Item 193
git_comments:

1. ----------

git_commits:

1. **summary:** [FLINK-10353][kafka] Support change of transactional semantics in Kafka Producer with existing state
   message: [FLINK-10353][kafka] Support change of transactional semantics in Kafka Producer with existing state This closes #7010.

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
2. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
3. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
4. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
5. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing,

Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)

6. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
   label: code-design
7. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
8. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
   label: code-design
9. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
   body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
10. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
11. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing,

Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)

12. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
13. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
14. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
    label: code-design
15. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
16. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)
17. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing,

Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)

18. **title:** [FLINK-10353][kafka] Support change of transactional semantics in Kaf…
    body: …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` I wonder if we should also add tests from/to the NONE semantics. The tests are pretty heavy weight for what they are doing. ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable)

github_pulls_comments:

1. Addressed the comments. Please have another look.
2. Thanks for the review @pnowojski. I will keep the hotfix commit separated. Merging.

github_pulls_reviews:

1. why do you need this change?
2. please deduplicate this code with `testMigrateFromAtLeastOnceToExactlyOnce` ``` private void testMigrateSemantic(Semantic from, Semantic to, String topicName) { // both try blocks } ```
3. **body:** start this from `45`. Now it's not obvious whether the result `44` comes from here or from above.
   label: code-design
4. also extend the second try block a little bit by completing at least one checkpoint to make sure that committing also works.
5. **body:** Could you provide overloaded version of `assertExactlyOnceForTopic` with default value for `long timeoutMillis = 30_000L` as separate commit? I know that it was like that before, but I have only now realised how duplicated (mostly by me) the magic constant `30_000L` is everywhere...
   label: code-design
6. Those changes should be in separate commit.
7. is this relevant to the bug fix or is this a clean up refactor?
8. Because otherwise the the restore of state from the `EXACTLY_ONCE` case will insert this key in the producer config and we will end up having a producer that is configured for transactions, which we don't want.
9. I think that should work, but the expected result also needs to be provided because it is different.
10. can do
11. **body:** A couple of other things in the test might be somewhat duplicated as well, I can just do a general pass and dedup stuff.
    label: code-design
12. This is relevant, there is a semantical difference. Small, but important.
13. Core problem here being that the config is somewhat a shared object that is written to in different attempts of initializing a producer.
14. you could provide expected result or do the assertion outside of the deduplicated method. I was thinking about the latter one, but maybe the first one is better. Either are fine for me.

jira_issues:

1. **summary:** Restoring a KafkaProducer with Semantic.EXACTLY_ONCE from a savepoint written with Semantic.AT_LEAST_ONCE fails with NPE
   description: If a KafkaProducer with {{Semantic.EXACTLY_ONCE}} is restored from a savepoint written with {{Semantic.AT_LEAST_ONCE}} the job fails on restore with the NPE below. This makes it impossible to upgrade an AT_LEAST_ONCE pipeline to an EXACTL_ONCE pipeline statefully. {quote} java.lang.NullPointerException at java.util.Hashtable.put(Hashtable.java:460) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initTransactionalProducer(FlinkKafkaProducer011.java:955) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:733) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:93) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.recoverAndCommitInternal(TwoPhaseCommitSinkFunction.java:373) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.initializeState(TwoPhaseCommitSinkFunction.java:333) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initializeState(FlinkKafkaProducer011.java:867) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.tryRestoreFunction(StreamingFunctionUtils.java:178) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.restoreFunctionState(StreamingFunctionUtils.java:160) at org.apache.flink.streaming.api.operators.AbstractUdfStreamOperator.initializeState(AbstractUdfStreamOperator.java:96) at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(AbstractStreamOperator.java:254) at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:738) at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:289) at org.apache.flink.runtime.taskmanager.Task.run(Task.java:711) at java.lang.Thread.run(Thread.java:748){quote} The reason is, that for {{Semantic.AT_LEAST_ONCE}} the snapshotted state of the {{TwoPhaseCommitFunction}} is of the form "TransactionHolder\{handle=KafkaTransactionState [transactionalId=null, producerId=-1, epoch=-1], transactionStartTime=1537175471175}".
2. **summary:** Restoring a KafkaProducer with Semantic.EXACTLY_ONCE from a savepoint written with Semantic.AT_LEAST_ONCE fails with NPE
   description: If a KafkaProducer with {{Semantic.EXACTLY_ONCE}} is restored from a savepoint written with {{Semantic.AT_LEAST_ONCE}} the job fails on restore with the NPE below. This makes it impossible to upgrade an AT_LEAST_ONCE pipeline to an EXACTL_ONCE pipeline statefully. {quote} java.lang.NullPointerException at java.util.Hashtable.put(Hashtable.java:460) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initTransactionalProducer(FlinkKafkaProducer011.java:955) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:733) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:93) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.recoverAndCommitInternal(TwoPhaseCommitSinkFunction.java:373) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.initializeState(TwoPhaseCommitSinkFunction.java:333) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initializeState(FlinkKafkaProducer011.java:867) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.tryRestoreFunction(StreamingFunctionUtils.java:178) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.restoreFunctionState(StreamingFunctionUtils.java:160) at

org.apache.flink.streaming.api.operators.AbstractUdfStreamOperator.initializeState(AbstractUdfStreamOperator.java:96) at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(AbstractStreamOperator.java:254) at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:738) at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:289) at org.apache.flink.runtime.taskmanager.Task.run(Task.java:711) at java.lang.Thread.run(Thread.java:748){quote} The reason is, that for {{Semantic.AT_LEAST_ONCE}} the snapshotted state of the {{TwoPhaseCommitFunction}} is of the form "TransactionHolder\{handle=KafkaTransactionState [transactionalId=null, producerId=-1, epoch=-1], transactionStartTime=1537175471175}".

