Comprehensive Guide: Kafka on Windows with Python (Producer/Consumer)

This guide will walk you through setting up Apache Kafka on a Windows machine and developing Python applications to send (produce) and receive (consume) data.

Target Audience: Developers using Windows with Python 3.8+ for local Kafka development.

1. System Prerequisites

Ensure these are installed on your Windows machine:

1. **Java Development Kit (JDK) 11 or higher:**
   - **Why:** Kafka is built on Java and Scala.
   - **Download:** Go to [Oracle JDK Downloads](). Choose JDK 11 or a later stable version for Windows (e.g., `x64 Installer`).
   - **Installation:** Run the downloaded `.exe` installer. Follow the prompts. It's recommended to add Java to your PATH environment variables during installation if prompted.
   - **Verification:**
     - Open **Command Prompt**.
     - Type: `java -version`
     - Expected Output: `openjdk version "11.0.XX" ...` or similar for your installed version.
2. **Python 3.8 or higher:**
   - **Why:** We'll use the `kafka-python` library for our applications.
   - **Download:** Go to [Python Downloads for Windows](). Download the latest 3.8.x installer (e.g., `Windows installer (64-bit)`).
   - **Installation:** Run the downloaded `.exe` installer. **Crucially, check the box "Add Python to PATH" during installation.** This makes `python` and `pip` commands accessible from the Command Prompt.
   - **Verification:**
     - Open **Command Prompt**.
     - Type: `python --version`
     - Expected Output: `Python 3.8.X`

2. Kafka Installation & Setup (Server-Side)

This sets up the Kafka broker and its dependency, ZooKeeper.

1. **Download Kafka:**
    - **Why:** This is the Kafka server software.
    - **Download:** Go to Apache Kafka Downloads.
    - Download the **latest stable binary release**. Look for a `.tgz` file, for example: `kafka_2.13-3.x.x.tgz` (where `3.x.x` is the version number, and `2.13` indicates the Scala version it was compiled with).
    - **Direct Link Example (may vary with version):** `https://downloads.apache.org/kafka/3.7.0/kafka_2.13-3.7.0.tgz`
2. **Extract Kafka:**
    - **Why:** Unpack the downloaded archive.
    - **Tool:** You'll need an archive tool that can handle `.tgz` (which is a gzipped tar archive). **7-Zip** (free and open-source) is highly recommended for Windows. Download from 7-Zip.org.
    - **Steps:**
        - Right-click the downloaded `kafka_2.13-3.x.x.tgz` file.
        - If using 7-Zip: `7-Zip` -> `Extract Here` (this will create a `.tar` file).
        - Right-click the newly extracted `.tar` file: `7-Zip` -> `Extract Here` (this will create the Kafka folder).
        - Rename the extracted folder (e.g., `kafka_2.13-3.x.x`) to simply `kafka` for easier navigation.
        - **Move this `kafka` folder to a convenient root directory, like `C:\`. So, your Kafka installation path will be `C:\kafka`.**
3. **Configure Kafka Data Directories:**
    - **Why:** Define where Kafka and ZooKeeper store their data (logs, metadata). This makes it easy to find and clear data for clean restarts.
    - **Navigate:** Open File Explorer and go to `C:\kafka\config`.
    - **Edit `zookeeper.properties`:**
        - Open `zookeeper.properties` with Notepad or a code editor (like VS Code).
        - Find the line `dataDir=/tmp/zookeeper`.
        - **Change it to:** `dataDir=C:/kafka/data/zookeeper` (Important: Use **forward slashes** `/` even on Windows).
        - Save the file.
    - **Edit `server.properties`:**
        - Open `server.properties` with Notepad or a code editor.
        - Find the line `log.dirs=/tmp/kafka-logs`.
        - **Change it to:** `log.dirs=C:/kafka/data/kafka-logs` (Important: Use **forward slashes** `/`).

Optional (but recommended for development): Add the following line at the end of the file to allow topic deletion from the command line:

Properties

```
delete.topic.enable = true
```

    -
    - Save the file.
  - **Create Data Folders:** Manually create these directories:
    - `C:\kafka\data\zookeeper`
    - `C:\kafka\data\kafka-logs`

3. Starting Kafka and ZooKeeper Servers

You will need **three separate Command Prompt windows** for this section.

1. **Start ZooKeeper Server:**
   - **Open a new Command Prompt.**
   - **Navigate to your Kafka directory:** `cd C:\kafka`

Run ZooKeeper:

Bash

```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

   -
   - **Wait:** You'll see many log messages. Look for a message indicating ZooKeeper has started successfully (it usually ends with something like `[NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181] - Accepted socket connection from ...`). Keep this window open.
2. **Start Kafka Server:**
   - **Open another new Command Prompt.**
   - **Navigate to your Kafka directory:** `cd C:\kafka`

Run Kafka:

Bash

```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

   -
   - **Wait:** Again, many logs. Look for a message like `[KafkaServer id=0] started (kafka.server.KafkaServer)` near the end. Keep this window open.
3. **Create a Kafka Topic:**
   - **Open a third new Command Prompt.**
   - **Navigate to your Kafka directory:** `cd C:\kafka`

Create a topic (e.g., `my_first_topic`):

Bash

```
.\bin\windows\kafka-topics.bat --create --topic my_first_topic --bootstrap-server localhost:9092
--partitions 1 --replication-factor 1
```

-
    - `--create`: Action to perform.
    - `--topic my_first_topic`: The name of your topic.
    - `--bootstrap-server localhost:9092`: Address of your Kafka broker.
    - `--partitions 1`: Number of partitions for the topic. For local testing, 1 is fine.
    - `--replication-factor 1`: How many copies of the data to keep. For single-broker setup, 1 is the only option.

