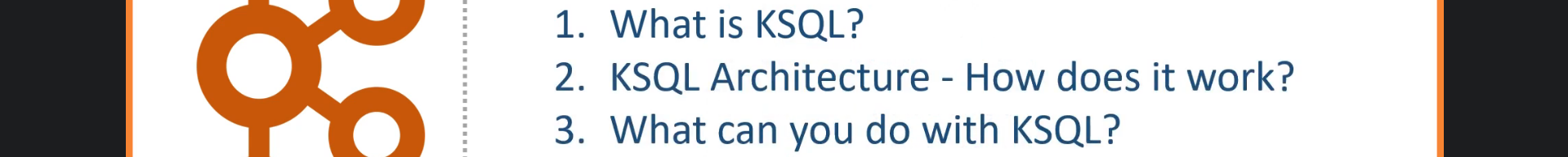
1. **Agenda**:
   1. Graphical user interface, text, application

      Description automatically generated
   2. We will give answers to the following questions:  
      
2. **KSQL**:
   1. 

Most of the things, you can do with **Kafka Stream**, can be done with **KSQL**.  
It means you can create **Scalable & Fault-Tolerant** **Stream** Processing workloads without the need to write code in a programming language like Java or Scala.

* 1. **KSQL** has two **operating modes**.  
     Graphical user interface

     Description automatically generated with medium confidence
     1. **Interactive Mode**:
        1. Uses either **CLI** or **Web-Based UI** to submit **KSQL** and get and immediate response.  
           **CLI** works like any DB SQL interface.
        2. **CLI Mode** is ideal for **Development Environment.**
     2. **Headless Mode**:
        1. The **Headless mode** is a **non-interactive mode** that allows you to submit your **KSQL file** which are executed by the **KSQL Server**.
        2. The **Headless** mode is ideal for the **Production Environment** whereas **CLI Mode** is ideal for **Development Environment.**

1.   
   Diagram

   Description automatically generated
   1. KSQL comes with three components.
      1. KSQL Engine:
      2. REST Interface:
      3. KSQL Client (CLI):  
         **KSQL Engine & REST Interface** together form the **KSQL Server**.  
         Text

         Description automatically generated with medium confidence
   2. **KSQL** can be deployed in one of the available modes 🡺 Interactive mode & Headless Mode.
   3. You candeploy multiple **KSQL** servers to form a **Scalable KSQL Cluster.**
   4. However, all the servers running in the **KSQL Cluster** must use the same deployment mode.
   5. **KSQL Engine** is the main component responsible for **KSQL Statements & Queries**.  
      Under the hood, the **KSQL Engine** is going to parse your **KSQL Statements** and build corresponding **Kafka Streams Topology** & run them as a **Streams Task** and  
      these Stream tasks are executed on the available **KSQL** Servers in the cluster.
   6. We can **dynamically** add more Servers to the cluster to scale out the cluster.
   7. Fault-Tolerant feature is inherent feature of the **Kafka Stream**.
   8. No need to mention that **KSQL Cluster** is different from **Kafka Cluster.**
   9. **KQL Server** will internally communicate with **Kafka Cluster** for reading and writing outputs.
   10. The **REST Interface** is to power the **KSQL Clients** and **KSQL** **Client** will send the received commands to **REST Interface** which will internally communicate with the **KSQL Engine** to execute your **KSQL Commands.**
2. 
   1. In nutshell, **KSQL** allows you to use your **Kafka Topic** as a table and fire SQL like queries over those **Topics**.  
      With that power in your hand, you can imagine the kinds of things you can do.
   2. Here are **some examples**:   
      Text

      Description automatically generated Text

      Description automatically generated
   3. So, the **KSQL** for **Kafka** is one step forward for **Kafka** to become a **Realtime Data Warehouse**.
   4. Those days are not too far when you might see **JDBC/ODBC connectors** being available for **KSQL** & **visualization tools** like **Tableau and QlikView** to start connecting with **KSQL**.

I am not predicting anything but that is the next step for **Kafka & KSQL**.