WEEK 5: Kafka Producer & Consumer with CPU Load Simulation

PRODUCER (Running on Google Colab)

Install Kafka Python client:

```
!pip install kafka-python
```

Python Script:

```
import time
from kafka import KafkaProducer
import requests
import json

# Kafka Producer Configuration
producer = KafkaProducer(
    bootstrap_servers='3.108.236.121:9092',  # Replace with your Kafka
broker address
    value_serializer=lambda v: json.dumps(v).encode('utf-8')  # Serialize
dictionary to JSON
)

while True:
    try:
        # Fetch Node Exporter Metrics
        response = requests.get('http://3.108.236.121:9100/metrics')  #
Replace with your URL
        metrics = response.text

        # Send subset of metrics to Kafka
        for line in metrics.splitlines():
            if line.startswith('node_cpu_seconds_total'):
                data = {"metric": line}
                print("Sending metric to Kafka:", data)
                producer.send('test-topic', value=data)

        print("Metrics sent to Kafka topic 'test-topic'")
        producer.flush()
        time.sleep(10)  # Send every 10 seconds

    except Exception as e:
        print(f"Error fetching or sending metrics: {e}")
        time.sleep(10)
```

CONSUMER (Running on Google Colab)

Install Kafka Python client:

```
!pip install kafka-python
```

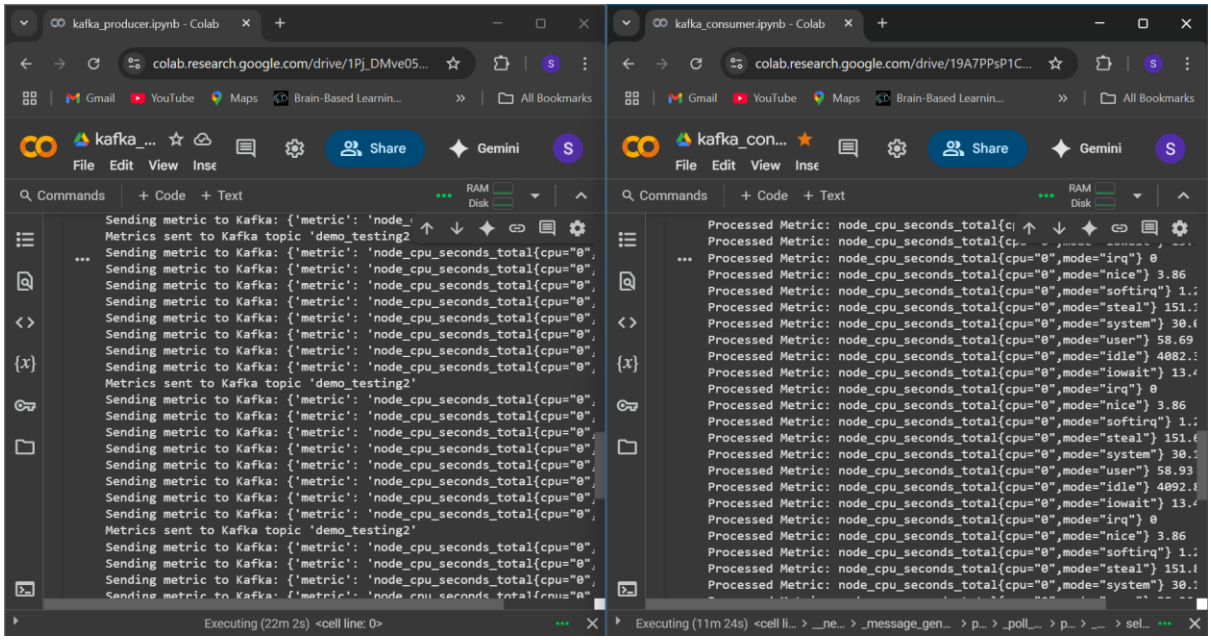
Python Script:

```
from kafka import KafkaConsumer
import json

# Kafka consumer
consumer = KafkaConsumer(
    'test-topic',
    bootstrap_servers='3.108.236.121:9092',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='metrics-consumer-group',
    value_deserializer=lambda v: json.loads(v.decode('utf-8'))
)

print("Listening for messages on topic 'test-topic'...")

try:
    for message in consumer:
        metric_name = message.value.get("metric", "N/A")
        print(f"Processed Metric: {metric_name}")
except KeyboardInterrupt:
    print("Consumer stopped.")
    consumer.close()
```



