KIP-304: Connect runtime mode improvements for container platforms

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA: TBD

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Currently Kafka Connect can be run in either standalone or distributed modes, the latter being recommended for production usage. However distributed mode and the concept of running a set of connectors in a shared cluster of workers has a lot of drawbacks, just to name a few:

- deploying/modifying connectors can cause rebalancing and restarts of other connectors, introducing delays/spikes in completely unrelated pipelines
- in extreme cases connector crashes can bring down other connectors running on the same worker
- upgrading is unnecessarily difficult, e.g. if one wants to upgrade a JAR for a certain connector, one has to restart the whole cluster, affecting all running connectors
- no ability to control resources per connector, one can run into situations where a misbehaving connector can cause OOM errors on the worker once again affecting co-located connectors
- configuring Kafka consumers/producers per connector is currently not possible (thought proposed in KIP-296)
- no ability to configure different connectors with different authentication credentials, as currently all of them share the same client configuration
- connector deployment is performed via a REST API which usually requires a customized deployment workflow, the situation is much worse if one also needs to deploy new JARs, as mentioned above

In general there seems to be a natural trend to isolate connectors as much as possible, as seen with KIP-75, KIP-296 etc.

All of the above problems can be solved by running each connector as a separate instance inside a container on a platform like Mesos or Kubernetes: each container gets a separate lifecycle, separate configuration, resources can be controlled individually and the deployment workflow is standardized -- all the same reasons why Kafka Streams is "just a library" and does not introduce any concepts of "clusters".

The standalone mode is a good candidate to run individual connectors per container, as the framework introduces little overhead and connectors can be configured statically, avoiding any interaction with the API when deploying. However currently it only supports file offset storage and requires persistence, which is not desired in container workloads. Also, the exposed REST API allows one to pause/modify/delete the connector, which might not be desirable.

Connectors that need to be scaled to multiple instances due to high data volumes could be run in distributed mode, however this complicates deployment as currently it is not possible to pre-configure connectors statically.

Public Interfaces

Offset storage could be made configurable in standalone mode using a new configuration property `offset.storage.store` with possible values of `file`, backed by `FileOffsetBackingStore` and `kafka`, backed by `KafkaBackingStore`. When using the `kafka` store the following configuration keys would be used additionally (types/values/defaults exactly the same as in distributed mode):

- `offset.storage.kafka.topic`
- `offset.storage.kafka.replication.factor`
- `offset.storage.kafka.partitions`

This would make it symmetric with the existing configuration key `offset.storage.file.filename` thus introducing a naming convention of `offset.storage.<store>.<keys*>`.

In distributed mode these would be synonyms to the current keys `offset.storage.topic`, `offset.storage.replication.factor` and `offset.storage.partitions`.

An boolean configuration key `rest.modify.enable` if set to `false` would disable all `POST/PUT/DELETE` methods (possibly with the exception of `PUT */validate`). The default value `false` would retain the current behavior.

The distributed mode CLI utility would accept connector configurations via command line arguments just like in standalone mode.

Proposed Changes

Offset storage configuration can be implemented straightforwardly by simply reusing the already existing `KafkaBackingStore` in the standalone mode CLI class.

The distributed mode CLI class would be modified to accept extra command line arguments with connector properties files, similarly like in standalone mode.

The REST API would return HTTP error code 403 Forbidden for all POST/PUT/DELETE methods when run in read-only mode.

Finally, the Connect documentation should provide best practices for running isolated connectors on container platforms such as Mesos or Kubernetes. In particular, for most cases one would use the official Docker image to run Connect in standalone mode, configure it by setting

- `offset.storage.store=kafka`
- `rest.modify.enable=false`

and placing a particular `connector.properties` file inside the image.

Distributed mode would be recommended only when dealing with high volumes of data and the need to scale arises. Similarly as with standalone mode, the image would be pre-configured with a particular `connector.properties` file and the REST API would be run in read-only mode acting as a health-check and diagnostic tool only, leaving all deployment and lifecycle concerns up to the container platform.

Compatibility, Deprecation, and Migration Plan

All proposed changes would be introduced in a backwards compatible way.

The following distributed mode configuration keys would become deprecated in favor of new ones:

- `offset.storage.topic` deprecated in favor of `offset.storage.kafka.topic`
- `offset.storage.replication.factor` deprecated in favor of `offset.storage.kafka.replication.factor`
- `offset.storage.partitions` deprecated in favor of `offset.storage.kafka.partitions`

Both names could be kept as synonyms during some migration period.

Rejected Alternatives

None so far.