Spring Boot Kafka Integration: Complete Developer Guide

A comprehensive guide covering Spring Boot's seamless integration with Apache Kafka, including auto-configuration, property-based setup, and embedded testing capabilities with extensive Java examples and production patterns.

Table of Contents

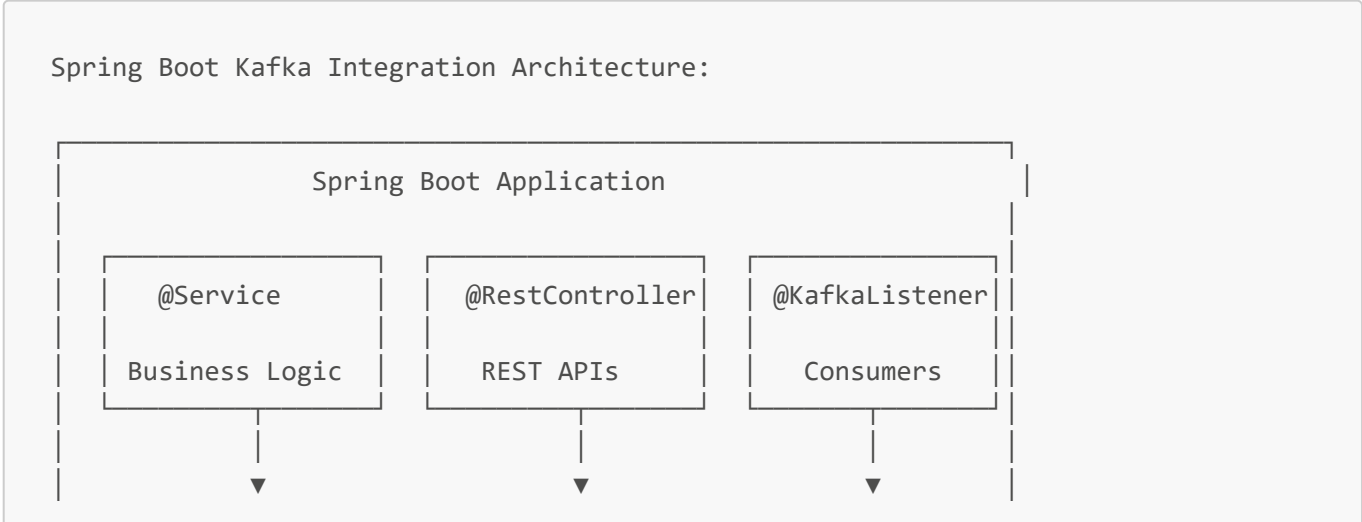
What is Spring Boot Kafka Integration?

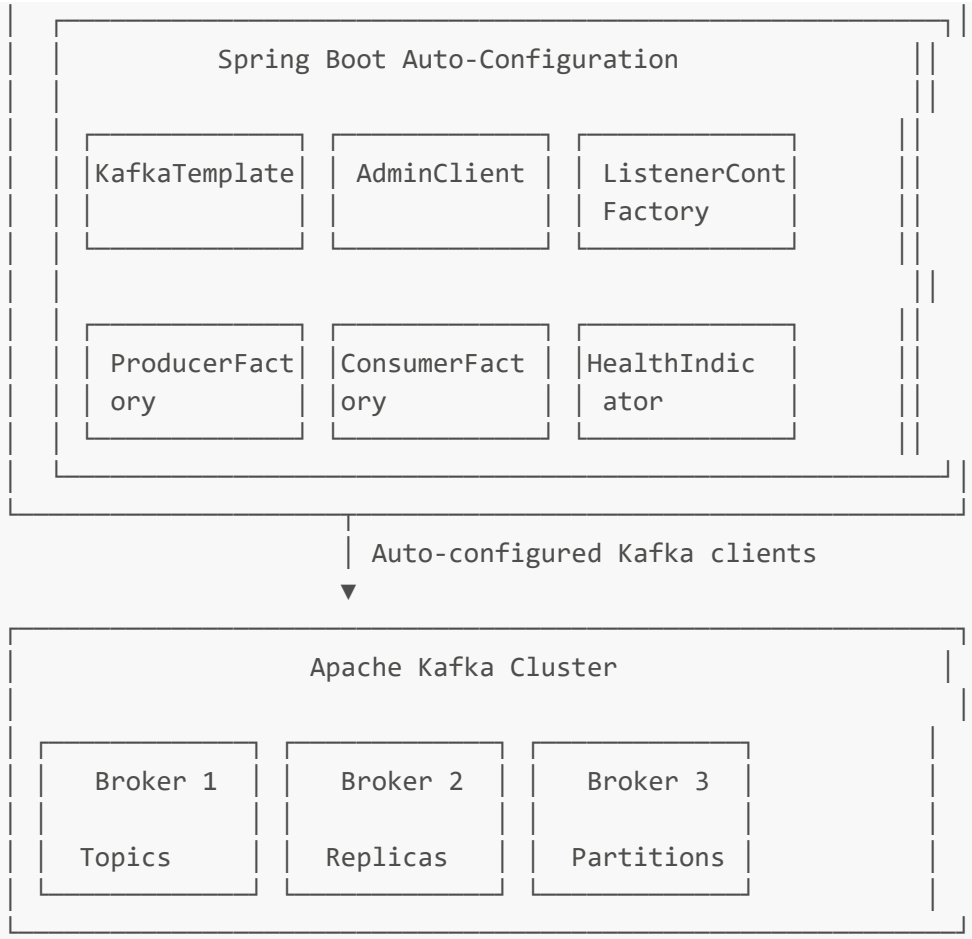
Simple Explanation: Spring Boot Kafka Integration provides seamless, convention-over-configuration setup for Apache Kafka in Spring Boot applications. It automatically configures producers, consumers, and admin clients based on properties, eliminating boilerplate configuration code and enabling rapid development of Kafka-based applications.

