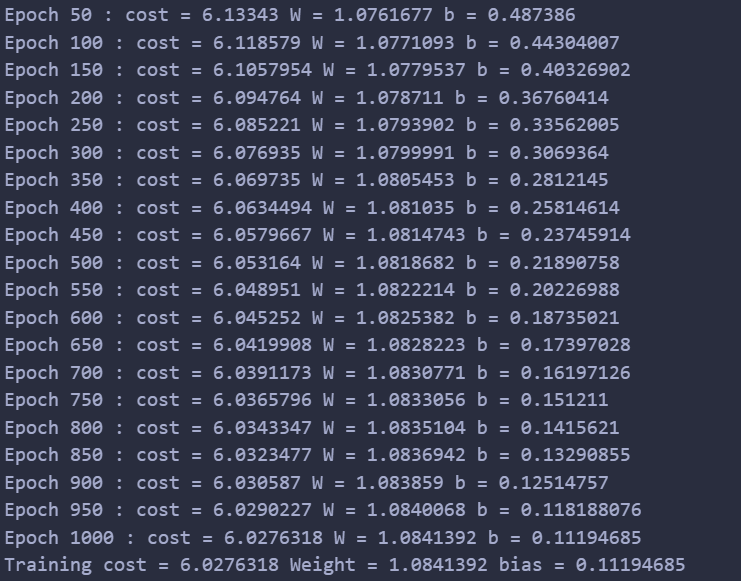
**The result of assignment 2**

The solution of Assignment 2 (consists of generating data, mounting files, and allocating data) is included in the file `Assignment2-processing.ipynb`. Here I’m not using the hostname to distribute data but passing data chunk indexes (defined in function ` *distribute\_and\_process\_data\_parallel`* ) to the container so that containers can get what they should process. For each of 4 containers, it will run the command `*python ass2\_volume/ass2\_data.pkl ass2\_volume/ass2\_process\_data.py {chunk\_index[0]} {chunk\_index[1]}*` to compute and get the result.

# Assignment 3 - AI Tasks in CM

The CM is extended to handle AI tasks by loading TensorFlow into Docker containers and A linear regression task using TensorFlow is executed across the cluster.

The solution is given in the file `*Assignment3-ML.ipynb*`. After generating the data, creating a container, and mounting files, the command `python /ass3\_volume/lreg.py` can be run to do the linear regression. I removed the section that draws images because matplotlib takes a lot of time to install.



**The result of assignment 3**

# Difficulty

To print out command-line statements of containers in real-time, I use `*container.exec\_run(command, stream = True)*` to detect each progress. However, `*exit\_code*` is always None when 'stream = True', which is necessary for recording, and online solutions are not that useful. Besides. some people said 'It cannot return stream and exit code at the same time' (from [How to get exec\_run exit\_code? · Issue #1381 · docker/docker-py (github.com)](https://github.com/docker/docker-py/issues/1381)). So, I just gave up using the option `*stream = True*`. Hope that I can find the solution of this problem.

# Summary and Conclusion

The project successfully achieves the creation of a versatile Cluster Manager, showcasing functionalities ranging from basic container management to data processing and machine learning tasks. The implementation is user-friendly, integrating seamlessly with Docker tools and providing valuable insights through extensive logging.

The assignment-wise breakdown highlights the advantages of the implemented solutions, such as data sharing, parallel processing, and TensorFlow compatibility. However, some limitations, such as security measures and potential size constraints, should be considered for future improvements.