# Yet Another Kafka

**YaK** or **Yet Another Kafka** is one of the projects that can be chosen as part of the Big Data Course (UE20CS322) at PES University.
It involves setting up a mini-Kafka on a student's system, complete with a Producer, Subscriber and a Publish-Subscribe architecture.

## Project Objectives and Outcomes

- You will gain a deeper understanding of Kafka Architecture.
- You will gain a deeper understanding of principles such as Fault-Tolerance, Recovery and Leader Election.
- At the end of this project, you will be able to setup a mini-Kafka on your system; complete with a mini-zookeeper, multiple Kafka Brokers, Producers and Consumers.

## Technologies/Languages to be Used

- You are free to use any language such as Python, Go, Java, among others.
- Ensure that the chosen language supports all the required functionality.
- You are allowed to use any external libraries or APIs if required.

## Marks

- 10 Marks

## Deadline

- Final code submission to GitHub should be done by 11:59PM on November 27th.
- Commits after the deadline will not be considered.
- The last commit before the deadline will be used for a plagiarism check.
- Rules of plagiarism check remain the same as the previous assignments.

## Evaluation

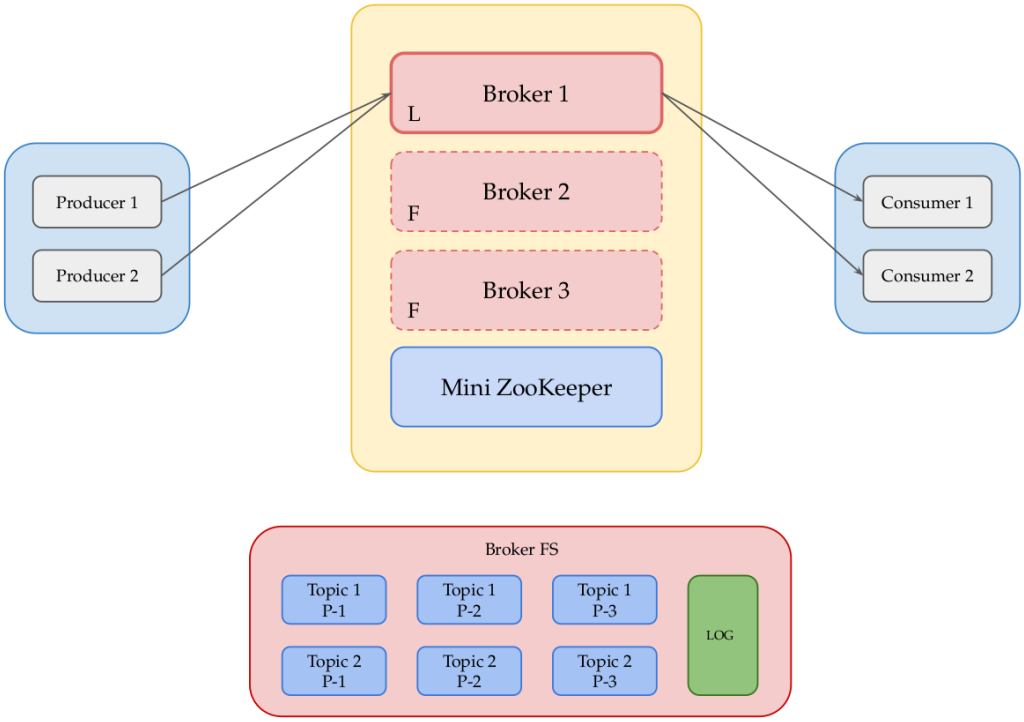
- Last working week of the semester, i.e., November 28th to December 2nd.
- Evaluation will be done in class hours and consist of project demonstration and viva.
- All the team members are **required** to be present and participating.

# Project Specification

## High-Level Overview

- You are required to set up a mini-Zookeeper, multiple Kafka Brokers, one of which is a leader, and multiple Producers and Consumers.
- The number of Producers and Consumers must be dynamic and not hard-coded, i.e., the user must be able to specify the number of Producers and Consumers.
- The number of topics must also be dynamic, the user should be able to create and delete topics on demand.

To help you get started with the project, the following sections will provide a detailed description of all the individual modules.




## mini-Zookeeper

- The main purpose of the mini-Zookeeper is to monitor the health of the Kafka Brokers and keep a track of the Leader.
- The method of health monitoring is up to you, but it must be able to detect a failure of a Kafka Broker.
  - Some examples are: Heartbeats, Polling, etc.
- In case of a failure of a Kafka Broker, the mini-Zookeeper must elect a new Leader.
  - The choice of a Leader Election algorithm is given to you.
- An assumption that is being made is that the mini-Zookeeper is always running and will never fail.

## Kafka Brokers

- The Kafka Brokers are responsible for creating and managing topics.
- They are also responsible for registering/de-registering any Producers and Consumers.
- In case a topic does not exist when a Producer/Consumer has been started, the Broker must create one.
- The Broker is also in charge of storing the messages that are being published by the Producers.
  - The Topics must be stored as directories in a file-system.
  - You are required to also dynamically create partitions for each Topic. The partition function that will be used for this is upto you.
  - The performance of the chosen partition function will be evaluated.
- They are also in charge of keeping track of which messages have reached a particular consumer and which have not.
- You are required to have **3 Kafka Brokers**.
- The Leader is responsible for handling all Publish Operations. Consume Operations can be handled through either the Leader, or any one of the replicas.
- The Leader maintains a log of all operations which must be replicated on the Followers as well.
- In case a publish or a consume operation is encountered when the Leader has died, the remaining Kafka Brokers must be able to handle this situation.
  - How this situation is handled is upto you. An example could be to ask the producer/consumer to retry in `x` amount of time, assuming a leader has been chosen by then.

## Kafka Producer

- The Producer is responsible for publishing messages to a Kafka Topic.
- The number of producers **must NOT be hard-coded**.
- The Producer should be able to register to any Kafka Topic or should notify the Broker to create one if necessary.
- The Producer must co-ordinate with the Broker to keep track of all the messages that a Broker has received.
  - In case the Broker does not receive a message, it must re-transmit the message.
  - This can be done by obtaining an acknowledgement from the Broker, on receiving a message.

## Kafka Consumer

- The Consumer is responsible for receiving messages from a Kafka Topic.

- The number of consumers **must NOT be hard-coded**.
- The Consumer should be able to register to any Kafka Topic or should notify the Broker to create one if necessary.
- A consumer must be able to receive all the messages from the time of creation of a Kafka Topic by using the `--from-beginning` flag.
  - When this flag is used, the Consumer should be able to obtain all messages that were sent since the time of creation of the topic.
  - If this flag is not used, the Consumer should only obtain messages sent after the Consumer registed with the topic, **not** since the creation of the topic itself.
- The Consumer must co-ordinate with the Broker to keep track of all the messages that it has received.
  - This can be done by sending an acknowledgement to the Broker on receiving a message.

## Scope and Simplifying Assumptions

- You are **NOT** required to handle mini-Zookeeper failures.
- The Leader Election algorithms can be as simple or as complex as you wish.

## Tips and Tricks

- A module like `subprocess` in python, or a similar library in the programming language of your choice can be used for Producer, Consumer and Broker nodes.
  - In addition, students can explore options such as Docker Containers, VMs and so on for the same if they wish to do so.
- Edge cases within the scope of the project should be taken care of.

**NOTE:** Plagiarism of any form will **not be tolerated**. Using existing solutions on GitHub or co-operating with other teams is strictly not allowed.