KIP-974: Docker Image for GraalVM based Native Kafka Broker

Status

Current state: Under Discussion

Discussion thread: *here*

Voting thread: *here*

JIRA: KAFKA-15444

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

Motivation

Existing Java-based Kafka brokers take a few seconds to startup. Although it is not affecting the criticality of long-running brokers in production workloads, it annoys developers instantiating hundreds of brokers for unit testing their applications during local development.

This KIP aims to deliver an experimental Apache Kafka docker image that can launch brokers with sub-second startup time and minimal memory footprint by leveraging a GraalVM based native Kafka binary and runs in the Kraft mode.

Thanks to Ozan Gunalp's work on kafka-native which inspired to formalise native Kafka artifact through this KIP.

Broker Startup Time, CPU Usage and Memory Footprint with GraalVM based Native Kafka Broker

GraalVM has provisions to build native standalone executables through ahead-of-time compilation of Java applications. These native executables generally have a smaller memory footprint and start faster than their JVM counterparts.
GraalVM based Kafka broker showed significantly faster startup time and less RAM consumption in benchmarking against JVM based broker. Details can be found below.

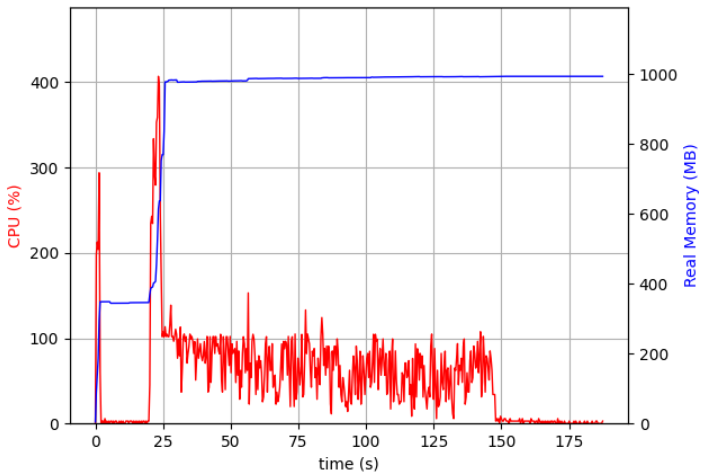
Avg Kafka Broker Startup Time GraalVM versus JVM

- Avg Kafka Server Startup Time Using JVM(default configs)**:** ~1150 ms - 1200ms
- Max Memory: ~1GB
- Following is the table of startup times of GraalVM based Native Kafka Broker depending on GC and PGO:

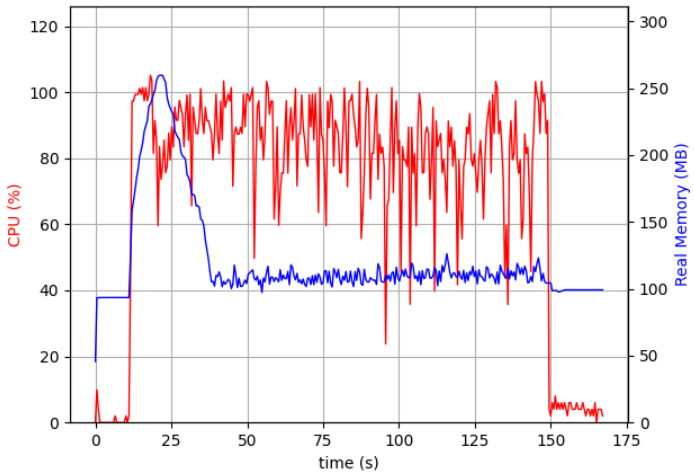
GC	PGO	Kafka Native Binary size	Avg Kafka Broker Startup Time	Max Memory
serial	disabled	96MB	~130ms	~250MB
serial	enabled	67MB	~110ms	~230MB
G1	disabled	128MB	~140ms	~520MB

