



LẬP TRÌNH JAVA SPRING BOOT

Bài 8: Spring Middlewares – Kafka, Redis,
Websocket

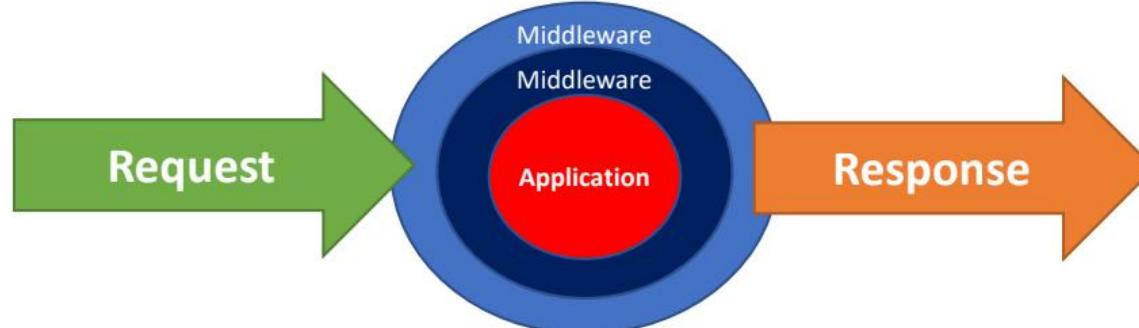
❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu Middlewares là gì
- ❖ Hiểu Message Broker là gì
- ❖ Biết cách triển khai Kafka
- ❖ Biết cách triển khai Redis
- ❖ Biết cách triển khai Websocket

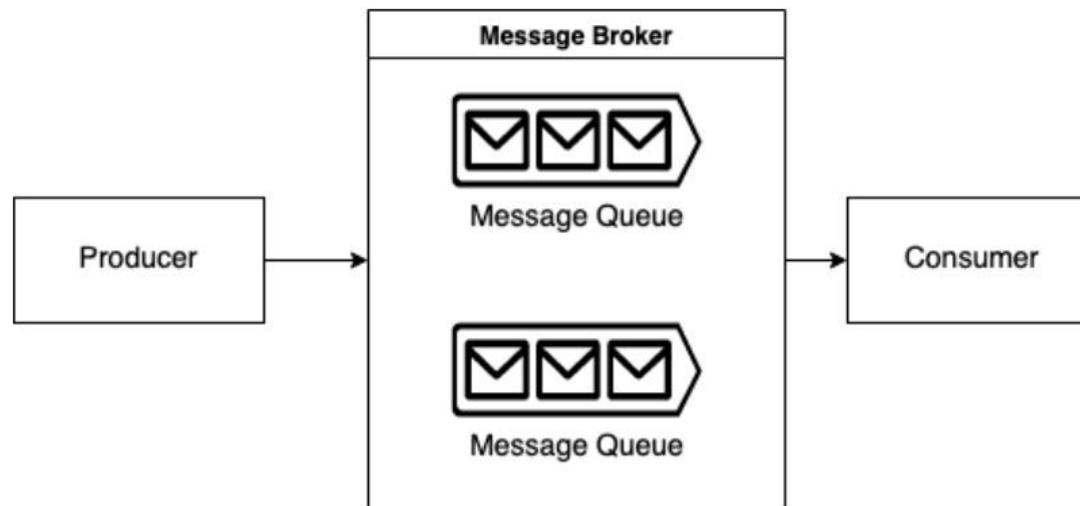
- ❑ **Middleware** là những đoạn mã trung gian nằm giữa các **request** và **response**
- ❑ Nó nhận các **request**, thi hành các mệnh lệnh tương ứng trên request đó. Sau khi hoàn thành nó **response** (trả về) hoặc chuyển kết quả ủy thác cho một Middleware khác trong hàng đợi

👉 Tại sao nên sử dụng nó?

- ❑ Sử dụng Middleware hiệu quả có thể **tối giản** được số lượng **dòng code**
- ❑ Với tư tưởng là cầu nối giữa **client** và **server** các hàm được dùng để **tiền xử lý**, **lọc** các request trước khi đưa vào xử lý **logic** hoặc điều chỉnh các **response** trước khi gửi về cho người dùng
- ❑ Ví dụ phổ biến thường dùng Middleware là các trang cho **admin** và **không** cho phép người dùng bình thường **truy cập**



- ❑ **Message broker** là một module **trung gian** trung chuyển message từ người **gửi** đến người **nhận**
- ❑ Nó là một **mô hình kiến trúc** (architectural pattern) để kiểm tra, trung chuyển và điều hướng **message**
- ❑ Làm trung gian giữa các ứng dụng với nhau, tối giản hóa giao tiếp giữa các ứng dụng và tăng hiệu quả tối đa cho việc tách ra các khối nhỏ hơn
- ❑ Nhiệm vụ chính của **Message broker** là tiếp nhận những message từ các ứng dụng và thực hiện một thao tác nào đó
- ❑ Hiện tại có rất nhiều các message broker software có thể kể đến như: Amazon Web Services (AWS), **Apache Kafka**, Apache ActiveMQ...



- ❑ Các ứng dụng **sử dụng Kafka**: Twitter, Linkedin, Netflix, Các hệ thống ngân hàng...
- ❑ Kafka là một trong những “trụ cột” chính của nền tảng dữ liệu IoT
- ❑ Hầu như mọi công ty **hàng đầu** trên thế giới đều đang sử dụng **kafka** trong nền tảng **cơ sở hạ tầng** của mình

Kafka Applications



Twitter



hotstar

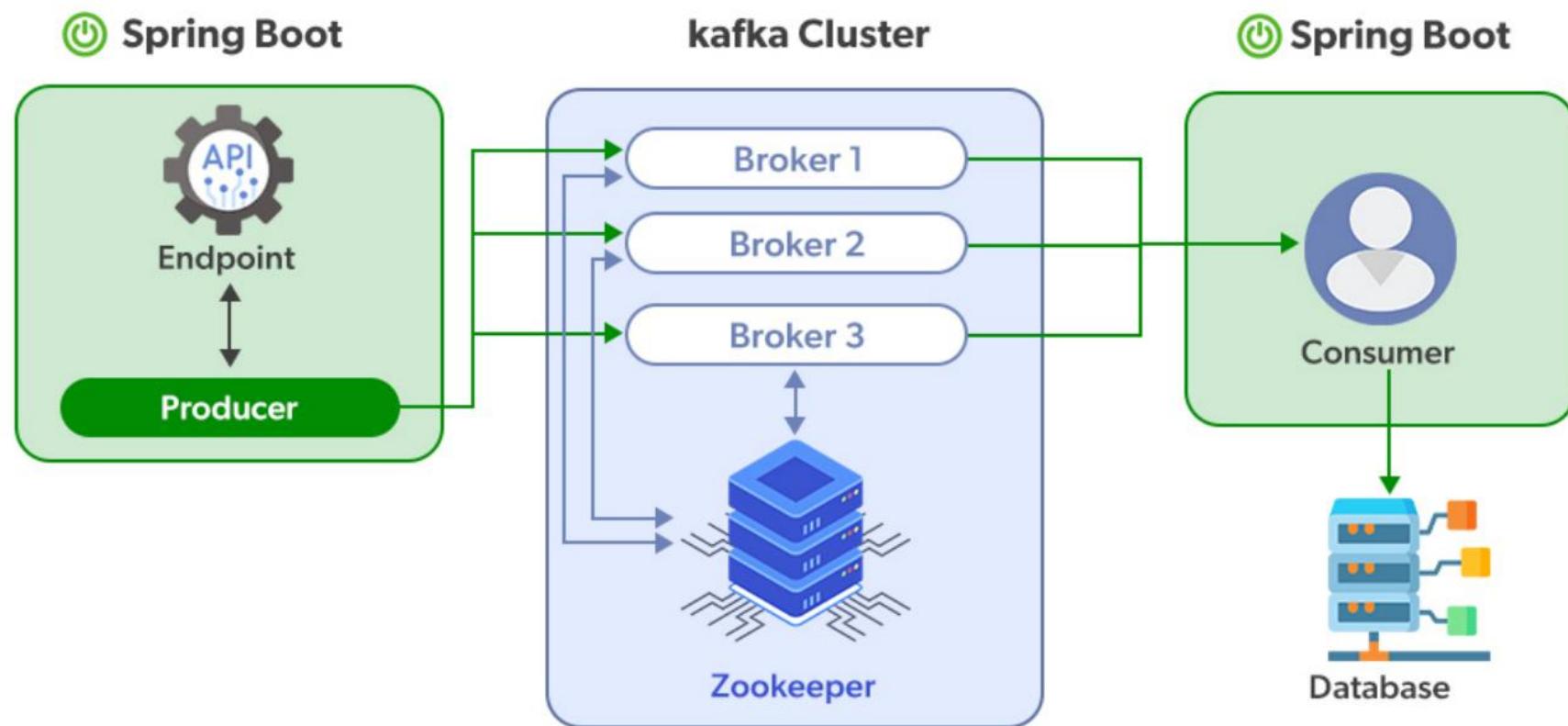


Netflix

- ❑ **Kafka** có tên gọi khác là **Apache Kafka** do hệ thống này được phát triển bởi tổ chức **Apache**
- ❑ Là một phần mềm sử dụng **mã nguồn mở**
- ❑ Kafka có nhiệm vụ xây dựng một **khuôn khổ để đọc, lưu trữ và phân tích dữ liệu trực tuyến**
- ❑ Phần mềm này được thiết kế để hoạt động trong môi trường “**phân tán**” (Distributed)
- ❑ Điều này có nghĩa là thay vì chạy trên máy tính của **một** người dùng, Kafka sẽ hoạt động trên **nhiều** máy chủ
- ❑ Nền tảng **Kafka stream** được viết bởi ngôn ngữ **Scala** và **Java** nhằm mục đích cung cấp dịch vụ xử lý sự kiện dựa trên **thời gian thực** với **độ trễ thấp và thông lượng cao**
- ❑ Nhờ đó, chúng có thể tận dụng **sức mạnh** xử lý bổ sung và dung lượng **lưu trữ hiệu quả**
- ❑ Vì thế, hệ thống được hơn 80% doanh nghiệp trong top 100 của Fortune tin dùng
- ❑ Confluent định nghĩa đây là một nền tảng **stream**, có nhiều bài viết khác lại nói về **Kafka** là một **message broker**