Verify Topic Creation (Optional, but good practice):

Bash

```
.\bin\windows\kafka-topics.bat --list --bootstrap-server localhost:9092
```

- You should see `my_first_topic` (and possibly `__consumer_offsets` which is an internal Kafka topic) listed. Keep this window open or close it; it's not strictly needed for the next steps.

4. Python Setup for Kafka Client

1. **Install `kafka-python` Library:**
    - **Why:** This is the official Python client library to interact with Kafka.
    - **Open a new Command Prompt** (or use your existing one that's not running Kafka/ZooKeeper).

Install:

Bash

```
pip install kafka-python
```

-
- This command downloads and installs the necessary Python package.

5. Python Producer and Consumer Code

Create these two Python files in a directory of your choice (e.g., `C:\KafkaPythonApp`).

1. **Kafka Producer Code (`producer.py`):**

- ○ Create a file named `producer.py`.
- ○ Paste the following code into it:

Python

```
from kafka import KafkaProducer
import json
import time

# Initialize Kafka Producer
# 'bootstrap_servers': A list of Kafka broker addresses (host:port).
# 'value_serializer': A function to convert your message value into bytes before sending.
#                    Here, we convert Python dict to JSON string, then encode to UTF-8 bytes.
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

topic_name = 'my_first_topic'

print(f"--- Kafka Producer Started ---")
print(f"Sending messages to topic: '{topic_name}'")

try:
    # Send 10 messages
    for i in range(10):
        message = {"id": i, "timestamp": time.time(), "content": f"Hello from Python Producer! Message
{i}"}
        # The 'send' method is asynchronous. It returns a future.
        future = producer.send(topic_name, value=message)

        # You can optionally block until the message is sent (useful for debugging/small scripts)
        record_metadata = future.get(timeout=10) # Wait up to 10 seconds for acknowledgment

        print(f"Sent: {message} | "
              f"Topic: {record_metadata.topic}, "
              f"Partition: {record_metadata.partition}, "
              f"Offset: {record_metadata.offset}")

        time.sleep(1) # Wait for 1 second before sending the next message

except Exception as e:
    print(f"An error occurred during production: {e}")
```

```
finally:
    # Ensure all messages are sent before closing the producer
    producer.flush()
    print("\nProducer finished sending messages and flushed.")
    # It's good practice to close the producer when done
    producer.close()
    print("Producer closed.")

print(f"--- Kafka Producer Finished ---")
```

2.
3. **Kafka Consumer Code (`consumer.py`):**
   - ○ Create a file named `consumer.py`.
   - ○ Paste the following code into it:

```
Python
from kafka import KafkaConsumer
import json
import time

# Initialize Kafka Consumer
# 'my_first_topic': The topic(s) to subscribe to. Can be a list.
# 'bootstrap_servers': A list of Kafka broker addresses.
# 'auto_offset_reset': Where to start reading if no committed offset is found for the group:
#                     'earliest': Start from the beginning of the topic.
#                     'latest': Start from the most recent messages.
# 'enable_auto_commit': If True, consumer offsets are periodically committed in the background.
# 'group_id': A unique identifier for this consumer group. Messages are distributed among consumers in the same group.
# 'value_deserializer': A function to convert received bytes back into your original message format.
#                      Here, we decode UTF-8 bytes to string, then parse JSON string to Python dict.
consumer = KafkaConsumer(
    'my_first_topic',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest', # Start reading from the beginning if no offset committed
    enable_auto_commit=True,      # Automatically commit offsets
    group_id='my_python_consumer_group', # Unique group ID for your consumer
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

print(f"--- Kafka Consumer Started ---")
print(f"Consumer is listening for messages on topic: 'my_first_topic'")
print(f"Using consumer group: 'my_python_consumer_group'")

# Loop indefinitely to continuously consume messages
try:
    for message in consumer:
        # The 'message' object contains:
        # .topic (str): The topic the message came from
        # .partition (int): The partition within the topic
        # .offset (int): The offset of the message in the partition
        # .key (bytes or None): The message key (if any)
        # .value (decoded type, from deserializer): The message value
        # .timestamp (int): The timestamp of the message creation
        # .timestamp_type (int): Type of timestamp
        # .headers (list of tuples): Message headers (if any)

        print(f"Received message: "
              f"Topic={message.topic}, "
              f"Partition={message.partition}, "
              f"Offset={message.offset}, "
              f"Key={message.key.decode('utf-8') if message.key else 'None'}, "
              f"Value={message.value}")

except KeyboardInterrupt:
    print("\nConsumer stopped by user (Ctrl+C).")
except Exception as e:
    print(f"\nAn unexpected error occurred in consumer: {e}")
finally:
    # It's good practice to close the consumer when done or on interruption
    consumer.close()
    print("Consumer closed.")

print(f"--- Kafka Consumer Finished ---")
```

4.

6. Running Your Python Applications

You will need **two separate Command Prompt windows** for this section.

1. **Start the Consumer:**
    - **Open a new Command Prompt.**

    - **Navigate to the directory where you saved `consumer.py`** (e.g., `cd C:\KafkaPythonApp`).

Run the consumer:
Bash

```
python consumer.py
```

    -
    - You should see `--- Kafka Consumer Started ---` and `Consumer is listening for messages....`

2. **Start the Producer:**
    - **Open another new Command Prompt.**
    - **Navigate to the directory where you saved `producer.py`** (e.g., `cd C:\KafkaPythonApp`).

Run the producer:
Bash

```
python producer.py
```

    -
    - As the producer sends messages, you'll see `Sent: {message}...` in its window. Simultaneously, the consumer window will display `Received message: ...` for each message. This demonstrates the data flowing from producer to Kafka to consumer.