CPU LOAD SIMULATION – M1

1. Simulating CPU Load via Math Computation

Create the directory and script:

```
mkdir cpu_simulation
cd cpu_simulation
nano cpu_load_simulator.py
```

Python Script:

```
import math
import time

# Simulate CPU load by continuously performing computations
def simulate_cpu_load():
    while True:
        _ = math.factorial(100000)  # Simulating CPU-bound operation
        time.sleep(0.1)

if __name__ == "__main__":
    simulate_cpu_load()
```

Run the script in the background:

```
python3 cpu_load_simulator.py &
```

---

2. View CPU Load Using `top`

```
top
```

This command provides a dynamic real-time view of CPU, memory usage, and running processes.

---

3. Using psutil to Export CPU Usage via Prometheus

Install and edit script:

```
nano cpu_load_simulator1.py
```

Python Script:

```
from prometheus_client import start_http_server, Gauge
import time
import psutil

# Create a metric to track CPU usage
cpu_usage_gauge = Gauge('python_cpu_seconds_total', 'Total CPU usage by
Python')
```

```
# Start Prometheus server to expose metrics
start_http_server(8000)

while True:
    cpu_usage = psutil.cpu_percent(interval=1)
    cpu_usage_gauge.set(cpu_usage)
    time.sleep(1)
```

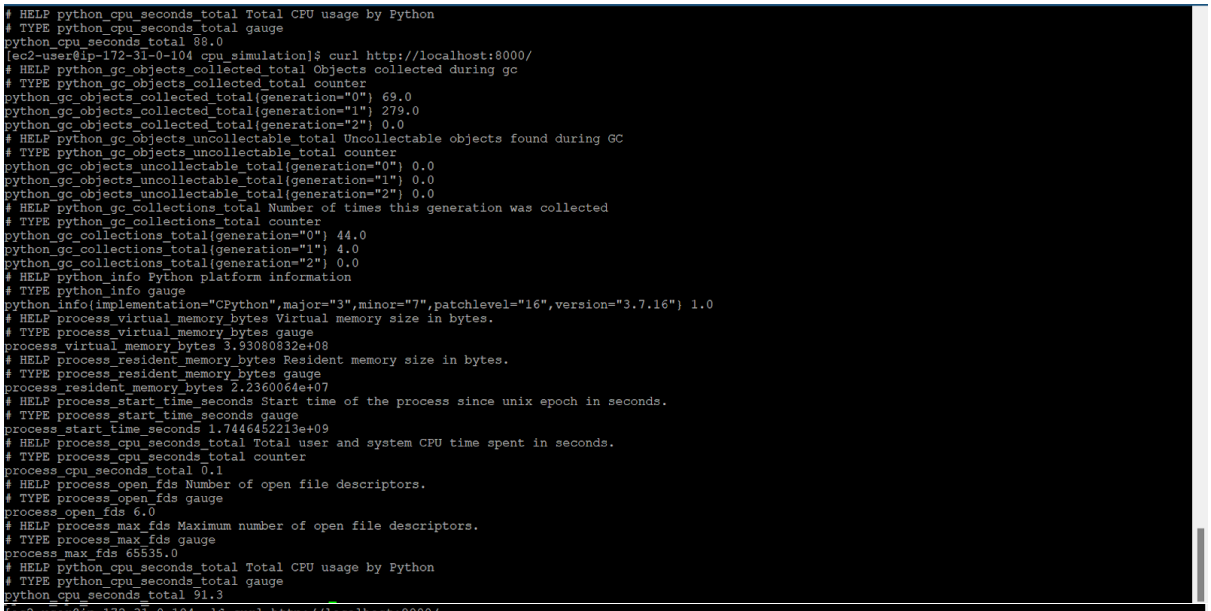
Run the script in background:

```
python3 cpu_load_simulator1.py &
```