3. **summary:** Restoring a KafkaProducer with Semantic.EXACTLY_ONCE from a savepoint written with Semantic.AT_LEAST_ONCE fails with NPE
   description: If a KafkaProducer with {{Semantic.EXACTLY_ONCE}} is restored from a savepoint written with {{Semantic.AT_LEAST_ONCE}} the job fails on restore with the NPE below. This makes it impossible to upgrade an AT_LEAST_ONCE pipeline to an EXACTL_ONCE pipeline statefully. {quote} java.lang.NullPointerException at java.util.Hashtable.put(Hashtable.java:460) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initTransactionalProducer(FlinkKafkaProducer011.java:955) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:733) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.recoverAndCommit(FlinkKafkaProducer011.java:93) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.recoverAndCommitInternal(TwoPhaseCommitSinkFunction.java:373) at org.apache.flink.streaming.api.functions.sink.TwoPhaseCommitSinkFunction.initializeState(TwoPhaseCommitSinkFunction.java:333) at org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.initializeState(FlinkKafkaProducer011.java:867) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.tryRestoreFunction(StreamingFunctionUtils.java:178) at org.apache.flink.streaming.util.functions.StreamingFunctionUtils.restoreFunctionState(StreamingFunctionUtils.java:160) at org.apache.flink.streaming.api.operators.AbstractUdfStreamOperator.initializeState(AbstractUdfStreamOperator.java:96) at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(AbstractStreamOperator.java:254) at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:738) at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:289) at org.apache.flink.runtime.taskmanager.Task.run(Task.java:711) at java.lang.Thread.run(Thread.java:748){quote} The reason is, that for {{Semantic.AT_LEAST_ONCE}} the snapshotted state of the {{TwoPhaseCommitFunction}} is of the form "TransactionHolder\{handle=KafkaTransactionState [transactionalId=null, producerId=-1, epoch=-1], transactionStartTime=1537175471175}".

jira_issues_comments:

1. StefanRRichter opened a new pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010 …ka Producer with existing state ## What is the purpose of the change This PR changes `FlinkKafkaProducer` and `FlinkKafkaProducer011` to support a change of transactional semantics when restoring from existing state, e.g. a savepoint. ## Brief change log - Introduced `KafkaTransactionState#isTransactional` to distinguish which transactional handling should be applied instead of relying on what is currently configured. - Call `initializeUserContext` in `TwoPhaseCommitSinkFunction` for all cases that did not recover a user context. - Consider removing tranactional id from the properties when creating a new producer to deactivate transactional semantics if no longer required. ## Verifying this change Added `FlinkKafkaProducer(11)ITCase#testMigrateFromAtLeastOnceToExactlyOnce` and `#testMigrateFromAtExactlyOnceToAtLeastOnce` ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (no) - If yes, how is the feature documented? (not applicable) ---------- 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
2. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231055211 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java ########## @@ -957,6 +928,11 @@ private void recycleTransactionalProducer(FlinkKafkaProducer<byte[], byte[]> pro return initProducer(registerMetrics); } + private FlinkKafkaProducer<byte[], byte[]> initNonTransactionalProducer(boolean registerMetrics) { + producerConfig.remove("transactional.id"); Review comment: why do you need this change? ---------- 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
3. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231060313 ########## File path: flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java ########## @@ -320,9 +320,9 @@ public void initializeState(FunctionInitializationContext context) throws Except state = context.getOperatorStateStore().getListState(stateDescriptor); + boolean recoveredUserContext = false; Review comment: is this relevant to the bug fix or is this a clean up refactor? ---------- 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
4. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231060083 ########## File path: flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducerITCase.java ########## @@ -163,7 +163,7 @@ public void testRestoreToCheckpointAfterExceedingProducersPool() throws Exceptio } @Test - public void testFlinkKafkaProducer10FailBeforeNotify() throws Exception { + public void testFlinkKafkaProducerFailBeforeNotify() throws Exception { Review comment: Those changes should be in separate commit. ---------- 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
5. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231058042 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); Review comment: also extend the second try block a little bit by completing at least one checkpoint to make sure that committing also works. ---------- 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

6. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231059615 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42, 43, 44 in directly flushed writes from at-least-once + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44), 30_000L); + deleteTestTopic(topic); + } + + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + // restore from snapshot + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records, after potentially lingering transactions + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42 and 43 in committed transactions + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44, 45), 30_000L); Review comment: Could you provide overloaded version of `assertExactlyOnceForTopic` with default value for `long timeoutMillis = 30_000L` as separate commit? I know that it was like that before, but I have only now realised how duplicated (mostly by me) the magic constant `30_000L` is everywhere... ---------- 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

7. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231057070 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); Review comment: start this from `45`. Now it's not obvious whether the result `44` comes from here or from above. ---------- 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

8. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231056814 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42, 43, 44 in directly flushed writes from at-least-once + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44), 30_000L); + deleteTestTopic(topic); + } + + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { Review comment: please deduplicate this code with `testMigrateFromAtLeastOnceToExactlyOnce` ``` private void testMigrateSemantic(Semantic from, Semantic to, String topicName) { // both try blocks } ``` ---------- 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

9. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231061192 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java ########## @@ -957,6 +928,11 @@ private void recycleTransactionalProducer(FlinkKafkaProducer<byte[], byte[]> pro return initProducer(registerMetrics); } + private FlinkKafkaProducer<byte[], byte[]> initNonTransactionalProducer(boolean registerMetrics) { + producerConfig.remove("transactional.id"); Review comment: Because otherwise the the restore of state from the `EXACTLY_ONCE` case will insert this key in the producer config and we will end up having a producer that is configured for transactions, which we don't want. ---------- 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

10. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231067462 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); +

testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42, 43, 44 in directly flushed writes from at-least-once + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44), 30_000L); + deleteTestTopic(topic); + } + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { Review comment: I think that should work, but the expected result also needs to be provided because it is different. ---------- 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

11. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231067562 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); Review comment: can do ---------- 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

12. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231067879 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42, 43, 44 in directly flushed writes from at-least-once + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44), 30_000L); + deleteTestTopic(topic); + } + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + // restore from snapshot + testHarness.initializeState(producerSnapshot); + testHarness.open(); + // write and commit more records, after potentially lingering transactions + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42 and 43 in committed transactions + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44, 45), 30_000L); Review comment: A couple of other things in the test might be somewhat duplicated as well, I can just do a general pass and dedup stuff. ---------- 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

13. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231068106 ########## File path: flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java ########## @@ -320,9 +320,9 @@ public void initializeState(FunctionInitializationContext context) throws Except state = context.getOperatorStateStore().getListState(stateDescriptor); + boolean recoveredUserContext = false; Review comment: This is relevant, there is a semantical difference. Small, but important. ---------- 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

14. StefanRRichter commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231068587 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java ########## @@ -957,6 +928,11 @@ private void recycleTransactionalProducer(FlinkKafkaProducer<byte[], byte[]> pro return initProducer(registerMetrics); } + private FlinkKafkaProducer<byte[], byte[]> initNonTransactionalProducer(boolean registerMetrics) { + producerConfig.remove("transactional.id"); Review comment: Core problem here being that the config is somewhat a shared object that is written to in different attempts of initializing a producer. ---------- 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

15. pnowojski commented on a change in pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#discussion_r231069235 ########## File path: flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java ########## @@ -566,6 +568,76 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, AT_LEAST_ONCE)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { + testHarness.setup(); + // restore from snapshot, all records until here should be persisted + testHarness.initializeState(producerSnapshot); + testHarness.open(); + + // write and commit more records + testHarness.processElement(44, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(45, 9); + } + + //now we should have: + // - records 42, 43, 44 in directly flushed writes from at-least-once + // - aborted transactions with records 44 and 45 + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42,

43, 44), 30_000L); + deleteTestTopic(topic); + } + + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, EXACTLY_ONCE)) { Review comment: you could provide expected result or do the assertion outside of the deduplicated method. I was thinking about the latter one, but maybe the first one is better. Either are fine for me. ---------- 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