7. Troubleshooting & Common Issues

- **`kafka.errors.NoBrokersAvailable: NoBrokersAvailable`**:
    1. **Cause:** Your Python script couldn't connect to the Kafka broker.
    2. **Solution:**
        - **Ensure ZooKeeper is running** (its Command Prompt window is open and logging).
        - **Ensure Kafka server is running** (its Command Prompt window is open and logging).
        - **Check `localhost:9092`**: Make sure the `bootstrap_servers` in your Python code is `['localhost:9092']` and that Kafka is configured to listen on that port in `server.properties` (`listeners=PLAINTEXT://:9092`).
        - **Firewall:** Windows Firewall might be blocking port 9092 (Kafka) or 2181 (ZooKeeper). Temporarily disable it for testing, or add inbound/outbound rules to allow these ports.
        - **Clean Restart:** If all else fails, perform a full clean restart (see below).

- **Kafka Server Logs showing `failed log directory C:\kafka\data\kafka-logs`:**
    1. **Cause:** Kafka cannot write to its data directory.
    2. **Solution:** This is likely due to full disk space, incorrect permissions, or a corrupted/deleted directory.
- **Clean Restart Procedure (Essential for Troubleshooting):**
    1. **Stop ALL Command Prompt windows related to Kafka and ZooKeeper.** (Just close them).
    2. **Delete Data Directories:**
        - Go to `C:\kafka\data\` in File Explorer.
        - **Delete the `kafka-logs` folder.**
        - **Delete the `zookeeper` folder.**
    3. **Re-run Steps 3 and 4 of "Starting Kafka and ZooKeeper Servers":**
        - Start ZooKeeper.
        - Start Kafka.
        - Re-create your topics using `kafka-topics.bat --create ...`.
    4. **Then, run your Python producer and consumer.**

Connecting Kafka with Grafana

Grafana doesn't directly connect to Kafka. You need an intermediary that can read data from Kafka and expose it in a format Grafana understands (like Prometheus, InfluxDB, or a database). For real-time and time-series data, the most common and robust approach is to:

1. **Read data from Kafka:** Use a Kafka consumer (e.g., Python, Java, Go) to subscribe to your topic.
2. **Process/Transform data:** Extract the relevant metrics and timestamps.
3. **Store data in a Time-Series Database (TSDB):** Push the processed data into a TSDB like Prometheus, InfluxDB, or TimescaleDB (PostgreSQL extension).
4. **Visualize with Grafana:** Configure Grafana to query the TSDB and display the data.

We'll focus on a popular and relatively straightforward setup using:

- **Kafka:** Your data source.
- **Python:** To consume from Kafka and publish to a TSDB.
- **Prometheus Node Exporter (optional but useful):** To monitor your system.
- **Prometheus:** As the time-series database.
- **Grafana:** For visualization.

You're moving into a powerful realm of data visualization! Visualizing Kafka data in Grafana for time-series and real-time insights is a fantastic use case.

Here's a comprehensive guide on how to achieve this.

The Core Idea:

Grafana doesn't directly connect to Kafka. You need an intermediary that can read data from Kafka and expose it in a format Grafana understands (like Prometheus, InfluxDB, or a database). For real-time and time-series data, the most common and robust approach is to:

1. **Read data from Kafka:** Use a Kafka consumer (e.g., Python, Java, Go) to subscribe to your topic.
2. **Process/Transform data:** Extract the relevant metrics and timestamps.
3. **Store data in a Time-Series Database (TSDB):** Push the processed data into a TSDB like Prometheus, InfluxDB, or TimescaleDB (PostgreSQL extension).
4. **Visualize with Grafana:** Configure Grafana to query the TSDB and display the data.

We'll focus on a popular and relatively straightforward setup using:

- **Kafka:** Your data source.
- **Python:** To consume from Kafka and publish to a TSDB.
- **Prometheus Node Exporter (optional but useful):** To monitor your system.
- **Prometheus:** As the time-series database.
- **Grafana:** For visualization.

Visualizing Kafka Data in Grafana (Time-Series & Real-Time)

Scenario: You have a Kafka topic (`my_time_series_data_topic`) with JSON messages containing a `timestamp` and a `value` (e.g., sensor readings, stock prices). You want to see this data as a real-time graph in Grafana.

Required Installations (beyond what you already have):

1. **Prometheus:** A monitoring system and time-series database.
2. **Grafana:** The open-source analytics and monitoring solution.
3. **(Optional but good practice) Prometheus Node Exporter:** To monitor your Windows machine's metrics (CPU, memory, disk). This helps confirm Prometheus is working.

Estimated Time: 1-2 hours for first-time setup.

Step 1: Install Prometheus

Prometheus is a powerful time-series database. Grafana has native support for it.

1. **Download Prometheus for Windows:**
    - Go to the Prometheus Downloads page.

    - Find the latest stable version for "Windows" (e.g., `prometheus-x.xx.x.windows-amd64.zip`).
2. **Extract Prometheus:**
    - Unzip the downloaded file.
    - Move the extracted folder (e.g., `prometheus-x.xx.x.windows-amd64`) to a convenient location, like `C:\Prometheus`.
3. **Configure Prometheus (`prometheus.yml`):**
    - Go to `C:\Prometheus`.
    - Open `prometheus.yml` in a text editor.
    - Under the `scrape_configs:` section, you'll see a default `job_name: 'prometheus'`. Keep this.
    - **Add a new job for your custom Kafka exporter (which we'll create in Python):**

You're moving into a powerful realm of data visualization! Visualizing Kafka data in Grafana for time-series and real-time insights is a fantastic use case.

Here's a comprehensive guide on how to achieve this.

