Kafka Cluster Topic Key Distribution Hot Partition Analyzer Tool

Efficient **Kafka key distribution** is fundamental to building scalable, high-performance event-driven systems. Kafka uses each record's key to determine which partition it belongs to—governing **data ordering**, **load balancing**, and **parallelism** across the cluster. When key distribution is uneven, some partitions become hot, processing far more traffic than others. These **hot partitions** lead to broker overload, consumer lag, and throttled throughput, undermining the scalability of your Kafka workloads.

This tool helps you **test**, **visualize**, and **validate** how record keys are distributed across topic partitions in your Kafka cluster. It generates records using configurable key patterns, publishes them to a target topic, and then consumes the data to analyze partition utilization and message distribution metrics.

By surfacing patterns of **data skew**, **low-key cardinality**, or **biased hashing**, the analyzer reveals whether your partitioning strategy is truly balanced. The results empower you to:

- Detect and diagnose **hot partitions** before they degrade performance.
- Experiment with **key-salting** or **hashing strategies** to improve balance.
- Optimize **consumer parallelism** and **broker load** for predictable throughput at scale.

Use this tool as a **proactive performance lens** on your Kafka topics—ensuring your cluster's data distribution is as efficient, scalable, and reliable as the workloads it powers.

Table of Contents

- **1.0 To get started**
  - **1.1 Download the Tool**
  - **1.2 Configure the Tool**
  - **1.3 Run the Tool**
    - **1.3.1 Did you notice we prefix `uv run` to `streamlit run src/tool.py`?**
    - **1.3.2 Troubleshoot Connectivity Issues (if any)**
  - **1.4 The Results**
- **2.0 How the Tool Works**
- **3.0 Resources**

1.0 To get started

Download ---> ***Configure*** ---> ***Run*** ---> ***Results***

1.1 Download the Tool

Clone the repo: `shell git clone https://github.com/j3-signalroom/kafka_cluster-topic-key_distribution-hot_partition_analyzer-tool.git`

Since this project was built using `uv`, please install it, and then run the following command to install all the project dependencies:

```
uv sync
```

1.2 Configure the Tool

Now, you need to set up the tool by creating a `.env` file in the root directory of your project.

1.3 Run the Tool

Navigate to the Project Root Directory

Open your Terminal and navigate to the root folder of the `kafka_cluster-topic-key_distribution-hot_partition_analyzer-tool/` repository that you have cloned. You can do this by executing:

```
cd path/to/kafka_cluster-topic-key_distribution-hot_partition_analyzer-
tool/
```

> Replace `path/to/` with the actual path where your repository is located.

Then enter the following command below to run the tool:

```
uv run streamlit run src/tool.py
```

1.3.1 Did you notice we prefix `uv run` to `streamlit run src/tool.py`?

You maybe asking yourself why. Well, `uv` is an incredibly fast Python package installer and dependency resolver, written in **Rust**, and designed to seamlessly replace `pip`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more in your workflows. By prefixing `uv run` to a command, you're ensuring that the command runs in an optimal Python environment.

Now, let's go a little deeper into the magic behind `uv run`:

- When you use it with a file ending in `.py` or an HTTP(S) URL, `uv` treats it as a script and runs it with a Python interpreter. In other words, `uv run file.py` is equivalent to `uv run python file.py`. If you're working with a URL, `uv` even downloads it temporarily to execute it. Any inline dependency metadata is installed into an isolated, temporary environment—meaning zero leftover mess! When used with `-`, the input will be read from `stdin`, and treated as a Python script.
- If used in a project directory, `uv` will automatically create or update the project environment before running the command.
- Outside of a project, if there's a virtual environment present in your current directory (or any parent directory), `uv` runs the command in that environment. If no environment is found, it uses the interpreter's environment.

So what does this mean when we put `uv run` before `streamlit run src/tool.py`? It means `uv` takes care of all the setup—fast and seamless—right in your local environment. If you think AI/ML is magic, the work the folks at Astral have done with `uv` is pure wizardry!

Curious to learn more about Astral's uv? Check these out:

- Documentation: Learn about uv.
- Video: uv IS THE FUTURE OF PYTHON PACKING!.

If you have Kafka connectivity issues, you can verify connectivity using the following command:

1.3.2 Troubleshoot Connectivity Issues (if any)

To verify connectivity to your Kafka cluster, you can use the `kafka-topics.sh` command-line tool. First, download the Kafka binaries from the Apache Kafka website and extract them. Navigate to the `bin` directory of the extracted Kafka folder. Second, create a `client.properties` file with your Kafka credentials:

```
# For SASL_SSL (most common for cloud services)
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required \
  username="<YOUR_KAFKA_API_KEY>" \
  password="<YOUR_KAFKA_API_SECRET>";

# Additional SSL settings if needed
ssl.endpoint.identification.algorithm=https
```

Finally, run the following command to list all topics in your Kafka cluster:

```
./kafka-topics.sh --list --bootstrap-server <YOUR_BOOTSTRAP_SERVER_URI> --
command-config ./client.properties
```

If the connection is successful, you should see a list of topics in your Kafka cluster. If you encounter any errors, double-check your credentials and network connectivity.

1.4 The Results

2.0 How the Tool Works

3.0 Resources

- The Importance of Standardized Hashing Across Producers

- librdkafka Configuration