Item 325
git_comments:

1. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
2. Projects depending on this project, won't depend on flink-table.
3. ---------- Utilities ----------
4. ---------- Runtime fields ----------
5. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
6. * * User defined configuration for the producer.
7. * * Number of unacknowledged records.
8. * * Pulsar Producer instance.
9. * * (Serializable) SerializationSchema for turning objects used with Flink into. * byte[] for Pulsar.
10. * * Errors encountered in the async producer are stored here.
11. * * If true, the producer will wait until all outstanding records have been send to the broker.
12. * * The pulsar service url.
13. * * Initializes the connection to pulsar. * * @param parameters configuration used for initialization * @throws Exception
14. make sure we propagate pending errors
15. ---------- Logic for handling checkpoint flushing ---------- //
16. * * User-provided key extractor for assigning a key to a pulsar message.
17. * * Produce Mode.
18. ---------- Properties ----------
19. if the flushed requests has errors, we should propagate it also and fail the checkpoint
20. * * @return pulsar key extractor.
21. * * Lock for accessing the pending records.
22. * * Sets this producer's operating mode. * * @param produceMode The mode of operation.
23. * * The name of the default topic this producer is writing data to.
24. ---------- Sink Methods ----------
25. * * Gets this producer's operating mode.
26. check for asynchronous errors and fail the checkpoint if necessary
27. * * If set to true, the Flink producer will wait for all outstanding messages in the Pulsar buffers * to be acknowledged by the Pulsar producer on a checkpoint. * This way, the producer can guarantee that messages in the Pulsar buffers are part of the checkpoint. * * @param flush Flag indicating the flushing mode (true = flush on checkpoint)
28. wait until all the messages are acknowledged
29. nothing to do
30. * * Flink Sink to produce data into a Pulsar topic.
31. * * The callback than handles error propagation or logging callbacks.
32. prevent double throwing
33. * * Base class for {@link PulsarTableSink} that serializes data in JSON format.
34. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or

agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
35. * * Create PulsarJsonTableSink. * * @param serviceUrl pulsar service url * @param topic topic in pulsar to which table is written * @param producerConf producer configuration * @param routingKeyFieldName routing key field name
36. * * The supported producing modes of operation for flink's pulsar producer.
37. * * Any produce failures will be ignored hence there could be data loss.
38. * * The producer will ensure that all the events are persisted in pulsar. * There could be duplicate events written though.
39. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
40. * * Returns the low-level producer.
41. * * Create serialization schema for converting table rows into bytes. * * @param rowSchema the schema of the row to serialize. * @return Instance of serialization schema
42. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
43. * * A key extractor that extracts the routing key from a {@link Row} by field name.
44. * * An append-only table sink to emit a streaming table as a Pulsar stream.
45. * * Create a deep copy of this sink. * * @return Deep copy of this sink
46. * * Retrieve a key from the value. * * @param in the value to extract a key. * @return key.
47. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
48. * * Extract key from a value.
49. * * Flag indicating whether to fail on a missing field.
50. * * Object mapper for parsing the JSON.
51. * * Configures the failure behaviour if a JSON field is missing. * * <p>By default, a missing field is ignored and the field is set to null. * * @param failOnMissingField Flag indicating whether to fail or not on a missing field.
52. * * Creates a JSON deserialization schema for the given fields and types. * * @param typeInfo Type information describing the result type. The field names are used * to parse the JSON file and so are the types.
53. * * Types to parse fields as. Indices match fieldNames indices.
54. Read the value as specified type
55. * * Deserialization schema from JSON to {@link Row}. * * <p>Deserializes the <code>byte[]</code> messages as a JSON object and reads * the specified fields. * * <p>Failure during deserialization are forwarded as wrapped IOExceptions.
56. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of

the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
57. * * Type information describing the result type.
58. * * Field names to parse. Indices match fieldTypes indices.
59. * * Serialization schema that serializes an object into a JSON bytes. * * <p>Serializes the input {@link Row} object into a JSON string and * converts it into <code>byte[]</code>. * * <p>Result <code>byte[]</code> messages can be deserialized using * {@link JsonRowDeserializationSchema}.
60. * * Fields names in the input Row object.
61. * * Object mapper that is used to create output JSON objects.
62. * * Creates a JSON serialization schema for the given fields and types. * * @param rowSchema The schema of the rows to encode.
63. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, * software distributed under the License is distributed on an * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY * KIND, either express or implied. See the License for the * specific language governing permissions and limitations * under the License.
64. check that no field is composite

git_commits:

1. **summary:** Add pulsar flink sink connector (#2434)
   message: Add pulsar flink sink connector (#2434) @XiaoZYang introduces a module for a pulsar sink connector for flink. This PR is moving the work from flink repo to pulsar repo, so the flink connector can be released faster along with Pulsar releases. Original Flink Github Issue: apache/flink#5845 Jira Issue: https://issues.apache.org/jira/browse/FLINK-9168 Original Author: @XiaoZYang (Zong Yang Xiao)

github_issues:

1. **title:** log4j2-appender producer name should be configurable
   body: When using the log4j2-appender it always gives a producer name of "pulsar-log4j2-appender-" + topic which means you can only send logs from one process to that topic at a time. When there are multiple instances of an application I think it makes sense to be able to consolidate them into a single topic.
2. **title:** log4j2-appender producer name should be configurable
   body: When using the log4j2-appender it always gives a producer name of "pulsar-log4j2-appender-" + topic which means you can only send logs from one process to that topic at a time. When there are multiple instances of an application I think it makes sense to be able to consolidate them into a single topic.
   label: documentation
3. **title:** log4j2-appender producer name should be configurable
   body: When using the log4j2-appender it always gives a producer name of "pulsar-log4j2-appender-" + topic which means you can only send logs from one process to that topic at a time. When there are multiple instances of an application I think it makes sense to be able to consolidate them into a single topic.
4. **title:** log4j2-appender producer name should be configurable
   body: When using the log4j2-appender it always gives a producer name of "pulsar-log4j2-appender-" + topic which means you can only send logs from one process to that topic at a time. When there are multiple instances of an application I think it makes sense to be able to consolidate them into a single topic.
5. **title:** log4j2-appender producer name should be configurable
   body: When using the log4j2-appender it always gives a producer name of "pulsar-log4j2-appender-" + topic which means you can only send logs from one process to that topic at a time. When there are multiple instances of an application I think it makes sense to be able to consolidate them into a single topic.

