KIP-906: Tools migration guidelines

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

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

Motivation

The tools migration effort is ongoing and being tracked in KAFKA-14525. This is part of a bigger initiative to split the core module into multiple modules (e. g. storage, network, security, tools), which is being tracked in KAFKA-14524.

The plan is to migrate tools and related classes in a fully compatible way from **kafka.tools** and **kafka.admin** packages (core module) to **org.apache.kafka.tools** package (tools module).

While kicking off this activity, we identified a number of potential compatibility issues:

1. **Missing wrapper**: some tools do not have a wrapper script.
    a. There are system tests that directly refer to the tool's fully qualified class name (FQCN) and expect the old package name when running on old Kafka releases.
    b. They are often used for troubleshooting or automation through the kafka-run-class.sh script which takes the FQCN as input parameter.
2. **SPI argument**: some tools have arguments for setting a custom SPI implementation to be used in place of the default implementation.
    a. Any custom SPI implementation depends on the old package name.
3. **Broken tool**: some tools do not work on supported releases.
4. **Core dependency**: some tools require access to non-trivial core classes that should be migrated first.

Public Interfaces

Detailed list of tools grouped by issue:

- Tools with missing wrapper script
    - kafka.tools.JmxTool
    - kafka.tools.StateChangeLogMerger
    - kafka.tools.GetOffsetShell

- Tools with no wrapper script and used in system tests with version older than DEV_BRANCH
    - kafka.tools.JmxTool
    - kafka.tools.TestEndToEndLatency

- Tools with no BAT wrapper script
    - kafka.tools.ClusterTool (KAFKA-14614)
    - kafka.admin.FeatureCommand
    - org.apache.kafka.shell.MetadataShell
    - org.apache.kafka.tools.VerifiableConsumer
    - org.apache.kafka.tools.VerifiableProducer
    - kafka.admin.ZkSecurityMigrator

- Tools with SPI argument
    - kafka.common.MessageReader used in kafka.tools.ConsoleProducer

- Tools that do not work on supported releases
    - kafka.tools.StateChangeLogMerger

- Tools with non trivial core classes dependencies

- kafka.tools.DumpLogSegments

Proposed Changes

According to the KIP process, command line tools and arguments have to be considered public interface, so we want to maintain full compatibility or provide a deprecation period.

Rather than discussing common migration issues on a case by case basis, we would like to define some guidelines on how to address them in a consistent way.

Missing wrapper

Every tool should have its own wrapper script in bin and bin/windows directories. If a system test refers to the tool's FQCN, it should be updated to use the wrapper script name.

During the package deprecation period, we should provide a redirection to the new compatible implementation via reflection. We should also add a note in the release changes and create a sub-task as a reminder to remove the redirection in the next major release.

Example:

```
package kafka.tools

@deprecated(since = "3.5")
object JmxTool {
 def main(args: Array[String]): Unit = {
   println("WARNING: The 'kafka.tools' package is deprecated and will change to 'org.apache.kafka.tools' in the
next major release.")
   val toolClass = Class.forName("org.apache.kafka.tools.JmxTool")
   val toolMethod = toolClass.getDeclaredMethod("main", classOf[Array[String]])
   toolMethod.invoke(null, args)
 }
}
```

SPI argument

We only have one tool with this issue (kafka.tools.ConsoleProducer) and a number of known external implementations. We can address this as originally proposed in KIP-641.

Broken tool

We only have one tool with this issue (kafka.tools.StateChangeLogMerger), which is tracked in KAFKA-7735. This tool has been broken since 2.0 release, so it means that there is not much interest and we should simply deprecate.

However, the work for migrating and fixing this tool is on-going, but not included in the scope of this KIP.

Core dependency

Most tools communicate via Kafka protocol by using client classes, so they can be migrated without dependencies on core. Instead, tools with non-trivial dependencies on core (e.g. log message parsers) should wait until all required dependencies are migrated. If present, we should set related tasks as blockers for the tool migration sub-task.

Compatibility, Deprecation, and Migration Plan

- Migrated tools must be fully compatible with the old implementation.
- The old package name must be deprecated in the target release (e.g. 3.5) and redirection removed in the next major release (e.g. 4.0).
- Existing users will get a deprecation warning when using the old package name, while old SPIs and classes will be marked as deprecated.

Test Plan

All unit/integration tests should be migrated or added in case they are missing. All affected system tests have to pass with the new implementation.

Rejected Alternatives

- One option to address the missing wrapper and SPI argument issues would be to create a :tools:deprecated sub module hosting all deprecated classes with the old package name. This is mainly driven by the single root package checkstyle limitation.
- Another option to address the missing wrapper issue would be to let the kafka-run-class script do the redirection by going through all input parameters and replacing the old package name with the new one. This adds some delay to every command.