The real power of Kafka comes from the ability to build applications that interact with it and make use of its features. Luckily, Kafka provides APIs that make the process of building such an application significantly easier. In this lesson, we will discuss the five Kafka APIs, and then we will demonstrate what it looks like to use one of these APIs by building a simple Kafka producer in Java.

Relevant Documentation

- Kafka APIs

Lesson Reference

1. Clone the starter project:

```
cd ~/
git clone https://github.com/linuxacademy/content-ccdak-kafka-java-connect.git
```

2. Add the necessary dependency to `build.gradle`:

```
vi build.gradle
```

3. Add the `kafka-client` dependency in the `dependencies {...}` block:

```
dependencies {
    implementation 'org.apache.kafka:kafka-clients:2.2.1'
    testImplementation 'junit:junit:4.12'
}
```

4. Edit the `main` class, and implement a simple producer:

```
vi src/main/java/com/linuxacademy/ccdak/kafkaJavaConnect/Main.java
```

```
package com.linuxacademy.ccdak.kafkaJavaConnect;

import org.apache.kafka.clients.producer.*;
import java.util.Properties;

public class Main {<pre><code>public static void main(String[] args) {
    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092");
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    Producer&lt;String, String&gt; producer = new KafkaProducer&lt;&gt;(props);
    for (int i = 0; i &lt; 100; i++) {
        producer.send(new ProducerRecord&lt;String, String&gt;("count-topic", "count", Integer.toString(i
    }
    producer.close();
}</code></pre>}
```

5. Run your code to produce some data to `count-topic`:

```
./gradlew run
```

6. Read from `count-topic` to verify that the data from your producer published to the topic successfully:

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic count-topic --from-beginning
```