github_issues_comments:

1. Thanks @tkram01 for report this requirements. > I think it makes sense to be able to consolidate them into a single topic The 'topic' in log4j2 appender is configurable, If user want to append all logs into a single topic, only need to set the same `topic` name. If want to append to different topic, user also could set the different `topic` name. I may not understand it well, Would you please provide an example for the requirement?
2. **body:** @jiazhai we might consider adding documentation for this.
   label: documentation
3. I add the following note in "reference-configure/log4j". If we need more info, please provide, and I'll improve it further in [PR-7297](https://github.com/apache/pulsar/pull/7297) > 'topic' in log4j2.appender is configurable. > - If you want to append all logs to a single topic, set the same topic name. > - If you want to append logs to different topics, you can set different topic names.

github_pulls:

1. **title:** Add pulsar flink sink connector
   body: @XiaoZYang introduces a module for a pulsar sink connector for flink. This PR is moving the work from flink repo to pulsar repo, so the flink connector can be released faster along with Pulsar releases. Original Flink Github Issue: apache/flink#5845 Jira Issue: https://issues.apache.org/jira/browse/FLINK-9168 Original Author: @XiaoZYang (Zong Yang Xiao)

github_pulls_comments:

1. Created a master ticket for all the efforts related to flink connectors #2441
2. @aahmed-se : since we are moving other people's PR into pulsar's repo, it is important to mention the original work and credits to the original author @XiaoZYang .
3. retest this please
4. @aahmed-se there are some license header issues. PTAL ``` 2018-08-27\T\19:50:00.378 [ERROR] Failed to execute goal com.mycila:license-maven-plugin:3.0.rc1:check (default-cli) on project pulsar-flink: Some files do not have the expected license header -> [Help 1] ```
5. should be fixed
6. retest this please
7. run java8 tests
8. Looks like PulsarProduceMode.AT_LEAST_ONE is still incorrect (see https://github.com/apache/flink/pull/5845/files/fff36440197e3f34caf3af0b7ed7ba5320003e3b#r187084563)

github_pulls_reviews:

1. can you move these properties to root pom file (where is the the central place for placing properties)?
2. "pulsar-storm-shade" => "pulsar-flink-shade"

jira_issues:

1. **summary:** Pulsar Sink Connector
   description: Flink does not provide a sink connector for Pulsar.

jira_issues_comments:

1. Related works will be done by [~sijie@apache.org], [~zhaijia]and [~xiaozongyang_bupt] .
2. GitHub user XiaoZYang opened a pull request: https://github.com/apache/flink/pull/5845 [FLINK-9168] [flink-connectors]Pulsar Sink connector ## What is the purpose of the change Provide a [pulsar] (https://github.com/apache/incubator-pulsar) sink connector for flink. ## Brief change log - add `PulsarTableSink` - add `PulsarJsonTableSink` ## Verifying this change Added test that validates that target classses (i.e. PulsarTableSink and PulsarJsonTableSink) work as expected ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (yes) upgrade `org.javassist` version to 3.20.0-GA - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (don't know) - The runtime per-record code paths (performance sensitive): (don't know) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (don't

know) ## Documentation - Does this pull request introduce a new feature? (yes) - If yes, how is the feature documented? (JavaDocs) You can merge this pull request into a Git repository by running: $ git pull https://github.com/XiaoZYang/flink pulsar-connector Alternatively you can review and apply these changes as the patch at: https://github.com/apache/flink/pull/5845.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #5845 ---- commit fff36440197e3f34caf3af0b7ed7ba5320003e3b Author: Sijie Guo <sijie@...> Date: 2018-03-02T05:08:53Z Add pulsar flink connector commit 447a998ce9d4de2dbb7b099b0568c5aed9f374bd Author: xiaozongyang <xiaozongyang@...> Date: 2018-04-13T07:51:49Z 1.update pom.xml to follow flink version ----