Why Spring Boot Kafka Integration Exists:

- **Zero Configuration**: Automatic setup of Kafka beans based on classpath detection
- **Convention Over Configuration**: Sensible defaults that work out of the box
- **Property-Based Setup**: Externalized configuration through application.properties/yaml
- **Production Ready**: Built-in health checks, metrics, and monitoring
- **Testing Support**: Embedded Kafka for integration testing
- **Enterprise Features**: Security, transactions, and reliability patterns

Spring Boot Kafka Architecture:






```
Auto-Configuration Flow:
1. Spring Boot detects spring-kafka on classpath
2. KafkaAutoConfiguration activates
3. Reads application.properties/yaml
4. Creates configured beans automatically
5. Enables @KafkaListener processing
6. Sets up health checks and metrics
```

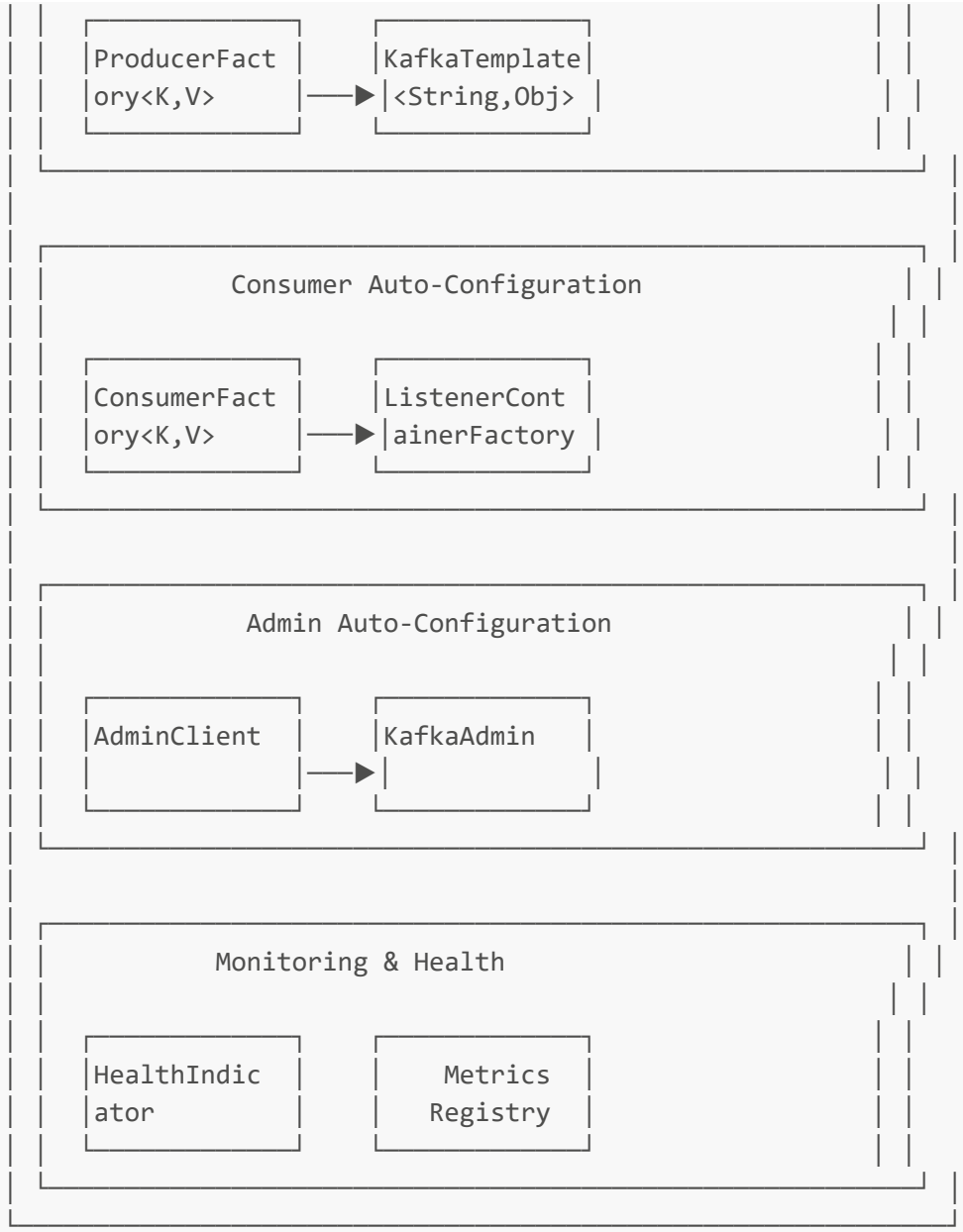
🚀 Auto-configuration (spring-kafka)

Simple Explanation: Spring Boot's auto-configuration for Kafka automatically creates and configures all necessary Kafka beans (KafkaTemplate, ConsumerFactory, ProducerFactory, etc.) when spring-kafka is on the classpath, using properties from application.yml/properties files to customize the configuration.

Auto-Configuration Components:

```
Spring Boot Kafka Auto-Configuration Components:

┌─────────────────────────────────────────────────────────┐
│              KafkaAutoConfiguration                     │
│                                                         │
│  ┌─────────────────────────────────────────────────┐    │
│  │            Producer Auto-Configuration          │    │
│  │                                                 │    │
```



Configuration Conditions:
☑ @ConditionalOnClass(KafkaTemplate.class)
☑ @ConditionalOnProperty("spring.kafka.bootstrap-servers")
☑ @ConditionalOnMissingBean(ProducerFactory.class)

Spring Boot Kafka Starter Dependencies

```xml
<!-- pom.xml - Spring Boot Kafka dependencies -->
<dependencies>
    <!-- Core Spring Boot starter -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <!-- Spring Kafka starter - enables auto-configuration -->
    <dependency>
```

```
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>

    <!-- Web starter (optional, for REST APIs) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- JSON processing support -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-json</artifactId>
    </dependency>

    <!-- Actuator for health checks and metrics -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

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

    <!-- Embedded Kafka for testing -->
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- Optional: Kafka Streams support -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
    </dependency>

    <!-- Optional: Schema Registry support -->
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>7.5.0</version>
    </dependency>
</dependencies>

<!-- Gradle equivalent - build.gradle -->
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter'
    implementation 'org.springframework.kafka:spring-kafka'
```

```
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-actuator'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.kafka:spring-kafka-test'

    // Optional dependencies
    implementation 'org.apache.kafka:kafka-streams'
    implementation 'io.confluent:kafka-avro-serializer:7.5.0'
}
```