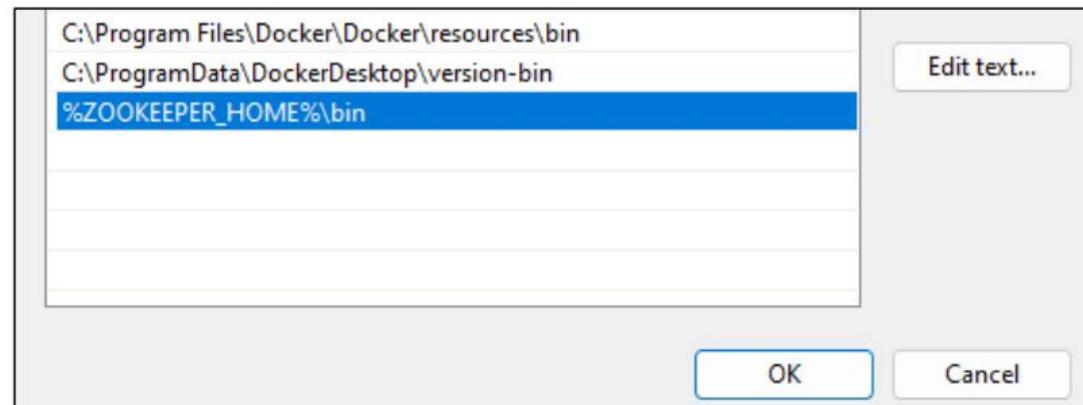
- ❑ **Kafka** là một hệ thống message theo cơ chế **Pub-Sub**. Nó cho phép các nhà sản xuất (gọi là **producer**) viết các message vào Kafka mà một, hoặc nhiều người tiêu dùng (gọi là **consumer**) có thể đọc, xử lý được những **message** đó
- ❑ **Producer**: Đây có thể là bất kỳ ứng dụng nào có chức năng publish message vào một topic
- ❑ **Topic**: Topic là một thể loại (category) hoặc tên nguồn cung cấp dữ liệu (feed name). Tại đây, bản ghi (record) sẽ được publish
- ❑ **Messages**: Đây chỉ đơn giản là một mảng byte. Chúng được nhà phát triển sử dụng nhằm lưu các object với bất kỳ format nào, thường sẽ là **JSON, Avro, String**
- ❑ **Partitions**: Những topic sẽ được chia đều vào các đoạn khác nhau, những đoạn này được gọi là **partitions**
- ❑ **Consumer**: Consumer có thể là ứng dụng nào với chức năng **subscribe** vào một **topic** và tiêu thụ các tin nhắn
- ❑ **Broker**: Một set các server được gọi là cụm **Kafka**, mỗi set này được gọi là một **broker**
- ❑ **Zookeeper**: Có chức năng quản lý và bố trí các **broker**

- ❑ Minh họa Kafka với dự án Spring Boot
- ❑ Cần cài đặt Apache Kafka, Apache Zookeeper trên máy local để lấy thông tin kết nối đưa vào Spring Boot

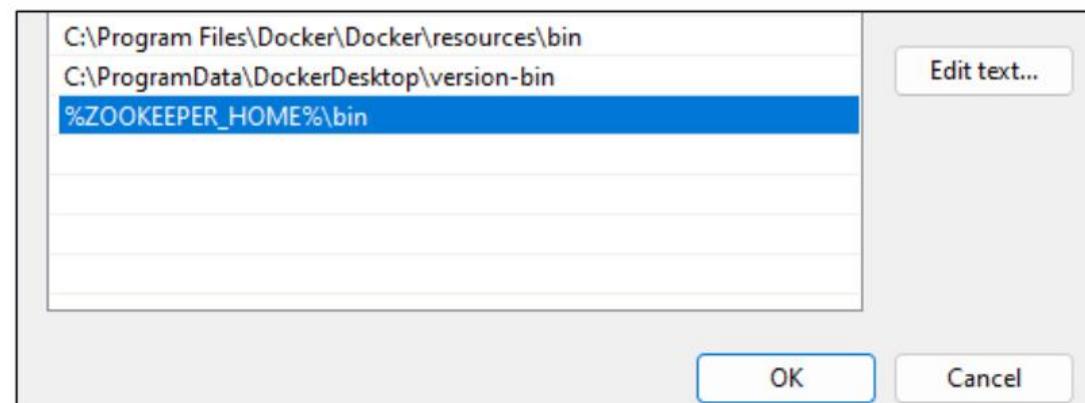


- ❑ Để demo **Kafka** với dự án **Spring Boot**
- ❑ Cần cài đặt **Apache Kafka**, **Apache Zookeeper** trên máy local để lấy thông tin kết nối đưa vào Spring Boot
- ❑ Tải về zookeeper và kafka ở hai link sau
 - ❖ Tải về **Zookeeper** version(3.6.3):
<http://zookeeper.apache.org/releases.html#download>
 - ❖ Tải về **Kafka** version(Binary Scala 2.13):
<http://kafka.apache.org/downloads.html>

- ❑ **Bước 1:** Ta giải nén file zookeeper vừa tải về, ví dụ ở đây giải nén ra được thư mục **C:\Users\PC\Downloads\apache-zookeeper-3.6.3-bin**
- ❑ **Bước 2:** Đổi tên file **zoo_sample.cfg** trong thư mục **conf** thành **zoo.cfg** (zookeeper sử dụng file **zoo.cfg** để config)
- ❑ **Bước 3:** Mở file **zoo.cfg** bằng notepad++ và sửa dòng
dataDir=/tmp/zookeeper → dataDir= apache-zookeeper-3.6.3-bin/data
- ❑ **Lưu ý:** **apache-zookeeper-3.6.3-bin** là tên thư mục cuối mà bạn cài đặt
- ❑ **Bước 4:** Tiếp theo các bạn tạo biến môi trường cho zookeeper như sau
- ❑ Tạo biến
ZOOKEEPER_HOME: là đường dẫn tới thư mục zookeeper vừa tải về



- ❑ **Bước 1:** Ta giải nén file zookeeper vừa tải về, ví dụ ở đây giải nén ra được thư mục **C:\Users\PC\Downloads\apache-zookeeper-3.6.3-bin**
- ❑ **Bước 2:** Đổi tên file **zoo_sample.cfg** trong thư mục **conf** thành **zoo.cfg** (zookeeper sử dụng file **zoo.cfg** để config)
- ❑ **Bước 3:** Mở file **zoo.cfg** bằng notepad++ và sửa dòng
dataDir=/tmp/zookeeper → dataDir= apache-zookeeper-3.6.3-bin/data
- ❑ **Lưu ý:** **apache-zookeeper-3.6.3-bin** là tên thư mục cuối mà bạn cài đặt
- ❑ **Bước 4:** Tiếp theo các bạn tạo biến môi trường cho zookeeper như sau
 - Tạo biến **ZOOKEEPER_HOME**: là đường dẫn tới thư mục zookeeper vừa tải về **C:\Users\PC\Downloads\apache-zookeeper-3.6.3-bin**
 - Thêm **%ZOOKEEPER_HOME%\bin** vào **PATH**



- ❑ **Bước 5:** kiểm tra cài đặt, mở cmd và gõ zkserver
- ❑ Cổng mặc định của zookeeper là **2181**

```
2022-07-19 14:08:59,404 [myid:] - INFO  [main:NIOServerCnxnFactory@674] - binding to port 0.0.0.0/0.0.0.0:2181
2022-07-19 14:08:59,416 [myid:] - INFO  [main:WatchManagerFactory@42] - Using org.apache.zookeeper.server.watch
2022-07-19 14:08:59,417 [myid:] - INFO  [main:WatchManagerFactory@42] - Using org.apache.zookeeper.server.watch
2022-07-19 14:08:59,417 [myid:] - INFO  [main:ZKDatabase@132] - zookeeper.snapshotSizeFactor = 0.33
2022-07-19 14:08:59,417 [myid:] - INFO  [main:ZKDatabase@152] - zookeeper.commitLogCount=500
2022-07-19 14:08:59,420 [myid:] - INFO  [main:SnapStream@61] - zookeeper.snapshot.compression.method = CHECKED
2022-07-19 14:08:59,421 [myid:] - INFO  [main:FileSnap@85] - Reading snapshot apache-zookeeper-3.6.3-bin\data\ve
2022-07-19 14:08:59,423 [myid:] - INFO  [main:DataTree@1737] - The digest value is empty in snapshot
2022-07-19 14:08:59,439 [myid:] - INFO  [main:ZKAuditProvider@42] - ZooKeeper audit is disabled.
2022-07-19 14:08:59,440 [myid:] - INFO  [main:FileTxnSnapLog@363] - 40 txns loaded in 13 ms
2022-07-19 14:08:59,441 [myid:] - INFO  [main:ZKDatabase@289] - Snapshot loaded in 23 ms, highest zxid is 0x28,
2022-07-19 14:08:59,441 [myid:] - INFO  [main:FileTxnSnapLog@470] - Snapshotting: 0x28 to apache-zookeeper-3.6.3
2022-07-19 14:08:59,443 [myid:] - INFO  [main:ZooKeeperServer@529] - Snapshot taken in 2 ms
2022-07-19 14:08:59,454 [myid:] - INFO  [ProcessThread(sid:0 cport:2181)::PrepRequestProcessor@136] - PrepReques
2022-07-19 14:08:59,455 [myid:] - INFO  [main:RequestThrottler@74] - zookeeper.request_throttler.shutdownTimeout
s=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0
2022-07-19 14:09:59,648 [myid:] - INFO  [SyncThread:0:FileTxnLog@284] - Creating new log file: log.29
```