The Core Idea:

Grafana doesn't directly connect to Kafka. You need an intermediary that can read data from Kafka and expose it in a format Grafana understands (like Prometheus, InfluxDB, or a database). For real-time and time-series data, the most common and robust approach is to:

1. **Read data from Kafka:** Use a Kafka consumer (e.g., Python, Java, Go) to subscribe to your topic.
2. **Process/Transform data:** Extract the relevant metrics and timestamps.
3. **Store data in a Time-Series Database (TSDB):** Push the processed data into a TSDB like Prometheus, InfluxDB, or TimescaleDB (PostgreSQL extension).
4. **Visualize with Grafana:** Configure Grafana to query the TSDB and display the data.

We'll focus on a popular and relatively straightforward setup using:

- **Kafka:** Your data source.
- **Python:** To consume from Kafka and publish to a TSDB.
- **Prometheus Node Exporter (optional but useful):** To monitor your system.
- **Prometheus:** As the time-series database.
- **Grafana:** For visualization.

Visualizing Kafka Data in Grafana (Time-Series & Real-Time)

Scenario: You have a Kafka topic (`my_time_series_data_topic`) with JSON messages containing a `timestamp` and a `value` (e.g., sensor readings, stock prices). You want to see this data as a real-time graph in Grafana.

Required Installations (beyond what you already have):

1. **Prometheus:** A monitoring system and time-series database.
2. **Grafana:** The open-source analytics and monitoring solution.
3. **(Optional but good practice) Prometheus Node Exporter:** To monitor your Windows machine's metrics (CPU, memory, disk). This helps confirm Prometheus is working.

Estimated Time: 1-2 hours for first-time setup.

Step 1: Install Prometheus

Prometheus is a powerful time-series database. Grafana has native support for it.

1. **Download Prometheus for Windows:**
    - Go to the Prometheus Downloads page.
    - Find the latest stable version for "Windows" (e.g., `prometheus-x.xx.x.windows-amd64.zip`).
2. **Extract Prometheus:**
    - Unzip the downloaded file.
    - Move the extracted folder (e.g., `prometheus-x.xx.x.windows-amd64`) to a convenient location, like `C:\Prometheus`.
3. **Configure Prometheus (`prometheus.yml`):**
    - Go to `C:\Prometheus`.
    - Open `prometheus.yml` in a text editor.
    - Under the `scrape_configs:` section, you'll see a default `job_name: 'prometheus'`. Keep this.
    - **Add a new job for your custom Kafka exporter (which we'll create in Python):**

prometheus.yml

```
# Global configuration settings.
global:
  scrape_interval: 15s # How frequently Prometheus scrapes targets (e.g., every 15 seconds).
  evaluation_interval: 15s # How frequently Prometheus evaluates alerting rules.

# Alerting configuration (optional, for setting up alerts)
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093 # Uncomment and configure if you set up Alertmanager later

# Rule files for recording rules or alerting rules (optional)
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# Scrape configurations: Define what Prometheus should monitor.
scrape_configs:
  # 1. Prometheus itself (already exists)
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]
        labels:
          app: "prometheus"

  # 2. Prometheus Node Exporter
  # This scrapes metrics from your Windows machine (CPU, memory, disk, network).
  # Assuming Node Exporter runs on its default port 9100.
  - job_name: "node_exporter"
    static_configs:
      - targets: ["localhost:9100"]
        labels:
          app: "node_exporter" # Add a descriptive label

  # 3. Your Custom Python Kafka Metrics Exporter
  # This scrapes the metrics you expose from your Python script.
  # Assuming your Python script runs on port 9000 (as in previous example).
  - job_name: "kafka_app_metrics" # A clear name for your application-specific Kafka metrics
    static_configs:
      - targets: ["localhost:9000"]
        labels:
          app: "kafka_consumer_metrics" # Label for these specific metrics

  # 4. (Optional) Kafka JMX Exporter
  # If you decide to set up JMX Exporter for Kafka broker internal metrics.
  # Assuming JMX Exporter runs on port 7071 (as suggested in previous explanation).
  - job_name: "kafka_broker_jmx" # Name for Kafka broker's internal JVM/JMX metrics
    static_configs:
```

```
    - targets: ["localhost:7071"]
      labels:
        app: "kafka_broker" # Label for the Kafka broker

  # 5. (Optional) Kafka Exporter (danielqsj/kafka-exporter)
  # If you decide to set up the dedicated Kafka Exporter for consumer lag and cluster health.
  # Assuming Kafka Exporter runs on port 9308 (its default).
  - job_name: "kafka_cluster_exporter" # Name for Kafka cluster health and consumer lag metrics
    static_configs:
    - targets: ["localhost:9308"]
      labels:
        app: "kafka_cluster" # Label for the Kafka cluster metrics
```

1. **Start Prometheus:**
   - Open a **Command Prompt**.
   - Navigate to your Prometheus directory: `cd C:\Prometheus`

Run Prometheus:
Bash

```
prometheus.exe --config.file=prometheus.yml
```

   -
   - You should see logs indicating it's starting.
   - **Verify:** Open your web browser and go to `http://localhost:9090`. You should see the Prometheus UI. Go to "Status" -> "Targets" to see if the `prometheus` job is "UP".

You're moving into a powerful realm of data visualization! Visualizing Kafka data in Grafana for time-series and real-time insights is a fantastic use case.

Here's a comprehensive guide on how to achieve this.

The Core Idea:

Grafana doesn't directly connect to Kafka. You need an intermediary that can read data from Kafka and expose it in a format Grafana understands (like Prometheus, InfluxDB, or a database). For real-time and time-series data, the most common and robust approach is to:

1. **Read data from Kafka:** Use a Kafka consumer (e.g., Python, Java, Go) to subscribe to your topic.
2. **Process/Transform data:** Extract the relevant metrics and timestamps.
3. **Store data in a Time-Series Database (TSDB):** Push the processed data into a TSDB like Prometheus, InfluxDB, or TimescaleDB (PostgreSQL extension).

4. **Visualize with Grafana:** Configure Grafana to query the TSDB and display the data.