Auto-Configuration in Action

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Spring Boot application with automatic Kafka configuration
 * No explicit @EnableKafka needed - auto-configuration handles it
 */
@SpringBootApplication  // Includes @EnableAutoConfiguration
public class KafkaSpringBootApplication {

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

        // Auto-configuration automatically creates:
        // - KafkaTemplate<String, Object>
        // - ProducerFactory<String, Object>
        // - ConsumerFactory<String, Object>
        // - ConcurrentKafkaListenerContainerFactory<String, Object>
        // - KafkaAdmin
        // - KafkaHealthIndicator
    }
}

/**
 * Auto-configured Kafka producer service
 * KafkaTemplate is automatically available for injection
 */
@Service
@lombok.extern.slf4j.Slf4j
public class AutoConfiguredKafkaProducer {

    // Auto-configured KafkaTemplate - no manual configuration needed
    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    /**
     * Send message using auto-configured KafkaTemplate
     */
```

```
    public void sendMessage(String topic, String key, Object message) {

        log.info("Sending auto-configured message: topic={}, key={}", topic, key);

        try {
            // KafkaTemplate is pre-configured with application.yml properties
            ListenableFuture<SendResult<String, Object>> future =
                kafkaTemplate.send(topic, key, message);

            // Add callback for success/failure handling
            future.addCallback(
                result -> log.info("Message sent successfully: offset={}",
                    result.getRecordMetadata().offset()),
                failure -> log.error("Failed to send message", failure)
            );

        } catch (Exception e) {
            log.error("Error sending auto-configured message", e);
            throw e;
        }
    }

    /**
     * Send to default topic (configured in properties)
     */
    public void sendToDefaultTopic(Object message) {
        kafkaTemplate.sendDefault(message);
    }

    /**
     * Send with headers
     */
    public void sendWithHeaders(String topic, String key, Object message,
                                Map<String, Object> headers) {

        ProducerRecord<String, Object> record = new ProducerRecord<>(topic, key,
message);

        // Add headers
        headers.forEach((headerKey, headerValue) ->
            record.headers().add(headerKey, headerValue.toString().getBytes()));

        kafkaTemplate.send(record);
    }
}

/**
 * Auto-configured Kafka consumer
 * @KafkaListener automatically works with auto-configuration
 */
@Component
@lombok.extern.slf4j.Slf4j
public class AutoConfiguredKafkaConsumer {
```

```
    /**
     * Simple consumer using auto-configured container factory
     */
    @KafkaListener(topics = "user-events", groupId = "user-service")
    public void consumeUserEvents(@Payload UserEvent userEvent,
                                  @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
                                  @Header(KafkaHeaders.RECEIVED_PARTITION) int
partition,
                                  @Header(KafkaHeaders.OFFSET) long offset) {

        log.info("Auto-configured consumer received: topic={}, partition={},
offset={}, user={}",
            topic, partition, offset, userEvent.getUserId());

        try {
            // Process the user event
            processUserEvent(userEvent);

        } catch (Exception e) {
            log.error("Error processing user event: userId={}, offset={}",
                userEvent.getUserId(), offset, e);
            throw e;
        }
    }

    /**
     * Batch consumer with auto-configuration
     */
    @KafkaListener(
        topics = "order-events",
        groupId = "order-batch-processor",
        containerFactory = "kafkaListenerContainerFactory"  // Auto-configured
    )
    public void consumeOrderEventsBatch(@Payload List<OrderEvent> orderEvents,
                                        @Header(KafkaHeaders.RECEIVED_TOPIC)
List<String> topics,
                                        @Header(KafkaHeaders.OFFSET) List<Long>
offsets,
                                        Acknowledgment ack) {

        log.info("Auto-configured batch consumer received: size={}, topics={}",
            orderEvents.size(),
topics.stream().distinct().collect(Collectors.toList()));

        try {
            // Process batch
            for (int i = 0; i < orderEvents.size(); i++) {
                OrderEvent order = orderEvents.get(i);
                Long offset = offsets.get(i);

                log.debug("Processing order: orderId={}, offset={}",
order.getOrderId(), offset);
                processOrderEvent(order);
            }
```

```
            // Manual acknowledgment
            ack.acknowledge();

            log.info("Batch processing completed: size={}", orderEvents.size());

        } catch (Exception e) {
            log.error("Error processing order events batch", e);
            throw e;
        }
    }

    /**
     * Error handling consumer
     */
    @KafkaListener(topics = "error-topic", groupId = "error-handler")
    public void handleErrors(@Payload String errorMessage,
                             @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
                             @Header(KafkaHeaders.EXCEPTION_MESSAGE) String
exceptionMessage) {

        log.error("Auto-configured error handler: topic={}, error={}, exception=
{}",
            topic, errorMessage, exceptionMessage);

        // Handle the error appropriately
        processError(errorMessage, exceptionMessage);
    }

    // Helper methods
    private void processUserEvent(UserEvent userEvent) {
        log.debug("Processing user event: {}", userEvent);
        // Business logic here
    }

    private void processOrderEvent(OrderEvent orderEvent) {
        log.debug("Processing order event: {}", orderEvent);
        // Business logic here
    }

    private void processError(String errorMessage, String exceptionMessage) {
        log.warn("Processing error: message={}, exception={}", errorMessage,
exceptionMessage);
        // Error handling logic here
    }
}

/**
 * Auto-configured admin operations
 */
@Service
@lombok.extern.slf4j.Slf4j
public class AutoConfiguredKafkaAdmin {
```

```
    // Auto-configured KafkaAdmin - no manual configuration needed
    @Autowired
    private KafkaAdmin kafkaAdmin;

    // Auto-configured AdminClient - available for advanced operations
    @Autowired
    private AdminClient adminClient;

    /**
     * Create topics using auto-configured admin
     */
    @PostConstruct
    public void createTopics() {

        log.info("Creating topics using auto-configured admin");

        try {
            List<NewTopic> topics = Arrays.asList(
                TopicBuilder.name("user-events")
                    .partitions(6)
                    .replicas(3)
                    .config(TopicConfig.RETENTION_MS_CONFIG, "86400000") // 1 day
                    .build(),

                TopicBuilder.name("order-events")
                    .partitions(12)
                    .replicas(3)
                    .config(TopicConfig.CLEANUP_POLICY_CONFIG,
TopicConfig.CLEANUP_POLICY_COMPACT)
                    .build(),

