**Spring Kafka**

### **Your Setup Breakdown**

#### **1. Kafka (Notification Service) — WSL2 Ubuntu**

* Kafka and Zookeeper installed and running.
* Expose Kafka to Windows host using:

**listeners=PLAINTEXT://0.0.0.0:9092**

**advertised.listeners=PLAINTEXT://<WSL\_IP>:9092**

* Get your WSL2 IP using: ip addr | grep eth0

ip addr | grep inet

#### **2. Producer Service — Spring Boot (Windows)**

* Sends messages to a Kafka topic like notifications.
* Kafka config in application.yml:

**bootstrap-servers: <WSL\_IP>:9092**

#### **Consumer Service — Spring Boot (Windows)**

* Subscribes to the notifications topic and processes messages.
* Same Kafka config as producer.

### **🗂️ Project Structure (3 Git Repos/Modules)**

* notification-broker → Setup guide for Kafka on WSL2.
* notification-producer → REST + KafkaTemplate to send events.
* notification-consumer → KafkaListener-based consumer.

## **✅ Step 1: Kafka Broker Setup on WSL2 (Ubuntu)**

### **🧰 Prerequisites**

Ensure you have:

* Java 11+ installed: java -version
* WSL2 Ubuntu up and running
* Network connectivity between WSL2 and Windows (test using ping)

### **📦 Install Kafka and Zookeeper**

# **Update and install Java**

sudo apt update sudo apt install default-jdk -y

# **Download Kafka (latest 3.x is fine)**

wget <https://downloads.apache.org/kafka/3.7.0/kafka_2.13-3.7.0.tgz>

tar -xvzf kafka\_2.13-3.7.0.tgz

mv kafka\_2.13-3.7.0 kafka

cd kafka

### **Configure server.properties**

Edit config/server.properties:

# **Replace with your actual WSL IP from: ip addr | grep inet**

listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT://:9092

zookeeper.connect=localhost:2181

# Start Zookeeper and Kafka

# **Terminal 1**

bin/zookeeper-server-start.sh config/zookeeper.properties

# **Terminal 2**

bin/kafka-server-start.sh config/server.properties

Verify with:

# **List topics (should return empty at first)**

bin/kafka-topics.sh --list --bootstrap-server :9092

### **Test Kafka Access from Windows**

On your Windows machine (with kafka-console-producer.sh available or using Spring Boot):

# **From Windows terminal (optional)**

kafka-console-producer.sh --broker-list :9092 --topic test

If you can produce/consume, it’s working.

======================================================================================

wsl2ubuntu@Stephen-Samuels:~$ ip addr | grep eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

inet 172.24.113.151/20 brd 172.24.127.255 scope global eth0

Great! Your **WSL2 IP address is 172.24.113.151** — that’s what your Windows-based Spring Boot producer and consumer services will use to connect to the Kafka broker.

======================================================================================

### **Update Kafka Config on WSL2**

Edit your Kafka server.properties file (kafka/config/server.properties) and update these lines:

listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT://172.24.113.151:9092

Restart Kafka Services (after config change)

# **Stop existing Kafka server (if running)**

pkill -f kafka.Kafka

# **Start Zookeeper (if not running)**

bin/zookeeper-server-start.sh config/zookeeper.properties

# **In another terminal, start Kafka**

bin/kafka-server-start.sh config/server.properties

Once that’s done, your Kafka broker is now listening on your WSL2 IP: 172.24.113.151:9092, accessible from Windows-hosted Spring Boot apps 🎯

======================================================================================

### **To Fix Kafka Not Reachable from Windows (WSL2 IPv6 Issue)**

You should **disable IPv6 inside WSL2**, using the following commands:

# **Disable IPv6 system-wide in WSL2 (run inside Ubuntu)**

sudo sysctl -w net.ipv6.conf.all.disable\_ipv6=1 sudo sysctl -w net.ipv6.conf.default.disable\_ipv6=1 sudo sysctl -w net.ipv6.conf.lo.disable\_ipv6=1

# **Make it persistent (optional)**

echo "net.ipv6.conf.all.disable\_ipv6=1" | sudo tee -a /etc/sysctl.conf

echo "net.ipv6.conf.default.disable\_ipv6=1" | sudo tee -a /etc/sysctl.conf

echo "net.ipv6.conf.lo.disable\_ipv6=1" | sudo tee -a /etc/sysctl.conf

sudo sysctl –p

Verify It’s Disabled

cat /proc/sys/net/ipv6/conf/all/disable\_ipv6

# Should return: 1

### **Why This Works**

Kafka sometimes binds to both IPv4 and IPv6, but Windows Java clients may attempt to resolve to an IPv6 address first — leading to timeouts or refused connections. Disabling IPv6 **forces Kafka to use IPv4 only**, which avoids that issue completely on WSL2.

