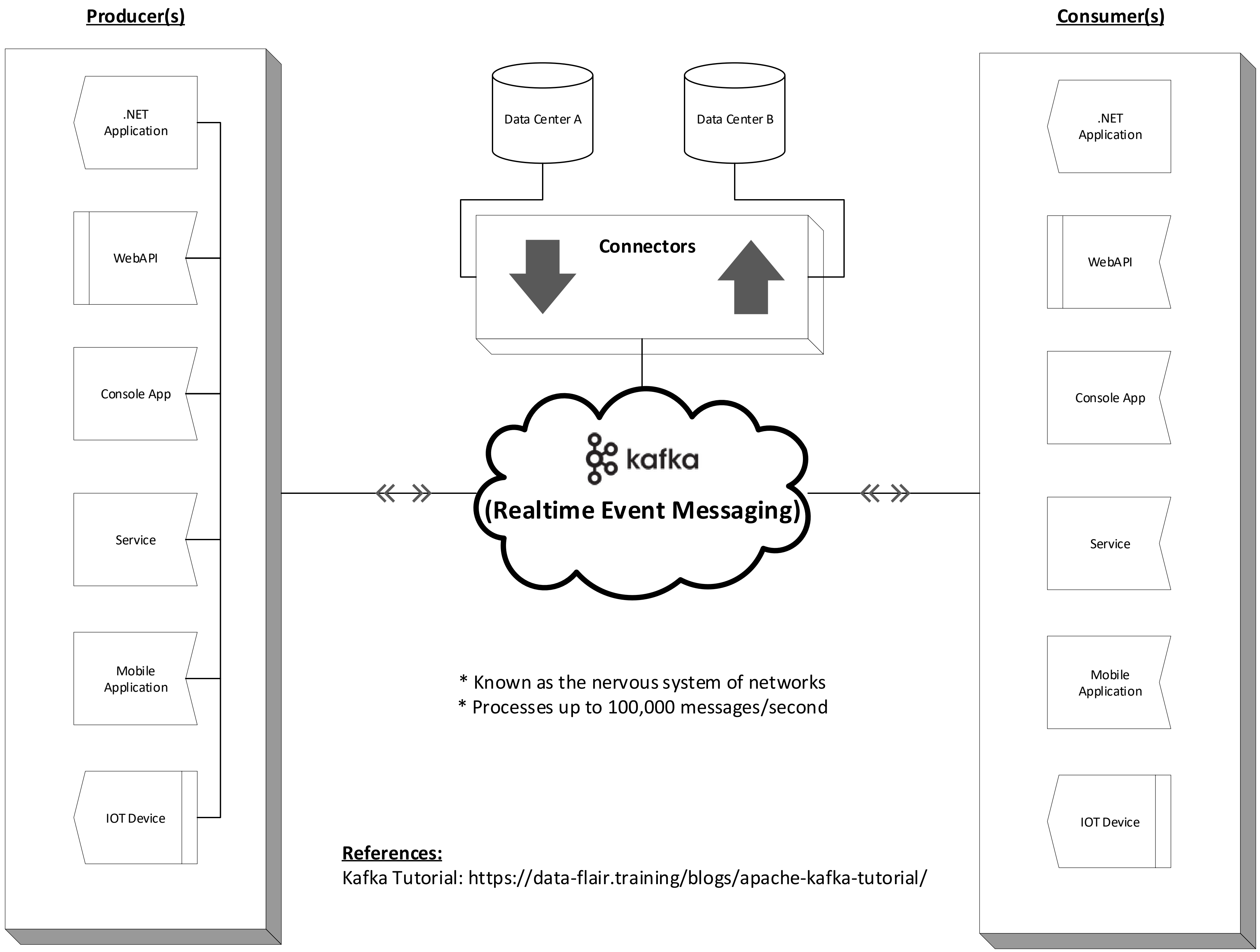


* Known as the nervous system of networks
* Processes up to 100,000 messages/second

References:
Kafka Tutorial: https://data-flair.training/blogs/apache-kafka-tutorial/

What are we trying to accomplish?

Inter-System communication

Centralized Event Transmission

Implementation Redundancy Reduction

Mass Usage & Scalability

Organized Computing

Producer/Consumer Design Pattern

The producer consumer pattern is a concurrency design pattern where one or more producer threads produce objects which are queued up, and then consumed by one or more consumer threads. The objects enqueued often represent some work that needs to be done.