- ❑ **Bước 1:** Giải nén file các bạn đã tải về. Đổi tên thư mục thành kafka
- ❑ Ví dụ sau khi giải nén và đổi tên: **C:\Users\PC\Downloads\kafka**
- ❑ **Bước 2:** Mở file **server.properties** trong thư mục config và sửa
 - log.dirs=/tmp/kafka-logs →
 - log.dirs=log.dirs=C:/Users/PC/Downloads/kafka/kafka-logs
- ❑ **Lưu ý:** đây là đường dẫn file các bạn đã tải về, tránh đặt tên thư mục quá dài sẽ gây ra lỗi không run được kafka
- ❑ Giữ nguyên **zookeeper.connect=localhost:2181** vì lúc cài đặt zookeeper chúng ta để port mặc định là **2181**
- ❑ **Bước 3:** Chạy Apache Kafka (**phải start Zookeeper trước đó**). Port mặc định của kafka là **9092**

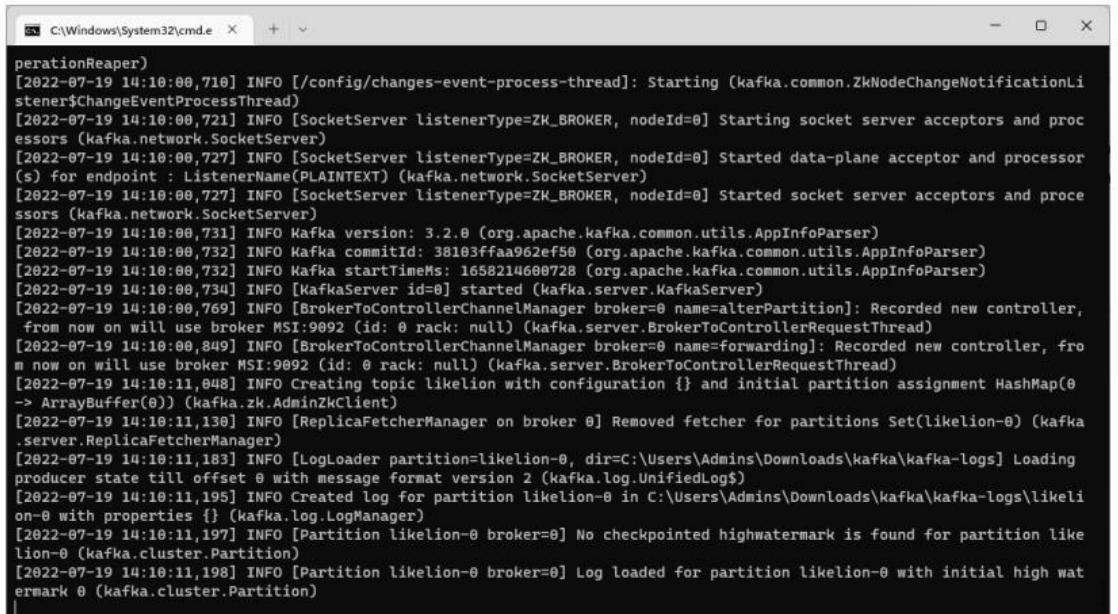
☐ Nếu dùng Window:

- vào thư mục **C:\Users\PC\Downloads\kafka\bin\windows**
- Gõ lệnh: kafka-server-start.bat ../../config/server.properties

☐ Nếu dùng Mac:

- vào thư mục **\Downloads\kafka\bin**
- Gõ lệnh: kafka-server-start.sh config/server.properties

☐ Kết quả chạy thành công



```
C:\Windows\System32\cmd.exe + - x

operationReaper)
[2022-07-19 14:10:00,710] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2022-07-19 14:10:00,721] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Starting socket server acceptors and processors (kafka.network.SocketServer)
[2022-07-19 14:10:00,727] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started data-plane acceptor and processor(s) for endpoint : ListenerName(PLAINTEXT) (kafka.network.SocketServer)
[2022-07-19 14:10:00,727] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started socket server acceptors and processors (kafka.network.SocketServer)
[2022-07-19 14:10:00,731] INFO Kafka version: 3.2.0 (org.apache.kafka.common.utils.AppInfoParser)
[2022-07-19 14:10:00,732] INFO Kafka commitId: 38103ffaa962ef50 (org.apache.kafka.common.utils.AppInfoParser)
[2022-07-19 14:10:00,732] INFO Kafka startTimeMs: 1658214600728 (org.apache.kafka.common.utils.AppInfoParser)
[2022-07-19 14:10:00,734] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2022-07-19 14:10:00,769] INFO [BrokerToControllerChannelManager broker=0 name=alterPartition]: Recorded new controller, from now on will use broker MSI:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2022-07-19 14:10:00,849] INFO [BrokerToControllerChannelManager broker=0 name=forwarding]: Recorded new controller, from now on will use broker MSI:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2022-07-19 14:10:11,048] INFO Creating topic likelion with configuration {} and initial partition assignment HashMap<@ -> ArrayBuffer<@>)) (kafka.zk.AdminZkClient)
[2022-07-19 14:10:11,130] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(likelion-0) (kafka.server.ReplicaFetcherManager)
[2022-07-19 14:10:11,183] INFO [LogLoader partition=likelion-0, dir=c:\Users\Admins\Downloads\kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2022-07-19 14:10:11,195] INFO Created log for partition likelion-0 in C:\Users\Admins\Downloads\kafka\kafka-logs\likelion-0 with properties {} (kafka.log.LogManager)
[2022-07-19 14:10:11,197] INFO [Partition likelion-0 broker=0] No checkpointed highwatermark is found for partition likelion-0 (kafka.cluster.Partition)
[2022-07-19 14:10:11,198] INFO [Partition likelion-0 broker=0] Log loaded for partition likelion-0 with initial high watermark 0 (kafka.cluster.Partition)
```

- ☐ Truy cập vào <https://start.spring.io/> tạo dự án **Spring Boot** và **Kafka**

The screenshot shows the configuration interface for generating a Spring Boot and Kafka application. The left panel contains the following settings:

- Project**: Maven Project (selected)
- Language**: Java (selected)
- Spring Boot**: 2.7.1 (selected)
- Project Metadata**:
 - Group: com.likelion
 - Artifact: kafka
 - Name: kafka
 - Description: (empty)
 - Package name: com.likelion.kafka
 - Packaging: Jar (selected)
 - Java: 8 (selected)

The right panel shows the selected dependencies:

- Dependencies**: ADD ... CTRL + B
- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Spring for Apache Kafka** (MESSAGING): Publish, subscribe, store, and process streams of records.

At the bottom are three buttons: GENERATE (CTRL + ⌘), EXPLORE (CTRL + SPACE), and SHARE... .

- ❑ Mở dự án bằng IntelliJ và kiểm tra file pom.xml với các dependency

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>

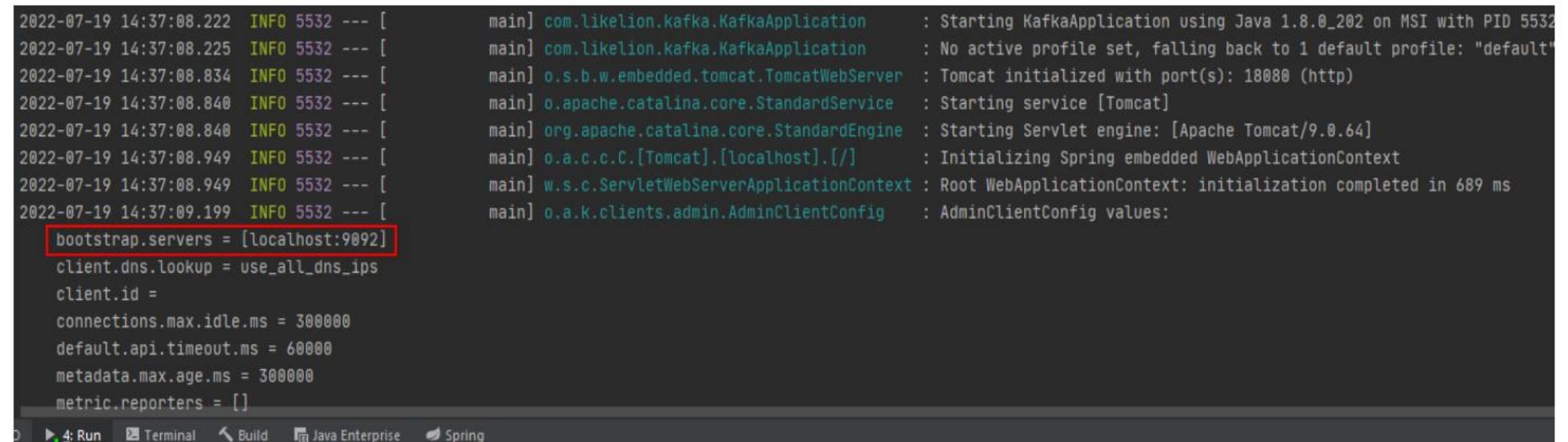
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

☐ Mở file application.properties và thêm như sau

```
# HTTP server port  
server.port=18080  
  
# Kafka  
# Kết nối đến Kafka với cổng mặc định 9092. Lưu ý phải start zookeeper và kafka trước  
spring.kafka.bootstrap-servers=localhost:9092
```

☐ Chạy thử Spring Boot

```
2022-07-19 14:37:08.222 INFO 5532 --- [           main] com.likelion.kafka.KafkaApplication      : Starting KafkaApplication using Java 1.8.0_202 on MSI with PID 5532  
2022-07-19 14:37:08.225 INFO 5532 --- [           main] com.likelion.kafka.KafkaApplication      : No active profile set, falling back to 1 default profile: "default"  
2022-07-19 14:37:08.834 INFO 5532 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 18080 (http)  
2022-07-19 14:37:08.840 INFO 5532 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]  
2022-07-19 14:37:08.840 INFO 5532 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]  
2022-07-19 14:37:08.949 INFO 5532 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[]        : Initializing Spring embedded WebApplicationContext  
2022-07-19 14:37:08.949 INFO 5532 --- [           main] w.s.c.ServletWebServerApplicationContext  : Root WebApplicationContext: initialization completed in 689 ms  
2022-07-19 14:37:09.199 INFO 5532 --- [           main] o.a.k.clients.admin.AdminClientConfig    : AdminClientConfig values:  
bootstrap.servers = [localhost:9092]  
client.dns.lookup = use_all_dns_ips  
client.id =  
connections.max.idle.ms = 300000  
default.api.timeout.ms = 60000  
metadata.max.age.ms = 300000  
metric.reporters = []
```



The screenshot shows a terminal window with the following details:

- Toolbar icons: Run, Terminal, Build, Java Enterprise, Spring.
- Log output from the application's main class (com.likelion.kafka.KafkaApplication) showing its startup process.
- A specific line in the log, "bootstrap.servers = [localhost:9092]", is highlighted with a red rectangular box.