Once this is done, restart Kafka and Zookeeper, and you’re good to go.

============================================================================

this LEADER\_NOT\_AVAILABLE warning means the **topic test hasn't been fully initialized yet**, which is common if the topic was **just created on the fly** or Kafka is still assigning a leader for its partition.

Here’s how to **fix it cleanly**:

### **✅ Option 1: Manually Create the Topic**

Instead of relying on auto-creation, manually create the topic first:

bin/kafka-topics.sh \

--create \

--topic test \

--bootstrap-server 172.24.113.151:9092 \

--replication-factor 1 \

--partitions 1

Then retry the producer:

bin/kafka-console-producer.sh \

--broker-list 172.24.113.151:9092 \

--topic test

### **Option 2: Wait a Few Seconds**

Sometimes Kafka just needs a few seconds to elect a leader for the topic partition — especially right after startup. So waiting and trying again might help, but manual creation is more reliable.

### **🧪 Verify Topic Created**

bin/kafka-topics.sh \

--list \

--bootstrap-server 172.24.113.151:9092

You should see test in the output.

Once this works, your Kafka is good to go!

======================================================================================

**Notification Microservice**: a **Spring Boot app** that exposes a REST API and **publishes messages to Kafka** running on WSL2.

## **Notification Microservice (Kafka Producer Gateway)**

### **🧱 Purpose:**

* Acts as a **gateway** — receives HTTP requests and forwards messages to Kafka.
* Connects to Kafka running at 172.24.113.151:9092 (your WSL2 broker).

### **📁 Project Structure**

css

CopyEdit

notification-service/  
├── src/  
│ └── main/  
│ ├── java/com/example/notification/  
│ │ ├── NotificationController.java  
│ │ └── KafkaProducerService.java  
│ └── resources/  
│ └── application.yml  
├── build.gradle / pom.xml  
└── ...

### **⚙️ Dependencies (Maven)**

<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>

</dependencies>

application.yml

server:

port: 8081

spring:

kafka:

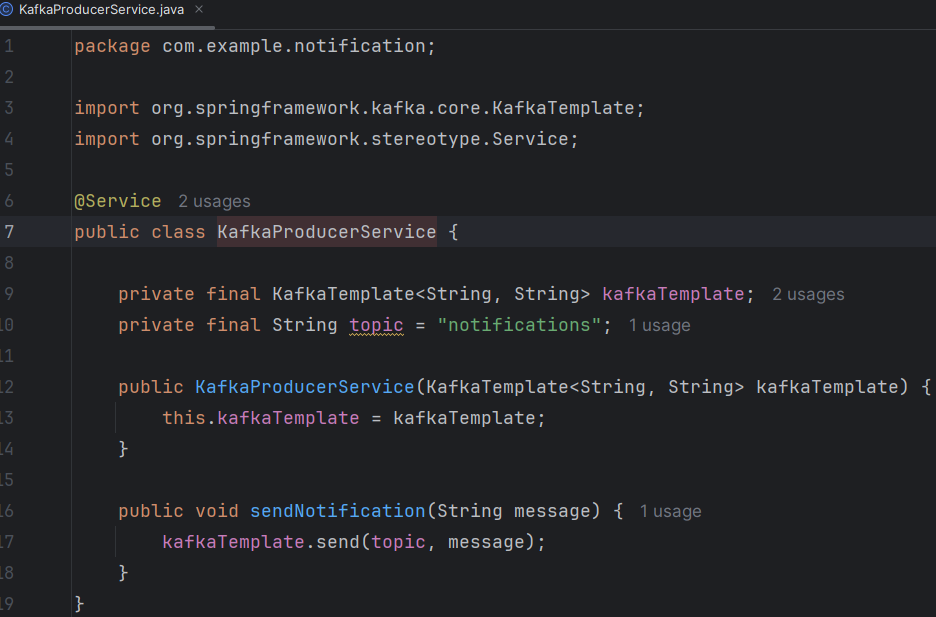
bootstrap-servers: 172.24.113.151:9092

producer:

