Kafka is a Stream processing application that is great for messaging between different components in a system architecture.

Kafka was developed by LinkedIn as a means to pipe data streams to various components in there system.

Kafka was designed to be high-throughput(HT), low-latency(LL), highly available(HA).

Kafka was designed to run in clusters so that individual instances of Kafka servers could receive similar messages and still dispatch those messages to consumers as though it was a single Kafka server.

Clustering ensures that Kafka can provide it services with HT, LL, and HA.

Producers are Kafka entities that are responsible for pushing messages to Kafka

Consumers are Kafka entities that are responsible for receiving messages from Kafka Producers

Produces publish messages to Kafka Topics. Topics are the general concept of the message and all messages with the same topic are store together and published to all consumers of that topic.

Kafka stores its messages according to Topics which are constructed as a queue in memory.

Each new message of a certain Topic is stored at the next offset for the topic. Offsets start at 0 and are incremented from there. The oldest messages for the topic are stored for a configurable amount of time and will exist in the topic even after being consumed.

Topics can also be partitioned so that multiple queues are created to store the messages. The partitions still act as a single partition but consumers can dictate from which partition they would like to read from and also from which offset they would like to begin reading from. Topic partitions ensure LL and HT

Creating messaging using spring boot.

You will need a few things to make spring boot work with kafka.

1. An instance of Apache Kafka server running on your machine
2. A spring boot application with the spring-kafka dependencies
3. Configure a ProducerFactory to help build Kafka Producer entities
4. Configure a KafkaTemplate which will use the ProducerFactory to create messages and send them
5. Configure a ConsumerFactory to help build Kafka Consumers entities
6. Configure a KafkaListenerContainerFactory which will use the ConsumerFactory to create Listeners to accept messages
7. The Kafka template is used to send messages, but you will need a different object to receive the messages. Annotating a class or method with @KafkaListener would tell the ListenerContainer which method to invoke for certain topics.

Once you have all these configured it is a simple fact of using a KafkaTemplate to send a message to a topic. These messages can be anything including Java Objects since the message is serialized before being published to the consumer.