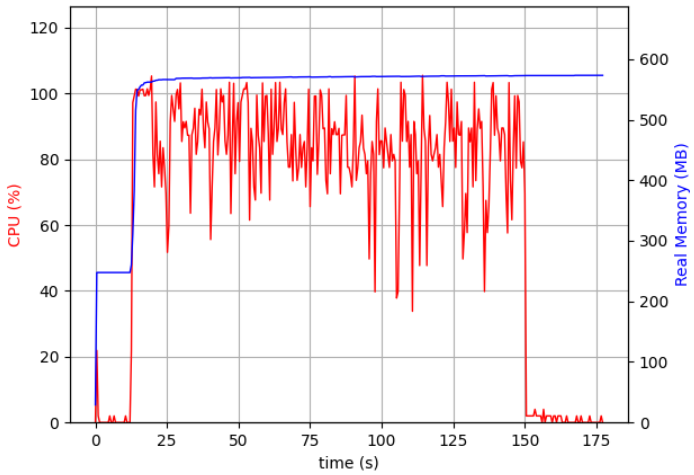
G1	enabled	82MB	~135ms	~540MB

Kafka Broker CPU and Memory usage for GraalVM versus JVM



Kafka Broker using JVM





Native-Binary-serialGC Native-Binary-G1GC

Observations:

- The startup time of the GraalVM broker is ~1/9th of the startup time of the JVM broker.
- No CPU spike is observed for GraalVM broker but JVM broker had sudden spikes both when the broker started and also when the load testing started.
- Considerable difference in memory usage is observed as well. For JVM broker the max memory usage went up to ~1GB as compared to the ~500MB in case of GraalVM broker with G1GC and ~250MB in case of GraalVM broker with serial GC.

Use-cases Tested for the benchmarking: Created a topic, produced to and consumed from it using default configurations.

Machine configuration used for benchmarking:

- Machine: m6i.2xlarge
- Architecture: x86_64
- OS: ubuntu
- Java version: openjdk 17.0.8 2023-07-18
- GraalVM native-image version:
  - native-image 17.0.8 2023-07-18
  - GraalVM Runtime Environment Oracle GraalVM 17.0.8+9.1 (build 17.0.8+9-LTS-jvmci-23.0-b14)

Public Interfaces

- There will be a new additional artifact i.e. Apache Kafka docker image for every Apache Kafka release.
- Sample `docker-compose.yml`
- Quick start examples
- Add public documentation

Proposed Changes

- There will be a docker image as an additional artifact for every Apache Kafka release.
- This docker image will consist of GraalVM native-image based kafka binary and will have support for Linux based AMD and ARM architectures.
- Add sanity tests to run against the new docker image for each tag.
- There will be a new build system for generating Kafka native executables for both ARM and AMD architectures. This new build system is in addition to the existing Java and Scala-based Kafka build systems.
- Add release script to extend the Apache release prcocess to publish a new docker image to public DockerHub.

Handle Multiple Entry-points in the Native Kafka Broker

While building the native image we need to provide the entry point of our application and when the native image will be run, it will start from that entry point.
Example: `native-image [options] -jar jarfile -cp kafka.Kafka`
Apache kafka provides multiple scripts for multiple use cases. They internally trigger the java classes which makes use of JVM. Consider a scenario:
i. We provide a docker image for Apache Kafka.
ii. User provides the `clusterId` value.
iii. We need to format the Kafka log directories using `kafka-storage.sh format -t <clusterid> -c config/kraft/server.properties`.
iv. We would need to support the `storage format` using native image as well, otherwise it will require JVM which will defeat our purpose.

Solution:

1. Create 2 native images with different entry points? This will increase the size of the docker image. Also, this is not scalable.
2. **[Preferred Approach]**Make changes in the Apache Kafka codebase to add a wrapper on the entry points.

```
object KafkaNativeWrapper extends Logging {
  def main(args: Array[String]): Unit = {
    if (args.length == 0) {
      throw new RuntimeException(s"Error: No operation input provided. " +
        s"Please provide a valid operation: 'storage-tool' or 'kafka'.")
    }
    val operation = args.head
    val arguments = args.tail
    operation match {
      case "storage-tool" => StorageTool.main(arguments)
      case "kafka" => Kafka.main(arguments)
      case _ =>
        throw new RuntimeException(s"Unknown operation $operation. " +
          s"Please provide a valid operation: 'storage-tool' or 'kafka'.")
    }
  }
}
```

Generate Metadata Configs for Building Native-Image Kafka Broker

The native-image, on building, does static analysis and includes only the classes and methods that are reachable from the application's entry point. It doesn't support dynamic class loading which can be an issue if our application uses reflection.
Hence, the program elements reflectively accessed at run time must be specified using a manual configuration.

Solution: GraalVM does provide an option to create these configs automatically by running the application normally with the native-image agent attached(`-agentlib:native-image-agent=config-output-dir=META-INF/native-image`).