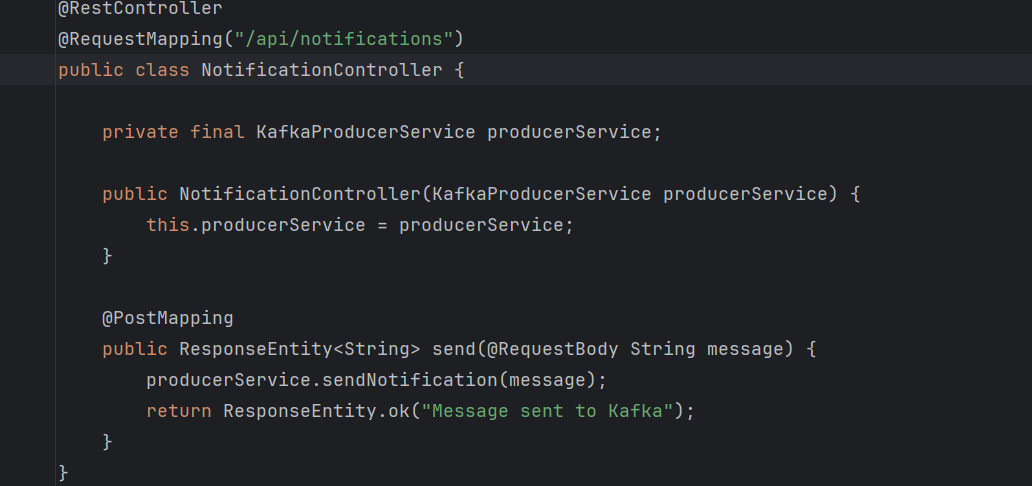
key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

**KafkaProducerService.java**



**NotificationController.ja**va



### **Run It:**

Start the app, then POST to it:

curl -X POST http://localhost:8081/api/notifications -H "Content-Type: text/plain" -d "Hello from Notification Service"

Use Real curl from Git Bash / WSL

#### **Use the Right IP from WSL to Access Windows Host**

In **WSL**, localhost refers to the WSL Linux VM — **not** your Windows host.  
 To call the Spring Boot app running on Windows **from WSL**, you need to use your **Windows host IP**.

Run this inside WSL:

cat /etc/resolv.conf | grep nameserver

This will return something like:

nameserver 172.20.112.1

That’s the IP of your **Windows host from WSL**.

### **✅ Try Again with Correct IP**

bash

CopyEdit

curl -X POST http://<windows-host-ip>:8081/api/notifications \  
 -H "Content-Type: text/plain" \  
 -d "Hello from Notification Service"

Example:

bash

CopyEdit

curl -X POST <http://172.20.112.1:8081/api/notifications> \  
 -H "Content-Type: text/plain" \  
 -d "Hello from Notification Service"

Once that works, you’ll see "Message sent to Kafka" in response, and it’ll appear in your Kafka topic.

### **Get the Correct Windows IP for WSL2**

Run this **inside WSL2**:

cat /etc/resolv.conf | grep nameserver

You'll get something like:

nameserver 172.22.240.1

This nameserver IP is how **WSL2 sees your Windows host**.

### **📤 Send the CURL Request Again**

Use that IP like this:

curl -X POST <http://172.22.240.1:8081/api/notifications> \  
 -H "Content-Type: text/plain" \  
 -d "Hello from Notifications Service"

Replace 172.22.240.1 with your actual output from cat /etc/resolv.conf.

### **🧠 Why This Happens**

In WSL2:

* localhost → refers to WSL2 itself, not Windows
* Windows host IP → accessible only via nameserver trick in /etc/resolv.conf

### **ix: Get Your Actual Windows Host IP from Inside WSL2**

Run this in WSL2 to extract the **real** Windows host IP:

#ip route | grep default

ip addr | grep inet

You’ll see something like:

#default via 172.22.208.1 dev eth0  
buntu@ICPL35327:~/kafka\_2.13-3.0.0$ ip addr | grep inet

inet 127.0.0.1/8 scope host lo

inet6 ::1/128 scope host

inet 172.26.252.235/20 brd 172.26.255.255 scope global eth0

inet6 fe80::215:5dff:fecb:4a57/64 scope link

👉 Here, 172.26.252.235 is the **Windows host IP** visible from WSL2.

### **📤 Try Your curl Again**

curl -X POST http://172.26.252.235:8081/api/notifications \  
 -H "Content-Type: text/plain" \  
 -d "Hello from Notification Service"

Replace 172.22.208.1 with the IP you got from ip route.

Let me know once you get the response "Message sent to Kafka" — then we’ll go ahead with the consumer microservice.

### **Option 1: Use Kafka Console Consumer (WSL2)**

You can use the built-in Kafka console consumer to check that the message landed in the topic:

bin/kafka-console-consumer.sh --bootstrap-server 172.26.252.235:9092 --topic notifications

--from-beginning

This will listen to messages on the notifications topic and print them as they arrive — including what your Spring Boot app sent.

### **✅ Option 2: Spring Boot Consumer Service (Next Step)**

You can also build a **Spring Boot consumer microservice** that uses @KafkaListener to consume messages — we’ll do this next.

### **Check if Topic Was Auto-Created**

Run this on WSL2:

bin/kafka-topics.sh --list --bootstrap-server 172.24.113.151:9092

If you see notifications listed, it was created automatically when the Spring Boot producer sent the first message.

