Kafka Cluster Topic Key Distribution Tester Tool

Kafka key distribution explains how Kafka assigns records to partitions within a topic based on their record key. It affects load balancing, order, and parallelism across partitions—key factors for Kafka's performance and scalability. Without proper key distribution, your topic can develop hot partitions. A hot partition in Kafka is one that receives or processes an unusually high amount of data or traffic compared to others in the same topic. This imbalance causes uneven load across brokers and consumers, which can reduce overall performance and throughput in the Kafka cluster.

This tool helps you test and analyze how keys are distributed in a Kafka topic within your cluster. It generates a specified number of records with different key patterns to a given topic, then consumes those records to examine how they are spread across the topic's partitions. The tool offers insights into the effectiveness of your key distribution strategy and can help identify potential issues like hot partitions.

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 `python 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_tester-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_tester-tool/` repository that you have cloned. You can do this by executing:

```
cd path/to/kafka_cluster-topic-key_distribution_tester-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 python src/tool.py
```

1.3.1 Did you notice we prefix `uv run` to `python 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 `python 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