We'll focus on a popular and relatively straightforward setup using:

- **Kafka:** Your data source.
- **Python:** To consume from Kafka and publish to a TSDB.
- **Prometheus Node Exporter (optional but useful):** To monitor your system.
- **Prometheus:** As the time-series database.
- **Grafana:** For visualization.

Visualizing Kafka Data in Grafana (Time-Series & Real-Time)

Scenario: You have a Kafka topic (`my_time_series_data_topic`) with JSON messages containing a `timestamp` and a `value` (e.g., sensor readings, stock prices). You want to see this data as a real-time graph in Grafana.

Required Installations (beyond what you already have):

1. **Prometheus:** A monitoring system and time-series database.
2. **Grafana:** The open-source analytics and monitoring solution.
3. **(Optional but good practice) Prometheus Node Exporter:** To monitor your Windows machine's metrics (CPU, memory, disk). This helps confirm Prometheus is working.

Estimated Time: 1-2 hours for first-time setup.

Step 1: Install Prometheus

Prometheus is a powerful time-series database. Grafana has native support for it.

1. **Download Prometheus for Windows:**
   - Go to the [Prometheus Downloads page](#).
   - Find the latest stable version for "Windows" (e.g., `prometheus-x.xx.x.windows-amd64.zip`).
2. **Extract Prometheus:**
   - Unzip the downloaded file.
   - Move the extracted folder (e.g., `prometheus-x.xx.x.windows-amd64`) to a convenient location, like `C:\Prometheus`.
3. **Configure Prometheus (`prometheus.yml`):**
   - Go to `C:\Prometheus`.

- Open `prometheus.yml` in a text editor.
- Under the `scrape_configs:` section, you'll see a default `job_name: 'prometheus'`. Keep this.
- **Add a new job for your custom Kafka exporter (which we'll create in Python):**

YAML

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds.

# A scrape configuration for the node exporter (if installed)
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090'] # Prometheus itself is exposing metrics on 9090

  # Add this block for your custom Python metrics exporter
  - job_name: 'kafka_exporter'
    static_configs:
      - targets: ['localhost:9000'] # This is where our Python exporter will expose metrics
    # Optional: Relabeling if your exporter serves multiple metrics
    # metrics_path: /metrics # Default path, can be explicit if needed
```

4.
   - Save the `prometheus.yml` file.
5. **Start Prometheus:**
   - Open a **Command Prompt**.
   - Navigate to your Prometheus directory: `cd C:\Prometheus`

Run Prometheus:

Bash

```
prometheus.exe --config.file=prometheus.yml
```

-
- You should see logs indicating it's starting.
- **Verify:** Open your web browser and go to `http://localhost:9090`. You should see the Prometheus UI. Go to "Status" -> "Targets" to see if the `prometheus` job is "UP".

Step 2: Install Grafana

1. **Download Grafana for Windows:**
   - Go to the [Grafana Downloads page](#).
   - Choose the "Windows installer" (usually `.msi`).
2. **Install Grafana:**
   - Run the downloaded `.msi` installer. Follow the prompts. It will install Grafana as a Windows service.
3. **Start Grafana Service:**
   - The installer typically starts the service automatically.
   - **Verify:** Open your web browser and go to `http://localhost:3000`.
   - **Login:** Default credentials are `admin` / `admin`. You'll be prompted to change the password.
4. **Add Prometheus Data Source to Grafana:**
   - After logging into Grafana, click on the **gear icon (Configuration)** on the left sidebar.
   - Click **"Data sources"**.
   - Click **"Add data source"**.
   - Select **"Prometheus"**.
   - **Name:** `Prometheus` (or whatever you prefer)
   - **URL:** `http://localhost:9090`
   - Click **"Save & test"**. You should see "Data source is working" if Prometheus is running.

Excellent! It's a great idea to consolidate all the successful steps. Here's a complete, step-by-step guide for visualizing Kafka data on Grafana using Prometheus on Windows, without Docker, with all the commands, file setups, and common pitfalls addressed.

Objective: Visualize time-series and real-time Kafka operational metrics on Grafana using Prometheus on a Windows machine (Python 3.8+).

Key Components:

- **Apache Kafka:** The distributed streaming platform.
- **Apache ZooKeeper:** Kafka's dependency for cluster coordination.
- **Prometheus JMX Exporter:** A Java agent that exposes Kafka's JMX metrics in a Prometheus-readable format.
- **Prometheus:** A monitoring system that scrapes and stores time-series data from exporters.
- **Grafana:** A visualization and dashboarding tool that queries data from Prometheus.
- **Python (Optional):** For custom real-time data producers.

Prerequisites:

- **Java Development Kit (JDK):** Version 8 or higher installed and `JAVA_HOME` environment variable set.
- **Python:** Version 3.8 or higher installed.
- **Administrative Privileges:** Required for certain installations and running services.

Step-by-Step Guide

1. Install and Set Up Kafka & ZooKeeper

Kafka requires ZooKeeper. Both are typically run from the same Kafka binary download.

a. Download Kafka:

- Go to the Apache Kafka downloads page (search "Apache Kafka download").
- Download the latest stable release (choose the **binary download**, e.g., `kafka_2.13-3.x.x.tgz` or `kafka_3.x.x.zip`).

b. Extract Kafka:

- Create a directory, e.g., `C:\Kafka`.
- Extract the downloaded archive (e.g., `kafka_3.x.x.zip`) directly into `C:\Kafka`.
    - This should result in a structure like `C:\Kafka\bin`, `C:\Kafka\config`, `C:\Kafka\libs`, etc.

c. Configure ZooKeeper:

- Navigate to `C:\Kafka\config`.
- Open `zookeeper.properties` in a text editor (e.g., Notepad++).

Ensure `dataDir` points to a valid directory:

Properties

```
dataDir=C:/Kafka/zookeeper-data
```

-
    - **Create this directory if it doesn't exist:** `C:\Kafka\zookeeper-data`.
- Save the file.

d. Configure Kafka Broker:

- Navigate to `C:\Kafka\config`.
- Open `server.properties` in a text editor.

Ensure `log.dirs` points to a valid directory:

Properties

```
log.dirs=C:/Kafka/kafka-logs
```

-
    - **Create this directory if it doesn't exist:** `C:\Kafka\kafka-logs`.

Important: Set `advertised.listeners` if you face connectivity issues from external machines (though `localhost` is fine for local setup):

Properties

```
advertised.listeners=PLAINTEXT://localhost:9092
```

-
- Save the file.

e. Start ZooKeeper:

- Open a **new Command Prompt as Administrator**.
- Navigate to your Kafka directory: `cd C:\Kafka`

Execute the ZooKeeper startup script:

Bash

```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

-
- **Keep this Command Prompt window open and running.**

2. Integrate Prometheus JMX Exporter with Kafka

This allows Prometheus to scrape metrics from Kafka.

a. Download JMX Exporter:

- Go to the Prometheus JMX Exporter GitHub releases page (search "Prometheus JMX Exporter GitHub").
- Download the latest `jmx_prometheus_javaagent-<version>.jar` file.
- Place this JAR file directly into your `C:\Kafka` directory (e.g., `C:\Kafka\jmx_prometheus_javaagent.jar`).

b. Create JMX Exporter Configuration File:

- Navigate to `C:\Kafka\config`.

- Create a new text file named `kafka-jmx-exporter.yml` (ensure the `.yml` extension, not `.txt`).

Paste the following basic configuration into it:

YAML

```
---
rules:
- pattern : "kafka.server<type=(.+), name=(.+), clientId=(.+), topic=(.+), partition=(.*)><>Value"
  name: "kafka_server_$1_$2"
  labels:
    clientId: "$3"
    topic: "$4"
    partition: "$5"
- pattern : "kafka.network<type=(.+), name=(.+), request=(.+), .*><>Value"
  name: "kafka_network_$1_$2_request"
  labels:
    request: "$3"
- pattern : "kafka.controller<type=(.+), name=(.+), .*><>Value"
  name: "kafka_controller_$1_$2"
- pattern : "kafka.log<type=(.+), name=(.+), topic=(.+), partition=(.*)><>Value"
  name: "kafka_log_$1_$2"
  labels:
    topic: "$3"
    partition: "$4"
- pattern : "kafka.producer<type=(.+), client-id=(.+), topic=(.+), .*><>Value"
  name: "kafka_producer_$1"
  labels:
    clientId: "$2"
    topic: "$3"
- pattern : "kafka.consumer<type=(.+), client-id=(.+), topic=(.+), .*><>Value"
  name: "kafka_consumer_$1"
  labels:
    clientId: "$2"
    topic: "$3"
```

-
- Save the file.

c. Modify Kafka Startup Script (`kafka-server-start.bat`):

- Navigate to `C:\Kafka\bin\windows`.
- Open `kafka-server-start.bat` in a text editor.

- **Locate the section where `KAFKA_HEAP_OPTS` is set.**

Add the `KAFKA_OPTS` environment variable definition *before* the line `"%~dp0kafka-run-class.bat" kafka.Kafka %*`.
Important: Ensure this is the *only* place `KAFKA_OPTS` is set with the `-javaagent` argument in this file.
Code snippet

```
rem Add default JVM options here. See kafka-server-start.sh for more details.
rem For example: set KAFKA_JVM_PERFORMANCE_OPTS="-server -XX:+UseG1GC
-XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35
-XX:+ExplicitGCInvokesConcurrent -XX:MaxInlineLevel=15"
rem
IF ["%KAFKA_LOG4J_OPTS%"] EQU [""] (
  set KAFKA_LOG4J_OPTS=-Dlog4j.configuration=file:%~dp0../../config/log4j.properties
)
IF ["%KAFKA_HEAP_OPTS%"] EQU [""] (
  rem detect OS architecture
  wmic os get osarchitecture | find /i "32-bit" >nul 2>&1
  IF NOT ERRORLEVEL 1 (
    rem 32-bit OS
    set KAFKA_HEAP_OPTS=-Xmx512M -Xms512M
  ) ELSE (
    rem 64-bit OS
    set KAFKA_HEAP_OPTS=-Xmx1G -Xms1G
  )
)

REM ************************************************************************
REM ** JMX EXPORTER INTEGRATION **
REM ************************************************************************

rem Set the KAFKA_OPTS environment variable which is picked up by kafka-run-class.bat
rem This will apply the JMX Exporter agent to the Kafka broker's JVM.
rem IMPORTANT: Ensure the path to the jmx_prometheus_javaagent.jar and its config are correct.
rem JMX Exporter will listen on port 7072 (this port is defined here and used by Prometheus).

set
KAFKA_OPTS=-javaagent:"C:\Kafka\jmx_prometheus_javaagent.jar"=7072:"C:\Kafka\config\kafka-jmx
-exporter.yml"

REM ************************************************************************
REM ** END OF JMX EXPORTER INTEGRATION **
REM **********************************************************************

"%~dp0kafka-run-class.bat" kafka.Kafka %*

EndLocal
```

-
- **Verify Paths:** Double-check `C:\Kafka\jmx_prometheus_javaagent.jar` and `C:\Kafka\config\kafka-jmx-exporter.yml` in the `set KAFKA_OPTS` line match your actual file locations and names.
- Save the `kafka-server-start.bat` file.

d. Start Kafka Broker:

- Open a **new Command Prompt as Administrator**.
- Navigate to your Kafka directory: `cd C:\Kafka`

Execute the Kafka startup script:
Bash

```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

-
- **Keep this Command Prompt window open and running.**
- **Verify JMX Exporter:** Open your web browser and go to `http://localhost:7072/metrics`. You should see a page full of Kafka metrics in Prometheus format. If not, check the console output of Kafka for errors.

3. Install and Configure Prometheus

Prometheus will pull metrics from the JMX Exporter.

a. Download Prometheus:

- Go to the Prometheus downloads page (search "Prometheus download").
- Download the latest stable release for Windows (e.g., `prometheus-<version>.windows-amd64.zip`).

b. Extract Prometheus:

- Create a directory, e.g., `C:\Prometheus`.
- Extract the downloaded ZIP file into `C:\Prometheus`.
  - This should result in `C:\Prometheus\prometheus.exe`, `C:\Prometheus\prometheus.yml`, etc.

c. Configure Prometheus:

- Navigate to `C:\Prometheus`.
- Open `prometheus.yml` in a text editor.

Add a scrape configuration for your Kafka JMX Exporter under the `scrape_configs` section.
Ensure the `targets` port matches the port you configured for the JMX Exporter (e.g., `7072`).
YAML

```
global:
  scrape_interval: 15s # How frequently to scrape targets
  evaluation_interval: 15s # How frequently to evaluate rules

alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - localhost:9093

rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

scrape_configs:
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"] # Prometheus's own metrics

  - job_name: 'kafka'
    static_configs:
      - targets: ['localhost:7072'] # Kafka JMX Exporter running on port 7072
        labels:
          instance: 'kafka-broker-1' # Optional: Add a label for easier identification
```

-
- Save the file.

d. Start Prometheus:

- Open a **new Command Prompt as Administrator**.
- Navigate to your Prometheus directory: `cd C:\Prometheus`

Execute Prometheus:
Bash

```
prometheus.exe --config.file=prometheus.yml
```

-
- **Keep this Command Prompt window open and running.**
- **Verify Prometheus:** Open your web browser and go to `http://localhost:9090`. You should see the Prometheus UI.
  - Go to **"Status" -> "Targets"**. Confirm that `kafka` (localhost:7072) is listed and has a "State" of "UP".

4. Install and Configure Grafana

Grafana will connect to Prometheus to visualize the data.

a. Download Grafana:

- Go to the Grafana downloads page (search "Grafana download").
- Download the **Windows installer (.exe)**.

b. Install Grafana:

- Run the downloaded installer. Follow the on-screen prompts (typically "Next" -> "Install" -> "Finish").
- Grafana is usually installed as a Windows service and starts automatically.

c. Access Grafana:

- Open your web browser and navigate to `http://localhost:3000`.
- **Default login:**
  - Username: `admin`
  - Password: `admin`
- You will be prompted to change the password on your first login.

d. Add Prometheus Data Source to Grafana:

- In the Grafana UI, click the **Gear icon (Configuration)** on the left sidebar.
- Click **Data sources**.
- Click **Add data source**.
- Select **Prometheus**.
- In the **HTTP** section, set the **URL** to `http://localhost:9090`.
- Click **Save & Test**. You should see a green "Data source is working" message.

5. Visualize Kafka Data in Grafana Dashboards

a. Import Pre-built Kafka Dashboards (Recommended for quick setup):

- Go to the Grafana Dashboards page: `https://grafana.com/grafana/dashboards/`
- Search for "Kafka Prometheus JMX" or "Kafka Overview".
    - Popular dashboard IDs include: `7589` (Kafka Overview), `11962` (Kafka Metrics).
- In Grafana:
    - Click the **"+" icon (Create)** on the left sidebar, then **Import**.
    - Enter the Dashboard ID (e.g., `11962`) in the "Import via grafana.com" field, or paste the JSON model from the dashboard's page.
    - Click **Load**.
    - On the next screen, select your **Prometheus** data source from the dropdown.
    - Click **Import**.
- This will create a pre-configured dashboard with many useful Kafka metrics.

b. Create Custom Dashboards (for specific metrics or real-time views):

- In Grafana, click the **"+" icon (Create)** on the left sidebar, then **Dashboard**.
- Click **Add new panel**.
- In the **Query** tab:
    - Ensure your **Prometheus** data source is selected.
    - Use the **Metric browser** to explore available metrics (start typing `kafka_`).
    - **Write PromQL queries** to retrieve the data you want.

Time Series Example (Messages In/Out per second):

Code snippet

```
rate(kafka_server_brokertopicmetrics_messagesinpersec[5m])
```

- (Shows the average messages per second over the last 5 minutes.)

Real-time Example (Current Consumer Lag):

Code snippet

```
kafka_consumergroup_group_lag_sum{consumergroup="your_consumer_group_name",
topic="your_topic_name"}
```

- (Shows the current sum of lag for a specific consumer group and topic.)

- **Visualization:**
    - Choose **"Graph (Time series)"** for historical trends.
    - Choose **"Stat"**, **"Gauge"**, or **"Bar Gauge"** for current, real-time values.
- **Panel Settings:** Configure title, units, axes, legends, and refresh rates as needed.
- **Dashboard Refresh Rate:** Set the dashboard's refresh rate (top right corner, e.g., `5s`, `10s`) to get a "real-time" view.

6. Optional: Python Producer for Custom Real-time Data

If you have custom application data you want to push to Kafka and visualize.

a. Install Kafka Python Client:

Open Command Prompt:
Bash