- ❑ Từ thư mục gốc của dự án(com.likelion.kafka) tạo package config
- ❑ Trong package config tạo class **KafkaTopicConfig**

```
@Configuration  
public class KafkaTopicConfig {  
  
    @Bean  
    public NewTopic likelionTopic(){  
        return TopicBuilder.name("likelion").build();  
    }  
}
```

- ❑ Chạy thử Spring Boot

```
2022-07-19 14:37:09.232  INFO 5532 --- [           main] o.a.kafka.common.utils.AppInfoParser : Kafka version: 3.1.1  
2022-07-19 14:37:09.232  INFO 5532 --- [           main] o.a.kafka.common.utils.AppInfoParser : Kafka commitId: 97671528ba54a138  
2022-07-19 14:37:09.232  INFO 5532 --- [           main] o.a.kafka.common.utils.AppInfoParser : Kafka startTimeMs: 1658216229231  
2022-07-19 14:37:09.465  INFO 5532 --- [| adminclient-1] o.a.kafka.common.utils.AppInfoParser : App info kafka.admin.client for adminclient-1 unregistered  
2022-07-19 14:37:09.468  INFO 5532 --- [| adminclient-1] org.apache.kafka.common.metrics.Metrics : Metrics scheduler closed  
2022-07-19 14:37:09.468  INFO 5532 --- [| adminclient-1] org.apache.kafka.common.metrics.Metrics : Closing reporter org.apache.kafka.common.metrics.JmxReporter  
2022-07-19 14:37:09.468  INFO 5532 --- [| adminclient-1] org.apache.kafka.common.metrics.Metrics : Metrics reporters closed  
2022-07-19 14:37:09.475  INFO 5532 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 18080 (http) with context path ''  
2022-07-19 14:37:09.481  INFO 5532 --- [           main] com.likelion.kafka.KafkaApplication : Started KafkaApplication in 1.527 seconds (JVM running for 1.994)
```

☐ Trong package config tạo class KafkaTopicConfig

```
@Configuration
public class KafkaProducerConfig {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    public Map<String, Object> producerConfig(){
        Map<String, Object> props = new HashMap<>();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        return props;
    }

    // ProducerFactory responsible for creating instances
    @Bean
    public ProducerFactory<String, String> producerFactory() {
        return new DefaultKafkaProducerFactory<>(producerConfig());
    }

    // KafkaTemplate allows us to create kafka producers we need a way for us to send messages
    @Bean
    public KafkaTemplate<String, String> kafkaTemplate(ProducerFactory<String, String> producerFactory){
        return new KafkaTemplate<>(producerFactory);
    }
}
```

- ❑ Mở file main Spring Boot **KafkaApplication** và thêm như sau để test send một mesage

```
@SpringBootApplication
public class KafkaApplication {

    public static void main(String[] args) {
        SpringApplication.run(KafkaApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(KafkaTemplate<String, String> kafkaTemplate) {
        return args -> {
            kafkaTemplate.send("likelion", "Hello kafka");
        };
    }
}
```

- ❑ Bean **CommandLineRunner** sẽ thực hiện chạy lần đầu khi khởi chạy ứng dụng
- ❑ “**likelion**” là topic đã được tạo tại bean **likelionTopic()**

☐ Chạy thử Spring Boot

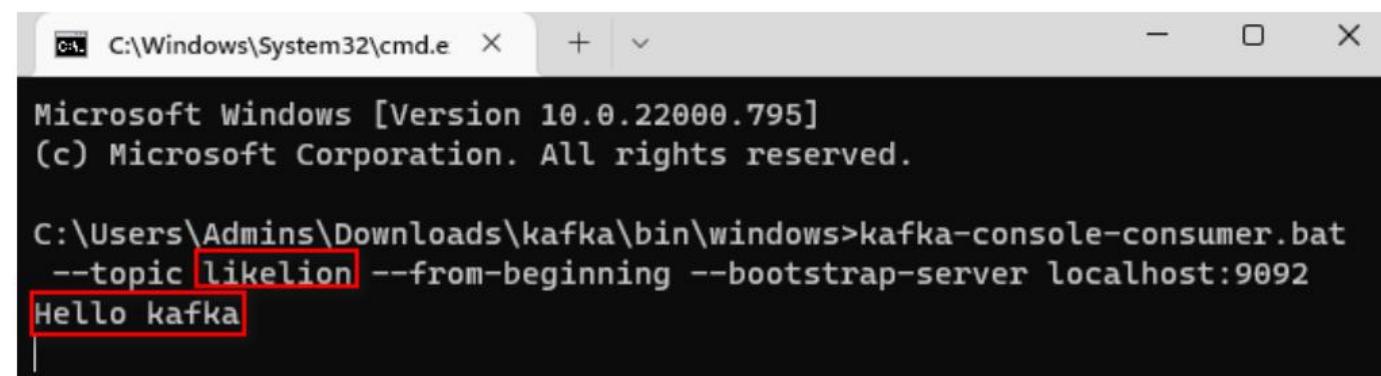


A screenshot of a Java IDE showing the output of a Kafka producer application. The logs are displayed in a terminal window, showing various INFO messages from the KafkaProducer and AppInfoParser classes. A red box highlights the last few lines of the log, which indicate the producer has been instantiated, the Kafka version is 3.1.1, and the commit ID is 97671528ba54a138. It also shows the start time in milliseconds (1658217915029) and the epoch set to 0. The log ends with a message about the producer ID being set to 1.

```
2022-07-19 15:05:15.020  INFO 13284 --- [           main] o.a.k.clients.producer.KafkaProducer      : [Producer clientId=producer-1] Instantiated an idempotent producer.
2022-07-19 15:05:15.030  INFO 13284 --- [           main] o.a.kafka.common.utils.AppInfoParser       : Kafka version: 3.1.1
2022-07-19 15:05:15.030  INFO 13284 --- [           main] o.a.kafka.common.utils.AppInfoParser       : Kafka commitId: 97671528ba54a138
2022-07-19 15:05:15.030  INFO 13284 --- [           main] o.a.kafka.common.utils.AppInfoParser       : Kafka startTimeMs: 1658217915029
2022-07-19 15:05:15.036  INFO 13284 --- [ad | producer-1] org.apache.kafka.clients.Metadata   : [Producer clientId=producer-1] Resetting the last seen epoch of part
2022-07-19 15:05:15.036  INFO 13284 --- [ad | producer-1] org.apache.kafka.clients.Metadata   : [Producer clientId=producer-1] Cluster ID: UJgCaZ7eTV6gEqSYCxmX-g
2022-07-19 15:05:15.037  INFO 13284 --- [ad | producer-1] o.a.k.c.p.internals.TransactionManager    : [Producer clientId=producer-1] ProducerId set to 1 with epoch 0
```

Below the terminal window, there is a toolbar with icons for TODO, Run, Terminal, Build, Java Enterprise, Spring, and Messages.

☐ Mở cmd và kiểm tra message như sau



A screenshot of a Windows Command Prompt window titled 'cmd' with the path 'C:\Windows\System32\cmd.e'. The window displays the Microsoft Windows version (10.0.22000.795) and copyright information. The user runs the command 'kafka-console-consumer.bat --topic likelion --from-beginning --bootstrap-server localhost:9092'. The output shows the message 'Hello kafka'.

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admins\Downloads\kafka\bin\windows>kafka-console-consumer.bat
--topic likelion --from-beginning --bootstrap-server localhost:9092
Hello kafka
```

☐ Trong package config tạo class KafkaConsumerConfig

```
@Configuration
public class KafkaConsumerConfig {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    public Map<String, Object> consumerConfig(){
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        return props;
    }

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        return new DefaultKafkaConsumerFactory<>(consumerConfig());
    }

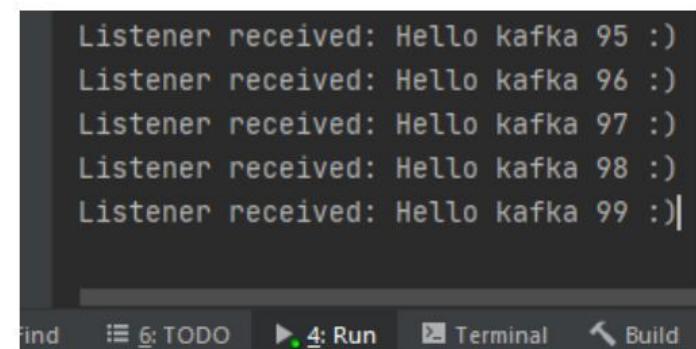
    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, String >> factory (
        ConsumerFactory<String, String> consumerFactory) {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory);
        return factory;
    }
}
```

❑ Trong package config tạo class **KafkaListeners**

```
@Component
public class KafkaListeners {

    @KafkaListener(
        topics = "likelion",
        groupId = "groupId"
    )
    void listener(String data) {
        System.out.println("Listener received: " + data + " :)");
    }
}
```

❑ Chạy thử Spring Boot



A screenshot of an IDE terminal window showing the output of a Kafka consumer application. The terminal displays five lines of text, each starting with 'Listener received:' followed by 'Hello kafka' and a number from 95 to 99, separated by a colon and a closing parenthesis. The terminal interface includes standard navigation keys like 'Find', 'TODO', 'Run', 'Terminal', and 'Build'.