                TopicBuilder.name("error-topic")
                    .partitions(3)
                    .replicas(3)
                    .config(TopicConfig.RETENTION_MS_CONFIG, "604800000") // 7
days
                    .build()
            );

            // KafkaAdmin automatically creates topics that don't exist
            kafkaAdmin.createOrModifyTopics(topics.toArray(new NewTopic[0]));

            log.info("Topics created successfully with auto-configuration");

        } catch (Exception e) {
            log.error("Error creating topics with auto-configuration", e);
            throw e;
        }
    }

    /**
     * Advanced admin operations using AdminClient
     */
    public void performAdvancedAdminOperations() {
```

```
        log.info("Performing advanced admin operations");

        try {
            // Describe cluster
            DescribeClusterResult clusterResult = adminClient.describeCluster();
            Collection<Node> nodes = clusterResult.nodes().get(30,
TimeUnit.SECONDS);

            log.info("Cluster nodes: {}", nodes.size());

            // List topics
            ListTopicsResult topicsResult = adminClient.listTopics();
            Set<String> topicNames = topicsResult.names().get(30,
TimeUnit.SECONDS);

            log.info("Available topics: {}", topicNames);

            // Describe topics
            DescribeTopicsResult describeResult =
adminClient.describeTopics(topicNames);
            Map<String, TopicDescription> descriptions =
describeResult.all().get(30, TimeUnit.SECONDS);

            descriptions.forEach((topicName, description) ->
                log.info("Topic: {}, Partitions: {}", topicName,
description.partitions().size()));

        } catch (Exception e) {
            log.error("Error in advanced admin operations", e);
            throw new RuntimeException("Admin operations failed", e);
        }
    }
}

/**
 * Health check and monitoring with auto-configuration
 */
@Component
@lombok.extern.slf4j.Slf4j
public class KafkaHealthMonitor {

    // Auto-configured health indicator is available
    @Autowired(required = false)
    private KafkaHealthIndicator kafkaHealthIndicator;

    @Autowired
    private MeterRegistry meterRegistry;

    /**
     * Custom health check using auto-configured components
     */
    @EventListener(ApplicationReadyEvent.class)
    public void checkKafkaHealth() {
```

```
        log.info("Checking Kafka health with auto-configuration");

        try {
            if (kafkaHealthIndicator != null) {
                Health health = kafkaHealthIndicator.health();

                log.info("Kafka health status: {}", health.getStatus());

                if (health.getStatus() != Status.UP) {
                    log.warn("Kafka health check failed: {}",
health.getDetails());
                }
            }

            // Register custom metrics
            registerCustomMetrics();

        } catch (Exception e) {
            log.error("Error checking Kafka health", e);
        }
    }

    private void registerCustomMetrics() {

        // Counter for successful messages
        Counter.builder("kafka.messages.sent.success")
            .description("Number of successfully sent messages")
            .register(meterRegistry);

        // Counter for failed messages
        Counter.builder("kafka.messages.sent.failure")
            .description("Number of failed message sends")
            .register(meterRegistry);

        // Gauge for consumer lag
        Gauge.builder("kafka.consumer.lag")
            .description("Consumer lag in messages")
            .register(meterRegistry, this, KafkaHealthMonitor::getConsumerLag);

        log.info("Custom Kafka metrics registered");
    }

    private double getConsumerLag(KafkaHealthMonitor monitor) {
        // Implementation would calculate actual consumer lag
        return 0.0;
    }
}

// Supporting data models
@lombok.Data
@lombok.AllArgsConstructor
@lombok.NoArgsConstructor
class UserEvent {
```

```
    private String userId;
    private String action;
    private String timestamp;
    private Map<String, Object> properties;
}

@lombok.Data
@lombok.AllArgsConstructor
@lombok.NoArgsConstructor
class OrderEvent {
    private String orderId;
    private String customerId;
    private BigDecimal amount;
    private String status;
    private String timestamp;
}
```

Customizing Auto-Configuration

```
/**
 * Customizing Spring Boot Kafka auto-configuration
 */
@Configuration
@EnableConfigurationProperties(KafkaProperties.class)
@lombok.extern.slf4j.Slf4j
public class KafkaCustomConfiguration {

    /**
     * Customize auto-configured KafkaTemplate
     */
    @Bean
    @Primary
    public KafkaTemplate<String, Object>
customKafkaTemplate(ProducerFactory<String, Object> producerFactory) {

        KafkaTemplate<String, Object> template = new KafkaTemplate<>
(producerFactory);

        // Set default topic
        template.setDefaultTopic("default-events");

        // Set producer interceptors
        template.setProducerInterceptors(Arrays.asList(new
CustomProducerInterceptor()));

        // Configure send timeout
        template.setTimeout(Duration.ofSeconds(30));

        // Set message converter
        template.setMessageConverter(new StringJsonMessageConverter());
```

```
        log.info("Customized KafkaTemplate configured");

        return template;
    }

    /**
     * Customize auto-configured listener container factory
     */
    @Bean
    @Primary
    public ConcurrentKafkaListenerContainerFactory<String, Object>
customKafkaListenerContainerFactory(
            ConsumerFactory<String, Object> consumerFactory) {

        ConcurrentKafkaListenerContainerFactory<String, Object> factory =
            new ConcurrentKafkaListenerContainerFactory<>();

        factory.setConsumerFactory(consumerFactory);

        // Set concurrency
        factory.setConcurrency(4);

        // Set batch listener support
        factory.setBatchListener(true);

        // Container properties
        ContainerProperties containerProps = factory.getContainerProperties();
        containerProps.setAckMode(ContainerProperties.AckMode.MANUAL_IMMEDIATE);
        containerProps.setSyncCommits(true);
        containerProps.setCommitLogLevel(LogIfLevelEnabled.Level.DEBUG);

        // Error handling
        factory.setCommonErrorHandler(new DefaultErrorHandler(
            new FixedBackOff(1000L, 3L) // 3 retries with 1 second intervals
        ));

        // Consumer interceptors
        factory.setConsumerInterceptors(Arrays.asList(new
CustomConsumerInterceptor()));