### **🛠️ Recommended: Manually Create the Topic**

Even if auto-creation works, in production and good practice, **you should explicitly create the topic** to control its partition count and replication factor:

bin/kafka-topics.sh \

--create \

--topic notifications \

--bootstrap-server 172.24.113.151:9092 \

--replication-factor 1 \

--partitions 1

Once created, the topic is ready for use by both the producer and the consumer.

==================================================================

### **How It Works**

If auto.create.topics.enable=true:

* You can just send messages to any topic name — Kafka will create that topic for you behind the scenes.
* Partitions: usually 1
* Replication: usually 1 (which is fine for dev but not for production)

If false:

* Producers/consumers will throw an error if the topic doesn’t already exist.
* You must explicitly create topics via CLI or Admin API.

### **🔧 Where to Check or Set It**

In your WSL2 Kafka broker config file:

nano ~/kafka/config/server.properties

Look for:

auto.create.topics.enable=true

If it’s not there, Kafka defaults to true. You can **add or change it** if needed.

After changing it, restart Kafka:

bin/kafka-server-stop.sh

bin/kafka-server-start.sh config/server.properties

================================================================================

### **Let’s Build the Consumer Microservice (Spring Boot)**

This service will:

* Connect to Kafka
* Listen on the notifications topic
* Automatically print or process messages

## **Kafka Consumer Microservice**

### **🧱 Purpose:**

* Listens to the notifications topic
* Logs or processes incoming messages from Kafka

### **📁 Project Structure**

css

CopyEdit

notification-consumer/  
├── src/  
│ └── main/  
│ ├── java/com/example/consumer/  
│ │ ├── KafkaConsumerService.java  
│ └── resources/  
│ └── application.yml  
├── build.gradle / pom.xml  
└── ...

### **⚙️ Dependencies (Maven)**

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

### **🧠 KafkaConsumerService.java**

@Service  
public class KafkaConsumerService {  
  
 @KafkaListener(topics = "notifications", groupId = "notification-group")  
 public void listen(String message) {  
 System.out.println("Received message: " + message);  
 }  
}

Make sure @EnableKafka is in your main application class:

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

### **⚙️ application.yml**

spring:  
 kafka:  
 bootstrap-servers: 172.24.113.151:9092  
 consumer:  
 group-id: notification-group  
 auto-offset-reset: earliest  
 key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

### **🏃 Run the App**

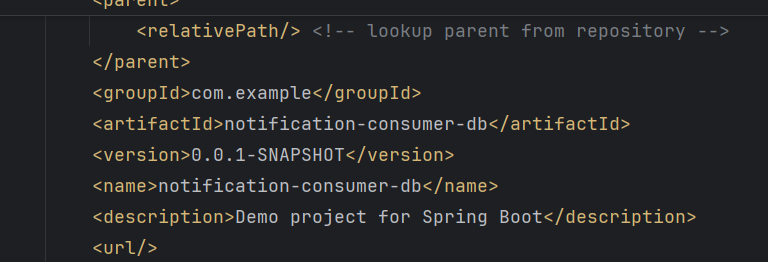
Start the consumer app — it will automatically connect to Kafka and start listening. When you send another message via the notification microservice, you'll see:

Received message: Hello from Notification Service

===========================================================================

**Consumer-DB**

================



<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 </dependency>  
  
 <!-- Spring Data JPA -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <!-- H2 or any other DB (for dev) -->  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </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>

server:  
 port: 8082  
  
spring:  
 datasource:  
 url: jdbc:h2:mem:testdb  
 driver-class-name: org.h2.Driver  
 username: sa  
 password: ""  
 h2:  
 console:  
 enabled: true  
 path: /h2-console  
 jpa:  
 hibernate:  
 ddl-auto: update  
 show-sql: true  
 kafka:  
 bootstrap-servers: 172.24.113.151:9092  
 consumer:  
 group-id: notification-group  
 auto-offset-reset: earliest  
 key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

package com.example.consumer.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
import java.time.LocalDateTime;  
  
@Entity  
public class Notification {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String message;  
  
 private LocalDateTime receivedAt = LocalDateTime.*now*();  
  
 public Notification() {  
 }  
  
 public Notification(String message) {  
 this.message = message;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public LocalDateTime getReceivedAt() {  
 return receivedAt;  
 }  
  
 public void setReceivedAt(LocalDateTime receivedAt) {  
 this.receivedAt = receivedAt;  
 }  
// getters/setters  
  
}

package com.example.consumer.repository;  
  
import com.example.consumer.entity.Notification;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface NotificationRepository extends JpaRepository<Notification, Long> {  
}

package com.example.consumer;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.kafka.annotation.EnableKafka;  
  
@SpringBootApplication  
@EnableKafka  
public class NotificationConsumerApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(NotificationConsumerApplication.class, args);  
 }  
  
}