```
Listener received: Hello kafka 95 :)
Listener received: Hello kafka 96 :)
Listener received: Hello kafka 97 :)
Listener received: Hello kafka 98 :)
Listener received: Hello kafka 99 :)|
```

- Trong package config tạo class **MessageController**

```
@RestController
@RequestMapping("/api/messages")
public class MessageController {

    @Autowired
    private KafkaTemplate kafkaTemplate;

    @GetMapping("/{message}")
    public void publish(@PathVariable("message") String message) {
        kafkaTemplate.send("likelion", message);
    }
}
```

- Restart Spring Boot

☐ Cài đặt như hình và send request

The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: `http://localhost:18080/api/messages/HelloWorld`
- Headers tab (selected):
 - Params
 - Auth
 - Headers (6)
 - Body
 - Pre-req.
 - Tests
 - Settings
- Body tab (selected):
 - Body type dropdown: Text
 - Raw content:

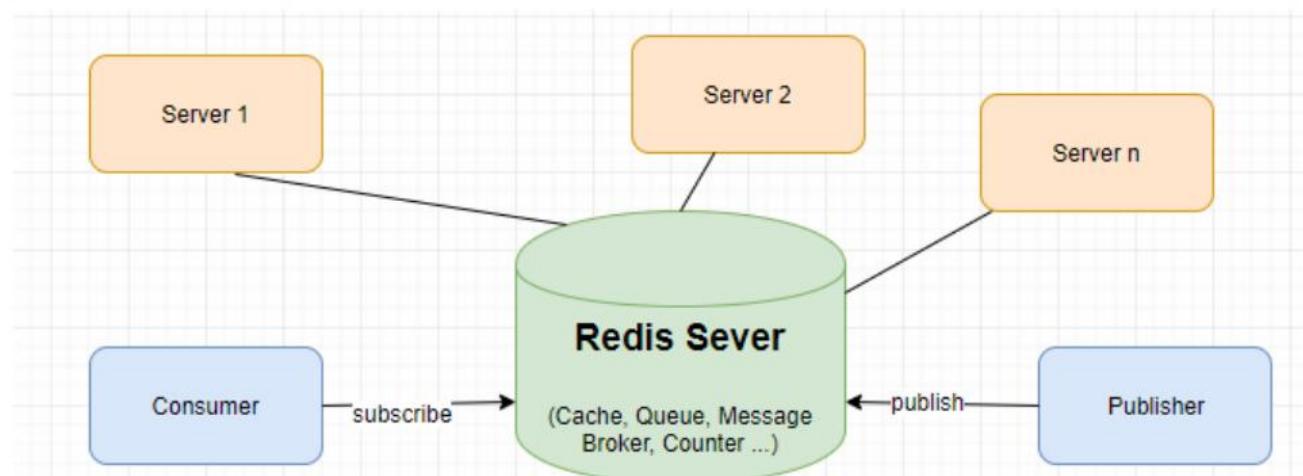
```
HelloWorld
```
 - Preview content:

```
HelloWorld
```
- Response status bar: 200 OK 75 ms 123 B

☐ Kiểm tra trong console trong intelliJ

```
Listener received: Hello kafka 95 :)
Listener received: Hello kafka 96 :)
Listener received: Hello kafka 97 :)
Listener received: Hello kafka 98 :)
Listener received: Hello kafka 99 :)
Listener received: HelloWorld :)
```

- ❑ Redis là tên viết tắt của Remote Dictionary Server, là kho dữ liệu **khóa-giá trị, trong bộ nhớ, mã nguồn mở** và có **tốc độ** truy cập nhanh để dùng để lưu trữ dữ liệu có **cấu trúc**, làm **database**, bộ nhớ **cache**
- ❑ Lợi ích của việc sử dụng Redis
 - ❖ **Kho dữ liệu trong bộ nhớ:** Toàn bộ dữ liệu Redis nằm trong bộ nhớ chính của máy chủ
 - ❖ **Cấu trúc dữ liệu linh hoạt:** Kiểu dữ liệu gồm có String, List, Set,...
 - ❖ **Đơn giản dễ sử dụng:** chỉ cần viết vài dòng lệnh là đã có thể truy cập, lưu trữ và sử dụng dữ liệu
 - ❖ **Khả năng mở rộng:** Redis là dự án mã nguồn mở được một cộng đồng đông đảo ủng hộ



- ❑ **Caching:** Sử dụng làm bộ nhớ đệm. Chính tốc độ đọc ghi nhanh mà Redis có thể làm bộ nhớ đệm, nơi chia sẻ dữ liệu giữa các ứng dụng hoặc làm database tạm thời
- ❑ **Counter:** Sử dụng làm bộ đếm. Với thuộc tính tăng giảm thông số rất nhanh trong khi dữ liệu được lưu trên RAM, sets và sorted sets. Chúng ta có thể sử dụng Counter để làm các bảng xếp hạng game, đếm số lượt view của 1 website
- ❑ **Machine Learning:** Các ứng dụng kiểu mới, chịu sự chi phối của dữ liệu yêu cầu machine learning phải có khả năng nhanh chóng xử lý được dữ liệu theo khối lượng lớn, đa dạng, tốc độ cao và tự động hóa quá trình ra quyết định
- ❑ **Publish/Suscribe (Pub/Sub):** Tạo kênh chia sẻ dữ liệu. Redis hỗ trợ tạo các channel để trao đổi dữ liệu giữa publisher và subscriber giống như channel trong Socket Cluster hay topic trong Apache Kafka
- ❑ **Queues:** Tạo hàng đợi để xử lý lần lượt các request. Redis cho phép lưu trữ theo list và cung cấp rất nhiều thao tác với các phần tử trong list, vì vậy nó còn được sử dụng như một message queue

- ❑ Hiện tại Redis Server vẫn chưa có bản **chính thức** dành cho windows.
(Không nên chạy Redis-Server trên Windows vì có thể nó sẽ có MỘT số lỗi, chưa tối ưu và không được update)
- ❑ Tuy nhiên bạn vẫn có thể download các phiên bản cũ tại
<https://github.com/MicrosoftArchive/redis/releases> để chạy thử và test trên windows
- ❑ Tải và cài đặt file .msi
- ❑ **Lưu ý:** chọn ô **Add the Redis ...** nó sẽ tự động thêm thư mục cài đặt vào biến môi trường, sau đó bạn có thể chạy lệnh redis-server, redis-cli ở bất kỳ thư mục nào trong màn hình cmd/powershell
- ❑ Chọn Port cho Redis Server (mặc định là 6379)
- ❑ Có thể giới hạn dung lượng tối đa dành cho Redis Server
- ❑ Sau khi cài đặt xong, mở màn hình cmd hoặc powershell và chạy lệnh *redis-server*
- ❑ **Lưu ý:** Nếu chạy lệnh redis-server trên Windows sẽ gặp message:
Creating Server TCP listening socket *:6379: bind: No such file or directory

- ❑ Hướng dẫn cách khắc phục khi gặp message: # *Creating Server TCP listening socket *:6379: bind: No such file or directory*

- **Bước 1:** vào thư mục cài đặt Redis, ví dụ: C:\Program Files\Redis
 - **Bước 2:** mở cmd và gõ lệnh redis-cli.exe
 - **Bước 3:** gõ lệnh *shutdown*
 - **Bước 4:** gõ lệnh *exit*
 - **Bước 5:** mở cmd mới và gõ lệnh *redis-server*

- ❑ Truy cập vào <https://start.spring.io/> tạo dự án Spring Boot và Redis
- ❑ Lưu ý: Chọn những item như **khoanh đỏ**

The screenshot shows the configuration interface for creating a Spring Boot project. It is divided into two main sections: Project Configuration and Dependencies.

Project Configuration:

- Project:** Maven Project (selected)
- Language:** Java (selected)
- Spring Boot:** 2.7.1 (selected)
- Project Metadata:**
 - Group: com.likelion
 - Artifact: redis
 - Name: redis
 - Description: (empty)
 - Package name: com.likelion.redis
 - Packaging: Jar (selected)
 - Java: 11 (selected)
 - 8
 - 17
 - 18

Dependencies:

- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
- Spring Data Redis (Access+Driver)** (NOSQL): Advanced and thread-safe Java Redis client for synchronous, asynchronous, and reactive usage. Supports Cluster, Sentinel, Pipelining, Auto-Reconnect, Codecs and much more.

- ❑ Mở dự án và kiểm tra file pom.xml với các dependency sau

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- ❑ Sau khi đã cài đặt Redis, để kết nối tới Redis, cần cung cấp địa chỉ **host** và **port**
- ❑ Thêm cấu hình kết nối Redis vào file **application.properties** như sau

```
# Redis
redis.host=localhost
redis.port=6379
```

- ❑ Từ thư mục gốc của dự án(com.lielion.redis) tạo package **config**
- ❑ Trong package **config** tạo class **RedisConfig**

```
@Configuration
public class RedisConfig {
    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port}")
    private int redisPort;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {
        // Tạo Standalone Connection tới Redis
        return new LettuceConnectionFactory(new RedisStandaloneConfiguration(redisHost, redisPort));
    }

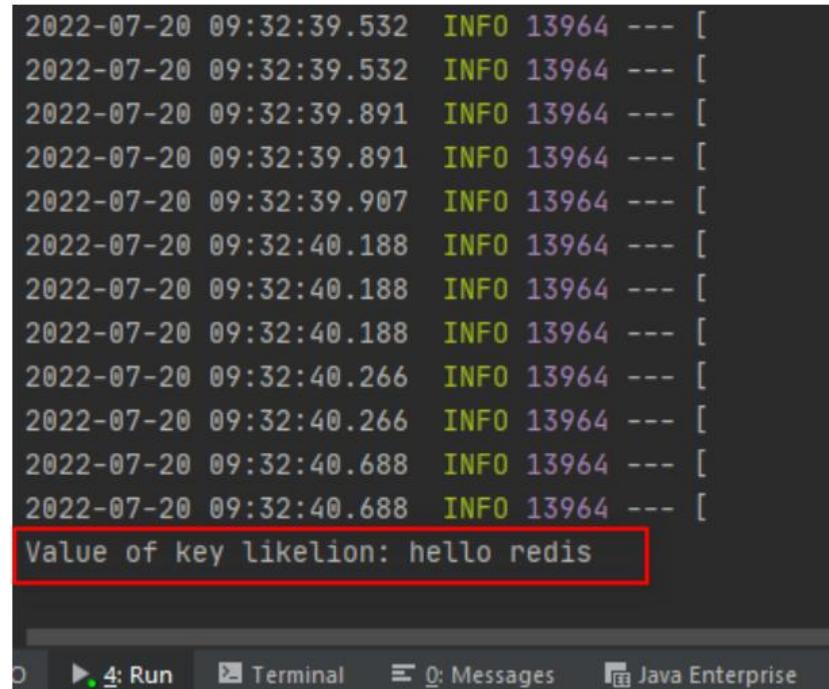
    @Bean
    @Primary
    public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        // tạo ra một RedisTemplate
        // Với Key là Object
        // Value là Object
        // RedisTemplate giúp chúng ta thao tác với Redis
        RedisTemplate<Object, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(redisConnectionFactory);
        return template;
    }
}
```

- ☐ Mở file main Spring Boot **RedisApplication** và thêm như sau để test redis