        // Message converter
        factory.setMessageConverter(new StringJsonMessageConverter());

        log.info("Customized KafkaListenerContainerFactory configured");

        return factory;
    }

    /**
     * Additional KafkaTemplate for specific use cases
     */
    @Bean("jsonKafkaTemplate")
    public KafkaTemplate<String, Object> jsonKafkaTemplate(ProducerFactory<String,
Object> producerFactory) {
```

```
        KafkaTemplate<String, Object> template = new KafkaTemplate<>
(producerFactory);

        // Configure for JSON messaging
        template.setMessageConverter(new JsonMessageConverter());
        template.setDefaultTopic("json-events");

        return template;
    }

    /**
     * Specialized consumer factory for different message types
     */
    @Bean("avroConsumerFactory")
    public ConsumerFactory<String, GenericRecord>
avroConsumerFactory(KafkaProperties properties) {

        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
properties.getBootstrapServers());
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "avro-consumer-group");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
KafkaAvroDeserializer.class);

        // Schema registry configuration
        props.put("schema.registry.url", "http://localhost:8081");
        props.put("specific.avro.reader", false);

        return new DefaultKafkaConsumerFactory<>(props);
    }

    /**
     * Custom producer interceptor
     */
    public static class CustomProducerInterceptor implements
ProducerInterceptor<String, Object> {

        @Override
        public ProducerRecord<String, Object> onSend(ProducerRecord<String,
Object> record) {
            // Add custom headers
            record.headers().add("producer-timestamp",
                String.valueOf(System.currentTimeMillis()).getBytes());
            record.headers().add("producer-id", "custom-producer".getBytes());

            return record;
        }

        @Override
        public void onAcknowledgement(RecordMetadata metadata, Exception
exception) {
```

```
            if (exception != null) {
                log.error("Producer interceptor - send failed: partition={},
offset={}",
                    metadata != null ? metadata.partition() : -1,
                    metadata != null ? metadata.offset() : -1,
                    exception);
            } else {
                log.debug("Producer interceptor - send successful: partition={},
offset={}",
                    metadata.partition(), metadata.offset());
            }
        }

        @Override
        public void close() {
            log.info("CustomProducerInterceptor closed");
        }

        @Override
        public void configure(Map<String, ?> configs) {
            log.info("CustomProducerInterceptor configured");
        }
    }

    /**
     * Custom consumer interceptor
     */
    public static class CustomConsumerInterceptor implements
ConsumerInterceptor<String, Object> {

        @Override
        public ConsumerRecords<String, Object> onConsume(ConsumerRecords<String,
Object> records) {
            log.debug("Consumer interceptor - received {} records",
records.count());
            return records;
        }

        @Override
        public void onCommit(Map<TopicPartition, OffsetAndMetadata> offsets) {
            log.debug("Consumer interceptor - committed offsets: {}", offsets);
        }

        @Override
        public void close() {
            log.info("CustomConsumerInterceptor closed");
        }

        @Override
        public void configure(Map<String, ?> configs) {
            log.info("CustomConsumerInterceptor configured");
        }
    }
}
```

```
/**
 * Auto-configuration conditions and customization
 */
@Configuration
@ConditionalOnProperty(prefix = "app.kafka", name = "enhanced-features",
havingValue = "true")
@lombok.extern.slf4j.Slf4j
public class EnhancedKafkaConfiguration {

    /**
     * Enhanced error handler with dead letter topic
     */
    @Bean
    @ConditionalOnMissingBean
    public CommonErrorHandler enhancedErrorHandler(KafkaTemplate<String, Object>
kafkaTemplate) {

        DeadLetterPublishingRecoverer recoverer = new
DeadLetterPublishingRecoverer(kafkaTemplate,
            (record, exception) -> new TopicPartition(record.topic() + "-dlt",
-1));

        return new DefaultErrorHandler(recoverer, new ExponentialBackOff(1000L,
2.0));
    }

    /**
     * Transaction support
     */
    @Bean
    @ConditionalOnProperty(prefix = "app.kafka", name = "enable-transactions",
havingValue = "true")
    public KafkaAwareTransactionManager
kafkaTransactionManager(ProducerFactory<String, Object> producerFactory) {
        return new KafkaAwareTransactionManager(producerFactory);
    }

    /**
     * Streams support
     */
    @Bean
    @ConditionalOnClass(StreamsBuilder.class)
    @ConditionalOnProperty(prefix = "app.kafka", name = "enable-streams",
havingValue = "true")
    public StreamsBuilder streamsBuilder() {
        return new StreamsBuilder();
    }

    @PostConstruct
    public void logEnhancedConfiguration() {
        log.info("Enhanced Kafka configuration enabled");
    }
}
```

⚙ Application.properties/yaml Setup

Simple Explanation: Spring Boot provides comprehensive property-based configuration for Kafka through application.properties or application.yml files, allowing you to externalize all Kafka settings without writing Java configuration code. This approach supports environment-specific configurations and follows Spring Boot's convention-over-configuration principle.

Configuration Property Structure:

```
# Complete Spring Boot Kafka Configuration Reference

spring:
  kafka:
    # ========================================
    # Core Connection Settings
    # ========================================
    bootstrap-servers: localhost:9092,localhost:9093,localhost:9094
    client-id: my-spring-boot-app

    # ========================================
    # Producer Configuration
    # ========================================
    producer:
      # Serialization
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer:
org.springframework.kafka.support.serializer.JsonSerializer

      # Reliability & Performance
      acks: all                    # all, 0, 1
      retries: 2147483647          # Max retries
      batch-size: 65536            # 64KB batches
      linger-ms: 10                # Wait time for batching
      buffer-memory: 33554432      # 32MB buffer
      max-block-ms: 60000          # 60 seconds max block

      # Compression
      compression-type: snappy     # none, gzip, snappy, lz4, zstd

      # Idempotency & Ordering
      enable-idempotence: true
      max-in-flight-requests-per-connection: 5

      # Timeouts
      request-timeout-ms: 30000
      delivery-timeout-ms: 120000

      # Transaction support
      transaction-id-prefix: tx-   # Enables transactions when set
```

```yaml
      # Custom properties (passed directly to Kafka producer)
      properties:
        security.protocol: SASL_SSL
        sasl.mechanism: SCRAM-SHA-256
        sasl.jaas.config: |
          org.apache.kafka.common.security.scram.ScramLoginModule required
          username="producer-user"
          password="producer-password";

