

# TP07 : Apache Kafka

kiram mohammed ali

## Introduction

This report presents the practical work of TP07 on Apache Kafka, based on the official instructions provided in the TP document.<sup>1</sup> The goal is to understand distributed messaging, create Kafka topics, use producers and consumers, and build an IoT simulation system using Kafka.

## 1. Starting Zookeeper

Kafka requires Zookeeper to manage metadata and broker coordination.

### Command

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

### Screenshot

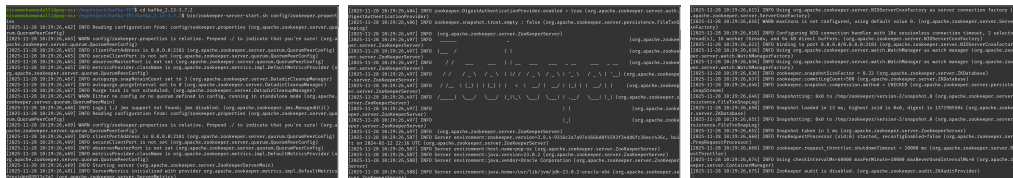


Figure 1: Zookeeper successfully started

<sup>1</sup>See TP07 PDF instructions.

## 2. Starting Kafka Broker

### Command

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

### Screenshot

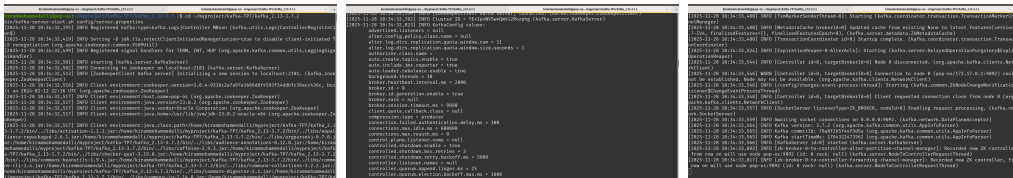


Figure 2: Kafka broker running

## 3. Creating a Kafka Topic

The topic used in this TP is named my-first-topic.

### Create Topic

```
bin/kafka-topics.sh --create \  
  --topic my-first-topic \  
  --bootstrap-server localhost:9092 \  
  --partitions 1 \  
  --replication-factor 1
```

### List Topics

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

## Screenshot

A terminal window showing the creation and verification of a Kafka topic. The user runs 'bin/kafka-topics.sh --create' with options for topic name, bootstrap server, partitions, and replication factor. The output shows the topic was created successfully. Then, the user runs 'bin/kafka-topics.sh --list' to verify the topic exists in the cluster.

```
kiramohammedalli@pop-os: ~/myproject/kafka-TP7/kafka_2.13-3.7.2
kiramohammedalli@pop-os: ~/myproject/kafka-TP7/kafka_2.13-3.7.2$ bin/kafka-topics.sh --create \
--topic my-first-topic \
--bootstrap-server localhost:9092 \
--partitions 1 \
--replication-factor 1
Created topic my-first-topic.
kiramohammedalli@pop-os: ~/myproject/kafka-TP7/kafka_2.13-3.7.2$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
my-first-topic
kiramohammedalli@pop-os: ~/myproject/kafka-TP7/kafka_2.13-3.7.2$
```

Figure 3: Topic creation and verification

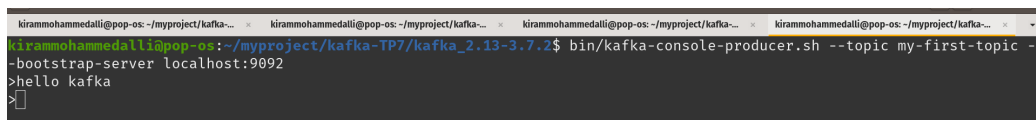
## 4. Producer - Sending Messages

### Command

```
bin/kafka-console-producer.sh \
--topic my-first-topic \
--bootstrap-server localhost:9092
```

After running the command, messages like `hello kafka` are typed.

## Screenshot

A terminal window showing the Kafka console producer in action. The user runs 'bin/kafka-console-producer.sh' with the topic and bootstrap server options. The prompt '>' appears, and the user types 'hello kafka', which is then sent to the topic.

```
kiramohammedalli@pop-os: ~/myproject/kafka-TP7/kafka_2.13-3.7.2$ bin/kafka-console-producer.sh --topic my-first-topic --bootstrap-server localhost:9092
>hello kafka
>
```

Figure 4: Kafka Console Producer sending messages

## 5. Consumer - Reading Messages

### Command

```
bin/kafka-console-consumer.sh \
--topic my-first-topic \
--bootstrap-server localhost:9092 \
--from-beginning
```

## Screenshot



Figure 5: Kafka Console Consumer receiving messages

## 6. IoT + Kafka Project

According to the TP instructions,<sup>2</sup> a simulated IoT system sends machine temperature and vibration data to Kafka. Alerts are raised when thresholds are exceeded.