```
@SpringBootApplication
public class RedisApplication {
    public static void main(String[] args) {
        SpringApplication.run(RedisApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(RedisTemplate<Object, Object> redisTemplate){
        return args -> {
            // Set giá trị của key "likelion" là "hello redis"
            redisTemplate.opsForValue().set("likelion","hello redis");
            // In ra màn hình Giá trị của key "likelion" trong Redis
            System.out.println("Value of key likelion: "+redisTemplate.opsForValue().get("likelion"));
        };
    }
}
```

❑ Khởi chạy Spring Boot



The screenshot shows a terminal window within a Java IDE. The window title bar includes icons for Run, Terminal, Messages, and Java Enterprise. The terminal content displays a series of INFO log entries from a Spring Boot application, each showing a timestamp, log level, process ID (13964), and a trailing bracket. Below these, a single line of text reads "Value of Key likelion: hello redis", which is highlighted with a red rectangular box.

```
2022-07-20 09:32:39.532  INFO 13964 --- [  
2022-07-20 09:32:39.532  INFO 13964 --- [  
2022-07-20 09:32:39.891  INFO 13964 --- [  
2022-07-20 09:32:39.891  INFO 13964 --- [  
2022-07-20 09:32:39.907  INFO 13964 --- [  
2022-07-20 09:32:40.188  INFO 13964 --- [  
2022-07-20 09:32:40.188  INFO 13964 --- [  
2022-07-20 09:32:40.188  INFO 13964 --- [  
2022-07-20 09:32:40.266  INFO 13964 --- [  
2022-07-20 09:32:40.266  INFO 13964 --- [  
2022-07-20 09:32:40.688  INFO 13964 --- [  
2022-07-20 09:32:40.688  INFO 13964 --- [  
  
Value of Key likelion: hello redis
```

- ## ❑ Vậy là chúng ta đã kết nối thành công tới **Redis** và lưu một cặp **key-value** vào trong đó

❑ Diễn giải

- ❖ **opsForValue()** được gọi là **Redis Operations**

❑ Spring Data hỗ trợ chúng ta **thao tác** với **Redis** thông qua các Operations như sau

- ❖ **opsForValue()**: Kiểu Key-Value thông thường. Với Value là 1 giá trị String tùy ý
- ❖ **opsForHash()**: Tương ứng với cấu trúc Hash trong Redis. Value là một Object có cấu trúc
- ❖ **opsForList()**: Tương ứng với cấu trúc List trong Redis. Value là một list
- ❖ **opsForSet()**: Tương ứng với cấu trúc Set trong Redis
- ❖ **opsForZSet()**: Tương ứng với cấu trúc ZSet trong Redis

☐ Sửa lại nội dung file **RedisApplication** như sau

```
@SpringBootApplication
public class RedisApplication {
    public static void main(String[] args) {
        SpringApplication.run(RedisApplication.class, args);
    }
    @Autowired
    private RedisTemplate template;

    @Bean
    CommandLineRunner commandLineRunner() {
        return args -> {
            listExample();
        };
    }

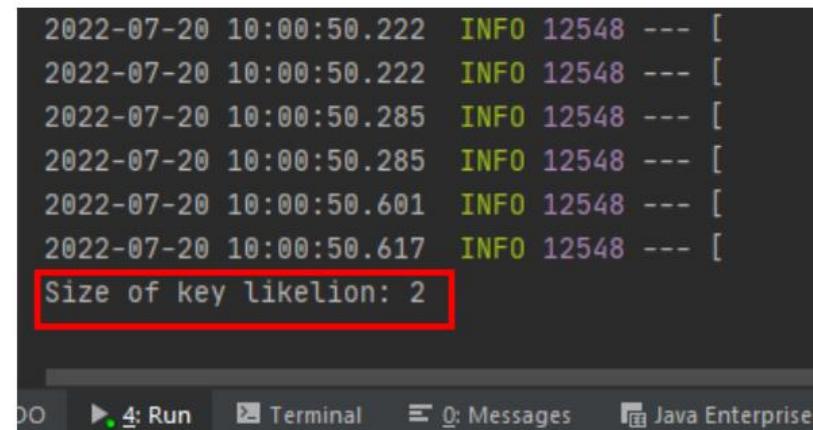
    public void listExample(){
        // Tạo ra một list gồm 2 phần tử
        List<String> list = new ArrayList<>();
        list.add("Hello");
        list.add("Redis");

        // Set giá trị có key likelion_list
        template.opsForList().rightPushAll("likelion_list", list);

        System.out.println("Size of key likelion: "+template.opsForList().size("likelion_list"));
    }
}
```

❑ Khởi chạy Spring Boot

```
2022-07-20 10:00:50.222 INFO 12548 --- [
2022-07-20 10:00:50.222 INFO 12548 --- [
2022-07-20 10:00:50.285 INFO 12548 --- [
2022-07-20 10:00:50.285 INFO 12548 --- [
2022-07-20 10:00:50.601 INFO 12548 --- [
2022-07-20 10:00:50.617 INFO 12548 --- [
Size of key likelion: 2
```



The screenshot shows a terminal window with several log entries from a Java application. The logs are timestamped and show the application's internal state. The last log entry, which displays the size of a key named 'likelion', is highlighted with a red rectangular box.

❑ Tới đây các bạn có thể dễ dàng thao tác với các kiểu cấu trúc trong Redis thông qua **RedisTemplate**

- ❑ **WebSocket** là giao thức hỗ trợ giao tiếp **hai chiều** giữa **client** và **server** để tạo một **kết nối** trao đổi **dữ liệu** một cách **mạnh mẽ**. WebSocket hay được sử dụng trong các trường hợp yêu cầu **thời gian thực (realtime)** như các ứng dụng **chat** hay **biểu đồ**

Socket là gì ?

- ❑ Socket là một **endpoint** (điểm cuối) của liên kết giao tiếp **hai chiều** của giữa **hai chương trình** chạy trên **mạng** hay bạn có thể hiểu đơn giản là **Socket** được sử dụng để cho phép **process** này nói chuyện được với **process** khác

Socket được hoạt động như thế nào ?

- ❑ Socket giúp ta có thể **kết nối** các ứng dụng để **truyền** và **nhận** dữ liệu trong môi trường có kết nối **Internet** bằng cách sử dụng các phương thức **TCP/IP** và **UDP**
- ❑ Khi ta muốn trao đổi dữ liệu giữa 2 ứng dụng thì ứng dụng này cần phải biết được thông tin về **IP** và **port** của ứng dụng kia

- ☐ Truy cập vào <https://start.spring.io/> tạo dự án **Spring Boot** và **WebSocket**

Project	Language
<input checked="" type="radio"/> Maven Project	<input checked="" type="radio"/> Java
<input type="radio"/> Gradle Project	<input type="radio"/> Kotlin
<input type="radio"/> Groovy	
 Spring Boot	
<input type="radio"/> 3.0.0 (SNAPSHOT)	<input checked="" type="radio"/> 3.0.0 (M3)
<input type="radio"/> 2.7.2 (SNAPSHOT)	<input checked="" type="radio"/> 2.7.1
<input type="radio"/> 2.6.10 (SNAPSHOT)	<input type="radio"/> 2.6.9
 Project Metadata	
Group	com.likelion
Artifact	websocket
Name	websocket
Description	
Package name	com.likelion.websocket
Packaging	<input checked="" type="radio"/> Jar
Java	<input type="radio"/> 18 <input type="radio"/> 17 <input checked="" type="radio"/> 11 <input type="radio"/> 8

Dependencies
ADD ... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

WebSocket MESSAGING
Build WebSocket applications with SockJS and STOMP.

- ❑ Mở dự án và kiểm tra file pom.xml với các dependency sau

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-websocket</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- ❑ Từ thư mục gốc của dự án(com.likelion.websocket) tạo package **config**
- ❑ Trong package **config** tạo class **WebSocketConfig**

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        // Từ STOMP trong tên phương thức đến từ Spring frameworks STOMP implementation
        // Đăng ký một websocket endpoint mà các máy khách sẽ sử dụng để kết nối với máy chủ /ws
        registry.addEndpoint("/ws").withSockJS();
    }
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        // Xác định các thư có đích bắt đầu bằng "/app" sẽ được định tuyến đến các phương thức xử lý tin nhắn
        registry.setApplicationDestinationPrefixes("/app");
        // Được định tuyến tới Message Broker(ở đây đang dùng bộ nhớ máy)
        // Có thể sử dụng Message Broker bất kỳ để thay thế ví dụ RabbitMQ hoặc ActiveMQ
        registry.enableSimpleBroker("/topic");
    }
}
```

❑ Diễn giải

- ❑ Annotation **@EnableWebSocketMessageBroker** để bật tính năng WebSocket
- ❑ Thực hiện implement interface **WebSocketMessageBrokerConfigurer** và cung cấp việc triển khai thực hiện một số phương thức của nó để cấu hình kết nối websocket
 - ❖ **STOMP** trong tên phương thức
 - ❖ **STOMP** là viết tắt của **Simple Text Oriented Messaging Protocol**. Nó là một giao thức nhắn tin xác định định dạng và quy tắc trao đổi dữ liệu
 - ❖ **WebSocket** chỉ là một **giao thức truyền thông**. Nó không xác định những thứ như - Cách gửi thư chỉ cho những người dùng đã đăng ký một **chủ đề** cụ thể hoặc **cách gửi thư đến một người dùng cụ thể**
 - ❖ Chúng ta cần **STOMP** cho các chức năng này