    # =======================================
    # Consumer Configuration
    # =======================================
    consumer:
      # Group and offset management
      group-id: my-consumer-group
      auto-offset-reset: earliest    # earliest, latest, none
      enable-auto-commit: false      # Use manual acknowledgment
      auto-commit-interval-ms: 5000

      # Deserialization
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer:
org.springframework.kafka.support.serializer.JsonDeserializer

      # Session management
      session-timeout-ms: 30000
      heartbeat-interval-ms: 10000
      max-poll-interval-ms: 300000
      max-poll-records: 500

      # Fetch configuration
      fetch-min-bytes: 1024          # 1KB minimum fetch
      fetch-max-wait-ms: 500
      max-partition-fetch-bytes: 1048576  # 1MB max per partition

      # Isolation level (for transactional producers)
      isolation-level: read_committed

      # Custom properties
      properties:
        security.protocol: SASL_SSL
        sasl.mechanism: SCRAM-SHA-256
        sasl.jaas.config: |
          org.apache.kafka.common.security.scram.ScramLoginModule required
          username="consumer-user"
          password="consumer-password";
        # JSON deserializer configuration
        spring.json.trusted.packages: com.example.events,com.example.models
        spring.json.value.default.type: com.example.events.GenericEvent

    # =======================================
    # Admin Client Configuration
    # =======================================
```

```yaml
    admin:
      close-timeout: 10s
      operation-timeout: 30s
      fail-fast: true
      modify-topic-configs: true

      # Custom admin properties
      properties:
        security.protocol: SASL_SSL
        sasl.mechanism: SCRAM-SHA-256
        sasl.jaas.config: |
          org.apache.kafka.common.security.scram.ScramLoginModule required
          username="admin-user"
          password="admin-password";

  # ========================================
  # Listener Configuration
  # ========================================
  listener:
    # Container type
    type: single    # single, batch

    # Acknowledgment
    ack-mode: manual_immediate    # record, batch, time, count, count_time,
manual, manual_immediate

    # Concurrency
    concurrency: 3

    # Poll configuration
    poll-timeout: 3s
    no-poll-threshold: 30s

    # Idle detection
    idle-between-polls: 0ms
    idle-partition-event-interval: 30s
    idle-event-interval: 30s

    # Missing topics handling
    missing-topics-fatal: true

    # Error handling
    immediate-stop: false
    log-container-config: false

    # Only for batch listeners
    only-log-record-metadata: true

  # ========================================
  # Streams Configuration (if using Kafka Streams)
  # ========================================
  streams:
    application-id: my-streams-app
    auto-startup: true
```

```
      bootstrap-servers: localhost:9092  # Can override main setting
      client-id: streams-client

      # Streams-specific properties
      properties:
        default.key.serde:
org.apache.kafka.common.serialization.Serdes$StringSerde
        default.value.serde:
org.springframework.kafka.support.serializer.JsonSerde
        commit.interval.ms: 30000
        cache.max.bytes.buffering: 10485760  # 10MB

    # =======================================
    # SSL Configuration (if using SSL)
    # =======================================
    ssl:
      trust-store-location: classpath:ssl/kafka.client.truststore.jks
      trust-store-password: truststore-password
      trust-store-type: JKS
      key-store-location: classpath:ssl/kafka.client.keystore.jks
      key-store-password: keystore-password
      key-store-type: JKS
      key-password: key-password
      protocol: TLSv1.3

    # =======================================
    # Security Configuration
    # =======================================
    security:
      protocol: SASL_SSL  # PLAINTEXT, SSL, SASL_PLAINTEXT, SASL_SSL

    jaas:
      enabled: true
      login-module: org.apache.kafka.common.security.scram.ScramLoginModule
      control-flag: required
      options:
        username: secure-user
        password: secure-password

    # =======================================
    # Common Properties (apply to all clients)
    # =======================================
    properties:
      # Connection settings
      connections.max.idle.ms: 540000
      receive.buffer.bytes: 65536
      send.buffer.bytes: 131072

      # Metric reporting
      metrics.recording.level: INFO
      metrics.num.samples: 2
      metrics.sample.window.ms: 30000

# =======================================
```

```yaml
# Application-specific Kafka Configuration
# ========================================
app:
  kafka:
    # Feature flags
    enhanced-features: true
    enable-transactions: false
    enable-streams: false
    enable-dlq: true

    # Topic configuration
    topics:
      user-events:
        name: user-events
        partitions: 6
        replication-factor: 3
        retention-ms: 86400000  # 1 day
        cleanup-policy: delete

      order-events:
        name: order-events
        partitions: 12
        replication-factor: 3
        retention-ms: 604800000  # 7 days
        cleanup-policy: compact

      dead-letter:
        name: dead-letter-topic
        partitions: 3
        replication-factor: 3
        retention-ms: 2592000000  # 30 days

    # Consumer groups
    consumer-groups:
      user-processor:
        group-id: user-event-processor
        max-poll-records: 100
        session-timeout-ms: 45000

      order-processor:
        group-id: order-event-processor
        max-poll-records: 50
        session-timeout-ms: 30000

    # Retry configuration
    retry:
      initial-interval: 1000ms
      max-interval: 10000ms
      multiplier: 2.0
      max-attempts: 5

# ========================================
# Environment-specific Profiles
# ========================================
```

```yaml
---
# Development Profile
spring:
  config:
    activate:
      on-profile: dev

  kafka:
    bootstrap-servers: localhost:9092
    security:
      protocol: PLAINTEXT
    consumer:
      auto-offset-reset: latest
    producer:
      retries: 3
    admin:
      fail-fast: false

---
# Testing Profile
spring:
  config:
    activate:
      on-profile: test

  kafka:
    bootstrap-servers: ${spring.embedded.kafka.brokers}
    consumer:
      auto-offset-reset: earliest
      group-id: test-group-${random.uuid}
    producer:
      retries: 0
      batch-size: 1

