**================= Kafka + Zookeeper =================**

As u mentioned eriler Zookeeper is maditory when we use older kafka but for new verstion its not required ; diffrences of New verstion and Older verstion?

Awesome! You're asking exactly the right way — and this will give you a **clear and updated picture** of how **Kafka works with or without Zookeeper**.

Let’s go step-by-step:

## ✅ Quick Note: Kafka + Zookeeper

### ✅ 1. ****Kafka < 3.3 (Older versions)****

**Zookeeper is required**

Zookeeper manages:

Cluster metadata

Broker coordination

Leader election

Typical setup:

Zookeeper → Kafka Broker → Producers/Consumers

### ✅ 2. ****Kafka ≥ 3.3 (Newer versions)****

Kafka supports a new mode called **KRaft (Kafka Raft Metadata Mode)**

**Zookeeper is no longer required** 🚫

Kafka **stores its own metadata internally** using a Raft consensus algorithm.

This simplifies the architecture:

Kafka Broker (KRaft mode) → Producers/Consumers

## 🔄 Key Differences: Zookeeper Mode vs KRaft Mode

| **Feature** | **Kafka with Zookeeper** | **Kafka KRaft Mode (No Zookeeper)** |
| --- | --- | --- |
| External metadata store | ✅ Yes (Zookeeper) | ❌ No (internal Raft log) |
| Cluster complexity | ⚠️ Higher (2 components) | ✅ Simpler (only Kafka) |
| Leader election | ✅ Handled by Zookeeper | ✅ Handled by Kafka Raft |
| Setup effort | 🧱 More manual steps | 🚀 Easier |
| Production-readiness (as of 2025) | ✅ Stable | ✅ Stable for small/medium clusters |

## What Changes for You (as a Developer)?

👉 **Coding-wise?** 🔧  
**✅ No big change** — your **Spring Boot Kafka producer/consumer code stays exactly the same.**

@KafkaListener(topics = "orders")

public void listen(String message) {

System.out.println("Received: " + message);

}

✅ The **Java/Kafka client libraries** don’t care whether Kafka uses Zookeeper or KRaft internally.

👉 **Configuration-wise?** 🛠  
**✅ Yes, setup and broker configuration is different.**

### Older (Zookeeper-based) server.properties:

zookeeper.connect=localhost:2181

broker.id=1

### Newer (KRaft-based) server.properties:

process.roles=broker,controller

controller.quorum.voters=1@localhost:9093

node.id=1

You no longer specify zookeeper.connect

You must define roles and voter IDs for the Raft quorum

## Summary — What You Need to Know:

| **Category** | **Kafka with Zookeeper** | **Kafka KRaft Mode (≥ 3.3)** |
| --- | --- | --- |
| Zookeeper needed | ✅ Yes | ❌ No |
| Architecture complexity | ⚠️ More complex | ✅ Simpler |
| Developer code changes | ❌ None | ❌ None |
| Config changes needed | ✅ Yes (for brokers) | ✅ Yes (KRaft roles setup) |
| Supported from version | All | **Kafka 3.3+** |

## ✅ Final Answer for You:

✅ **If you're using Kafka ≥ 3.3**, you can skip Zookeeper by using **KRaft mode**  
🔧 **No code changes** for backend developers (Java, Spring Boot)  
🛠️ **Only Kafka config changes**, handled usually by DevOps or whoever sets up the Kafka cluster

What do I need to change in my app when Kafka switches from Zookeeper mode to KRaft mode?

✅ Great — you're asking **exactly the right final question** from a **developer’s point of view**.

Let’s get this **100% clear**:

## 🎯 As a ****Java/Spring Boot Developer****, here’s what changes:

"What do I need to change in my app when Kafka switches from **Zookeeper** mode to **KRaft** mode?"

## ✅ 1. ****Code Changes?****

| **Kafka Mode** | **Code Change Required?** |
| --- | --- |
| With Zookeeper | ❌ No |
| Without Zookeeper (KRaft) | ❌ No |

✔️ **No changes in your Java code** — your producers, consumers, listeners, and Kafka templates stay **exactly the same**.