- ❑ Từ thư mục gốc của dự án(com.likelion.websocket) tạo package **model**
- ❑ Trong package **model** tạo class **ChatMessage**

```
@Data  
public class ChatMessage {  
    private MessageType type;  
    private String content;  
    private String sender;  
  
    public enum MessageType {  
        CHAT,  
        JOIN,  
        LEAVE  
    }  
}
```

- ❑ **ChatMessage** model là nơi chứa tin nhắn sẽ được trao đổi giữa khách hàng và máy chủ

- ❑ Từ thư mục gốc của dự án(com.likelion.websocket) tạo package **controller**
- ❑ Trong package controller tạo class **ChatController**

```
@Controller
public class ChatController {

    // Một thư có đích /app/chat.sendMessage sẽ được định tuyến đến tới phương thức sendMessage()
    @MessageMapping("/chat.sendMessage")
    @SendTo("/topic/public")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage) {
        return chatMessage;
    }

    // Một thư có đích /app/chat.addUser sẽ được định tuyến đến tới phương thức addUser()
    @MessageMapping("/chat.addUser")
    @SendTo("/topic/public")
    public ChatMessage addUser(@Payload ChatMessage chatMessage,
                               SimpMessageHeaderAccessor headerAccessor) {
        // Add username in web socket session
        headerAccessor.getSessionAttributes().put("username", chatMessage.getSender());
        return chatMessage;
    }
}
```

- ☐ Từ thư mục gốc của dự án(com.likelion.websocket) tạo tạo class **WebSocketEventListener**

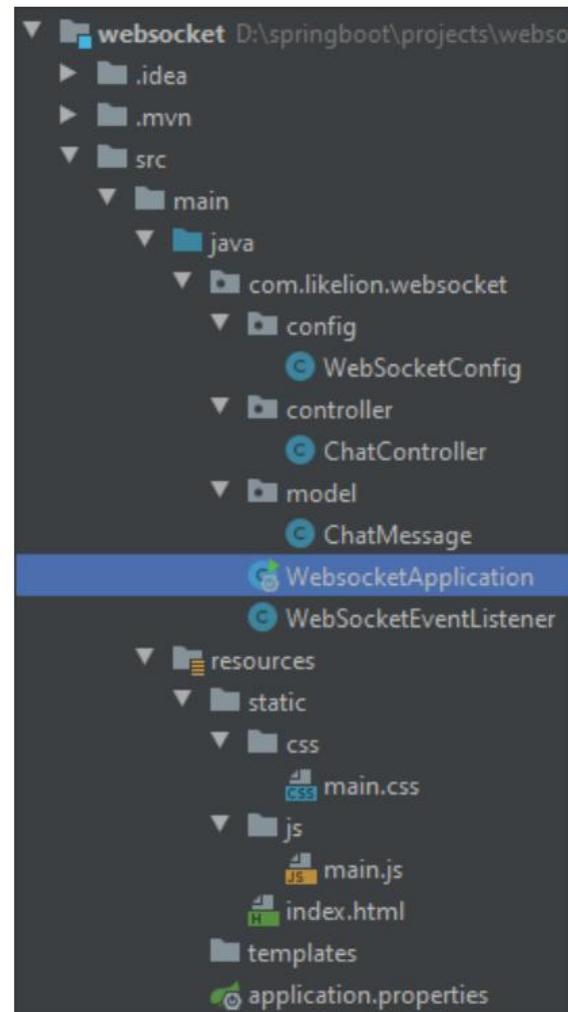
```
@Component
public class WebSocketEventListener {
    private static final Logger logger = LoggerFactory.getLogger(WebSocketEventListener.class);

    @Autowired
    private SimpMessageSendingOperations messagingTemplate;

    // sự kiện SessionConnectedEvent, dùng tham gia trong phương thức addUser() được xác định bên trong ChatController
    @EventListener
    public void handleWebSocketConnectListener(SessionConnectedEvent event) {
        logger.info("Received a new web socket connection");
    }

    // sự kiện SessionDisconnect, trích xuất tên người dùng từ websocket session
    // và phát sóng sự kiện người dùng rời khỏi cho tất cả các khách hàng được kết nối
    @EventListener
    public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {
        StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());
        String username = (String) headerAccessor.getSessionAttributes().get("username");
        if(username != null){
            logger.info("User Disconnected : " + username);
            ChatMessage chatMessage = new ChatMessage();
            chatMessage.setType(ChatMessage.MessageType.LEAVE);
            chatMessage.setSender(username);
            messagingTemplate.convertAndSend("/topic/public", chatMessage);
        }
    }
}
```

❑ Cấu trúc các file của dự án



☐ Tạo giao diện người dùng bên trong thư mục src/main/resources

```
static
  └── css
      └── main.css
  └── js
      └── main.js
  └── index.html
```

☐ File index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Spring Chat Application</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.7/dist/js/bootstrap.bundle.min.js" integrity="sha384-Olpc5wGNpzbKnEzJm9x7ItI4oJr6y7gkqoWsfwCj41mQK8Z8a4QfWVJ+PZtLp" crossorigin="anonymous"></script>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.7/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Olpc5wGNpzbKnEzJm9x7ItI4oJr6y7gkqoWsfwCj41mQK8Z8a4QfWVJ+PZtLp" crossorigin="anonymous"/>
    <script src="main.js" type="text/javascript"></script>
  </head>
  <body>
    <div class="container" style="text-align: center; margin-top: 10px;">
      <div class="row" style="margin-bottom: 10px;">
        <div class="col-4" style="text-align: left; padding-right: 10px;">
          <input type="text" id="username" style="width: 100%; height: 30px; border: 1px solid #ccc; border-radius: 5px; padding: 5px; font-size: 14px;" placeholder="Nhập tên người dùng" />
        <div style="text-align: right; margin-top: -10px;">
          <button type="button" id="join" style="border: 1px solid #ccc; border-radius: 5px; width: 100px; height: 30px; background-color: #f0f0f0; color: #007bff; font-size: 14px; font-weight: bold; padding: 5px; margin-right: 10px;">Join</button>
          <button type="button" id="leave" style="border: 1px solid #ccc; border-radius: 5px; width: 100px; height: 30px; background-color: #f0f0f0; color: #007bff; font-size: 14px; font-weight: bold; padding: 5px;">Leave</button>
        </div>
      </div>
      <div class="col-4" style="text-align: center; position: relative; height: 350px; border: 1px solid #ccc; border-radius: 5px; overflow: hidden; border-bottom: none;">
        <div style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; background-color: #f0f0f0; border-radius: 5px; border-bottom: 1px solid #ccc; z-index: 1;">
          <div style="position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); width: 150px; height: 150px; border: 1px solid #ccc; border-radius: 50%; background-color: #fff; border-bottom: 1px solid #ccc; z-index: 2;">
            <img alt="Placeholder for user profile picture" style="width: 100%; height: 100%; border-radius: 50%; border: 1px solid #ccc; border-bottom: 1px solid #ccc; object-fit: cover; border: 1px solid #ccc; border-radius: 50%; border-bottom: 1px solid #ccc; z-index: 3;" />
          </div>
        </div>
        <div style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; background-color: #f0f0f0; border-radius: 5px; border-bottom: 1px solid #ccc; z-index: 1;">
          <div style="position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); width: 150px; height: 150px; border: 1px solid #ccc; border-radius: 50%; background-color: #fff; border-bottom: 1px solid #ccc; z-index: 2;">
            <img alt="Placeholder for user profile picture" style="width: 100%; height: 100%; border-radius: 50%; border: 1px solid #ccc; border-bottom: 1px solid #ccc; object-fit: cover; border: 1px solid #ccc; border-radius: 50%; border-bottom: 1px solid #ccc; z-index: 3;" />
          </div>
        </div>
      </div>
    </div>
    <div style="text-align: center; margin-top: 10px;">
      <div style="border: 1px solid #ccc; border-radius: 5px; width: 300px; height: 30px; margin-bottom: 10px;">
        <input type="text" id="message" style="width: 100%; height: 100%; border: 0; border-radius: 5px; padding: 5px; font-size: 14px;" placeholder="Nhập tin nhắn..." />
      <button type="button" id="send" style="border: 1px solid #ccc; border-radius: 5px; width: 100px; height: 30px; background-color: #f0f0f0; color: #007bff; font-size: 14px; font-weight: bold; padding: 5px; margin-right: 10px;">Send</button>
      <button type="button" id="clear" style="border: 1px solid #ccc; border-radius: 5px; width: 100px; height: 30px; background-color: #f0f0f0; color: #007bff; font-size: 14px; font-weight: bold; padding: 5px;">Clear</button>
    </div>
  </div>
</body>
```

☐ File main.js

```
$(document).ready(function() {
  var socket = io('ws://localhost:8080');
  var messageArea = $('#message-area');
  var messageForm = $('#message-form');
  var messageInput = $('#message-input');
  var messageSend = $('#send');
  var messageClear = $('#clear');

  messageForm.on('submit', function(e) {
    e.preventDefault();
    var message = $(this).find('input').val();
    if (message === '') {
      alert('Nhập tin nhắn');
      return;
    }
    socket.emit('new message', {username: 'User', message: message});
    $(this).find('input').val('');
  });

