Application Reset Tool

You can reset an application and force it to reprocess its data from scratch by using the application reset tool. This can be useful for development and testing, or when fixing bugs.

The application reset tool handles the Kafka Streamsuser topics (input, output, and intermediate topics) andinternal topics differently when resetting the application.

Here's what the application reset tool does for each topic type:

- Input topics: Reset offsets to specified position. By default they are reset to the beginning of the topic.
- Intermediate topics: Skip to the end of the topic, i.e., set the application's committed consumer offsets for all partitions to each partition's `logSize` (for consumer group `application.id`).
- Internal topics: Delete the internal topic (this automatically deletes any committed offsets).

The application reset tool does not:

- Reset output topics of an application. If any output (or intermediate) topics are consumed by downstream applications, it is your responsibility to adjust those downstream applications as appropriate when you reset the upstream application.
- Reset the local environment of your application instances. It is your responsibility to delete the local state on any machine on which an application instance was run. See the instructions in section Step 2: Reset the local environments of your application instanceson how to do this.

Prerequisites

- All instances of your application must be stopped. Otherwise, the application may enter an invalid state, crash, or produce incorrect results. You can verify whether the consumer group with ID `application.id` is still active by using `bin/kafka-consumer-groups`.
- Use this tool with care and double-check its parameters: If you provide wrong parameter values (e.g., typos in `application.id`) or specify parameters inconsistently (e.g., specify the wrong input topics for the application), this tool might invalidate the application's state or even impact other applications, consumer groups, or your Apache Kafka® topics.
- You should manually delete and re-create any intermediate topics before running the application reset tool. This will free up disk space in Kafka brokers.
- You should delete and recreate intermediate topics before running the application reset tool, unless the following applies:

  > - You have external downstream consumers for the application's intermediate topics.
  > - You are in a development environment where manually deleting and re-creating intermediate topics is unnecessary.

Step 1: Run the application reset tool

Invoke the application reset tool from the command line

```
<path-to-confluent>/bin/kafka-streams-application-reset
```

The tool accepts the following parameters:

```
Option (* = required)                 Description
---------------------                 -----------
* --application-id <String: ID>       The |kstreams| application ID
                                        (application.id).
--bootstrap-servers <String: urls>    Comma-separated list of broker urls with
                                        format: HOST1:PORT1,HOST2:PORT2
                                        (default: localhost:9092)
--by-duration <String: urls>          Reset offsets to offset by duration from
                                      current timestamp. Format: 'PnDTnHnMnS'
--config-file <String: file name>     Property file containing configs to be
                                        passed to admin clients and embedded
                                        consumer.
--dry-run                             Display the actions that would be
                                        performed without executing the reset
                                        commands.
--from-file <String: urls>            Reset offsets to values defined in CSV
                                      file.
--input-topics <String: list>         Comma-separated list of user input
                                        topics. For these topics, the tool will
                                        reset the offset to the earliest
                                        available offset.
--intermediate-topics <String: list>  Comma-separated list of intermediate user
                                        topics (topics used in the through()
                                        method). For these topics, the tool
                                        will skip to the end.
--shift-by <Long: number-of-offsets>  Reset offsets shifting current offset by
                                      'n', where 'n' can be positive or
                                      negative
--to-datetime <String>                Reset offsets to offset from datetime.
                                        Format: 'YYYY-MM-DDTHH:mm:SS.sss'
--to-earliest                         Reset offsets to earliest offset.
--to-latest                           Reset offsets to latest offset.
--to-offset <Long>                    Reset offsets to a specific offset.
--zookeeper                           Zookeeper option is deprecated by
                                        bootstrap.servers, as the reset tool
                                        would no longer access Zookeeper
                                        directly.
```

Parameters can be combined as needed. For example, if you want to restart an application from an empty internal state, but not reprocess previous data, simply omit the parameters `--input-topics` and `--intermediate-topics`.

Step 2: Reset the local environments of your application instances

For a complete application reset, you must delete the application's local state directory on any machines where the application instance was run. You must do this before restarting an application instance on the same machine. You can use either of these methods:

- The API method `KafkaStreams#cleanUp()` in your application code.
- Manually delete the corresponding local state directory (default location: `/var/lib/kafka-streams/<application.id>`). For more information, see state.dir StreamsConfig class.

Example

In this example you are developing and testing an application locally and you want to iteratively improve your application via run-reset-modify cycles.

```
package io.confluent.examples.streams;

import ...;

public class ResetDemo {

  public static void main(String[] args) throws Exception {
    // Kafka Streams configuration
    Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "my-streams-app");
    // ...and so on...

    // Define the processing topology
    StreamsBuilder builder = new StreamsBuilder();
    builder.stream("my-input-topic")
        .selectKey(...)
        .through("rekeyed-topic")
        .groupByKey()
        .count("global-count")
        .to("my-output-topic");

    KafkaStreams app = new KafkaStreams(builder.build(), props);

    // Delete the application's local state.
    // Note: In real application you'd call `cleanUp()` only under
    // certain conditions.  See tip on `cleanUp()` below.
    app.cleanUp();

    app.start();

    // Note: In real applications you would register a shutdown hook
    // that would trigger the call to `app.close()` rather than
    // using the sleep-then-close example we show here.
    Thread.sleep(30 * 1000L);
    app.close();
  }

}
```

Tip: To avoid the corresponding recovery overhead, you should not cal `cleanUp()` unconditionally and every time an application instance is restarted or resumed. For exammple, in a production application you could use command line arguments to enable or disable the `cleanUp()` call on an as-needed basis.

You can then perform run-reset-modify cycles as follows:

```
# Run your application
  bin/kafka-run-class io.confluent.examples.streams.ResetDemo

# After stopping all application instances, reset the application
  bin/kafka-streams-application-reset --application-id my-streams-app \
                                      --input-topics my-input-topic \
                                      --intermediate-topics rekeyed-topic

# Now you can modify/recompile as needed and then re-run the application again.
# You can also experiment, for example, with different input data without
# modifying the application.
```