- To generate configs we intend to use the existing Apache Kafka System Tests as they are quite exhaustive.
- We will run the the system tests using GraalVM JIT and attaching the native-image agent to the process
- Merge the configs of all the tests.

```
JAVAG="/graalvm-jdk-17.0.8+9.1/bin/java"

# Launch mode
if [ "x$DAEMON_MODE" = "xtrue" ]; then
  nohup "$JAVAG" -agentlib:native-image-agent=config-merge-dir=native-configs $KAFKA_HEAP_OPTS
$KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp "$CLASSPATH" $KAFKA_OPTS
"$@" > "$CONSOLE_OUTPUT_FILE" 2>&1 < /dev/null &
else
  exec "$JAVAG" -agentlib:native-image-agent=config-merge-dir=native-configs $KAFKA_HEAP_OPTS
$KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp "$CLASSPATH" $KAFKA_OPTS
"$@"
fi
```

Note: The above will be record only the code path which is executed. *We can never be 100% sure of having an exhaustive coverage of all the reflection classes making it hard to support in production.*

GraalVM Version for Building Native-Image Kafka Broker

GraalVM is available as the following distributions:

[Preferred Approach]GraalVM Community Edition(Java 21)

1. This is based on OpenJDK and features like G1 garbage collector, profile guided optimisations etc are not available in GraalVM Community Edition.
2. It includes Serial GC.
3. Being community edition, we won't face any licensing issues.

Note: GraalVM provides the docker images for the above distribution which can be leveraged for multi layer docker builds.

Docker Base Image

Native binaries operate independently and do not require specific packages to run. Consequently, opting for the most minimal base images will enable us to produce compact Docker images.
We propose to make use of **alpine** image as the base image.

While Alpine images offer a lightweight solution, contributing to a smaller Docker image size, there are certain considerations to bear in mind

- Alpine uses musl libc, but for native image compatibility, we require glibc. To address this, we'll need to install gcompat.
- Alpine uses an older shell instead of bash, necessitating the installation of bash to run our helper scripts.
- Alpine employs the apk package manager, which, being relatively less popular, may pose challenges in the future. There's a potential risk that certain libraries we might need could lack support from apk.

Alpine vs Ubuntu Docker Base Image

The next best option I explored is the Ubuntu Docker image(https://hub.docker.com/_/ubuntu/tags) which is a more complete image.

- Size: It has a size of 70MB compared to the 15MB of the Alpine image (post-installation of glibc and bash), resulting in a difference of 55MB.
- Performance: I executed produce/consume performance scripts on the Kafka native Docker image using both Alpine and Ubuntu, and the results indicated comparable performance between the two.

Image Naming

Image naming should:

1. Transparently communicate the packaged Kafka version.
2. Maintain the above point in the event of CVEs/bugs requiring a dedicated Docker release.

Adhering to the outlined constraints, image tagging can follow this format
<image-name>:<kafka-version>

- **kafka-native:3.7.0**
    - Name of the image: **kafka-native**
      For example, for 3.7.0 version of kafka, the image name with tagging would be apache/kafka-native:3.7.0
    - `native` indicates that the image consists of the native binary.

NOTE: The JVM based Apache Kafka docker image will be named as `apache/kafka:<version>`

Directory Structure

A new directory named `docker` will be added to the repository. This directory will contain all the Docker related code.
Directory Structure:

```
kafka/
    - docker/

        - native-image/
            - Dockerfile          #Dockerfile for the GraalVM native-image based Apache Kafka Docker image.
        - jvm/
            - Dockerfile          #Dockerfile for the JVM-based Apache Kafka Docker image.
        - resources/              #Contains resources needed to create the Docker image.
        - test/                   #Contains sanity tests for the Docker image.
        - docker_build_test.py    #Python script for building and testing the Docker image.
        - docker_release.py       #Python script for building the Docker image and pushing it to Docker Hub.
```

NOTE: This structure is designed with the anticipation of introducing another Docker image based on the native Apache Kafka Broker (as per KIP-975). Both images will share the same resources for image building.

Configuring Properties

We offer two methods for passing the above properties to the container:

1. **File Mounting**: Users can mount a properties file to a specific path within the container (we will clearly document this path). This file will then be utilized to start up Kafka.
2. **Using Environment Variables**: Alternatively, users have the option to provide configurations via environment variables. Here's how to structure these variables:

    - Replace **`.`** with _
    - Replace _ with __`(double underscore)`
    - Replace **`-`** with ___`(triple underscore)`
    - Prefix the result with **KAFKA_**

    Examples:

    - For **`abc.def`**, use **KAFKA_ABC_DEF**
    - For **`abc-def`**, use **KAFKA_ABC___DEF**
    - For **`abc_def`**, use **KAFKA_ABC__DEF**

This way, you have flexibility in how you pass configurations to the container, making it more adaptable to various user preferences and requirements.
NOTE:

1. Secrets will be provided to the container using folder mount.
2. If a property is provided both in the mounted file and as an environment variable, the value from the environment variable will take precedence.