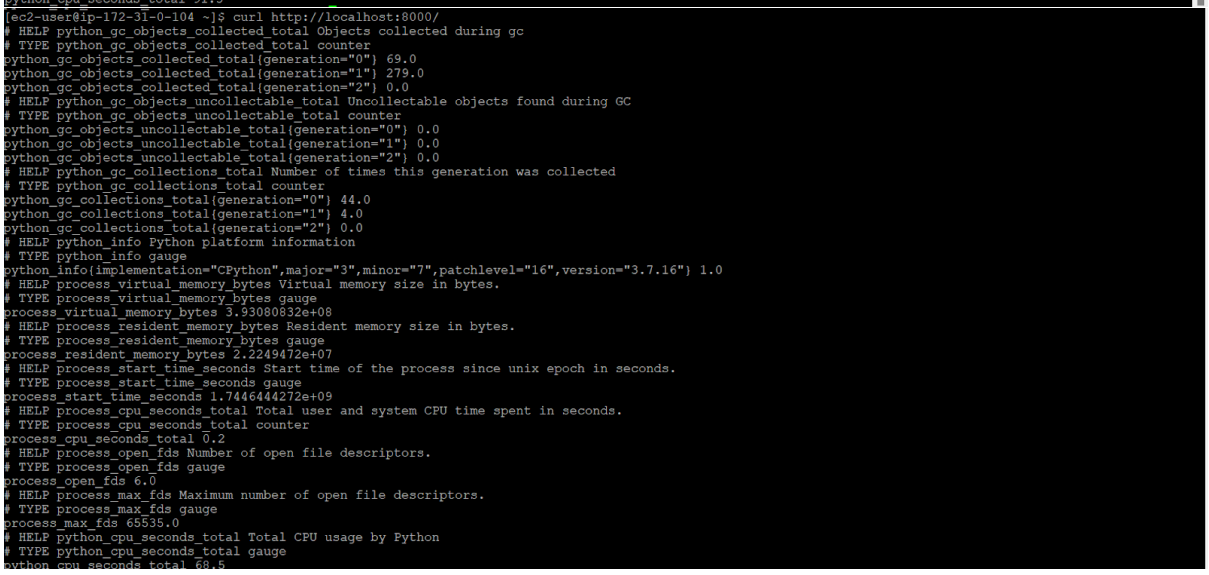
Test with curl:

```
curl http://localhost:8000/
```

OUTPUT SCREENSHOT:



```
# HELP python_cpu_seconds_total Total CPU usage by Python
# TYPE python_cpu_seconds_total gauge
python_cpu_seconds_total 88.0
[ec2-user@ip-172-31-0-104 cpu_simulation]$ curl http://localhost:8000/
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 69.0
python_gc_objects_collected_total{generation="1"} 279.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 44.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="7",patchlevel="16",version="3.7.16"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 3.93080832e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.2360064e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.7446452213e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.1
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 65535.0
# HELP python_cpu_seconds_total Total CPU usage by Python
# TYPE python_cpu_seconds_total gauge
python_cpu_seconds_total 91.3
```



```
[ec2-user@ip-172-31-0-104 ~]$ curl http://localhost:8000/
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 69.0
python_gc_objects_collected_total{generation="1"} 279.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 44.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="7",patchlevel="16",version="3.7.16"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 3.93080832e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.2249472e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.7446444272e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.2
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 65535.0
# HELP python_cpu_seconds_total Total CPU usage by Python
# TYPE python_cpu_seconds_total gauge
python_cpu_seconds_total 68.5
```

4. Stop Any Running Python CPU Scripts

Find and kill process:

```
ps aux | grep python
kill <PID>  # Replace <PID> with actual process ID
```

CPU LOAD SIMULATION – M2

Using `stress` to Simulate CPU Load on EC2

Install `stress`:

```
sudo yum install -y epel-release
sudo amazon-linux-extras install epel -y
sudo yum install stress -y
```

Run stress:

```
stress --cpu 2 --timeout 60
```

Use a second terminal to monitor with:

```
top
```

OUTPUT SCREENSHOT: