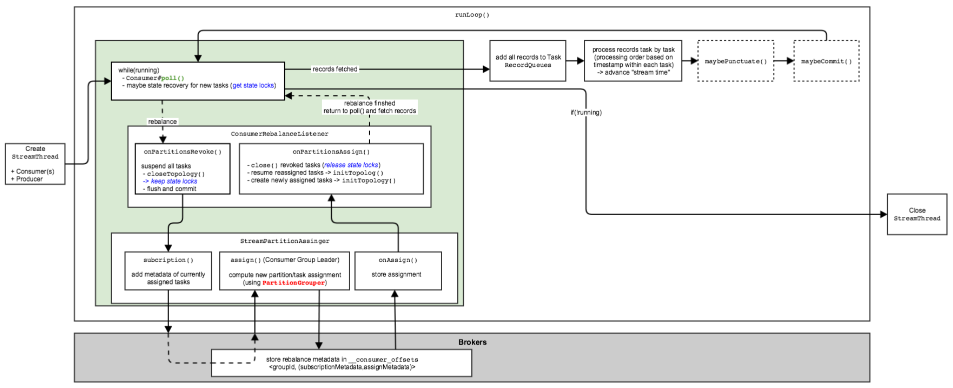
Kafka Streams Architecture

Warning

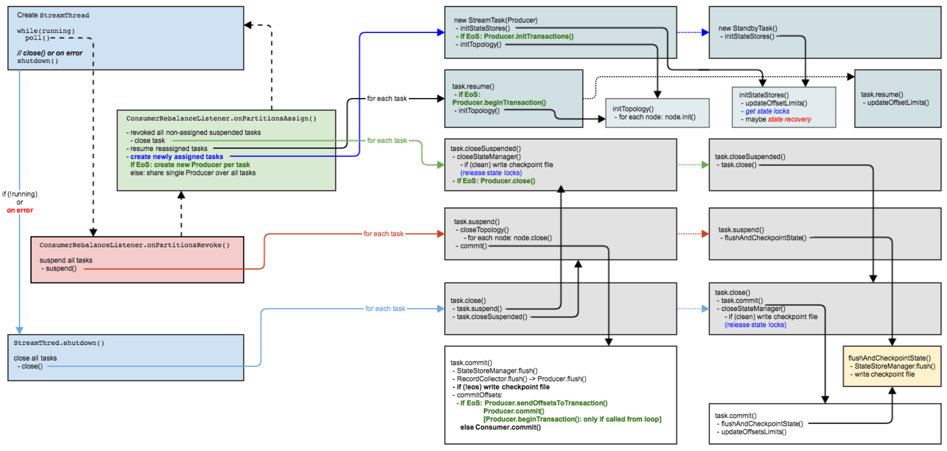
We try to keep this doc up to date, however, as it describes internals that might change at any point in time, there is no guarantee that this doc reflects the latest state of the code base.

Lifecycle of a `StreamThread`



runLoop()
Create StreamThread
+ Consumer(s)
+ Producer
while(running)
- Consumer#poll()
- maybe state recovery for new tasks (get state locks)
records fetched
add all records to Task RecordQueues
process records task by task (processing order based on timestamp within each task) -> advance "stream time"
maybePunctuate()
maybeCommit()
if(!running)
Close StreamThread
rebalance
rebalance finshed
return to poll() and fetch records
ConsumerRebalanceListener
onPartitionsRevoke()
suspend all tasks
- closeTopology()
-> keep state locks
- flush and commit
onPartitionsAssign()
- close() revoked tasks (release state locks)
- resume reassigned tasks -> initTopolog()
- create newly assigned tasks -> initTopology()
StreamPartitionAssinger
subcription()
add metadata of currently assigned tasks
assign() (Consumer Group Leader)
compute new partition/task assignment (using PartitionGrouper)
onAssign()
store assignment
Brokers
store rebalance metadata in __consumer_offsets
<groupId, (subscriptionMetadata,assignMetadata)>

Lifecycle of a `StreamTask` and `StandbyTask`



Exception Handling

A Kafka Streams client need to handle multiple different types of exceptions. We try to summarize what kind of exceptions are there, and how Kafka Streams should handle those. In general, Kafka Streams should be resilient to exceptions and keep processing even if some internal exceptions occur.

Types of Exceptions:

There are different categories how exceptions can be categoriezed.

First, we can distinguish between recoverable and fatal exceptions. Recoverable exception should be handled internally and never bubble out to the user. For fatal exceptions, Kafka Streams is doomed to fail and cannot start/continue to process data.

Related to this are retriable exception. While retriable exception are recoverable in general, it might happen that the (configurable) retry counter is exceeded; for this case, we end up with an fatal exception.

The second category are "external" vs "internal" exception. By "external" we refer to any exception that could be returned by the brokers. "Internal" exceptions are those that are raised locally.

For "external" exceptions, we need to consider `KafkaConsumer`, `KafkaProducer`, and `KafkaAdmintClient`. For internal exceptions, we have for example (de)serialization, state store, and user code exceptions as well as any other exception Kafka Streams raises itself (e.g., configuration exceptions).

Last but not least, we distinguish between exception that should never occur. If those exception do really occur, they indicate a bug and thus all those exception are fatal. All regular Java exception (eg, NullPointerException) are in this category.

Coding implications:

- We should never try to handle any fatal exceptions but clean up and shutdown
  - We should catch all those exceptions for clean up only and rethrow unmodified (they will eventually bubble out of the thread and trigger uncaught exception hander if one is registered)
  - We should only log those exception (with `ERROR` level) once at the thread level before they bubble out of the thread to avoid duplicate logging
- We need to do fine grained exception handling, ie, catch exceptions individually instead of coarse grained and react accordingly
- All methods should have complete JavaDocs about exception they might throw
- All exception classes must have strictly defined semantics that are documented in their JavaDocs
- In *runtime* code, we should never throw any regular Java excepiton (except it's fatal) but define our own exceptions if required (this allows us to destinguish between bugs and our own exceptions)
- We should catch, wrap, and rethrow exceptions each time we can add important information to it that helps users and us to figure out the root cause of what when wrong

To be discussed:

- How to handle `Throwable ?`
  - Should we try to catch-and-rethrow in order to clean up?
    - `Throwable` is fatal, so clean up might fail anyway?
    - Furthermore, should we assume that the whole JVM is dying anyway?
  - Should we be harsh and call `System.exit` (note, we are a library – but maybe we are "special" enough to justify this?
    - Note, if a thread dies without clean up, but other threads are still running fine, we might end up in a deadlock as locks are not released
    - Could also be configurable
    - Could also be a hybrid: try to clean up on `Throwable` but call `System.exit` if clean up fails (as we would end up in a deadlock anyway – maybe only if running with more than one thread?)
  - Should we force users to provide uncaught exception handler via `KafkaStreams` constructor to make sure they get notified about dying streams?
- Restructure exception class hierarchy:
  - Remove all sub-classed of `StreamsException` from public API (we only hand out this one to the user)
  - A `StreamsException` inidicates a fatal error (we could sub-class `StreamsException` with more detailed fatal errors if required – but don't think this is necessary)
  - We sub-class `StreamsException` with (an abstract?) `RecoverableStreamsException` in internal package for any internal exception that should be handled by Streams and never bubble out
    - As an alternative (that I would prefer) we could introduce this as an independet and *checked* exception instead of inheriting from `StreamsException` (this forces us to declare and handle those exceptions in our code and makes it hart do miss – otherwise, one might bubble out due to a bug
  - We sub-class inidividual recoverable exceptions in a fine grained manner from `RecoverableStreamsException` for individual errors
  - We can further group all retriable exceptions by sub-classing them from `abstract RetriableStreamsException extends RecoverableStreamsException` – the more details/categories the better?