3. Github user tzulitai commented on the issue: https://github.com/apache/flink/pull/5845 Thanks for this contribution @XiaoZYang! Pulsar seems like an interesting project. I think before we proceed with this, we need to make it clear (community-wise) that if we do accept this connector, there will be a maintainer for it. In the past, we had connector contributions that remained stale (because nobody was maintaining it) after it was merged to the codebase, which eventually started causing problems such as test instabilities and even the connector itself being in an unusable state without nobody realizing it. Because of that, our previous approach was to redirect new connector contributions to the Apache Bahir project. I personally think that the Flink community could still benefit from good connector contributions, and would be best if we can somehow keep them within Flink but still have good eyes on them. Maybe it would be a good idea to start a thread in the developer mailing lists about this connector, and whether or not it will be maintained (by either the original contributor or someone else) after it gets merged. What do you think?
4. Github user sijie commented on the issue: https://github.com/apache/flink/pull/5845 @tzulitai thank you for your comments. Glad to hear your opinions about pulsar connectors. I was the original person who initiated the idea of flink pulsar connectors with @XiaoZYang, I am also from Pulsar IPMC. Although Pulsar is a young project, it is a very active developing project. We have committers from various companies and pretty good adoption. from pulsar community perspective, we are very happy committed to developing/maintaining pulsar connectors. hope this can help clear some of your concerns. As the next step, I am happy to start the email thread at flink mailing list. should this be a FLIP? or just an general discussion email thread?
5. Github user tzulitai commented on the issue: https://github.com/apache/flink/pull/5845 Welcome to the community! @sijie I think for this a general discussion email thread will be enough.
6. Github user sijie commented on the issue: https://github.com/apache/flink/pull/5845 @tzulitai thank you very much for you help. just sent an email to dev@flink. look forward to feedback from flink community and collaboration between flink and pulsar communities.
7. Github user pluppens commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r187085104 --- Diff: flink-connectors/flink-connector-pulsar/src/test/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducerTest.java --- @@ -0,0 +1,100 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.flink.streaming.connectors.pulsar; +import org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.streaming.connectors.pulsar.serde.IntegerSerializationSchema; +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.junit.Test; +import org.junit.runner.RunWith; +import org.powermock.api.mockito.PowerMockito; +import org.powermock.core.classloader.annotations.PrepareForTest; +import org.powermock.modules.junit4.PowerMockRunner; +import static org.junit.Assert.assertEquals; +import static org.junit.Assert.assertSame; +import static org.mockito.Matchers.any; +import static org.mockito.Matchers.anyString; +import static org.mockito.Mockito.mock; +import static org.mockito.Mockito.spy; +import static org.mockito.Mockito.when; +/** + * Unit test of {@link FlinkPulsarProducer}. + */ +@RunWith(PowerMockRunner.class) +@PrepareForTest(PulsarClient.class) +public class FlinkPulsarProducerTest { + private static final String MOCK_SERVICE_URIL = "http://localhost:8080"; --- End diff -- `MOCK_SERVICE_URL` or `URI`?

8. Github user pluppens commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r187084563 --- Diff: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/PulsarProduceMode.java --- @@ -0,0 +1,36 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.flink.streaming.connectors.pulsar; + +/** + * The supported producing modes of operation for flink's pulsar producer. + */ +public enum PulsarProduceMode { + + /** + * Any produce failures will be ignored hence there could be data loss. + */ + AT_MOST_ONCE, + + /** + * The producer will ensure that all the events are persisted in pulsar. + * There could be duplicate events written though. + */ + AT_LEAST_ONE, --- End diff -- Is this is intentional? `AT_LEAST_ONCE` seems more appropriate?
9. Github user sijie commented on the issue: https://github.com/apache/flink/pull/5845 @pluppens thank you for your comments. I was busy with pulsar 2.0 release. I will try to pick this up again soon :)
10. Github user sijie commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r188867435 --- Diff: flink-connectors/flink-connector-pulsar/src/test/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducerTest.java --- @@ -0,0 +1,100 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.flink.streaming.connectors.pulsar; + +import org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.streaming.connectors.pulsar.serde.IntegerSerializationSchema; + +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.junit.Test; +import org.junit.runner.RunWith; +import org.powermock.api.mockito.PowerMockito; +import org.powermock.core.classloader.annotations.PrepareForTest; +import org.powermock.modules.junit4.PowerMockRunner; + +import static org.junit.Assert.assertEquals; +import static org.junit.Assert.assertSame; +import static org.mockito.Matchers.any; +import static org.mockito.Matchers.anyString; +import static org.mockito.Mockito.mock; +import static org.mockito.Mockito.spy; +import static org.mockito.Mockito.when; + +/** + * Unit test of {@link FlinkPulsarProducer}. + */ +@RunWith(PowerMockRunner.class) +@PrepareForTest(PulsarClient.class) +public class FlinkPulsarProducerTest { + + private static final String MOCK_SERVICE_URIL = "http://localhost:8080"; --- End diff -- thank you for catching this. it should be 'URL'.
11. Github user sijie commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r188867505 --- Diff: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/PulsarProduceMode.java --- @@ -0,0 +1,36 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.flink.streaming.connectors.pulsar; + +/** + * The supported producing modes of operation for flink's pulsar producer. + */ +public enum PulsarProduceMode { + + /** + * Any produce failures will be ignored hence there could be data loss. + */ + AT_MOST_ONCE, + + /** + * The producer will ensure that all the events are persisted in pulsar. +

* There could be duplicate events written though. + */ + AT_LEAST_ONE, --- End diff -- nice catch. It should be `AT_LEAST_ONCE`

12. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 @sijie would you mind help reviewing this commit?
13. Github user XiaoZYang closed the pull request at: https://github.com/apache/flink/pull/5845
14. GitHub user XiaoZYang reopened a pull request: https://github.com/apache/flink/pull/5845 [FLINK-9168][flink-connectors]Pulsar Sink connector ## What is the purpose of the change Provide a [pulsar](https://github.com/apache/incubator-pulsar) sink connector for flink. ## Brief change log - add `PulsarTableSink` - add `PulsarJsonTableSink` ## Verifying this change Added test that validates that target classses (i.e. PulsarTableSink and PulsarJsonTableSink) work as expected ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (yes) upgrade `org.javassist` version to 3.20.0-GA - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (don't know) - The runtime per-record code paths (performance sensitive): (don't know) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (don't know) ## Documentation - Does this pull request introduce a new feature? (yes) - If yes, how is the feature documented? (JavaDocs) You can merge this pull request into a Git repository by running: $ git pull https://github.com/XiaoZYang/flink pulsar-connector Alternatively you can review and apply these changes as the patch at: https://github.com/apache/flink/pull/5845.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #5845 ---- commit fff36440197e3f34caf3af0b7ed7ba5320003e3b Author: Sijie Guo <sijie@...> Date: 2018-03-02T05:08:53Z Add pulsar flink connector commit 447a998ce9d4de2dbb7b099b0568c5aed9f374bd Author: xiaozongyang <xiaozongyang@...> Date: 2018-04-13T07:51:49Z 1.update pom.xml to follow flink version commit ed74e0eee39bf5e175e45402564630f195a37dfb Author: xiaozy <xiaozy@...> Date: 2018-05-20T04:01:47Z add unit test of PulsarTableSink and FlinkPulsarProducer commit b0050c2af0a47d40bf3e0745ce272d7048afa37e Author: xiaozy <xiaozy@...> Date: 2018-05-20T04:01:47Z add unit test of PulsarTableSink and FlinkPulsarProducer commit c1ebbec7fde5d9243ddae25f46504f2e5ce41373 Author: xiaozy <xiaozy@...> Date: 2018-05-20T09:19:40Z Merge branch 'pulsar-connector' of https://github.com/XiaoZYang/flink into pulsar-connector ----
15. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 ping @sijie
16. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 Hi @sijie @tzulitai, since there will be serial of commits about *pulsar connector*, what about close this pr and reopen a new PR which is going to merge to a new branch named "pulsar-connector". So we can complete all works by incremental commits, then we merge the branch pulsar-connector to master. By doing so, we can avoid resolving conflicts again and again.
17. Github user sijie commented on the issue: https://github.com/apache/flink/pull/5845 @tzulitai what is your opinion about @XiaoZYang 's comment?
18. Github user surryr commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r193793061 --- Diff: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducer.java --- @@ -0,0 +1,304 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.flink.streaming.connectors.pulsar; + +import org.apache.flink.api.common.functions.RuntimeContext; +import org.apache.flink.api.common.serialization.SerializationSchema; +import org.apache.flink.api.java.ClosureCleaner; +import org.apache.flink.configuration.Configuration; +import org.apache.flink.runtime.state.FunctionInitializationContext; +import org.apache.flink.runtime.state.FunctionSnapshotContext; +import org.apache.flink.streaming.api.checkpoint.CheckpointedFunction; +import org.apache.flink.streaming.api.functions.sink.RichSinkFunction; +import org.apache.flink.streaming.api.operators.StreamingRuntimeContext; +import

org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.util.SerializableObject; + +import org.apache.pulsar.client.api.Message; +import org.apache.pulsar.client.api.MessageBuilder; +import org.apache.pulsar.client.api.MessageId; +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.slf4j.Logger; +import org.slf4j.LoggerFactory; + +import java.util.function.Function; + +import static org.apache.flink.util.Preconditions.checkNotNull; + +/** + * Flink Sink to produce data into a Pulsar topic. + */ +public class FlinkPulsarProducer<IN> + extends RichSinkFunction<IN> + implements CheckpointedFunction { + + private static final Logger LOG = LoggerFactory.getLogger(FlinkPulsarProducer.class); + + /** + * The pulsar service url. + */ + protected final String serviceUrl; + + /** + * User defined configuration for the producer. + */ + protected final ProducerConfiguration producerConfig; + + /** + * The name of the default topic this producer is writing data to. + */ + protected final String defaultTopicName; + + /** + * (Serializable) SerializationSchema for turning objects used with Flink into. + * byte[] for Pulsar. + */ + protected final SerializationSchema<IN> schema; + + /** + * User-provided key extractor for assigning a key to a pulsar message. + */ + protected final PulsarKeyExtractor<IN> flinkPulsarKeyExtractor; + + /** + * Produce Mode. + */ + protected PulsarProduceMode produceMode = PulsarProduceMode.AT_LEAST_ONE; + + /** + * If true, the producer will wait until all outstanding records have been send to the broker. + */ + protected boolean flushOnCheckpoint; + + // ------------------------------- Runtime fields --------------------------------------- + + /** Pulsar Producer instance. */ + protected transient Producer producer; + + /** The callback than handles error propagation or logging callbacks. */ + protected transient Function<MessageId, MessageId> successCallback = msgId -> { + acknowledgeMessage(); + return msgId; + }; + + protected transient Function<Throwable, MessageId> failureCallback; + + /** Errors encountered in the async producer are stored here. */ + protected transient volatile Exception asyncException; + + /** Lock for accessing the pending records. */ + protected final SerializableObject pendingRecordsLock = new SerializableObject(); + + /** Number of unacknowledged records. */ + protected long pendingRecords; + + public FlinkPulsarProducer(String serviceUrl, + String defaultTopicName, + SerializationSchema<IN> serializationSchema, + ProducerConfiguration producerConfig, + PulsarKeyExtractor<IN> keyExtractor) { + this.serviceUrl = checkNotNull(serviceUrl, "Service url not set"); + this.defaultTopicName = checkNotNull(defaultTopicName, "TopicName not set"); + this.schema = checkNotNull(serializationSchema, "Serialization Schema not set"); + this.producerConfig = checkNotNull(producerConfig, "Producer Config is not set"); + this.flinkPulsarKeyExtractor = getOrNullKeyExtractor(keyExtractor); + ClosureCleaner.ensureSerializable(serializationSchema); + } + + // ---------------------------------- Properties ------------------------- + + /** + * @return pulsar key extractor. + */ + public PulsarKeyExtractor<IN> getKeyExtractor() { + return flinkPulsarKeyExtractor; + } + + /** + * Gets this producer's operating mode. + */ + public PulsarProduceMode getProduceMode() { + return this.produceMode; + } + + /** + * Sets this producer's operating mode. + * + * @param produceMode The mode of operation. + */ + public void setProduceMode(PulsarProduceMode produceMode) { + this.produceMode = checkNotNull(produceMode); + } + + /** + * If set to true, the Flink producer will wait for all outstanding messages in the Pulsar buffers + * to be acknowledged by the Pulsar producer on a checkpoint. + * This way, the producer can guarantee that messages in the Pulsar buffers are part of the checkpoint. + * + * @param flush Flag indicating the flushing mode (true = flush on checkpoint) + */ + public void setFlushOnCheckpoint(boolean flush) { + this.flushOnCheckpoint = flush; + } + + // ---------------------------------- Sink Methods ------------------------- + + @SuppressWarnings("unchecked") + private static final <T> PulsarKeyExtractor<T> getOrNullKeyExtractor(PulsarKeyExtractor<T> extractor) { + if (null == extractor) { + return PulsarKeyExtractor.NULL; + } else { + return extractor; + } + } + + private Producer createProducer(ProducerConfiguration configuration) throws Exception { + PulsarClient client = PulsarClient.create(serviceUrl); + return client.createProducer(defaultTopicName, configuration); + } + + /** + * Initializes the connection to pulsar. + * + * @param parameters configuration used for initialization + * @throws Exception + */ + @Override + public void open(Configuration parameters) throws Exception { + this.producer = createProducer(producerConfig); + + RuntimeContext ctx = getRuntimeContext(); + + LOG.info("Starting FlinkPulsarProducer ({}/{}) to produce into pulsar topic {}", + ctx.getIndexOfThisSubtask() + 1, ctx.getNumberOfParallelSubtasks(), defaultTopicName); + + if (flushOnCheckpoint && !((StreamingRuntimeContext) this.getRuntimeContext()).isCheckpointingEnabled()) { + LOG.warn("Flushing on checkpoint is enabled, but checkpointing is not enabled. Disabling flushing."); + flushOnCheckpoint = false; + } + + if (PulsarProduceMode.AT_MOST_ONCE == produceMode) { + this.failureCallback = cause -> { + LOG.error("Error while sending record to Pulsar : " + cause.getMessage(), cause); + return null; + }; + }