16. StefanRRichter commented on issue #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#issuecomment-436235901 Addressed the comments. Please have another look. ---------- 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
17. StefanRRichter commented on issue #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010#issuecomment-436580544 Thanks for the review @pnowojski. I will keep the hotfix commit separated. Merging. ---------- 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
18. asfgit closed pull request #7010: [FLINK-10353][kafka] Support change of transactional semantics in Kaf… URL: https://github.com/apache/flink/pull/7010 This is a PR merged from a forked repository. As GitHub hides the original diff on merge, it is displayed below for the sake of provenance: As this is a foreign pull request (from a fork), the diff is supplied below (as it won't show otherwise due to GitHub magic): diff --git a/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java b/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java index 1cf4acf5406..c7f84c36b6a 100644 --- a/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java +++ b/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011.java @@ -689,7 +689,7 @@ protected KafkaTransactionState beginTransaction() throws FlinkKafka011Exception if (currentTransaction != null && currentTransaction.producer != null) { return new KafkaTransactionState(currentTransaction.producer); } - return new KafkaTransactionState(initProducer(true)); + return new KafkaTransactionState(initNonTransactionalProducer(true)); default: throw new UnsupportedOperationException("Not implemented semantic"); } @@ -712,73 +712,46 @@ protected void preCommit(KafkaTransactionState transaction) throws FlinkKafka011 @Override protected void commit(KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - transaction.producer.commitTransaction(); - recycleTransactionalProducer(transaction.producer); - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + transaction.producer.commitTransaction(); + recycleTransactionalProducer(transaction.producer); } } @Override protected void recoverAndCommit(KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - try (FlinkKafkaProducer<byte[], byte[]> producer = - initTransactionalProducer(transaction.transactionalId, false)) { - producer.resumeTransaction(transaction.producerId, transaction.epoch); - producer.commitTransaction(); - } catch (InvalidTxnStateException | ProducerFencedException ex) { - // That means we have committed this transaction before. - LOG.warn("Encountered error {} while recovering transaction {}. " + - "Presumably this transaction has been already committed before", - ex, transaction); - } - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + try (+ FlinkKafkaProducer<byte[], byte[]> producer = + initTransactionalProducer(transaction.transactionalId, false)) { + producer.resumeTransaction(transaction.producerId, transaction.epoch); + producer.commitTransaction(); + } catch (InvalidTxnStateException | ProducerFencedException ex) { + // That means we have committed this transaction before. + LOG.warn("Encountered error {} while recovering transaction {}. " + + "Presumably this transaction has been already committed before", + ex, transaction); + } } } @Override protected void abort(KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - transaction.producer.abortTransaction(); - recycleTransactionalProducer(transaction.producer); - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + transaction.producer.abortTransaction(); + recycleTransactionalProducer(transaction.producer); } } @Override protected void recoverAndAbort(KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - try (FlinkKafkaProducer<byte[], byte[]> producer = - initTransactionalProducer(transaction.transactionalId, false)) { - producer.initTransactions(); - } - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + try (+ FlinkKafkaProducer<byte[], byte[]> producer = + initTransactionalProducer(transaction.transactionalId, false)) { + producer.initTransactions(); + } } } @@ -905,9 +878,7 @@ private void cleanUpUserContext() { private void resetAvailableTransactionalIdsPool(Collection<String> transactionalIds) { availableTransactionalIds.clear(); - for (String transactionalId : transactionalIds) { - availableTransactionalIds.add(transactionalId); - } + availableTransactionalIds.addAll(transactionalIds); } // ------------------------------------ Utilities ------------------------- @@ -957,6 +928,11 @@ private void recycleTransactionalProducer(FlinkKafkaProducer<byte[], byte[]> pro return initProducer(registerMetrics); } + private FlinkKafkaProducer<byte[], byte[]> initNonTransactionalProducer(boolean registerMetrics) { + producerConfig.remove("transactional.id"); + return initProducer(registerMetrics); + } + private FlinkKafkaProducer<byte[], byte[]> initProducer(boolean registerMetrics) { FlinkKafkaProducer<byte[], byte[]> producer = new FlinkKafkaProducer<>(this.producerConfig); @@ -1075,7 +1051,7 @@ public int compare(PartitionInfo o1, PartitionInfo o2) { } KafkaTransactionState(- String transactionalId, + @Nullable String transactionalId, long producerId, short epoch, FlinkKafkaProducer<byte[], byte[]> producer) { @@ -1085,6 +1061,10 @@ public int compare(PartitionInfo o1, PartitionInfo o2) { this.producer = producer; } + boolean isTransactional() { + return transactionalId != null; + } + @Override public String toString() { return String.format(diff --git a/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/internal/FlinkKafkaProducer.java b/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/internal/FlinkKafkaProducer.java index 8faff38749f..fa672f03ccd 100644 --- a/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/internal/FlinkKafkaProducer.java +++ b/flink-connectors/flink-connector-kafka-0.11/src/main/java/org/apache/flink/streaming/connectors/kafka/internal/FlinkKafkaProducer.java @@ -210,6 +210,7 @@ public void resumeTransaction(long producerId, short epoch) { } } + @Nullable public String getTransactionalId() { return transactionalId; } diff --git a/flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java b/flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java index 57b7e77dc7f..ca75b1a637b 100644 --- a/flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java +++ b/flink-connectors/flink-connector-kafka-0.11/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer011ITCase.java @@ -48,11 +48,13 @@ import java.util.stream.IntStream; import static org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.Semantic; +import static org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.Semantic.AT_LEAST_ONCE; +import static org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011.Semantic.EXACTLY_ONCE; import static org.apache.flink.util.ExceptionUtils.findThrowable; import static org.apache.flink.util.Preconditions.checkState; import static org.hamcrest.Matchers.lessThan; -import static org.junit.Assert.assertFalse; import static org.junit.Assert.assertThat; +import static org.junit.Assert.fail; /** * IT cases for the {@link FlinkKafkaProducer011}. @@ -87,7 +89,7 @@ public void resourceCleanUpNone() throws Exception { @Test public void resourceCleanUpAtLeastOnce() throws Exception { - resourceCleanUp(Semantic.AT_LEAST_ONCE); + resourceCleanUp(AT_LEAST_ONCE); } /** @@ -153,7 +155,7 @@ public void testRestoreToCheckpointAfterExceedingProducersPool() throws Exceptio testHarness2.open(); } - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42), 30_000L); + +

assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42)); deleteTestTopic(topic); } catch (Exception ex) { @@ -183,7 +185,7 @@ public void testFlinkKafkaProducer011FailBeforeNotify() throws Exception { try { testHarness.processElement(44, 4); testHarness.snapshot(2, 5); - assertFalse(true); + fail(); } catch (Exception ex) { // expected @@ -201,7 +203,7 @@ public void testFlinkKafkaProducer011FailBeforeNotify() throws Exception { testHarness.initializeState(snapshot); testHarness.close(); - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @@ -216,7 +218,7 @@ public void testFlinkKafkaProducer011FailTransactionCoordinatorBeforeNotify() th topic, integerKeyedSerializationSchema, properties, - Semantic.EXACTLY_ONCE); + EXACTLY_ONCE); OneInputStreamOperatorTestHarness<Integer, Object> testHarness1 = new OneInputStreamOperatorTestHarness<>(new StreamSink<>(kafkaProducer), @@ -250,7 +252,7 @@ public void testFlinkKafkaProducer011FailTransactionCoordinatorBeforeNotify() th testHarness2.open(); } - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @@ -296,7 +298,7 @@ public void testFailBeforeNotifyAndResumeWorkAfterwards() throws Exception { // - aborted transactions with records 44 and 45 // - committed transaction with record 46 // - pending transaction with record 47 - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 46), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 46)); testHarness.close(); deleteTestTopic(topic); @@ -345,7 +347,7 @@ public void testFailAndRecoverSameCheckpointTwice() throws Exception { //now we should have: // - records 42 and 43 in committed transactions // - aborted transactions with records 44 and 45 - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @@ -367,7 +369,7 @@ public void testScaleDownBeforeFirstCheckpoint() throws Exception { preScaleDownParallelism, preScaleDownParallelism, subtaskIndex, - Semantic.EXACTLY_ONCE); + EXACTLY_ONCE); preScaleDownOperator.setup(); preScaleDownOperator.open(); @@ -382,7 +384,7 @@ public void testScaleDownBeforeFirstCheckpoint() throws Exception { // there might not be any close) // After previous failure simulate restarting application with smaller parallelism - OneInputStreamOperatorTestHarness<Integer, Object> postScaleDownOperator1 = createTestHarness(topic, 1, 1, 0, Semantic.EXACTLY_ONCE); + OneInputStreamOperatorTestHarness<Integer, Object> postScaleDownOperator1 = createTestHarness(topic, 1, 1, 0, EXACTLY_ONCE); postScaleDownOperator1.setup(); postScaleDownOperator1.open(); @@ -397,7 +399,7 @@ public void testScaleDownBeforeFirstCheckpoint() throws Exception { // - records 42, 43, 44 and 45 in aborted transactions // - committed transaction with record 46 // - pending transaction with record 47 - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(46), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(46)); postScaleDownOperator1.close(); // ignore ProducerFencedExceptions, because postScaleDownOperator1 could reuse transactional ids. @@ -466,8 +468,7 @@ public void testScaleUpAfterScalingDown() throws Exception { createProperties(), topic, 0, - IntStream.range(0, parallelism1 + parallelism2 + parallelism3).boxed().collect(Collectors.toList()), - 30_000L); + IntStream.range(0, parallelism1 + parallelism2 + parallelism3).boxed().collect(Collectors.toList())); deleteTestTopic(topic); } @@ -483,7 +484,7 @@ public void testScaleUpAfterScalingDown() throws Exception { for (int subtaskIndex = 0; subtaskIndex < parallelism; subtaskIndex++) { OneInputStreamOperatorTestHarness<Integer, Object> testHarness = - createTestHarness(topic, maxParallelism, parallelism, subtaskIndex, Semantic.EXACTLY_ONCE); + createTestHarness(topic, maxParallelism, parallelism, subtaskIndex, EXACTLY_ONCE); testHarnesses.add(testHarness); testHarness.setup(); @@ -539,7 +540,7 @@ public void testRecoverCommittedTransaction() throws Exception { testHarness.initializeState(checkpoint0); // recover state 0 - producerA recover and commit txn 0 testHarness.close(); - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42)); deleteTestTopic(topic); } @@ -566,6 +567,50 @@ public void testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + testRecoverWithChangeSemantics(topic, AT_LEAST_ONCE, EXACTLY_ONCE); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44, 45)); + deleteTestTopic(topic); + } + + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + testRecoverWithChangeSemantics(topic, EXACTLY_ONCE, AT_LEAST_ONCE); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 45, 46, 47)); + deleteTestTopic(topic); + } + + private void testRecoverWithChangeSemantics(+ String topic, + Semantic fromSemantic, + Semantic toSemantic) throws Exception { + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, fromSemantic)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, toSemantic)) { + testHarness.setup(); + testHarness.initializeState(producerSnapshot); + testHarness.open(); + testHarness.processElement(45, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(46, 9); + testHarness.notifyOfCompletedCheckpoint(2); + testHarness.processElement(47, 9); + } + } + // shut down a Kafka broker private void failBroker(int brokerId) { KafkaServer toShutDown = null; @@ -604,7 +649,13 @@ private void closeIgnoringProducerFenced(AutoCloseable autoCloseable) throws Exc } private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(String topic) throws Exception { - return createTestHarness(topic, 1, 1, 0, Semantic.EXACTLY_ONCE); + return createTestHarness(topic, Semantic.EXACTLY_ONCE); + } + + private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(+ String topic, + Semantic semantic) throws Exception { + return createTestHarness(topic, 1, 1, 0, semantic); } private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(diff --git a/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaProducerTestBase.java b/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaProducerTestBase.java index 69eb94a2e8f..cf68c3de776 100644 --- a/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaProducerTestBase.java +++ b/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaProducerTestBase.java @@ -329,7 +329,7 @@ protected void testExactlyOnce(boolean regularSink, int sinksCount) throws Excep } TypeInformationSerializationSchema<Integer> schema = new TypeInformationSerializationSchema<>(BasicTypeInfo.INT_TYPE_INFO, new ExecutionConfig()); - KeyedSerializationSchema<Integer> keyedSerializationSchema = new KeyedSerializationSchemaWrapper(schema); + KeyedSerializationSchema<Integer> keyedSerializationSchema = new KeyedSerializationSchemaWrapper<>(schema); StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment(); env.enableCheckpointing(500); diff --git a/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaTestBase.java b/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaTestBase.java index 1a20d7e82ad..2d4ac43fa9a 100644 --- a/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaTestBase.java +++ b/flink-connectors/flink-connector-kafka-base/src/test/java/org/apache/flink/streaming/connectors/kafka/KafkaTestBase.java @@ -241,6 +241,14 @@ protected void assertAtLeastOnceForTopic(fail(String.format("Expected to contain all of: <%s>, but was: <%s>", expectedElements, actualElements)); } + protected void assertExactlyOnceForTopic(+ Properties properties, + String topic, + int partition, + List<Integer> expectedElements) { + assertExactlyOnceForTopic(properties, topic, partition, expectedElements, 30_000L); + } + /** * We manually handle the timeout instead of using JUnit's timeout to return failure instead of timeout error. * After timeout we assume that there are missing records and there is a bug, not that the test has run out of time. @@ -250,7 +258,7 @@ protected void assertExactlyOnceForTopic(String topic, int partition, List<Integer> expectedElements, - long timeoutMillis) throws Exception { + long timeoutMillis) { long startMillis = System.currentTimeMillis(); List<Integer> actualElements = new ArrayList<>(); diff --git a/flink-connectors/flink-connector-kafka/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer.java b/flink-connectors/flink-connector-kafka/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer.java index 0ac2f906bde..df1a4b5727f 100644 --- a/flink-connectors/flink-connector-kafka/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer.java +++ b/flink-