### Producer Code (Python)

```
from kafka import KafkaProducer
import json
import random
import time

producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def generate_machine_data(machine_id):
    return {
        "machine_id": machine_id,
        "temperature": round(random.uniform(40, 110), 2),
        "vibration": round(random.uniform(0, 5), 2),
        "status": "OK"
    }

if __name__ == "__main__":
    while True:
```

---

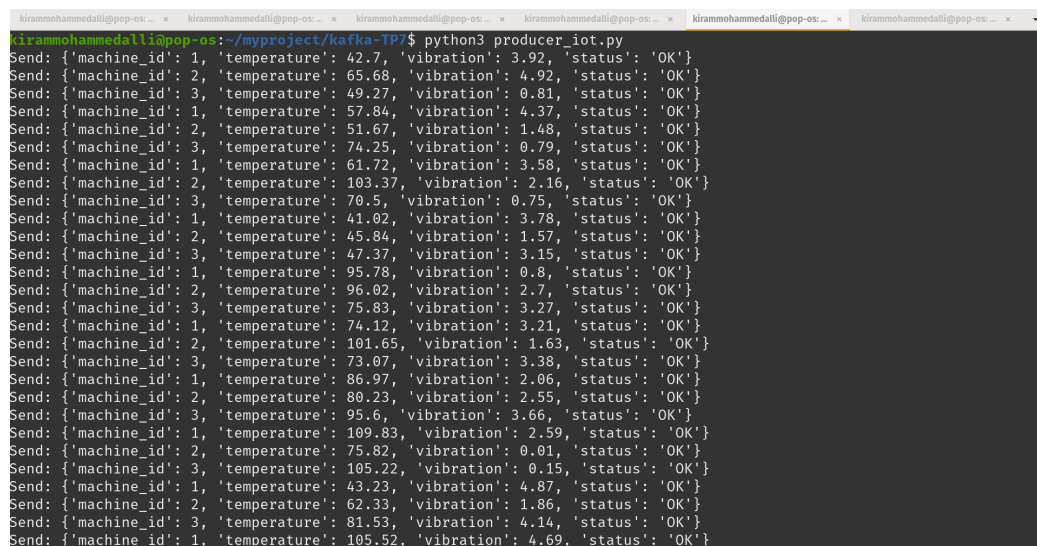
<sup>2</sup>See TP07 IoT section.

```

for machine_id in range(1, 4):
    data = generate_machine_data(machine_id)
    print("Send:", data)
    producer.send('machines-data', data)
    time.sleep(1)

```

## Producer Output



```

kirammohammedalli@pop-os: ~/myproject/kafka-TP7$ python3 producer_iot.py
Send: {'machine_id': 1, 'temperature': 42.7, 'vibration': 3.92, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 65.68, 'vibration': 4.92, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 49.27, 'vibration': 0.81, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 57.84, 'vibration': 4.37, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 51.67, 'vibration': 1.48, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 74.25, 'vibration': 0.79, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 61.72, 'vibration': 3.58, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 103.37, 'vibration': 2.16, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 70.5, 'vibration': 0.75, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 41.02, 'vibration': 3.78, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 45.84, 'vibration': 1.57, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 47.37, 'vibration': 3.15, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 95.78, 'vibration': 0.8, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 96.02, 'vibration': 2.7, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 75.83, 'vibration': 3.27, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 74.12, 'vibration': 3.21, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 101.65, 'vibration': 1.63, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 73.07, 'vibration': 3.38, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 86.97, 'vibration': 2.06, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 80.23, 'vibration': 2.55, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 95.6, 'vibration': 3.66, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 109.83, 'vibration': 2.59, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 75.82, 'vibration': 0.01, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 105.22, 'vibration': 0.15, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 43.23, 'vibration': 4.87, 'status': 'OK'}
Send: {'machine_id': 2, 'temperature': 62.33, 'vibration': 1.86, 'status': 'OK'}
Send: {'machine_id': 3, 'temperature': 81.53, 'vibration': 4.14, 'status': 'OK'}
Send: {'machine_id': 1, 'temperature': 105.52, 'vibration': 4.69, 'status': 'OK'}

```

Figure 6: IoT Producer sending machine sensor data

## Consumer Code (Python)

```

from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    'machines-data',
    bootstrap_servers='localhost:9092',
    value_deserializer=lambda m: json.loads(m.decode('utf-8')))

TEMP_LIMIT = 90

