KIP-921: OpenJDK CRaC support

Status

Current state: Under discussion

Discussion thread: https://lists.apache.org/thread/5k8vlj06o5973h2zm5bfxbd23gml5f2t

JIRA: TBD

Motivation

OpenJDK CRaC intends to provide a way to checkpoint (snapshot) and persist a running Java application and later restore it, possibly on a different computer. This can be used to significantly speed up the boot process (from seconds or minutes to tens of milliseconds), live replication or migration of the heated up application.

This is not entirely transparent to the application; the application can register for notification when this is happening, and sometime has to assist with that to prevent unexpected state after restore - e.g. close network connections and files. CRaC is not integrated yet into the mainline JDK; JEP is being prepared, and users are welcome to try out custom builds. However even when this gets into JDK users won't jump onto the latest release immediately; therefore a facade package org.crac delegates to the implementation, if it is present in the running JDK, or provides a no-op implementation.

The goal of this KIP is to create an epic for support of CRaC in Kafka, allowing new and existing applications that use Kafka to be subject to checkpoint and restore, ideally without any code changes. It does not have to include all components - for some there might not be much benefit - especially standalone ones like the broker. Therefore at least the initial focus would be on client.

Public Interfaces

Ideally there would be no changes in the API; it is possible that some methods will leak, though. This might be avoided using intermediate interfaces if needed (e.g. places in implementation that now use the public interface `Foo` implemented by `FooImpl` would be changed to use new intermediate interface `FooInternal` which `FooImpl` would implement). Alternatively the new methods might be introduced with noop default implementation.

Proposed Changes

With or without the implementation, the support for CRaC in the application should be designed to have a minimal impact on performance (few extra objects, some volatile reads...). On the other hand the checkpoint operation itself can be non-trivial in this matter. Therefore the main consideration should be about the maintenance costs - keeping a small JAR in dependencies and some extra code in networking and persistence.

The exact scope is yet to be determined. The first PR modifies the `o.a.k.clients.producer.internal.Sender` but some interfaces in `o.a.k.clients` are modified, too. The consumer side would be the next one.

Compatibility, Deprecation, and Migration Plan

- There should be no impact on existing code
- *If we are changing behavior how will we phase out the older behavior?* Only the behaviour during C/R - something that was not performed beforehand - will be changed.
- *If we need special migration tools, describe them here.* No migration.
- *When will we remove the existing behavior?* Not applicable.

Test Plan

Since OpenJDK CRaC requires custom build of JVM the most convenient way to test is inside a container. To an extent the support code can be tested without performing the C/R - this can ensure that application behaviour is correct after C/R but does not ensure that the C/R process will be successful.

Rejected Alternatives

CRaC support can be postponed until it appears in mainline JDK, or this feature can be ignored completely.