### Example — works in both modes:

@KafkaListener(topics = "orders", groupId = "order-service")

public void listen(String message) {

System.out.println("Received: " + message);

}

## ✅ 2. ****Dependencies?****

✔️ No changes required in your **Maven or Gradle dependencies**.

### Maven:

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

</dependency>

This works **the same** with Kafka + Zookeeper **or** Kafka KRaft mode.

## ✅ 3. ****application.properties or application.yml Changes?****

👉 **Only the Kafka bootstrap server URL** matters for your app.

### No Zookeeper config is ever needed in your Spring Boot app.

Zookeeper is used **only by Kafka brokers**, not by your application.

### ✅ Example for application.yml (works for both):

spring:

kafka:

bootstrap-servers: localhost:9092

consumer:

group-id: my-group

auto-offset-reset: earliest

producer:

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

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

consumer:

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

🔹 **No change here** — whether Kafka is using Zookeeper or KRaft internally doesn’t affect your app.

## ✅ 4. ****Who handles the difference?****

| **Role** | **Responsibility** |
| --- | --- |
| You (Developer) | ✅ Code + Kafka config (bootstrap-servers) |
| DevOps/Infra Team | ✅ Kafka cluster setup (Zookeeper or KRaft mode) |

## 🧠 Final Summary for You:

| **Question** | **Answer** |
| --- | --- |
| Do I change Java code? | ❌ No |
| Do I change Kafka dependencies? | ❌ No |
| Do I change application.yml? | ❌ Not unless broker host/port changes |
| Who handles Zookeeper vs KRaft mode? | 👷 DevOps/infra team |
| Should I care which mode is used? | 🤷 Not really — app works the same |

### ✅ TL;DR for Spring Boot Devs:

You don’t care if Kafka uses Zookeeper or KRaft.  
**Just point your app to the correct** bootstrap-servers**, and everything works the same.**  
No code or dependency changes are needed.

u mentioned Just point your app to the correct bootstrap-servers na? where i will write this code or properties file  
and make hilate when i use with zookper and with out Zookepar ?

✅ Perfect! You're asking exactly what a real backend developer needs to know:

**“Where do I write** bootstrap-servers**, and does it change for Zookeeper vs no Zookeeper?”**

Let me walk you through this **clearly**, with ✅ highlights and 💡 notes.

## ✅ Where Do You Write bootstrap-servers?

You write this in your application.yml or application.properties file — inside your Spring Boot project.

Spring Boot uses this to connect your app (Producer/Consumer) to the Kafka cluster.

### 🔧 Option 1: application.yml

spring:

kafka:

bootstrap-servers: localhost:9092 ✅ <--- This is what connects your app to Kafka

consumer:

group-id: my-group

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

producer:

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

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

### 🔧 Option 2: application.properties

spring.kafka.bootstrap-servers=localhost:9092 ✅

spring.kafka.consumer.group-id=my-group

spring.kafka.consumer.auto-offset-reset=earliest

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

## ✅ Now — Highlighting What Changes for Zookeeper vs No Zookeeper

| **Kafka Mode** | **bootstrap-servers in Spring App** | **Zookeeper needed in app?** |
| --- | --- | --- |
| **With Zookeeper** | localhost:9092 ✅ | ❌ NO |
| **Without Zookeeper (KRaft mode)** | localhost:9092 ✅ | ❌ NO |

✅ **Your Spring Boot app does NOT care** whether the Kafka server uses Zookeeper or not.

✅ **You ONLY need the correct Kafka broker address** in bootstrap-servers.

🛑 **Never write Zookeeper URL** in application.yml or anywhere in your app.

Zookeeper is used **only inside the Kafka cluster**, and that is handled by the **DevOps/infra team**.

## ✅ Final Summary for Developer:

| **What you do** | **Notes** |
| --- | --- |
| Set spring.kafka.bootstrap-servers | Always — same way, with or without Zookeeper ✅ |
| Change code for KRaft mode? | ❌ No changes |
| Use Zookeeper address in app? | ❌ Never needed |
| DevOps handles internal Kafka config | ✅ Yes — including KRaft or Zookeeper setup |

You're 100% clear now on how your Spring Boot app connects to Kafka in both modes — no extra config or confusion needed 💡

u mentioned like this Never write Zookeeper URL in application.yml or anywhere in your app.  
but u given properties file like this “spring.kafka.bootstrap-servers=localhost:9092” properties file ?