else if (PulsarProduceMode.AT_LEAST_ONE == produceMode){ + this.failureCallback = cause -> { + if (null == asyncException) { + if (cause instanceof Exception) { + asyncException = (Exception) cause; + } else { + asyncException = new Exception(cause); + } + } + return null; + }; + } else { + throw new UnsupportedOperationException("Unsupported produce mode " + produceMode); + } + } + + @Override + public void invoke(IN value, Context context) throws Exception { + checkErroneous(); + + byte[] serializedValue = schema.serialize(value); + + MessageBuilder msgBuilder = MessageBuilder.create(); + if (null != context.timestamp()) { + msgBuilder = msgBuilder.setEventTime(context.timestamp()); + } + String msgKey = flinkPulsarKeyExtractor.getKey(value); + if (null != msgKey) { + msgBuilder = msgBuilder.setKey(msgKey); + } + Message message = msgBuilder + .setContent(serializedValue) + .build(); + + if (flushOnCheckpoint) { + synchronized (pendingRecordsLock) { + pendingRecords++; + } + } + producer.sendAsync(message) + .thenApply(successCallback) + .exceptionally(failureCallback); + } + + @Override + public void close() throws Exception { + if (producer != null) { + producer.close(); + } + + // make sure we propagate pending errors + checkErroneous(); + } + + // ------------------- Logic for handling checkpoint flushing ------------------------- // + + private void acknowledgeMessage() { + if (flushOnCheckpoint) { + synchronized (pendingRecordsLock) { + pendingRecords--; + if (pendingRecords == 0) { + pendingRecordsLock.notifyAll(); + } + } + } + } + + @Override + public void snapshotState(FunctionSnapshotContext context) throws Exception { + // check for asynchronous errors and fail the checkpoint if necessary + checkErroneous(); + + if (flushOnCheckpoint) { + // wait until all the messages are acknowledged + synchronized (pendingRecordsLock) { + while (pendingRecords > 0) { + pendingRecordsLock.wait(100); + } + } + + // if the flushed requests has errors, we should propagate it also and fail the checkpoint + checkErroneous(); + } + } + + @Override + public void initializeState(FunctionInitializationContext context) throws Exception { + // nothing to do + } + + // -- Utilities ------------------------- + + protected void checkErroneous() throws Exception { + Exception e = asyncException; + if (e != null) { + // prevent double throwing + asyncException = null; + throw new Exception("Failed to send data to Kafka: " + e.getMessage(), e); --- End diff -- Seems like typo is here.

19. Github user XiaoZYang commented on a diff in the pull request: https://github.com/apache/flink/pull/5845#discussion_r194076473 --- Diff: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducer.java --- @@ -0,0 +1,304 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.flink.streaming.connectors.pulsar; + +import org.apache.flink.api.common.functions.RuntimeContext; +import org.apache.flink.api.common.serialization.SerializationSchema; +import org.apache.flink.api.java.ClosureCleaner; +import org.apache.flink.configuration.Configuration; +import org.apache.flink.runtime.state.FunctionInitializationContext; +import org.apache.flink.runtime.state.FunctionSnapshotContext; +import org.apache.flink.streaming.api.checkpoint.CheckpointedFunction; +import org.apache.flink.streaming.api.functions.sink.RichSinkFunction; +import org.apache.flink.streaming.api.operators.StreamingRuntimeContext; +import org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.util.SerializableObject; + +import org.apache.pulsar.client.api.Message; +import org.apache.pulsar.client.api.MessageBuilder; +import org.apache.pulsar.client.api.MessageId; +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.slf4j.Logger; +import org.slf4j.LoggerFactory; + +import java.util.function.Function; + +import static org.apache.flink.util.Preconditions.checkNotNull; + +/** + * Flink Sink to produce data into a Pulsar topic. + */ +public class FlinkPulsarProducer<IN> + extends RichSinkFunction<IN> + implements CheckpointedFunction { + + private static final Logger LOG = LoggerFactory.getLogger(FlinkPulsarProducer.class); + + /** + * The pulsar service url. + */ + protected final String serviceUrl; + + /** + * User defined configuration for the producer. + */ + protected final ProducerConfiguration producerConfig; + + /** + * The name of the default topic this producer is writing data to. + */ + protected final String defaultTopicName; + + /** + * (Serializable) SerializationSchema for