---
# Production Profile
spring:
  config:
    activate:
      on-profile: prod

  kafka:
    bootstrap-servers: ${KAFKA_BOOTSTRAP_SERVERS:kafka-1:9092,kafka-2:9092,kafka-
3:9092}
    security:
      protocol: SASL_SSL
    ssl:
      trust-store-location: ${KAFKA_SSL_TRUSTSTORE_PATH}
      trust-store-password: ${KAFKA_SSL_TRUSTSTORE_PASSWORD}
      key-store-location: ${KAFKA_SSL_KEYSTORE_PATH}
      key-store-password: ${KAFKA_SSL_KEYSTORE_PASSWORD}
      key-password: ${KAFKA_SSL_KEY_PASSWORD}
    consumer:
```

```
      auto-offset-reset: earliest
      session-timeout-ms: 30000
      max-poll-records: 100
    producer:
      acks: all
      retries: 2147483647
      enable-idempotence: true
      compression-type: snappy
    admin:
      fail-fast: true

    properties:
      sasl.mechanism: SCRAM-SHA-512
      sasl.jaas.config: |
        org.apache.kafka.common.security.scram.ScramLoginModule required
        username="${KAFKA_SASL_USERNAME}"
        password="${KAFKA_SASL_PASSWORD}";

# ========================================
# Management and Monitoring
# ========================================
management:
  endpoints:
    web:
      exposure:
        include: health,metrics,info,kafka
  endpoint:
    health:
      show-details: always
    kafka:
      enabled: true
  metrics:
    export:
      prometheus:
        enabled: true
    tags:
      application: ${spring.application.name}
      environment: ${spring.profiles.active}

# Logging configuration
logging:
  level:
    org.springframework.kafka: INFO
    org.apache.kafka: INFO
    com.example.kafka: DEBUG
  pattern:
    console: "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"
```

Properties File Format (application.properties)

```
# application.properties equivalent

# Core configuration
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.client-id=my-spring-boot-app

# Producer configuration
spring.kafka.producer.key-
serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-
serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.producer.acks=all
spring.kafka.producer.retries=2147483647
spring.kafka.producer.batch-size=65536
spring.kafka.producer.linger-ms=10
spring.kafka.producer.compression-type=snappy
spring.kafka.producer.enable-idempotence=true

# Consumer configuration
spring.kafka.consumer.group-id=my-consumer-group
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.enable-auto-commit=false
spring.kafka.consumer.key-
deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-
deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.session-timeout-ms=30000
spring.kafka.consumer.heartbeat-interval-ms=10000
spring.kafka.consumer.max-poll-records=500

# Listener configuration
spring.kafka.listener.ack-mode=manual_immediate
spring.kafka.listener.concurrency=3
spring.kafka.listener.type=single

# Admin configuration
spring.kafka.admin.fail-fast=true

# Custom properties for all clients
spring.kafka.properties[security.protocol]=SASL_SSL
spring.kafka.properties[sasl.mechanism]=SCRAM-SHA-256

# JSON deserializer configuration
spring.kafka.consumer.properties[spring.json.trusted.packages]=com.example.events,
com.example.models
spring.kafka.consumer.properties[spring.json.value.default.type]=com.example.event
s.GenericEvent

# SSL configuration
spring.kafka.ssl.trust-store-location=classpath:ssl/kafka.client.truststore.jks
spring.kafka.ssl.trust-store-password=truststore-password
spring.kafka.ssl.key-store-location=classpath:ssl/kafka.client.keystore.jks
spring.kafka.ssl.key-store-password=keystore-password
```

```
spring.kafka.ssl.key-password=key-password

# Profile-specific overrides
spring.kafka.bootstrap-servers[dev]=localhost:9092
spring.kafka.bootstrap-servers[prod]=${KAFKA_BOOTSTRAP_SERVERS}
spring.kafka.security.protocol[dev]=PLAINTEXT
spring.kafka.security.protocol[prod]=SASL_SSL
```

Property-Based Configuration Usage

```
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.context.properties.EnableConfigurationProperties;

/**
 * Property-based Kafka service demonstrating automatic configuration
 */
@Service
@EnableConfigurationProperties(KafkaConfigProperties.class)
@lombok.extern.slf4j.Slf4j
public class PropertyBasedKafkaService {

    // Auto-injected from application.yml properties
    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    @Autowired
    private KafkaConfigProperties kafkaConfig;

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

    @Value("${spring.kafka.producer.acks}")
    private String producerAcks;

    /**
     * Send message using property-configured template
     */
    public void sendMessageWithProperties(String topicKey, Object message) {

        // Get topic name from properties
        String topicName = kafkaConfig.getTopics().get(topicKey).getName();

        log.info("Sending message with property configuration: topic={},
bootstrap-servers={}, acks={}",
            topicName, bootstrapServers, producerAcks);

        try {
            ListenableFuture<SendResult<String, Object>> future =
                kafkaTemplate.send(topicName, message);

            future.addCallback(
```

```
                result -> log.info("Property-based message sent: offset={}",
                    result.getRecordMetadata().offset()),
                failure -> log.error("Property-based message failed", failure)
            );

        } catch (Exception e) {
            log.error("Error sending property-based message", e);
            throw e;
        }
    }

    /**
     * Conditional message processing based on properties
     */
    @KafkaListener(
        topics = "${app.kafka.topics.user-events.name}",
        groupId = "${app.kafka.consumer-groups.user-processor.group-id}",
        condition = "${app.kafka.enhanced-features:false}"
    )
    public void processUserEventsConditionally(@Payload UserEvent userEvent,
                                               @Header(KafkaHeaders.RECEIVED_TOPIC)
String topic) {

        log.info("Processing user event with property-based conditional listener:
userId={}, topic={}",
            userEvent.getUserId(), topic);

        // Process the event
        processUserEvent(userEvent);
    }

    /**
     * Environment-specific processing
     */
    @Profile("prod")
    @KafkaListener(topics = "${app.kafka.topics.order-events.name}")
    public void processOrderEventsProduction(@Payload OrderEvent orderEvent) {

        log.info("Processing order event in PRODUCTION mode: orderId={}",
orderEvent.getOrderId());

        // Production-specific processing logic
        processOrderEventSecurely(orderEvent);
    }

    @Profile("dev")
    @KafkaListener(topics = "${app.kafka.topics.order-events.name}")
    public void processOrderEventsDevelopment(@Payload OrderEvent orderEvent) {

        log.info("Processing order event in DEVELOPMENT mode: orderId={}",
orderEvent.getOrderId());