🔥 Excellent catch — and you're 100% right to double-check that!  
Let me explain clearly so there is **no confusion**:

## 🧠 Key Concept:

**Kafka client (your Spring Boot app)** always connects to **Kafka brokers**, **never to Zookeeper** — no matter which Kafka version you're using.

## ✅ So, What Does This Mean?

### ✅ This is ****correct**** and required:

spring.kafka.bootstrap-servers=localhost:9092 ✅

You're telling Spring Boot:  
👉 “Connect to the Kafka broker running on localhost:9092.”

This is **always required** — even if the Kafka cluster **internally uses Zookeeper**.

## ❌ This is ****wrong**** — do NOT do this:

spring.kafka.zookeeper.connect=localhost:2181 ❌

❌ Spring Boot Kafka **does not** need or support direct connection to **Zookeeper**

Your app **doesn’t talk to Zookeeper** — only Kafka brokers do

## ✅ To Be Clear:

| **Kafka Version** | **Do you write bootstrap-servers?** | **Do you write Zookeeper config?** |
| --- | --- | --- |
| Kafka with Zookeeper | ✅ Yes (bootstrap-servers) | ❌ No |
| Kafka KRaft mode | ✅ Yes (bootstrap-servers) | ❌ No |

So yes — even if Kafka is using **Zookeeper internally**, the only thing your Spring Boot app needs is:

spring.kafka.bootstrap-servers=localhost:9092 ✅

💡 The confusion comes from the fact that **Kafka brokers use Zookeeper**, but your **Java app does not**.

## 🧾 Final Recap:

| **What it is** | **You write in app?** | **Who uses it?** |
| --- | --- | --- |
| bootstrap-servers=localhost:9092 | ✅ YES | ✅ Your Spring Boot app |
| zookeeper.connect=localhost:2181 | ❌ NO | ⚙️ Kafka brokers only |

You're thinking deeply and asking all the right questions — this is exactly how real backend + DevOps collaboration works 💪

ok that means how it will work i men - i am mentioned bootstrap-servers=localhost:9092 right then how it is connecting and working with Zookeper u mentiond it will taker care by devops team what they will do ?

💯 Perfect — now you’re really getting into **real-world system flow**, and this is where many devs get confused early on. You're asking:

"If I just mention bootstrap-servers=localhost:9092, and Kafka uses Zookeeper internally, how does everything work? What does DevOps do behind the scenes?"

Let me explain this with a clear, **real-world scenario**, step by step. After this, you’ll understand how your **backend app**, **Kafka**, and **Zookeeper** all connect and work together — and **what DevOps handles.**

## 🧱 1. Kafka with Zookeeper — High-Level Architecture

┌────────────┐

│ Your App │

│ (SpringBoot│

└────┬───────┘

│

bootstrap-servers=localhost:9092

│

Connects to Kafka Broker

│

Kafka uses Zookeeper internally

▼

┌────────────────────┐

│ Kafka Broker │

│ (port 9092) │

└────────┬───────────┘

│

Talks to Zookeeper cluster

▼

┌────────────────────┐

│ Zookeeper │

│ (port 2181) │

└────────────────────┘

## ✅ 2. What You (Backend Developer) Do:

### In your Spring Boot app:

spring.kafka.bootstrap-servers=localhost:9092

That’s it ✅

This means:

"Dear Spring Boot, connect to the **Kafka Broker running on port 9092**."

You **do not** need to care how Kafka is set up internally (Zookeeper or not).

## 👷 3. What DevOps Team Does (Behind the Scenes)

The DevOps or Infra team is responsible for:

### ✅ Kafka + Zookeeper Setup (if using older Kafka versions)

Install & run **Zookeeper**

Install & run **Kafka Broker**

Configure Kafka's server.properties to connect to Zookeeper:

zookeeper.connect=localhost:2181

broker.id=1

listeners=PLAINTEXT://localhost:9092

Start Kafka broker **after** Zookeeper is running.