turning objects used with Flink into. + * byte[] for Pulsar. + */ + protected final SerializationSchema<IN> schema; + + /** + * User-provided key extractor for assigning a key to a pulsar message. + */ + protected final PulsarKeyExtractor<IN> flinkPulsarKeyExtractor; + + /** + * Produce Mode. + */ + protected PulsarProduceMode produceMode = PulsarProduceMode.AT_LEAST_ONE; + + /** + * If true, the producer will wait until all outstanding records have been send to the broker. + */ + protected boolean flushOnCheckpoint; + + // ------------------------------- Runtime fields --------------------------------------- + + /** Pulsar Producer instance. */ + protected transient Producer producer; + + /** The callback than handles error propagation or logging callbacks. */ + protected transient Function<MessageId, MessageId> successCallback = msgId -> { + acknowledgeMessage(); + return msgId; + }; + + protected transient Function<Throwable, MessageId> failureCallback; + + /** Errors encountered in the async producer are stored here. */ + protected transient volatile Exception asyncException; + + /** Lock for accessing the pending records. */ + protected final SerializableObject pendingRecordsLock = new SerializableObject(); + + /** Number of unacknowledged records. */ + protected long pendingRecords; + + public FlinkPulsarProducer(String serviceUrl, + String defaultTopicName, + SerializationSchema<IN> serializationSchema, + ProducerConfiguration producerConfig, + PulsarKeyExtractor<IN> keyExtractor) { + this.serviceUrl = checkNotNull(serviceUrl, "Service url not set"); + this.defaultTopicName = checkNotNull(defaultTopicName, "TopicName not set"); + this.schema = checkNotNull(serializationSchema, "Serialization Schema not set"); + this.producerConfig = checkNotNull(producerConfig, "Producer Config is not set"); + this.flinkPulsarKeyExtractor = getOrNullKeyExtractor(keyExtractor); + ClosureCleaner.ensureSerializable(serializationSchema); + } + + // ------------------------------- Properties ---------------------------- + + /** + * @return pulsar key extractor. + */ + public PulsarKeyExtractor<IN> getKeyExtractor() { + return flinkPulsarKeyExtractor; + } + + /** + * Gets this producer's operating mode. + */ + public PulsarProduceMode getProduceMode() { + return this.produceMode; + } + + /** + * Sets this producer's operating mode. + * + * @param produceMode The mode of operation. + */ + public void setProduceMode(PulsarProduceMode produceMode) { + this.produceMode = checkNotNull(produceMode); + } + + /** + * If set to true, the Flink producer will wait for all outstanding messages in the Pulsar buffers + * to be acknowledged by the Pulsar producer on a checkpoint. + * This way, the producer can guarantee that messages in the Pulsar buffers are part of the checkpoint. + * + * @param flush Flag indicating the flushing mode (true = flush on checkpoint) + */ + public void setFlushOnCheckpoint(boolean flush) { + this.flushOnCheckpoint = flush; + } + + // ------------------------------- Sink Methods ---------------------------- + + @SuppressWarnings("unchecked") + private static final <T> PulsarKeyExtractor<T> getOrNullKeyExtractor(PulsarKeyExtractor<T> extractor) { + if (null == extractor) { + return PulsarKeyExtractor.NULL; + } else { + return extractor; + } + } + + private Producer createProducer(ProducerConfiguration configuration) throws Exception { + PulsarClient client = PulsarClient.create(serviceUrl); + return client.createProducer(defaultTopicName, configuration); + } + + /** + * Initializes the connection to pulsar. + * + * @param parameters configuration used for initialization + * @throws Exception + */ + @Override + public void open(Configuration parameters) throws Exception { + this.producer = createProducer(producerConfig); + + RuntimeContext ctx = getRuntimeContext(); + + LOG.info("Starting FlinkPulsarProducer ({}/{}) to produce into pulsar topic {}", + ctx.getIndexOfThisSubtask() + 1, ctx.getNumberOfParallelSubtasks(), defaultTopicName); + + if (flushOnCheckpoint && !((StreamingRuntimeContext) this.getRuntimeContext()).isCheckpointingEnabled()) { + LOG.warn("Flushing on checkpoint is enabled, but checkpointing is not enabled. Disabling flushing."); + flushOnCheckpoint = false; + } + + if (PulsarProduceMode.AT_MOST_ONCE == produceMode) { + this.failureCallback = cause -> { + LOG.error("Error while sending record to Pulsar : " + cause.getMessage(), cause); + return null; + }; + } else if (PulsarProduceMode.AT_LEAST_ONE == produceMode){ + this.failureCallback = cause -> { + if (null == asyncException) { + if (cause instanceof Exception) { + asyncException = (Exception) cause; + } else { + asyncException = new Exception(cause); + } + } + return null; + }; + } else { + throw new UnsupportedOperationException("Unsupported produce mode " + produceMode); + } + } + + @Override + public void invoke(IN value, Context context) throws Exception { + checkErroneous(); + + byte[] serializedValue = schema.serialize(value); + + MessageBuilder msgBuilder = MessageBuilder.create(); + if (null != context.timestamp()) { + msgBuilder = msgBuilder.setEventTime(context.timestamp()); + } + String msgKey = flinkPulsarKeyExtractor.getKey(value); + if (null != msgKey) { + msgBuilder = msgBuilder.setKey(msgKey); + } + Message message = msgBuilder + .setContent(serializedValue) + .build(); + + if (flushOnCheckpoint) { + synchronized (pendingRecordsLock) { + pendingRecords++; + } + } + producer.sendAsync(message) + .thenApply(successCallback) + .exceptionally(failureCallback); + } + + @Override + public void close() throws Exception { + if (producer != null) { + producer.close(); + } + + // make sure we propagate pending errors + checkErroneous(); + } + + // ------------------- Logic for