Compatibility, Deprecation, and Migration Plan

- For existing apache kafka users there will be no impact as native-image based kafka docker image will be a new feature.
- The GraalVM native-image based Apache Kafka docker image will be an experimental docker image.
- Unlike JVM, GraalVM native-image performs ahead-of-time compilation and does not support dynamic class loading. It requires extensive testing to understand the total broker functionality support and performance through GraalVM native-image. The GraalVM native-image based container is recommended only for development, and testing and not for production workloads.
- For docker image catering production workloads refer the KIP-975.

Test Plan

GraalVM based Apache Kafka Image is an experimental docker image for local development and testing usage. GraalVM Native-Image tool is still in maturing stage, hence the usage of this image for production can't be recommended.
Testing of the Docker Image: Sanity Tests for the P0 functionalities like Image coming up, topics creation, producing, consuming, restart etc will be added. We will also try to run the existing system tests on the built Apache Kafka native executable.

Build, Test and Scanning Pipeline

This section will be same as mentioned for the JVM Docker Image in KIP-975 build and test pipeline.

Build and Test

Prior to release, the Docker images must undergo building, testing, and vulnerability scanning. To streamline this process, we'll be setting up a GitHub Actions workflow. This workflow will generate two reports: one for test results and another for scanning results. These reports will be available for community review before voting.

Scanning Previously Released Images

We intend to setup a nightly cron job using GitHub Actions and leverage an open-source vulnerability scanning tool like trivy (https://github.com/aquasecurity/trivy), to get vulnerability reports on all supported images. This tool offers a straightforward way to integrate vulnerability checks directly into our GitHub Actions workflow.

Release Process

Following is the plan to release the Docker image:

1. RM would have generated and pushed Apache Kafka's Release Candidate artifacts to apache sftp server hosted in blocked URLhome.apache.org by release.py script
2. Run the automation to build the docker image(**using the above Release Candidate tarball URL**) and test the image.
3. The docker image needs to be pushed to some Dockerhub repo(eg. Release Manager's) for the evaluation of RC Docker image.
4. Start the Voting for RC, which will include the Docker image as well as docker sanity tests report.
5. In case any docker image specific issue is detected, that will be evaluated by the community, if it's a release blocker or not.
6. Once the vote passes, the image will be pushed to `apache/kafka-native` with the version as tag.
7. Steps for the Docker image release will be included in the Release Process doc of Apache Kafka
8. eg. for AK release 3.7.0 and image released will be `apache/kafka-native:3.7.0` (=> image contains AK 3.7.0)

Ownership of the Docker Images' Release

- **Suggestion:** The docker image release should be owned by the Release Manager.
- As per the current release process, only PMC members should be allowed to push to apache/kafka docker image.
- If the RM is not a PMC member, they'll need to take help from a PMC member to release the image.

Rejected Alternatives

GraalVM Version for Building Native-Image Kafka Broker

Oracle GraalVM: This is based on Oracle JDK and includes all the GraalVM features for free under GraalVM Free Terms and Conditions (GFTC) license

1. **GraalVM for JDK 21**
    a. Provides G1 GC support for both linux/amd64 and linux/aarch64 architectures.
    b. It will receive updates under the GFTC, until September 2026.
    c. Subsequent updates of GraalVM for JDK 21 will be licensed under the GraalVM OTN License Including License for Early Adopter Versions (GOTN) and production use beyond the limited free grants of the GraalVM OTN license will require a fee.
2. **GraalVM for JDK 17**
    a. Does not provide G1 GC support for linux/aarch64.
    b. It will receive updates under the GFTC, until September 2024.

Post the free updates, the above distributions will require a fee. Therefore, the community version appears to be the most viable option for us.

Docker Base Image

The Distroless image was also evaluated as a potential base image. It required additional packages to be installed for our application to function.. Ultimately, we found that Alpine offered more flexibility, although there's no strong preference..

Image Name

Another option considered for the image was **kafka-local:3.5.1**

- local indicates that this image is intended only for the local development use. 3.5.1 is the sample kafka version
- This image may be used in the production in the future. In that case, the name "kafka-local" will be counter-intuitive.