```

```

VIB_LIMIT = 3

print("Listening for machine data...")

for message in consumer:
    data = message.value
    machine_id = data["machine_id"]
    temp = data["temperature"]
    vib = data["vibration"]

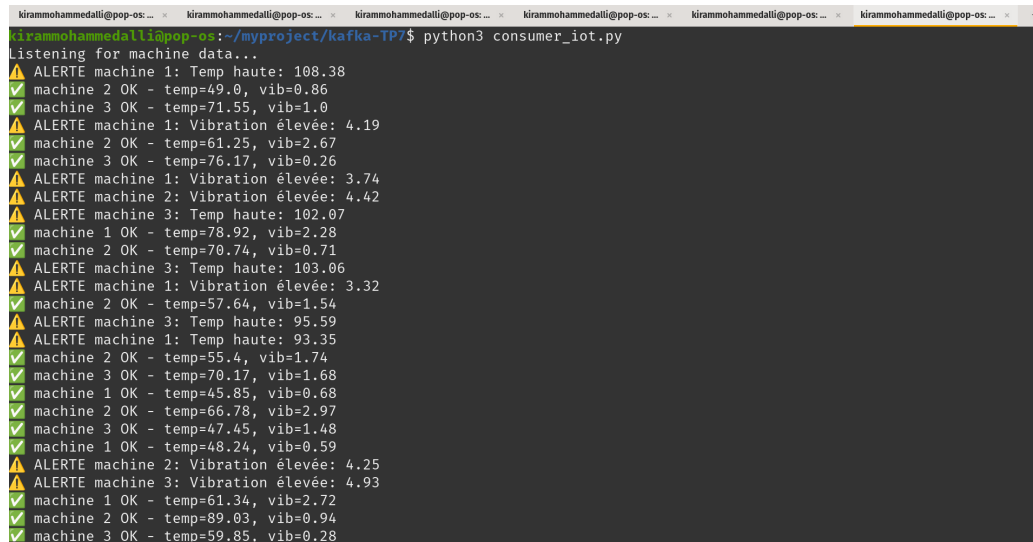
    alert = []

    if temp > TEMP_LIMIT:
        alert.append(f"Temp haute: {temp}")
    if vib > VIB_LIMIT:
        alert.append(f"Vibration élevée: {vib}")

    if alert:
        print(f"[WARNING] ALERTE machine {machine_id}: {' '.join(alert)}")
    else:
        print(f"[OK] machine {machine_id} OK - temp={temp}, vib={vib}")

```

## Consumer Output



```
kiramohammedati@pop-os: ... x kiramohammedati@pop-os: ... x kiramohammedati@pop-os: ... x kiramohammedati@pop-os: ... x kiramohammedati@pop-os: ... x kiramohammedati@pop-os: ... x
kiramohammedati@pop-os: ~/myproject/kafka-IP7$ python3 consumer_iot.py
Listening for machine data...
⚠️ALERTE machine 1: Temp haute: 108.38
✅machine 2 OK - temp=49.0, vib=0.86
✅machine 3 OK - temp=71.55, vib=1.0
⚠️ALERTE machine 1: Vibration élevée: 4.19
✅machine 2 OK - temp=61.25, vib=2.67
✅machine 3 OK - temp=76.17, vib=0.26
⚠️ALERTE machine 1: Vibration élevée: 3.74
⚠️ALERTE machine 2: Vibration élevée: 4.42
⚠️ALERTE machine 3: Temp haute: 102.07
✅machine 1 OK - temp=78.92, vib=2.28
✅machine 2 OK - temp=70.74, vib=0.71
⚠️ALERTE machine 3: Temp haute: 103.06
⚠️ALERTE machine 1: Vibration élevée: 3.32
✅machine 2 OK - temp=57.64, vib=1.54
⚠️ALERTE machine 3: Temp haute: 95.59
⚠️ALERTE machine 1: Temp haute: 93.35
✅machine 2 OK - temp=55.4, vib=1.74
✅machine 3 OK - temp=70.17, vib=1.68
✅machine 1 OK - temp=45.85, vib=0.68
✅machine 2 OK - temp=66.78, vib=2.97
✅machine 3 OK - temp=47.45, vib=1.48
✅machine 1 OK - temp=48.24, vib=0.59
⚠️ALERTE machine 2: Vibration élevée: 4.25
⚠️ALERTE machine 3: Vibration élevée: 4.93
✅machine 1 OK - temp=61.34, vib=2.72
✅machine 2 OK - temp=89.03, vib=0.94
✅machine 3 OK - temp=59.85, vib=0.28
```

Figure 7: IoT Consumer generating alerts based on thresholds

## Conclusion

This TP demonstrates the full workflow of Apache Kafka: starting Zookeeper, launching Kafka brokers, creating topics, using producers and consumers, and building an IoT real-time data processing system. Kafka proves to be a reliable and powerful distributed streaming platform for real-time analytics.