```
pip install kafka-python
```

- (Alternatively, `pip install confluent-kafka` for a more robust client, but `kafka-python` is simpler for basic examples).

b. Example Python Producer (`producer.py`):

Create a file `producer.py` and add:
Python

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

BOOTSTRAP_SERVERS = 'localhost:9092'
TOPIC_NAME = 'my_custom_data_topic' # Create this topic in Kafka if it doesn't exist

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

print(f"Producing messages to topic: {TOPIC_NAME}")

try:
    for i in range(1, 21): # Produce 20 messages for demo
        data = {
            'timestamp': int(time.time() * 1000), # Unix timestamp in milliseconds
            'sensor_id': f'sensor_{random.randint(1, 3)}',
            'temperature': round(random.uniform(20.0, 30.0), 2),
            'humidity': round(random.uniform(50.0, 70.0), 2),
            'reading_id': i
        }
        producer.send(TOPIC_NAME, value=data)
        print(f"Sent: {data}")
        time.sleep(1) # Send a message every second
except Exception as e:
    print(f"Error producing message: {e}")
finally:
    producer.flush()
    producer.close()
    print("Producer finished.")
```

-

c. Run the Python Producer:

- Open Command Prompt: `python producer.py`

d. Visualize Custom Data (requires custom Prometheus Exporter):

- **To get this custom data into Prometheus/Grafana, you'll need to create a *separate* Python application that acts as a Prometheus Exporter.**
- This exporter would:
  - Consume messages from your `my_custom_data_topic` Kafka topic.
  - Parse the JSON data.
  - Expose the data as Prometheus metrics (e.g., using `prometheus_client` library).
  - Run a simple HTTP server that Prometheus can scrape.

Example Python Custom Exporter (`custom_metrics_exporter.py`):

Python

```
from prometheus_client import start_http_server, Gauge
import json
import time
from kafka import KafkaConsumer
import threading

# Prometheus metrics definition
TEMPERATURE_GAUGE = Gauge('sensor_temperature_celsius', 'Current temperature in Celsius',
['sensor_id'])
HUMIDITY_GAUGE = Gauge('sensor_humidity_percent', 'Current humidity percentage',
['sensor_id'])

# Kafka consumer configuration
BOOTSTRAP_SERVERS = 'localhost:9092'
TOPIC_NAME = 'my_custom_data_topic'

def consume_kafka_and_expose_metrics():
    consumer = KafkaConsumer(
        TOPIC_NAME,
        bootstrap_servers=BOOTSTRAP_SERVERS,
        auto_offset_reset='latest', # Start consuming from the latest message
        enable_auto_commit=True,
        group_id='custom_metrics_group',
        value_deserializer=lambda x: json.loads(x.decode('utf-8'))
    )
    print(f"Custom metrics exporter: Consumer started for topic: {TOPIC_NAME}")
    try:
        for message in consumer:
            data = message.value
            sensor_id = data.get('sensor_id')
            temperature = data.get('temperature')
            humidity = data.get('humidity')

            if sensor_id and temperature is not None:
                TEMPERATURE_GAUGE.labels(sensor_id=sensor_id).set(temperature)
            if sensor_id and humidity is not None:
                HUMIDITY_GAUGE.labels(sensor_id=sensor_id).set(humidity)
            print(f"Custom metrics exporter: Processed message: {data}")
    except Exception as e:
        print(f"Custom metrics exporter: Error in consumer: {e}")
    finally:
        consumer.close()

if __name__ == '__main__':
    # Start the Prometheus HTTP server on a chosen port (e.g., 8000)
    exporter_port = 8000
    start_http_server(exporter_port)
    print(f"Prometheus custom exporter serving metrics on port {exporter_port}")

    # Start the Kafka consumer in a separate thread
    consumer_thread = threading.Thread(target=consume_kafka_and_expose_metrics)
    consumer_thread.daemon = True # Allows main program to exit if thread is still running
    consumer_thread.start()

    # Keep the main thread alive indefinitely to keep the HTTP server running
    while True:
        time.sleep(1)
```

-

- **Run the Custom Exporter:**
  - Install `prometheus_client`: `pip install prometheus_client`
  - Run: `python custom_metrics_exporter.py`
- **Configure Prometheus to Scrape Custom Exporter:**

Edit `C:\Prometheus\prometheus.yml` and add a new `scrape_config`:

YAML

```
scrape_configs:
  # ... (your existing kafka and prometheus jobs) ...

  - job_name: 'my_custom_app_metrics'
    static_configs:
      - targets: ['localhost:8000'] # Port where your Python exporter is running
```

  - Save `prometheus.yml` and **restart Prometheus**.
- Now, in Grafana, you can create new panels using metrics like `sensor_temperature_celsius` and `sensor_humidity_percent`.

Troubleshooting Checklist:

1. **Check Console Logs:** Always review the command prompt windows for Kafka, ZooKeeper, JMX Exporter, Prometheus, and any custom scripts. Error messages are usually descriptive.
2. **Port Conflicts:** Ensure no other applications are using:
   - `2181` (ZooKeeper)
   - `9092` (Kafka)
   - `7072` (JMX Exporter - verify this port in your `kafka-server-start.bat` and Prometheus config)
   - `9090` (Prometheus)
   - `3000` (Grafana)
   - `8000` (Custom Python Exporter, if used)
   - Use `netstat -ano | findstr :<port_number>` in Command Prompt to check.
3. **File Paths and Names:** Double-check all paths and file names in your `.bat` files and YAML configurations. Typos are common.
4. **Prometheus Targets:** Access `http://localhost:9090/targets` in your browser. All your configured jobs (`kafka`, `prometheus`, `my_custom_app_metrics`) should show "UP".
5. **Grafana Data Source Test:** In Grafana, always click "Save & Test" for your Prometheus data source to ensure connectivity.

6. **Full Restarts:** After any configuration changes, especially to `.bat` files or `prometheus.yml`, ensure you fully stop and restart the relevant components. For Kafka, always stop ZooKeeper and Kafka, verify no `java.exe` processes are lingering (Task Manager), then start ZooKeeper, then Kafka.