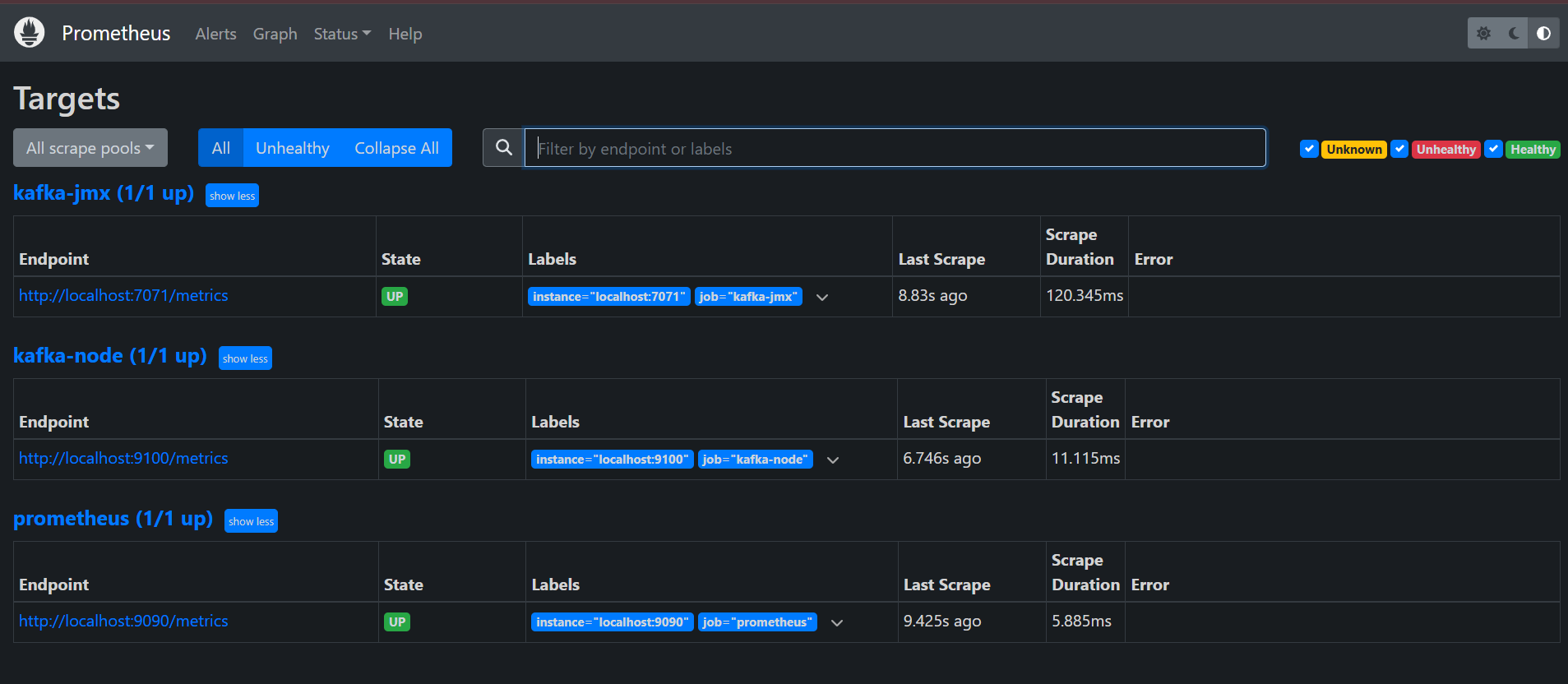
**Documentation: Data collection of CPU\_Usage when stressed**

Monitoring Setup:

Node Exporter: Installed on Kafka machine to expose system metrics at :9100/metrics.

JMX Exporter: Attached to Kafka broker to expose Kafka metrics at :7071/metrics.

Prometheus: Configured to scrape both exporters and store time-series metrics.



Producer Script (Run on Google Colab)

from kafka import KafkaProducer

import json

import time

import random

from datetime import datetime

# Kafka producer configuration

producer = KafkaProducer(

    bootstrap\_servers='100.27.223.212:9092',

    value\_serializer=lambda v: json.dumps(v).encode('utf-8'),

    linger\_ms=5,

    batch\_size=32768,

    compression\_type='gzip'

)

print("KafkaProducer initialized successfully!")

topic = "demo\_testing2"

try:

    while True:

        messages\_to\_send = random.randint(100, 200)

        for \_ in range(messages\_to\_send):

            data = {

                "sensor\_id": random.randint(1, 100),

                "temperature": round(random.uniform(15.0, 45.0), 2),

                "status": random.choice(["ok", "warn", "fail"]),

                "timestamp": datetime.now().isoformat()

            }

            producer.send(topic, value=data)

        producer.flush()

        print(f"Sent {messages\_to\_send} messages")

        # Tight loop with minimal delay

        time.sleep(1)

except KeyboardInterrupt:

    print("Producer stopped by user.")

finally:

    producer.flush()

    producer.close()

Consumer Script (Run on Google Colab)

from kafka import KafkaConsumer

import json

# Kafka consumer

consumer = KafkaConsumer(

    'demo\_testing2',  # Topic name

    bootstrap\_servers='100.27.223.212:9092',  # Kafka broker (EC2 public IP)

    auto\_offset\_reset='earliest',

    enable\_auto\_commit=True,

    value\_deserializer=lambda v: v.decode('utf-8')  # Decode as plain text first

)

print("Listening for messages on topic 'demo\_testing2'...")