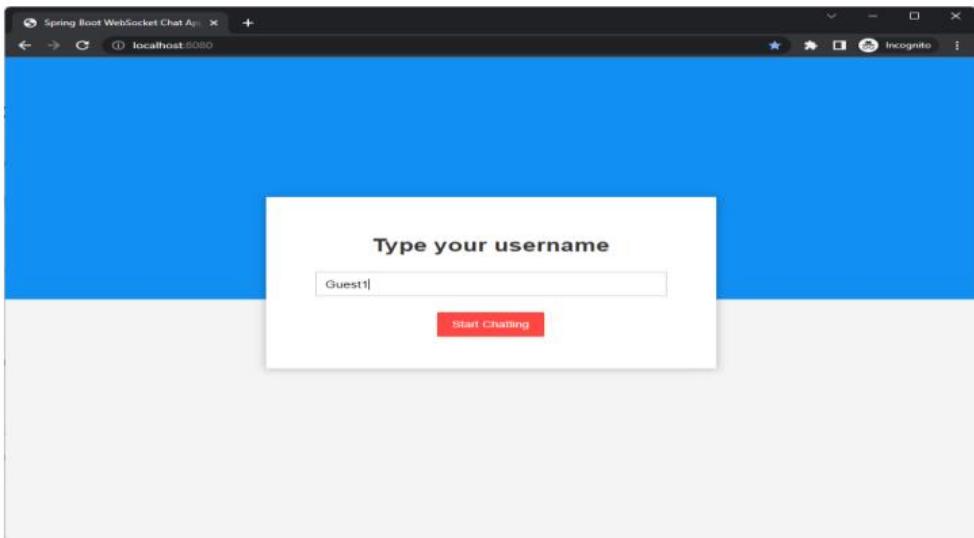
  messageSend.on('click', function() {
    messageForm.submit();
  });

  messageClear.on('click', function() {
    messageInput.val('');
  });

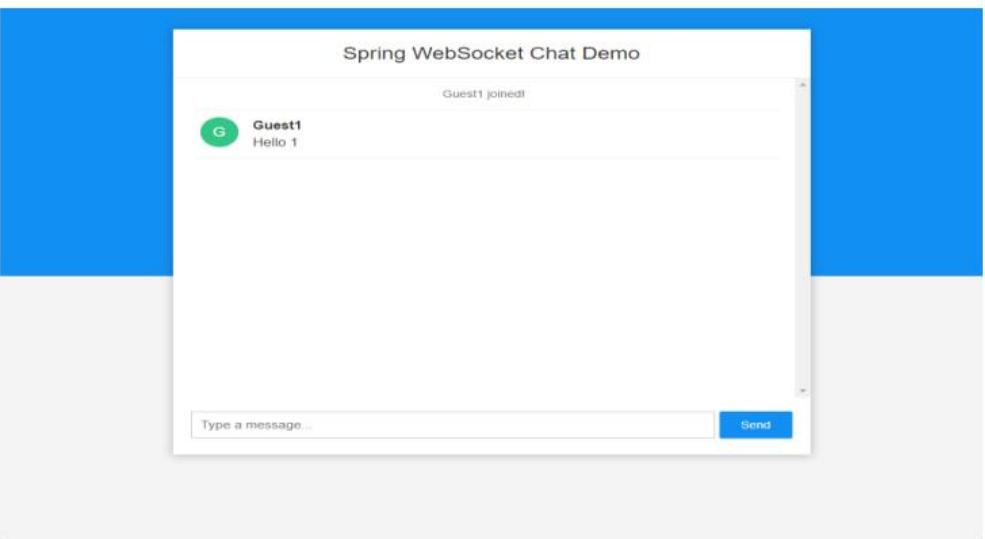
  socket.on('new message', function(data) {
    var messageDiv = $(`<div>${data.message}</div>`);
    messageArea.append(messageDiv);
    messageDiv.scrollTop(messageArea[0].scrollHeight);
  });
});
```

☐ File main.css

- ❑ Khởi chạy Spring Boot
- ❑ Truy cập vào <http://localhost:8080/>
- ❑ Mở trình duyệt **ẩn danh**, nhập tên
Guest1



- ❑ Bắt đầu chat: Hello 1

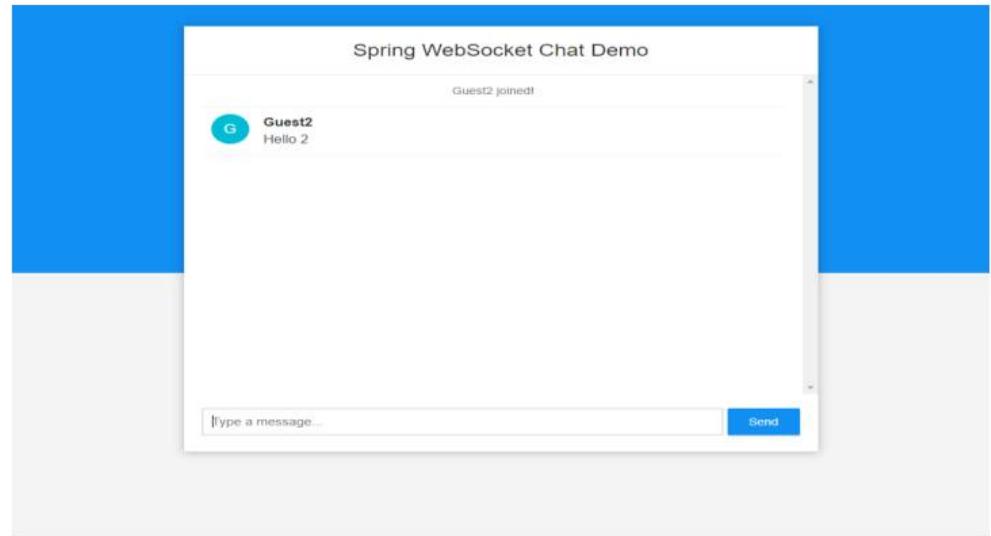


- ❑ Mở tab mới và Truy cập vào

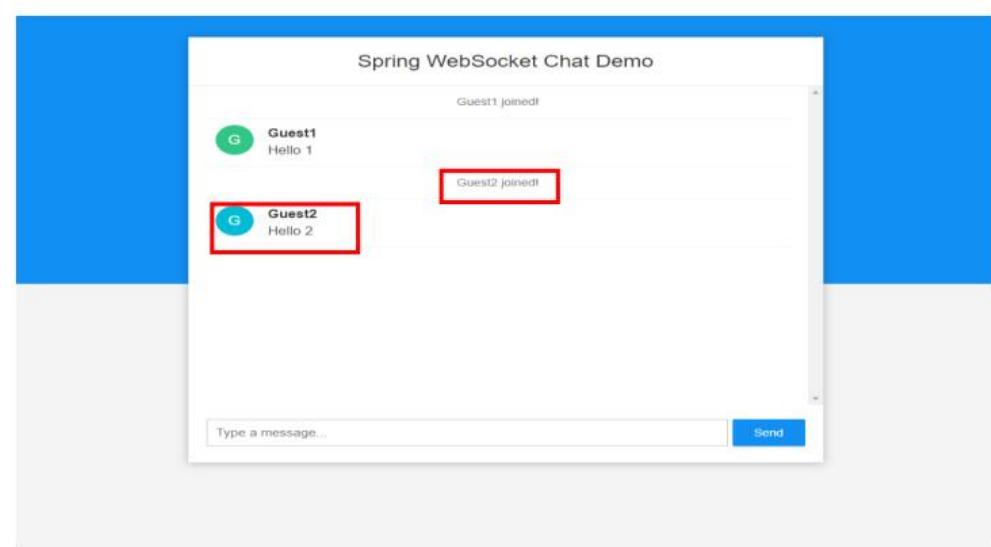
<http://localhost:8080/>

- ❑ Nhập tên **Guest2**

- ❑ Bắt đầu chat: Hello 2



- ❑ Quay về tab **ẩn danh**, cửa sổ của **Guest1** xem kết quả



- ❑ **Middleware** là những đoạn mã trung gian nằm giữa các **request** và **response**, thi hành các mệnh lệnh tương ứng trên request đó. Sau khi hoàn thành nó response (trả về) hoặc chuyển kết quả ủy thác cho một Middleware khác trong hàng đợi
- ❑ **Message broker** là một module trung gian trung chuyển message từ người **gửi** đến người **nhận**
- ❑ **Message broker** là tiếp nhận những message từ các ứng dụng và thực hiện một thao tác nào đó
- ❑ Hiện tại có rất nhiều các **message broker software** có thể kể đến như: Amazon Web Services (**AWS**), Apache **Kafka**, Apache **ActiveMQ**...
- ❑ **Hầu như** mọi công ty hàng đầu trên thế giới đều đang sử dụng **kafka** trong nền tảng **cơ sở hạ tầng** của mình
- ❑ Nền tảng Kafka stream được viết bởi ngôn ngữ **Scala** và **Java** nhằm mục đích cung cấp dịch vụ **xử lý** sự kiện dựa trên **thời gian thực** với **độ trễ thấp** và **thông lượng cao**
- ❑ **Confluent** định nghĩa đây là một nền tảng **stream**, có nhiều bài viết khác lại nói về **Kafka** là một **message broker**

- ❑ **Kafka** là một hệ thống message theo cơ chế **Pub-Sub**
- ❑ Các thành phần trong Kafka: **Producer, Topic, Messages, Partitions, Consumer, Broker, Zookeeper**
- ❑ **Redis** là tên viết tắt của **Remote Dictionary Server**, là kho dữ liệu **khóa-giá trị**, trong **bộ nhớ**, mã nguồn mở và có **tốc độ** truy cập nhanh để dùng để lưu trữ dữ liệu có **cấu trúc**, làm **database**, bộ nhớ **cache**
- ❑ **Lợi ích** của việc sử dụng **Redis**: kho dữ liệu **trong bộ nhớ**, cấu trúc dữ liệu **linh hoạt**, đơn giản **dễ sử dụng**, khả năng **mở rộng**
- ❑ **Một số ứng dụng** của **Redis**: **Caching, Counter, Machine Learning, Publish/Suscribe (Pub/Sub), Queues**
- ❑ **WebSocket** là giao thức hỗ trợ giao tiếp hai chiều giữa **client** và **server** để tạo một kết nối **trao đổi dữ liệu** một cách **mạnh mẽ**. **WebSocket** hay được sử dụng trong các trường hợp yêu cầu **thời gian thực (realtime)** như các ứng dụng **chat** hay **biểu đồ**
- ❑ **Socket** giúp ta có thể kết nối các ứng dụng bằng cách sử dụng các phương thức **TCPIP** và **UDP**
- ❑ Khi ta muốn trao đổi dữ liệu giữa 2 ứng dụng thì ứng dụng này cần phải biết được thông tin về **IP** và **port** của ứng dụng kia

Cảm Ơn Bạn Đã Chăm Chỉ!