        // Development-specific processing (maybe with more logging)
        processOrderEventWithDebug(orderEvent);
```

```
    }

    // Helper methods
    private void processUserEvent(UserEvent userEvent) {
        log.debug("Processing user event: {}", userEvent);
    }

    private void processOrderEventSecurely(OrderEvent orderEvent) {
        log.debug("Processing order event securely: {}", orderEvent);
    }

    private void processOrderEventWithDebug(OrderEvent orderEvent) {
        log.debug("Processing order event with debug info: {}", orderEvent);
    }
}

/**
 * Custom configuration properties
 */
@ConfigurationProperties(prefix = "app.kafka")
@lombok.Data
@Component
public class KafkaConfigProperties {

    private boolean enhancedFeatures = false;
    private boolean enableTransactions = false;
    private boolean enableStreams = false;
    private boolean enableDlq = true;

    private Map<String, TopicConfig> topics = new HashMap<>();
    private Map<String, ConsumerGroupConfig> consumerGroups = new HashMap<>();
    private RetryConfig retry = new RetryConfig();

    @lombok.Data
    public static class TopicConfig {
        private String name;
        private int partitions = 3;
        private int replicationFactor = 1;
        private long retentionMs = 86400000L; // 1 day
        private String cleanupPolicy = "delete";
    }

    @lombok.Data
    public static class ConsumerGroupConfig {
        private String groupId;
        private int maxPollRecords = 500;
        private long sessionTimeoutMs = 30000L;
    }

    @lombok.Data
    public static class RetryConfig {
        private Duration initialInterval = Duration.ofSeconds(1);
        private Duration maxInterval = Duration.ofSeconds(10);
        private double multiplier = 2.0;
```

```
        private int maxAttempts = 5;
    }
}

/**
 * Property validation and monitoring
 */
@Component
@lombok.extern.slf4j.Slf4j
public class KafkaPropertyValidator {

    @Autowired
    private KafkaProperties kafkaProperties;

    @Autowired
    private KafkaConfigProperties customProperties;

    @EventListener(ApplicationReadyEvent.class)
    public void validateKafkaProperties() {

        log.info("Validating Kafka property configuration");

        List<String> warnings = new ArrayList<>();
        List<String> errors = new ArrayList<>();

        // Validate bootstrap servers
        List<String> bootstrapServers = kafkaProperties.getBootstrapServers();
        if (bootstrapServers == null || bootstrapServers.isEmpty()) {
            errors.add("Bootstrap servers must be configured");
        } else {
            log.info("Bootstrap servers configured: {}", bootstrapServers);
        }

        // Validate producer configuration
        KafkaProperties.Producer producer = kafkaProperties.getProducer();
        if (producer.getAcks() == null) {
            warnings.add("Producer acks not specified - using default");
        }

        if (!producer.getEnableIdempotence()) {
            warnings.add("Producer idempotence disabled - may lead to
duplicates");
        }

        // Validate consumer configuration
        KafkaProperties.Consumer consumer = kafkaProperties.getConsumer();
        if (consumer.getGroupId() == null || consumer.getGroupId().isEmpty()) {
            errors.add("Consumer group ID must be specified");
        }

        if (consumer.getEnableAutoCommit() == null ||
consumer.getEnableAutoCommit()) {
            warnings.add("Auto-commit enabled - consider manual acknowledgment for
reliability");
```

```
        }

        // Validate listener configuration
        KafkaProperties.Listener listener = kafkaProperties.getListener();
        if (listener.getAckMode() == null) {
            warnings.add("Listener ack mode not specified - using default");
        }

        // Validate custom properties
        validateCustomProperties(warnings, errors);

        // Log results
        if (!errors.isEmpty()) {
            log.error("Kafka property configuration errors:");
            errors.forEach(error -> log.error("  ❌ {}", error));
            throw new IllegalStateException("Invalid Kafka property
configuration");
        }

        if (!warnings.isEmpty()) {
            log.warn("Kafka property configuration warnings:");
            warnings.forEach(warning -> log.warn("  ⚠️  {}", warning));
        }

        if (errors.isEmpty() && warnings.isEmpty()) {
            log.info("✅ Kafka property configuration validation passed");
        }

        // Log configuration summary
        logConfigurationSummary();
    }

    private void validateCustomProperties(List<String> warnings, List<String>
errors) {

        // Validate topic configurations
        customProperties.getTopics().forEach((key, topicConfig) -> {
            if (topicConfig.getName() == null || topicConfig.getName().isEmpty())
{
                errors.add("Topic name must be specified for key: " + key);
            }

            if (topicConfig.getPartitions() <= 0) {
                errors.add("Topic partitions must be positive for: " +
topicConfig.getName());
            }

            if (topicConfig.getReplicationFactor() <= 0) {
                errors.add("Topic replication factor must be positive for: " +
topicConfig.getName());
            }
        });

        // Validate consumer group configurations
```

```
        customProperties.getConsumerGroups().forEach((key, groupConfig) -> {
            if (groupConfig.getGroupId() == null ||
groupConfig.getGroupId().isEmpty()) {
                errors.add("Consumer group ID must be specified for key: " + key);
            }
        });
    }

    private void logConfigurationSummary() {

        log.info("=== Kafka Configuration Summary ===");
        log.info("Bootstrap servers: {}", kafkaProperties.getBootstrapServers());
        log.info("Producer acks: {}", kafkaProperties.getProducer().getAcks());
        log.info("Producer idempotence: {}",
kafkaProperties.getProducer().getEnableIdempotence());
        log.info("Consumer group: {}",
kafkaProperties.getConsumer().getGroupId());
        log.info("Consumer auto-commit: {}",
kafkaProperties.getConsumer().getEnableAutoCommit());
        log.info("Listener ack mode: {}",
kafkaProperties.getListener().getAckMode());
        log.info("Enhanced features: {}", customProperties.isEnhancedFeatures());
        log.info("Topics configured: {}", customProperties.getTopics().size());
        log.info("Consumer groups configured: {}",
customProperties.getConsumerGroups().size());
        log.info("====================================");
    }
}
```

This completes Part 1 of the Spring Boot Kafka Integration guide, covering auto-configuration and property-based setup. The guide continues with embedded Kafka testing in the next part.