connectors/flink-connector-kafka/src/main/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducer.java @@ -691,7 +691,7 @@ public void close() throws FlinkKafkaException { if (currentTransaction != null && currentTransaction.producer != null) { return new FlinkKafkaProducer.KafkaTransactionState(currentTransaction.producer); } - return new FlinkKafkaProducer.KafkaTransactionState(initProducer(true)); + return new FlinkKafkaProducer.KafkaTransactionState(initNonTransactionalProducer(true)); default: throw new UnsupportedOperationException("Not implemented semantic"); } @@ -714,73 +714,46 @@ protected void preCommit(FlinkKafkaProducer.KafkaTransactionState transaction) t @Override protected void commit(FlinkKafkaProducer.KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - transaction.producer.commitTransaction(); - recycleTransactionalProducer(transaction.producer); - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + transaction.producer.commitTransaction(); + recycleTransactionalProducer(transaction.producer); } } @Override protected void recoverAndCommit(FlinkKafkaProducer.KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - try (FlinkKafkaInternalProducer<byte[], byte[]> producer = + if (transaction.isTransactional()) { + try (+ FlinkKafkaInternalProducer<byte[], byte[]> producer = initTransactionalProducer(transaction.transactionalId, false)) { - producer.resumeTransaction(transaction.producerId, transaction.epoch); - producer.commitTransaction(); - } catch (InvalidTxnStateException | ProducerFencedException ex) { - // That means we have committed this transaction before. - LOG.warn("Encountered error {} while recovering transaction {}. " + - "Presumably this transaction has been already committed before", - ex, transaction); - } - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + producer.resumeTransaction(transaction.producerId, transaction.epoch); + producer.commitTransaction(); + } catch (InvalidTxnStateException | ProducerFencedException ex) { + // That means we have committed this transaction before. + LOG.warn("Encountered error {} while recovering transaction {}. " + + "Presumably this transaction has been already committed before", + ex, transaction); + } } } @Override protected void abort(FlinkKafkaProducer.KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - transaction.producer.abortTransaction(); - recycleTransactionalProducer(transaction.producer); - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + if (transaction.isTransactional()) { + transaction.producer.abortTransaction(); + recycleTransactionalProducer(transaction.producer); } } @Override protected void recoverAndAbort(FlinkKafkaProducer.KafkaTransactionState transaction) { - switch (semantic) { - case EXACTLY_ONCE: - try (FlinkKafkaInternalProducer<byte[], byte[]> producer = + if (transaction.isTransactional()) { + try (+ FlinkKafkaInternalProducer<byte[], byte[]> producer = initTransactionalProducer(transaction.transactionalId, false)) { - producer.initTransactions(); - } - break; - case AT_LEAST_ONCE: - case NONE: - break; - default: - throw new UnsupportedOperationException("Not implemented semantic"); + producer.initTransactions(); } } @@ -895,7 +868,7 @@ protected void finishRecoveringContext() { LOG.info("Recovered transactionalIds {}", getUserContext().get().transactionalIds); } - protected FlinkKafkaInternalProducer createProducer() { + protected FlinkKafkaInternalProducer<byte[], byte[]> createProducer() { return new FlinkKafkaInternalProducer<>(this.producerConfig); } @@ -911,9 +884,7 @@ private void cleanUpUserContext() { private void resetAvailableTransactionalIdsPool(Collection<String> transactionalIds) { availableTransactionalIds.clear(); - for (String transactionalId : transactionalIds) { - availableTransactionalIds.add(transactionalId); - } + availableTransactionalIds.addAll(transactionalIds); } // ---------------------------- Utilities ---------------------------- @@ -963,6 +934,11 @@ private void recycleTransactionalProducer(FlinkKafkaInternalProducer<byte[], byt return initProducer(registerMetrics); } + private FlinkKafkaInternalProducer<byte[], byte[]> initNonTransactionalProducer(boolean registerMetrics) { + producerConfig.remove("transactional.id"); + return initProducer(registerMetrics); + } + private FlinkKafkaInternalProducer<byte[], byte[]> initProducer(boolean registerMetrics) { FlinkKafkaInternalProducer<byte[], byte[]> producer = createProducer(); @@ -1081,7 +1057,7 @@ public int compare(PartitionInfo o1, PartitionInfo o2) { } KafkaTransactionState(- String transactionalId, + @Nullable String transactionalId, long producerId, short epoch, FlinkKafkaInternalProducer<byte[], byte[]> producer) { @@ -1091,6 +1067,10 @@ public int compare(PartitionInfo o1, PartitionInfo o2) { this.producer = producer; } + boolean isTransactional() { + return transactionalId != null; + } + @Override public String toString() { return String.format(diff --git a/flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducerITCase.java b/flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducerITCase.java index b14b3e2524b..29f157ffb4f 100644 --- a/flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducerITCase.java +++ b/flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/FlinkKafkaProducerITCase.java @@ -49,8 +49,8 @@ import static org.apache.flink.util.ExceptionUtils.findThrowable; import static org.apache.flink.util.Preconditions.checkState; import static org.hamcrest.Matchers.lessThan; -import static org.junit.Assert.assertFalse; import static org.junit.Assert.assertThat; +import static org.junit.Assert.fail; /** * IT cases for the {@link FlinkKafkaProducer}. @@ -151,7 +151,7 @@ public void testRestoreToCheckpointAfterExceedingProducersPool() throws Exceptio testHarness2.open(); } - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42)); deleteTestTopic(topic); } catch (Exception ex) { @@ -163,7 +163,7 @@ public void testRestoreToCheckpointAfterExceedingProducersPool() throws Exceptio } @Test - public void testFlinkKafkaProducer10FailBeforeNotify() throws Exception { + public void testFlinkKafkaProducerFailBeforeNotify() throws Exception { String topic = "flink-kafka-producer-fail-before-notify"; OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic); @@ -181,7 +181,7 @@ public void testFlinkKafkaProducer10FailBeforeNotify() throws Exception { testHarness.processElement(44, 4); testHarness.snapshot(2, 5); - assertFalse(true); + fail(); } catch (Exception ex) { // expected @@ -199,13 +199,13 @@ public void testFlinkKafkaProducer10FailBeforeNotify() throws Exception { testHarness.initializeState(snapshot); testHarness.close(); - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @Test - public void testFlinkKafkaProducer10FailTransactionCoordinatorBeforeNotify() throws Exception { + public void testFlinkKafkaProducerFailTransactionCoordinatorBeforeNotify() throws Exception { String topic = "flink-kafka-producer-fail-transaction-coordinator-before-notify"; Properties properties = createProperties(); @@ -248,7 +248,7 @@ public void testFlinkKafkaProducer10FailTransactionCoordinatorBeforeNotify() thr testHarness2.open(); } - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @@ -294,7 +294,7 @@ public void testFailBeforeNotifyAndResumeWorkAfterwards() throws Exception { // - committed transaction with record 46 // - pending transaction with record 47 assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 46), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 46)); testHarness.close(); deleteTestTopic(topic); @@ -343,7 +343,7 @@ public void testFailAndRecoverSameCheckpointTwice() throws Exception { // - aborted transactions with records 44 and 45 assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43)); deleteTestTopic(topic); } @@ -395,7 +395,7 @@ public void testScaleDownBeforeFirstCheckpoint() throws Exception { // - pending transaction with record 47 assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(46), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(46)); postScaleDownOperator1.close(); // ignore ProducerFencedExceptions, because postScaleDownOperator1 could reuse transactional ids. @@ -464,8 +464,7 @@ public void testScaleUpAfterScalingDown() throws Exception { createProperties(), topic, 0, - IntStream.range(0, parallelism1 + parallelism2 + parallelism3).boxed().collect(Collectors.toList()), - 30_000L); + IntStream.range(0, parallelism1 + parallelism2 + parallelism3).boxed().collect(Collectors.toList())); deleteTestTopic(topic); } @@ -537,7 +536,7 @@ public void testRecoverCommittedTransaction() throws Exception { testHarness.close(); - assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42), 30_000L); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42)); deleteTestTopic(topic); } @@ -564,6 +563,52 @@ public void testRecoverCommittedTransaction() throws Exception { public void