for message in consumer:

    try:

        \_ = json.loads(message.value)  # Consume and parse JSON

        # No printing or processing, just consuming

    except json.JSONDecodeError:

        # Just consume non-JSON messages silently

        pass

Python Script to log metrics to CSV

import requests

import csv

import time

from datetime import datetime

JMX\_URL = "http://100.27.223.212:7071/metrics"

NODE\_URL = "http://100.27.223.212:9100/metrics"

CSV\_FILE = "stressed\_node\_metrics.csv"

JMX\_METRICS = {

    "bytes\_in": "kafka\_server\_BrokerTopicMetrics\_BytesInPerSec\_total",

    "bytes\_out": "kafka\_server\_BrokerTopicMetrics\_BytesOutPerSec\_total"

}

def parse\_prometheus\_text(text):

    metrics = {}

    for line in text.splitlines():

        if line.startswith("#") or line.strip() == "":

            continue

        parts = line.split()

        if len(parts) == 2:

            key, val = parts

            try:

                metrics[key] = float(val)

            except ValueError:

                continue

    return metrics

def get\_jmx\_metrics():

    res = requests.get(JMX\_URL)

    metrics = parse\_prometheus\_text(res.text)

    return {k: metrics.get(v, 0.0) for k, v in JMX\_METRICS.items()}

def get\_cpu\_times(metrics):

    cpu\_times = {"idle": 0.0, "total": 0.0}

    for k, v in metrics.items():

        if k.startswith("node\_cpu\_seconds\_total{") and "cpu=" in k:

            cpu\_times["total"] += v

            if 'mode="idle"' in k:

                cpu\_times["idle"] += v

    return cpu\_times

def write\_to\_csv(timestamp, jmx\_data, node\_data):

    fieldnames = ["timestamp", "bytes\_in", "bytes\_out", "cpu\_usage"]

    try:

        with open(CSV\_FILE, "r"):

            file\_exists = True

    except FileNotFoundError:

        file\_exists = False

    with open(CSV\_FILE, "a", newline="") as csvfile:

        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        if not file\_exists:

            writer.writeheader()

        row = {

            "timestamp": timestamp,

            \*\*jmx\_data,

            \*\*node\_data

        }

        writer.writerow(row)

def main():

    prev\_cpu = None

    while True:

        try:

            timestamp = datetime.now().strftime("%d-%m-%Y %H:%M:%S")

            # Fetch current metrics

            jmx\_data = get\_jmx\_metrics()

            node\_res = requests.get(NODE\_URL)

            node\_metrics = parse\_prometheus\_text(node\_res.text)

            cpu\_times = get\_cpu\_times(node\_metrics)

            # Compute CPU usage from delta

            if prev\_cpu:

                idle\_delta = cpu\_times["idle"] - prev\_cpu["idle"]

                total\_delta = cpu\_times["total"] - prev\_cpu["total"]

                if total\_delta > 0:

                    cpu\_usage = 100 \* (1 - idle\_delta / total\_delta)

                else:

                    cpu\_usage = 0.0

            else:

                cpu\_usage = 0.0  # First iteration, can't compute

            prev\_cpu = cpu\_times

            node\_data = {"cpu\_usage": round(cpu\_usage, 2)}

            write\_to\_csv(timestamp, jmx\_data, node\_data)

            print(f"[{timestamp}] Metrics recorded: bytes\_in={jmx\_data['bytes\_in']}, bytes\_out={jmx\_data['bytes\_out']}, cpu\_usage={node\_data['cpu\_usage']}")

        except Exception as e:

            print(f"Error: {e}")

        time.sleep(4)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Stressing the CPU Load:

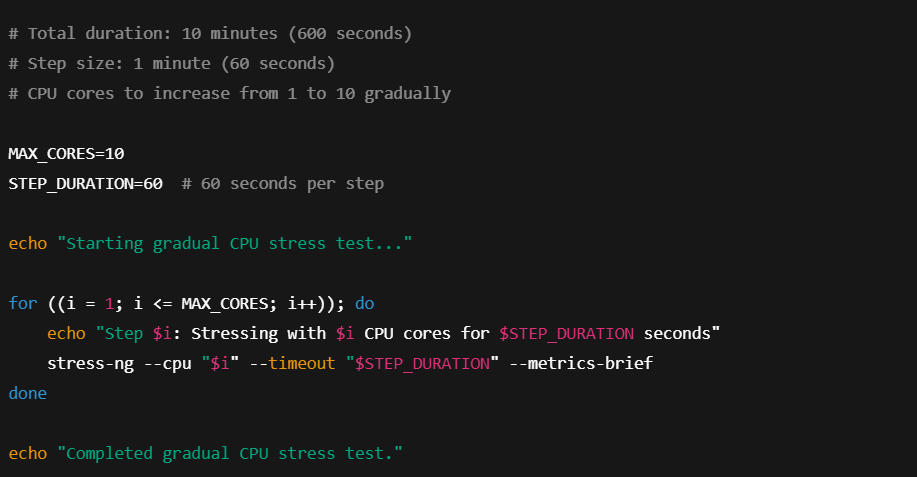
To **gradually increase CPU load from 0% to 100% over 10 minutes using** stress-ng first

1) Make sure stress-ng is installed:

sudo apt update

sudo apt install stress-ng

2) Save the script as gradual\_cpu\_load.sh



3) Make it executable:

chmod +x gradual\_cpu\_load.sh

4) Run it:

./gradual\_cpu\_load.sh