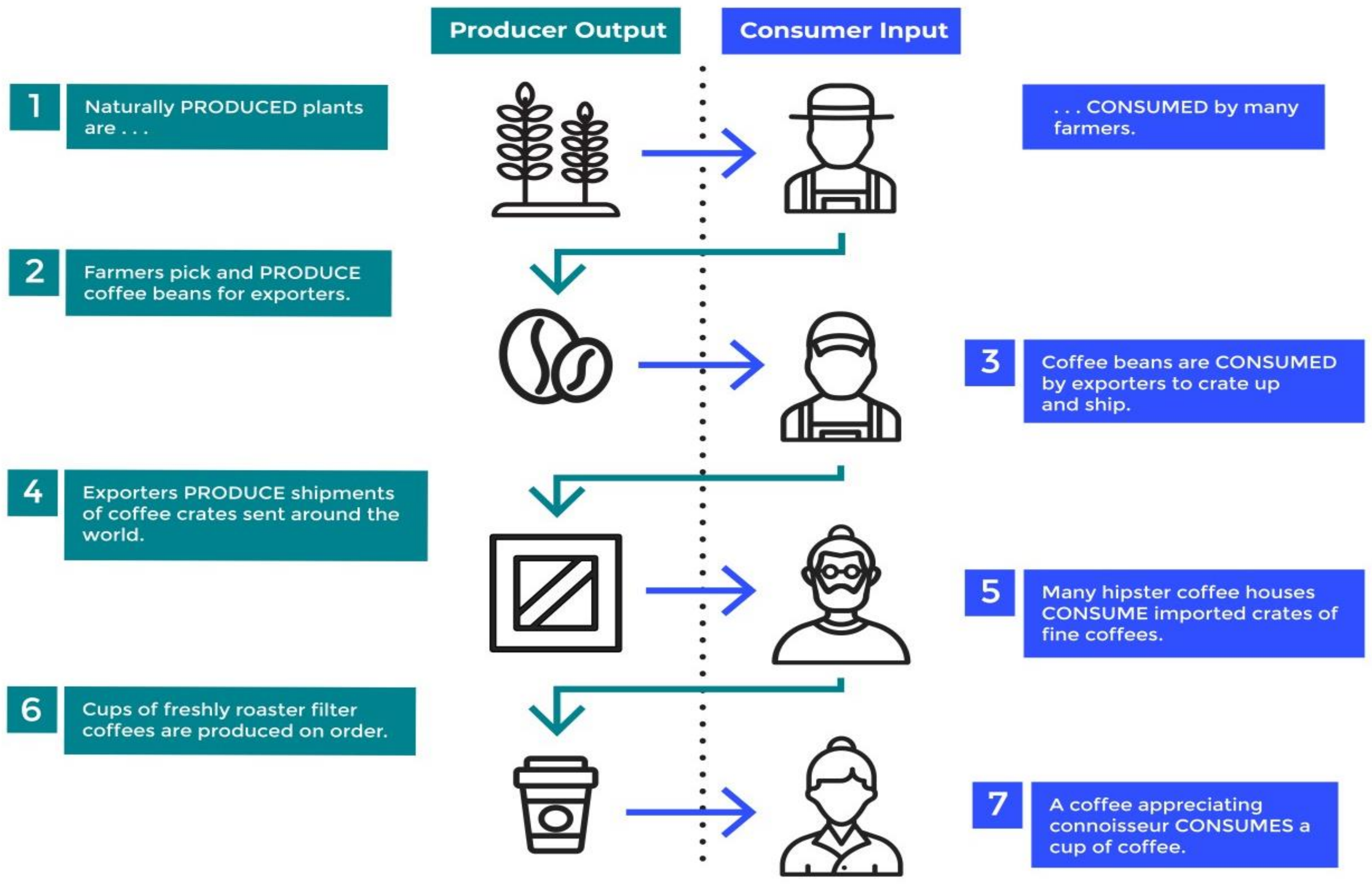
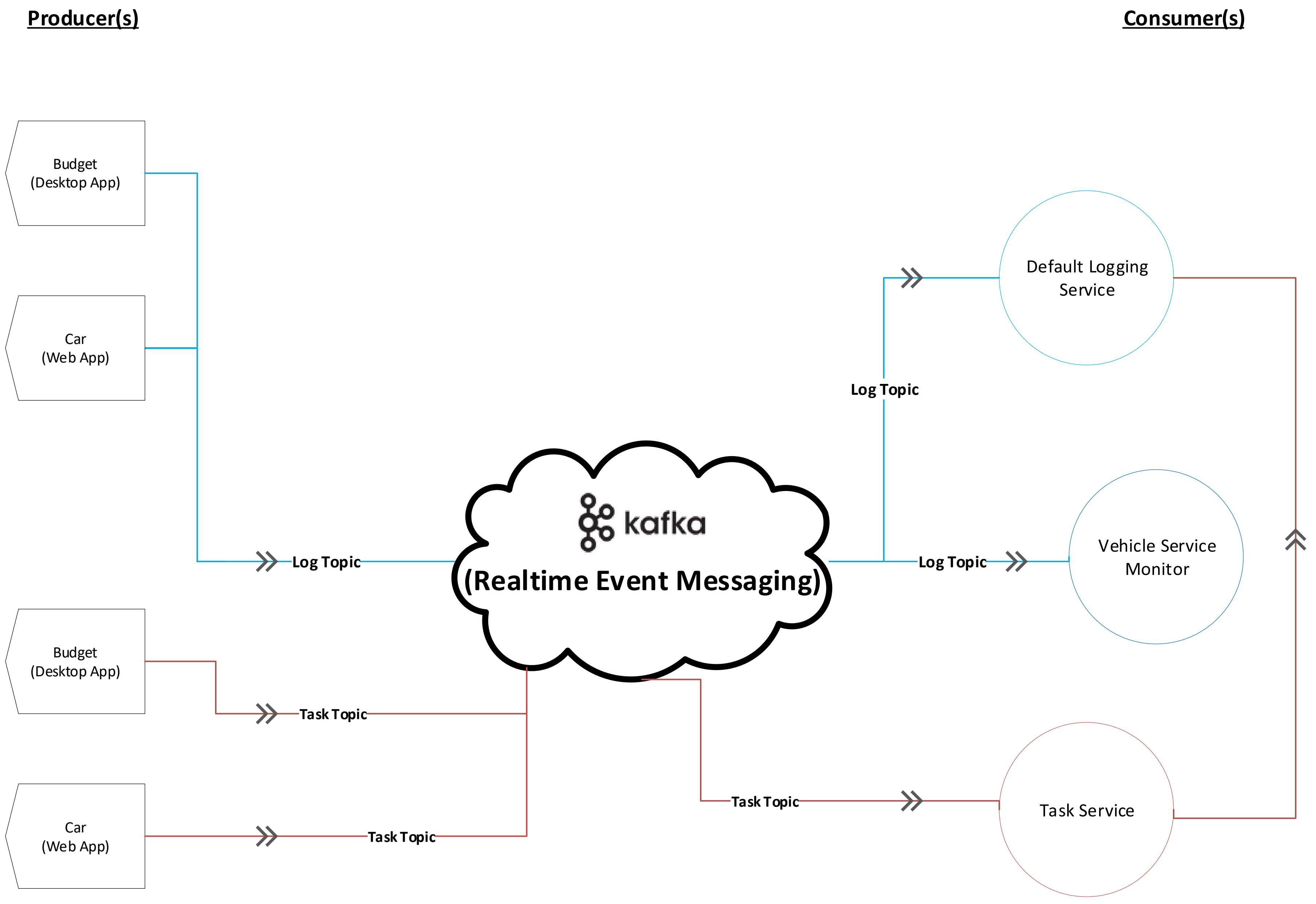

