Questions

What are the components that are absolutely necessary to build any kind of system no matter the requirement or the scale or the users?

For a very basic system, we need the following components for sure.

Database - Persistence is something that is required for every backend application. So, A Database is utmost necessary
Server - For any system that needs to scale, we need to have a server, so that we can have an API driven architecture so that if the frontend changes, the new frontend can easily consume the existing API's.
Network - We need a Network as a component for the communication between the client and the server and between the server and the database.

Not necessary

Components like a frontend are not necessary because even a console will work or even a curl command will work.
Stuff like Load Balancers, Caches are there for efficient and highly scalable systems, they are not required for basic applications.

what load balancer do if one server fails?
If one server fails, a load balancer will typically detect the failure through a health check mechanism that periodically pings the server to ensure that it is still responding. Once the failure is detected, the load balancer will stop forwarding traffic to the failed server and redirect it to the remaining healthy servers.
There are several ways that a load balancer can handle a failed server, depending on its configuration and the specific implementation. Here are some common methods:

- Round-robin: Load balancer will simply skip over the failed server and forward requests to the next available server in the rotation.
- Weighted distribution: Load balancer can be configured to adjust the distribution of traffic to each server based on their capacity or health status. For example, if one server fails, the load balancer can increase the traffic to the remaining servers to compensate for the loss of capacity.
- Hot standby: In a hot standby configuration, the load balancer will maintain a backup server that is ready to take over if one of the active servers fails. The backup server can be configured to take over automatically, or the load balancer administrator can manually initiate the failover.

In general, the goal of a load balancer is to ensure that traffic is distributed evenly and efficiently across multiple servers.

What is Apache kafka? When should we use it?
Apache Kafka is a distributed streaming platform that is used for building real-time data pipelines and streaming applications. It allows multiple applications or components to send and receive high-volume, high-throughput data streams in real-time(Ex: stock trading system,gaming application,video streaming etc).
Here are some scenarios where Kafka can be a good fit:

- Real-time data processing: Kafka is well-suited for processing high volumes of data in real-time. It can be used to build real-time data processing pipelines that ingest data from multiple sources, process and transform the data in real-time, and then store the results in a database or data warehouse for later analysis.
- Stream processing: Kafka supports stream processing, allowing data to be processed in real-time as it is received. This makes it a good choice for building real-time applications that can perform complex data processing operations on large volumes of data in real-time.

- Distributed architecture: Kafka is built on top of a distributed cluster architecture, making it highly scalable and fault-tolerant. It can be deployed across multiple servers or clusters, providing scalability and fault-tolerance for processing high volumes of data.
- Message queueing: Kafka can be used as a high-performance message queue for decoupling message producers and consumers. It can store messages for a period of time before they are consumed, providing a buffer between message producers and consumers.
- Integration with other data sources: Kafka provides a number of built-in connectors for integrating with popular data sources and data sinks, such as databases, Hadoop, and Elasticsearch. This makes it easy to build end-to-end data processing pipelines that can ingest data from multiple sources, process and transform the data in real-time, and then store the results in a database or data warehouse for later analysis.

Here are some scenarios in which Kafka may not be the best choice:

- Small data volumes: Kafka is designed to handle large volumes of data. If you have small data volumes, Kafka may be overkill.
- Simple applications: If you have a simple application that doesn't require real-time data processing or asynchronous communication, you may not need Kafka.
- Limited resources: Kafka requires significant resources to operate, including storage, memory, and CPU. If you have limited resources, Kafka may not be the best choice.

Overall, Kafka is a powerful tool for building real-time data processing pipelines and streaming applications. Its distributed, fault-tolerant architecture and support for high-throughput, real-time data processing make it a popular choice for modern data architectures.

both web sockets and gRPC allows real-time communication then which one should i use in my application and when?

While both WebSockets and gRPC allow real-time communication, they serve different purposes and have different tradeoffs.

WebSockets are ideal for applications where real-time communication is critical and the communication is bidirectional between the client and server. WebSockets enable a persistent, full-duplex communication channel between the client and server, which allows for instant messaging, online gaming, and other real-time applications. WebSockets also have wide browser support and are easy to use.

gRPC, on the other hand, is a lightweight, high-performance RPC framework that is ideal for microservices architecture. gRPC enables efficient communication between microservices by using protocol buffers, a compact binary format for data serialization. It supports streaming, flow control, and other advanced features that make it suitable for building complex microservices-based applications.

In summary, if you need bidirectional real-time communication between the client and server, use WebSockets. If you need efficient communication between microservices, use gRPC.

If both gRPC and kafka are good fit for microservices communication then in which kind of application should I use kafka or gRPC?

If the application requires real-time communication with low latency and high throughput, then Kafka is a better fit. Kafka is designed to handle large amounts of real-time data streams and can support a high volume of messages. It provides durable message storage, fault-tolerance, and scalability. Kafka is commonly used for applications that involve stream processing, real-time analytics, and event-driven architectures.

On the other hand, if the application requires a fast, lightweight, and efficient communication protocol between services, then gRPC can be a better fit. gRPC uses a binary format and protocol buffers for efficient data serialization and transmission. It supports multiple programming languages and platforms, making it easier to integrate with existing systems. gRPC is commonly used for applications that require microservices architecture, where services need to communicate with each other efficiently and reliably.