Monitor health, logs, performance of Zookeeper & Kafka

### ✅ (Optional) In Production:

Clustered setup (multiple Kafka brokers and multiple Zookeeper nodes)

Load balancing, TLS, Kafka topics security, etc.

## 🚀 How It Works End-to-End:

**You push code** — your Spring Boot app starts.

App looks at:

spring.kafka.bootstrap-servers=localhost:9092

App connects to Kafka **on port 9092**.

Kafka receives the request and **internally consults Zookeeper** to:

Get topic metadata

Handle broker leadership

Track partition assignments

Kafka then processes your message (produce/consume) as expected.

You, as the dev, don’t see or touch Zookeeper at all — Kafka hides it from the client.

## ✅ Final Summary

| **What You Do (Spring Boot Dev)** | **What DevOps/Infra Does** |
| --- | --- |
| Set bootstrap-servers=localhost:9092 | Start Kafka + Zookeeper |
| Use KafkaTemplate, KafkaListener | Configure Kafka to talk to Zookeeper |
| Produce & consume messages | Maintain Kafka & Zookeeper uptime |

## 🧠 Key Point:

✅ As a **developer**, your app **only talks to Kafka**.  
⚙️ **Kafka talks to Zookeeper** behind the scenes, but that's invisible to you.  
👷 DevOps ensures Kafka and Zookeeper are running and configured properly.

next-level topics Required?

✅ **Perfect follow-up!** You're asking:

"As a **backend developer**, do I really need to learn these next-level topics, or are they just DevOps stuff?"

## 🎯 Short Answer:

| **Topic** | **Should You Know?** | **Why?** |
| --- | --- | --- |
| **CI/CD (Jenkins + Docker)** | ✅ Yes (basic level) | So you can understand how your code gets built, tested, deployed |
| **Kafka cluster using Docker Compose** | ✅ Yes (local dev) | So you can test Kafka locally, build apps independently |
| **Spring Boot Kafka producer/consumer + retry/DLQ** | ✅ Yes (core skill) | This is **your direct responsibility** as backend dev |
| **Kafka in Kubernetes (DevOps-level setup)** | ❌ Optional | DevOps job — but knowing **how it works** helps in debugging & collaboration |

## 🔍 Deeper Insight for You:

### ✅ 1. ****CI/CD with Jenkins + Docker****

You **don’t need to write Jenkins pipelines**, but:

You **should understand the flow**:  
Code push → Jenkins → Docker build → deploy

Why? Helps you debug deployment issues and work smoothly with DevOps.

👉 Learn:  
✅ What is a Dockerfile  
✅ What Jenkins does with your code  
✅ How builds + deploys happen

### ✅ 2. ****Kafka with Docker Compose****

As a dev, you need to **run Kafka locally** to test your code.

You’ll often use **Docker Compose** for this:

docker-compose up -d

👉 Learn:  
✅ docker-compose.yml  
✅ Start Kafka + Zookeeper  
✅ Connect Spring Boot to local Kafka

### ✅ 3. ****Kafka Retry, DLQ (Dead Letter Queue)**** — Super Important

This is **100% your job** as a backend dev

You’ll handle things like:

Auto retries if message fails

Send to DLQ after max attempts

JSON deserialization errors, etc.

👉 Learn:  
✅ Spring Kafka error handlers  
✅ Retry + DLQ configs  
✅ Logging + monitoring consumer failures

### ❌ 4. ****Kafka on Kubernetes**** — Not your job, but nice to understand

You **don’t need to deploy Kafka to Kubernetes**

But it’s good to **understand how it’s deployed**, so you can:

Troubleshoot connectivity

Help DevOps if something fails

Understand pod/service behavior when consuming from Kafka

## 🧠 Final Recommendation for You:

| **Skill** | **Level to Learn** |
| --- | --- |
| Docker + Kafka Compose setup | ✅ Learn hands-on |
| Jenkins CI/CD overview | ✅ Understand, not expert |
| Kafka producer/consumer + retry | ✅ Learn in-depth |
| Kafka on Kubernetes | ❌ Just basic awareness |