In Apache Kafka, committing offsets is an essential aspect of consumer behavior to ensure that messages are not processed multiple times and to provide fault tolerance. When using the Kafka consumer API in a Spring Boot application, you typically leverage Kafka consumer factories provided by the Spring Kafka integration. Here's how you can commit offsets in a Spring Boot application:

Configure Kafka Consumer Factory:

First, you need to configure a Kafka consumer factory bean that will be used to create Kafka consumers. This factory is usually configured with properties such as bootstrap servers, group ID, key and value deserializers, etc.

```
@Configuration
public class KafkaConfig {
   @Value("${spring.kafka.bootstrap-servers}")
   private String bootstrapServers;
   @Value("${spring.kafka.consumer.group-id}")
   private String groupId;
   @Bean
   public ConsumerFactory<String, String> consumerFactory() {
      Map<String, Object> config = new HashMap<>();
      config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
      config.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
      // Other configuration properties
      return new DefaultKafkaConsumerFactory<>(config, new StringDeserializer(), new
StringDeserializer());
   }
   // Other Kafka related beans...
}
```

Configure Kafka Listener Container Factory:

Next, you configure a Kafka listener container factory to create Kafka message listener containers. This factory is responsible for creating message listener containers that will be used to receive messages from Kafka topics.

```
@Configuration
public class KafkaConfig {
   // Kafka consumer factory bean
   @Bean
   public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory() {
      ConcurrentKafkaListenerContainerFactory<String, String> factory = new
ConcurrentKafkaListenerContainerFactory<>();
      factory.setConsumerFactory(consumerFactory());
      // Other configuration properties
      return factory;
   }
   // Other Kafka related beans...
}
```

Write Kafka Consumer: Now, you can write your Kafka consumer. You can use the @KafkaListener annotation to mark methods that should be invoked to process messages from Kafka topics. In this method, you can process the received messages and then commit the offsets.

```
@Service
public class KafkaConsumerService {
   @KafkaListener(topics = "your-topic-name", groupId = "${spring.kafka.consumer.group-id}")
   public void listen(ConsumerRecord<String, String> record) {
      // Process the received message
      System.out.println("Received message: " + record.value());
      // Commit the offset manually
      Acknowledgment acknowledgment = AcknowledgmentUtils.getAcknowledgment(record);
      if (acknowledgment != null) {
         acknowledgment.acknowledge();
      }
   }
}
```

Commit Offsets Manually: In the Kafka consumer method, you can see that we're manually committing offsets using the Acknowledgment object. This object is provided by the Spring Kafka library and allows you to acknowledge the successful processing of messages and commit offsets manually.

Enable Auto Commit (Optional): Alternatively, you can configure your consumer to automatically commit offsets by setting the ***enable.auto.commit*** property to ***true*** in the **consumer configuration**. However, manual offset committing gives you more control over when offsets are committed, which can be useful in certain scenarios.

"So, in summary, committing offsets in a Spring Boot application using the Spring Kafka integration involves configuring Kafka consumer factories, listener container factories, writing Kafka consumers with @KafkaListener annotation, and manually committing offsets in the consumer method."