Producer Output
Consumer Input
1
Naturally PRODUCED plants are . . .
. . . CONSUMED by many farmers.
2
Farmers pick and PRODUCE coffee beans for exporters.
3
Coffee beans are CONSUMED by exporters to crate up and ship.
4
Exporters PRODUCE shipments of coffee crates sent around the world.
5
Many hipster coffee houses CONSUME imported crates of fine coffees.
6
Cups of freshly roaster filter coffees are produced on order.
7
A coffee appreciating connoisseur CONSUMES a cup of coffee.

Image: Courtesy of OpenClassrooms


Producer(s)
Consumer(s)
Budget (Desktop App)
Car (Web App)
Log Topic
kafka
(Realtime Event Messaging)
Budget (Desktop App)
Task Topic
Car (Web App)
Task Topic
Default Logging Service
Log Topic
Vehicle Service Monitor
Log Topic
Task Topic
Task Service

```
const string centralizedLoggingTopic = "applications:central_logging";
bool successfulDelivery = false;
string logMessage = "{
        message: 'There was a connection error when attempting to gather data from SQL Server: [server_name].meag.org',
        payload: {
                application: 'Pool Cars',
                username: jpowell',
                operation: 'LoadVehicleUsages',
                params: {
                        paramA: 'test_parameter_valueA',
                        paramB: 'test_parameter_valueB'
                }
        }
}";

Action<DeliveryReport<Null, string>> handler = r => successfulDelivery = r.Error.IsError ? false : true;

using (var producer = new ProducerBuilder<Null, string>(new ProducerConfig { BootstrapServers = bootstrapServers }).Build())
{
        producer.Produce(centralizedLoggingTopic, new Message<Null, string> { Timestamp = Timestamp.Default, Value = logMessage }, handler);

        // wait for up to 10 seconds for any inflight messages to be delivered.
        p.Flush(TimeSpan.FromSeconds(10));
}

ConsumeResult<Null, string> result;
using (IConsumer<Null, string> consumer =
        new ConsumerBuilder<Null, string>(new ConsumerConfig { GroupId = groupId, BootstrapServers = Constants.BootstrapServers }).Build()) {

        consumer.Subscribe(topics);

        // Listen for incoming messages and print them out
        while (true)
        {
                result = consumer.Consume();

                handler(result);
        }
}
```