handling checkpoint flushing ---------- // + + private void acknowledgeMessage() { + if (flushOnCheckpoint) { + synchronized (pendingRecordsLock) { + pendingRecords--; + if (pendingRecords == 0) { + pendingRecordsLock.notifyAll(); + } + } + } + + @Override + public void snapshotState(FunctionSnapshotContext context) throws Exception { + // check for asynchronous errors and fail the checkpoint if necessary + checkErroneous(); + + if (flushOnCheckpoint) { + // wait until all the messages are acknowledged + synchronized (pendingRecordsLock) { + while (pendingRecords > 0) { + pendingRecordsLock.wait(100); + } + } + + // if the flushed requests has errors, we should propagate it also and fail the checkpoint + checkErroneous(); + } + + @Override + public void initializeState(FunctionInitializationContext context) throws Exception { + // nothing to do + } + + // ---------- Utilities ---------- + + protected void checkErroneous() throws Exception { + Exception e = asyncException; + if (e != null) { + // prevent double throwing + asyncException = null; + throw new Exception("Failed to send data to Kafka: " + e.getMessage(), e); --- End diff -- Yes, thanks for pointing out this mistake, it's correct in the newest commit. ping @surryr

20. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 @tzulitai @surryr @pluppens @sijie This PR is going to be closed and I will open another PR to brunch pulsar-connector to this issue.
21. Github user XiaoZYang closed the pull request at: https://github.com/apache/flink/pull/5845
22. GitHub user XiaoZYang reopened a pull request: https://github.com/apache/flink/pull/5845 [FLINK-9168][flink-connectors]Pulsar Sink connector ## What is the purpose of the change Provide a [pulsar](https://github.com/apache/incubator-pulsar) sink connector for flink. ## Brief change log - add `PulsarTableSink` - add `PulsarJsonTableSink` ## Verifying this change Added test that validates that target classses (i.e. PulsarTableSink and PulsarJsonTableSink) work as expected ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (yes) upgrade `org.javassist` version to 3.20.0-GA - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (don't know) - The runtime per-record code paths (performance sensitive): (don't know) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (don't know) ## Documentation - Does this pull request introduce a new feature? (yes) - If yes, how is the feature documented? (JavaDocs) You can merge this pull request into a Git repository by running: $ git pull https://github.com/XiaoZYang/flink pulsar-connector Alternatively you can review and apply these changes as the patch at: https://github.com/apache/flink/pull/5845.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #5845 ---- commit fff36440197e3f34caf3af0b7ed7ba5320003e3b Author: Sijie Guo <sijie@...> Date: 2018-03-02T05:08:53Z Add pulsar flink connector commit 447a998ce9d4de2dbb7b099b0568c5aed9f374bd Author: xiaozongyang <xiaozongyang@...> Date: 2018-04-13T07:51:49Z 1.update pom.xml to follow flink version commit ed74e0eee39bf5e175e45402564630f195a37dfb Author: xiaozy <xiaozy@...> Date: 2018-05-20T04:01:47Z add unit test of PulsarTableSink and FlinkPulsarProducer commit b0050c2af0a47d40bf3e0745ce272d7048afa37e Author: xiaozy <xiaozy@...> Date: 2018-05-20T04:01:47Z add unit test of PulsarTableSink and FlinkPulsarProducer commit c1ebbec7fde5d9243ddae25f46504f2e5ce41373 Author: xiaozy <xiaozy@...> Date: 2018-05-20T09:19:40Z Merge branch 'pulsar-connector' of https://github.com/XiaoZYang/flink into pulsar-connector ----
23. Github user tzulitai commented on the issue: https://github.com/apache/flink/pull/5845 Sorry for delays on my reply here. I'll take a look at this week, over the next days. If there is going to be a new PR, please also let me know. Thanks!
24. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 I prefer to open a new PR and a new branch. But I am not authorized to do that.
25. Github user hsaputra commented on the issue: https://github.com/apache/flink/pull/5845 @XiaoZYang You can close this PR and create new branch to submit new PR since you are the creator os this one. Did you see any error or something preventing you to close this one?
26. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 @hsaputra There is no error but the issue is 1. when I create a PR to the flink repo from my fork repo I need to choose a branch to which the commits are merged to 2. I'm expected to merge the commits are merged to a branch name like 'pulsar-connector', but I can't create a new branch at flink repo Is there a better way to do that? Thank for you helping me !
27. Github user zentol commented on the issue: https://github.com/apache/flink/pull/5845 it is not possible to create a PR against a non-existing branch. We will either have to create a pulsar-connector branch up front, or open the first PR against master and merge it into a new branch. I would suggest the latter option.

