1. Graphical user interface, text

   Description automatically generated  
   
2. **Agenda**:
   1. Let’s try to understand the need for multi-threading with a realistic example.
3. 
4. Let’s assume we have a stock market data provider app as shown in the figure.  
   Graphical user interface, diagram

   Description automatically generated
   1. The app receives tick by tick stock data packet from the stock exchange over a TCP/IP Socket.  
      The data packets are arriving at high frequency.  
      So you decided to create a multithreaded Data Handler.
   2. The main thread listens to the socket and reads the data packet as it arrives.  
      But the main thread immediately hands over the packet to a different thread for sending the data to the Kafka Broker.   
      And the main thread again starts reading the next data packet.
   3. Other threads are responsible for the followings.
      1. Uncompressing the packet.
      2. Reading the individual msgs from the packet.
      3. Validating the msgs.
      4. And eventually sending to the Kafka Broker.
   4. Similar scenarios are applicable to other apps where data is arriving at high speed/frequency and you may need multiple application threads to handle such a load.
   5. Kafka Producer is thread-safe, so you can share the same KafkaProducer.java object across the multiple threads and send the msgs in parallel using the same **KafkaProducer.java** instance.  
      It is not recommended to create numerous KafkaProducer instances inside the same app instance.  
      Sharing the same KafkaProducer across multiple threads will be faster and less resource intensive.

