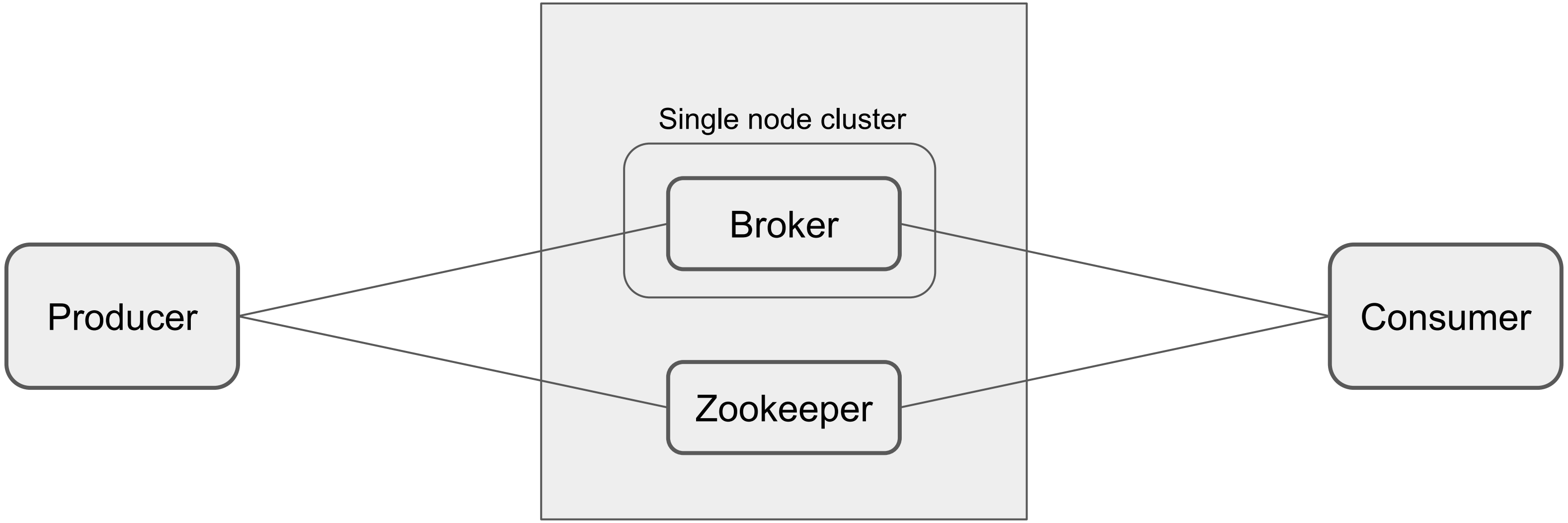
Exercise 4

Distributed Software Systems — Prof. Paolo Ciancarini
Università di Bologna
A.Y. 2023/2024
Riccardo Scotti (0001060133)

Architecture of the application

For simplicity reasons, everything will be running on a single machine, but Kafka is, obviously, designed specifically for distributed contexts.



Starting up the system

1) Starting up the zookeper `$bin/zookeeper-server-start.sh config/zookeeper.properties`
2) Starting up the broker `$bin/kafka-server-start.sh config/server.properties`

The zookeeper does not do much in this setup, because the configuration is extremely simple; nonetheless, a broker will not work properly without exchanging information with the zookeeper.

It is now possible to start consumer and producer in any order

Producer and consumer

Thanks to the Python package provided by Kafka, it is possible to create a producer and a consumer by creating an instance of classes KafkaProducer and KafkaConsumer.

When creating and using those objects, it is necessary to specify some information, such as the addresses of bootstrap servers (in this case only one) and the topic.

It is also possible to send data in a specific format by providing a serializing function; for this exercise, data are sent in JSON format.

Producer process

The producer process will perform a loop and send a message every 10 seconds.

This program, as simple as it is, is not so unrealistic: it may as well resemble a sensor which does not produce a large amount of information, whose data are then aggregated and processed along the pipeline.

Sending messages is done via the producer method `send`, that only requires to specify the topic and the message.

Consumer process

In order to consume the messages, it is required to iterate over the Consumer object.

In this case, the consumer simply prints the messages sent by the producer.

It is interesting to note that, even if the consumer is run *after* the producer has produced messages, it will still receive all those past messages, even if it were not up at the time. This demonstrate that Kafka allows not only space but also time decoupling.