	KafkaConsumer	**KafkaProducer**	**StreamsKafakClient**	**AdminClient**	**Streams API**

fatal (should never occur)	local: - IllegalArgumentExcetpion - IllegalStateException - WakeupException - InterruptExcetpion remote: - UnknownServerException	local: - IllegalArgumentExcetpion - IllegalStateException - WakeupException - InterruptExcetpion remote: - UnknownServerException - OffsetMetadataTooLarge - SerializationException (we use <`byte[],byte[]`> as types)	local: - IllegalArgumentExcetpion - IllegalStateException - WakeupException - InterruptExcetpion remote: - UnknownServerException - InvalidTopicException	local: - IllegalArgumentExcetpion - IllegalStateException - WakeupException - InterruptExcetpion remote: - UnknownServerException - InvalidTopicExcetpion	local:
fatal	local: - ConfigException remote: - AuthorizationException (including all subclasses) - AuthenticationException (inlcuding all subclasses) - SecurityDisabledException - InvalidTopicException	local: - ConfigException remote: - AuthorizationException (including all subclasses) - AuthenticationException (inlcuding all subclasses) - SecurityDisabledExcetpion - InvalidTopicException - UnkownTopicOrPartitionsException (retyable? refresh metadata?) - RecordBatchTooLargeException - RecordTooLargeException	local: - ConfigException remote: - AuthorizationException (including all subclasses) - AuthenticationException (inlcuding all subclasses) - SecurityDisabledExcetpion	local: - ConfigException remote: - AuthorizationException (including all subclasses) - AuthenticationException (inlcuding all subclasses) - SecurityDisabledExcetpion	local: - ConfigException - SerializationException
retriable	local: remote: - InvalidOffsetException (OffsetOutOfRangeException, NoOffsetForPartitionsException) - CommitFailedException - TimeoutException - QuotaViolationException?	local: remote: - CorruptedRecordException - NotEnoughReplicasAfterAppendException - OffsetOutOfRangeException (when can producer get this?) - TimeoutException - QuotaViolationException? - BufferExhausedException (verify)	local: remote:	local: remote:	local:
recoverable	local: remote:	local: remote: - ProducerFencedException	local: remote:	local: remote:	local: - LockException

Having a look at all `KafkaException` there are some exception we need to double check if they could bubble out any client (or maybe we should not care, an treat all of them as fatal/remote exceptions).

-> DataException, SchemaBuilderExcetpion, SchemaProjectorException, RequestTargetException, NotAssignedException, IllegalWorkerStateException, ConnectRestException, BadRequestException, AlreadyExistsException (might be possible to occur, or only TopicExistsException), NotFoundException, ApiException, InvalidTimestampException, InvalidGroupException, InvalidReplicationFactorException (might be possible, but inidcate bug), o.a.k.common.erros.InvalidOffsetExcetpion and o.a.k.common.errors.OffsetOutOfRangeException (*side note: do those need cleanup – seems to be duplicates?*), ReplicaNotAvailalbeException, UnknowServerException, OperationNotAttempedException, PolicyViolationException, InvalidConfigurationException, InvalidFetchSizeException, InvalidReplicaAssignmentException, InconsistentGroupProtocolException, ReblanceInProgressException, LogDirNotFoundException, BrokerNotAvailableException, InvalidOffsetCommitSizeException, InvalidTxnTimeoutException, InvalidPartitionsException, TopicExistsException (cf. AlreadyExistException), InvalidTxnStateException,, UnsupportedForMessageFormatException, InvalidSessionTimeoutException, InvalidRequestException, IllegalGenerationException, InvalidRequiredAckException,

-> RetryableException, CoordinatorNotAvailalbeException, RetryableCommitException, DuplicateSequenceNumberException, NotEnoughReplicasException, NotEnoughReplicasAfterAppendException, InvalidRecordException, DisconnectException, InvalidMetaDataException (NotLeaderForPartitionException, NoAvailableBrokersException, UnkonwTopicOrPartitionException, KafkaStoreException, LeaderNotAvailalbeException), GroupCoordinatorNotAvailableException

Should never happen:

Handled by client (consumer, producer, admin) internally and should never bubble out of a client: (verify)

 - ConnectionException, RebalanceNeededException, InvalidPidMappingException, ConcurrentTransactionException, NotLeaderException, TransactionalCoordinatorFencedException, ControllerMovedException, UnkownMemberIdException, OutOfOrderSequenceException, CoordinatorLoadInProgressException, GroupLoadInProgressException, NotControllerException, NotCoordinatorException, NotCoordinatorForGroupException, StaleMetadataException, NetworkException,