28. Github user XiaoZYang commented on the issue: https://github.com/apache/flink/pull/5845 @zentol got it! So what you mean is that @sijie and I will keep updating commits until this work is done and you guys will merge this PR to a new branch. Later PR will be against the new branch. Did I misunderstand you ?
29. chrismiller commented on a change in pull request #5845: [FLINK-9168][flink-connectors]Pulsar Sink connector URL: https://github.com/apache/flink/pull/5845#discussion_r219230230 ########## File path: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducer.java ########## @@ -0,0 +1,304 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.flink.streaming.connectors.pulsar; + +import org.apache.flink.api.common.functions.RuntimeContext; +import org.apache.flink.api.common.serialization.SerializationSchema; +import org.apache.flink.api.java.ClosureCleaner; +import org.apache.flink.configuration.Configuration; +import org.apache.flink.runtime.state.FunctionInitializationContext; +import org.apache.flink.runtime.state.FunctionSnapshotContext; +import org.apache.flink.streaming.api.checkpoint.CheckpointedFunction; +import org.apache.flink.streaming.api.functions.sink.RichSinkFunction; +import org.apache.flink.streaming.api.operators.StreamingRuntimeContext; +import org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.util.SerializableObject; + +import org.apache.pulsar.client.api.Message; +import org.apache.pulsar.client.api.MessageBuilder; +import org.apache.pulsar.client.api.MessageId; +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.slf4j.Logger; +import org.slf4j.LoggerFactory; + +import java.util.function.Function; + +import static org.apache.flink.util.Preconditions.checkNotNull; + +/** + * Flink Sink to produce data into a Pulsar topic. + */ +public class FlinkPulsarProducer<IN> + extends RichSinkFunction<IN> + implements CheckpointedFunction { + + private static final Logger LOG = LoggerFactory.getLogger(FlinkPulsarProducer.class); + + /** + * The pulsar service url. + */ + protected final String serviceUrl; + + /** + * User defined configuration for the producer. + */ + protected final ProducerConfiguration producerConfig; + + /** + * The name of the default topic this producer is writing data to. + */ + protected final String defaultTopicName; + + /** + * (Serializable) SerializationSchema for turning objects used with Flink into. + * byte[] for Pulsar. + */ + protected final SerializationSchema<IN> schema; + + /** + * User-provided key extractor for assigning a key to a pulsar message. + */ + protected final PulsarKeyExtractor<IN> flinkPulsarKeyExtractor; + + /** + * Produce Mode. + */ + protected PulsarProduceMode produceMode = PulsarProduceMode.AT_LEAST_ONE; + + /** + * If true, the producer will wait until all outstanding records have been send to the broker. + */ + protected boolean flushOnCheckpoint; + + // ---------------------------------- Runtime fields --------------------------------------- + + /** KafkaProducer instance. */ + protected transient Producer producer; + + /** The callback than handles error propagation or logging callbacks. */ + protected transient Function<MessageId, MessageId> successCallback = msgId -> { Review comment: This won't deserialise correctly, it will end up null. The solution is to initialise successCallback in the open() method instead, similar to how failureCallback is initialised. ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
30. chrismiller commented on a change in pull request #5845: [FLINK-9168][flink-connectors]Pulsar Sink connector URL: https://github.com/apache/flink/pull/5845#discussion_r219230230 ########## File path: flink-connectors/flink-connector-pulsar/src/main/java/org/apache/flink/streaming/connectors/pulsar/FlinkPulsarProducer.java ########## @@ -0,0 +1,304 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 +

* + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.flink.streaming.connectors.pulsar; + +import org.apache.flink.api.common.functions.RuntimeContext; +import org.apache.flink.api.common.serialization.SerializationSchema; +import org.apache.flink.api.java.ClosureCleaner; +import org.apache.flink.configuration.Configuration; +import org.apache.flink.runtime.state.FunctionInitializationContext; +import org.apache.flink.runtime.state.FunctionSnapshotContext; +import org.apache.flink.streaming.api.checkpoint.CheckpointedFunction; +import org.apache.flink.streaming.api.functions.sink.RichSinkFunction; +import org.apache.flink.streaming.api.operators.StreamingRuntimeContext; +import org.apache.flink.streaming.connectors.pulsar.partitioner.PulsarKeyExtractor; +import org.apache.flink.util.SerializableObject; + +import org.apache.pulsar.client.api.Message; +import org.apache.pulsar.client.api.MessageBuilder; +import org.apache.pulsar.client.api.MessageId; +import org.apache.pulsar.client.api.Producer; +import org.apache.pulsar.client.api.ProducerConfiguration; +import org.apache.pulsar.client.api.PulsarClient; +import org.slf4j.Logger; +import org.slf4j.LoggerFactory; + +import java.util.function.Function; + +import static org.apache.flink.util.Preconditions.checkNotNull; + +/** + * Flink Sink to produce data into a Pulsar topic. + */ +public class FlinkPulsarProducer<IN> + extends RichSinkFunction<IN> + implements CheckpointedFunction { + + private static final Logger LOG = LoggerFactory.getLogger(FlinkPulsarProducer.class); + + /** + * The pulsar service url. + */ + protected final String serviceUrl; + + /** + * User defined configuration for the producer. + */ + protected final ProducerConfiguration producerConfig; + + /** + * The name of the default topic this producer is writing data to. + */ + protected final String defaultTopicName; + + /** + * (Serializable) SerializationSchema for turning objects used with Flink into. + * byte[] for Pulsar. + */ + protected final SerializationSchema<IN> schema; + + /** + * User-provided key extractor for assigning a key to a pulsar message. + */ + protected final PulsarKeyExtractor<IN> flinkPulsarKeyExtractor; + + /** + * Produce Mode. + */ + protected PulsarProduceMode produceMode = PulsarProduceMode.AT_LEAST_ONE; + + /** + * If true, the producer will wait until all outstanding records have been send to the broker. + */ + protected boolean flushOnCheckpoint; + + // ------------------------------- Runtime fields --------------------------------------- + + /** KafkaProducer instance. */ + protected transient Producer producer; + + /** The callback than handles error propagation or logging callbacks. */ + protected transient Function<MessageId, MessageId> successCallback = msgId -> { Review comment: This won't deserialise correctly, it will end up null and result in invoke() generating a NPE. The solution is to initialise successCallback in the open() method instead, similar to how failureCallback is initialised. -- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org

31. I'm sorry but I'm closing this because of the newer efforts in FLINK-14146.