testRunOutOfProducersInThePool() throws Exception { deleteTestTopic(topic); } + @Test + public void testMigrateFromAtLeastOnceToExactlyOnce() throws Exception { + String topic = "testMigrateFromAtLeastOnceToExactlyOnce"; + testRecoverWithChangeSemantics(topic, FlinkKafkaProducer.Semantic.AT_LEAST_ONCE, FlinkKafkaProducer.Semantic.EXACTLY_ONCE); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 44, 45)); + deleteTestTopic(topic); + } + + @Test + public void testMigrateFromAtExactlyOnceToAtLeastOnce() throws Exception { + String topic = "testMigrateFromExactlyOnceToAtLeastOnce"; + testRecoverWithChangeSemantics(topic, FlinkKafkaProducer.Semantic.EXACTLY_ONCE, FlinkKafkaProducer.Semantic.AT_LEAST_ONCE); + assertExactlyOnceForTopic(createProperties(), topic, 0, Arrays.asList(42, 43, 45, 46, 47)); + deleteTestTopic(topic); + } + + private void testRecoverWithChangeSemantics(+ String topic, + FlinkKafkaProducer.Semantic fromSemantic, + FlinkKafkaProducer.Semantic toSemantic) throws Exception { + OperatorSubtaskState producerSnapshot; + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, fromSemantic)) { + testHarness.setup(); + testHarness.open(); + testHarness.processElement(42, 0); + testHarness.snapshot(0, 1); + testHarness.processElement(43, 2); + testHarness.notifyOfCompletedCheckpoint(0); + producerSnapshot = testHarness.snapshot(1, 3); + testHarness.processElement(44, 4); + } + + try (OneInputStreamOperatorTestHarness<Integer, Object> testHarness = createTestHarness(topic, toSemantic)) { + testHarness.setup(); + testHarness.initializeState(producerSnapshot); + testHarness.open(); + testHarness.processElement(45, 7); + testHarness.snapshot(2, 8); + testHarness.processElement(46, 9); + testHarness.notifyOfCompletedCheckpoint(2); + testHarness.processElement(47, 9); + } + } + + // -- + // shut down a Kafka broker private void failBroker(int brokerId) { KafkaServer toShutDown = null; @@ -602,7 +647,13 @@ private void closeIgnoringProducerFenced(AutoCloseable autoCloseable) throws Exc } private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(String topic) throws Exception { - return createTestHarness(topic, 1, 1, 0, FlinkKafkaProducer.Semantic.EXACTLY_ONCE); + return createTestHarness(topic, FlinkKafkaProducer.Semantic.EXACTLY_ONCE); + } + + private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(+ String topic, + FlinkKafkaProducer.Semantic semantic) throws Exception { + return createTestHarness(topic, 1, 1, 0, semantic); } private OneInputStreamOperatorTestHarness<Integer, Object> createTestHarness(diff --git a/flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java b/flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java index 03f12b585ae..d2735d566ee 100644 --- a/flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java +++ b/flink-streaming-java/src/main/java/org/apache/flink/streaming/api/functions/sink/TwoPhaseCommitSinkFunction.java @@ -320,9 +320,9 @@ public void initializeState(FunctionInitializationContext context) throws Except state = context.getOperatorStateStore().getListState(stateDescriptor); + boolean recoveredUserContext = false; if (context.isRestored()) { LOG.info("{} - restoring state", name()); - for (State<TXN, CONTEXT> operatorState : state.get()) { userContext = operatorState.getContext(); List<TransactionHolder<TXN>> recoveredTransactions = operatorState.getPendingCommitTransactions(); @@ -337,11 +337,13 @@ public void initializeState(FunctionInitializationContext context) throws Except if (userContext.isPresent()) { finishRecoveringContext(); + recoveredUserContext = true; } } } + // if in restore we didn't get any userContext or we are initializing from scratch - if (userContext == null) { + if (!recoveredUserContext) { LOG.info("{} - no state to restore", name()); userContext = initializeUserContext(); ---------- 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

19. Merged in: master: 85f895b083 release-1.7: 090119ac96 release-1.6: b609ede4b4