KIP-271: Add NetworkClient redirector

Status

Current state: Under Discussion

Discussion thread: here

JIRA: KAFKA-6669

Motivation

The purpose of the change is to accommodate environments without proper DNS support, when KafkaProducer or KafkaConsumer tries to connect to Kafka brokers that are inside another network.

Currently, Kafka client connection fails with "java.io.IOException: Can't resolve address" after Kafka cluster metadata is updated with internal DNS names of brokers, unreachable by Client. The example of such configuration might be a Java Client calling Kafka Cluster from outside of Kubernetes cluster or AWS network.

The KIP improves NetworkClient to redirect the call to alternative network address and provides developers with additional Configurable interface to specify redirection rules.

Public Interfaces

Introduce new **org.apache.kafka.common.network.Redirector** interface that will be used by **org.apache.kafka.common.network.Selector** to redirect to alternative broker address.

org.apache.kafka.common.network.Redirector

```
package org.apache.kafka.common.network;
import java.net.InetSocketAddress;
import org.apache.kafka.common.Configurable;

public interface Redirector extends Configurable {
        public InetSocketAddress redirect(InetSocketAddress address);
}
```

Proposed Changes

With respect to proposed interface the following changes in existing classes are required:

Change **org.apache.kafka.common.network.Selector** class to accept **Redirector** instance through constructor and execute it before opening the connection to broker:

org.apache.kafka.common.network.Selector

```diff
@@ -121,6 +121,7 @@ public class Selector implements Selectable, AutoCloseable {
     private final IdleExpiryManager idleExpiryManager;
     private final MemoryPool memoryPool;
     private final long lowMemThreshold;
+    private final Redirector redirector;
     //indicates if the previous call to poll was able to make progress in reading already-buffered data.
     //this is used to prevent tight loops when memory is not available to read any more data
     private boolean madeReadProgressLastPoll = true;
@@ -147,7 +148,8 @@ public class Selector implements Selectable, AutoCloseable {
             boolean recordTimePerConnection,
             ChannelBuilder channelBuilder,
             MemoryPool memoryPool,
-            LogContext logContext) {
+            LogContext logContext,
+            Redirector redirector) {
         try {
             this.nioSelector = java.nio.channels.Selector.open();
         } catch (IOException e) {
@@ -174,6 +176,21 @@ public class Selector implements Selectable, AutoCloseable {
         this.memoryPool = memoryPool;
         this.lowMemThreshold = (long) (0.1 * this.memoryPool.size());
         this.log = logContext.logger(Selector.class);
+        this.redirector = redirector;
+    }
+
+    public Selector(int maxReceiveSize,
+            long connectionMaxIdleMs,
+            Metrics metrics,
+            Time time,
+            String metricGrpPrefix,
+            Map<String, String> metricTags,
+            boolean metricsPerConnection,
+            boolean recordTimePerConnection,
+            ChannelBuilder channelBuilder,
+            MemoryPool memoryPool,
+            LogContext logContext) {
+        this(maxReceiveSize, connectionMaxIdleMs, metrics, time, metricGrpPrefix, metricTags,
metricsPerConnection, recordTimePerConnection, channelBuilder, memoryPool, logContext, null);
     }
     public Selector(int maxReceiveSize,
@@ -188,10 +205,15 @@ public class Selector implements Selectable, AutoCloseable {
         this(maxReceiveSize, connectionMaxIdleMs, metrics, time, metricGrpPrefix, metricTags,
metricsPerConnection, false, channelBuilder, MemoryPool.NONE, logContext);
     }
+    public Selector(long connectionMaxIdleMS, Metrics metrics, Time time, String metricGrpPrefix,
ChannelBuilder channelBuilder, LogContext logContext, Redirector redirector) {
+        this(NetworkReceive.UNLIMITED, connectionMaxIdleMS, metrics, time, metricGrpPrefix, Collections.
<String, String>emptyMap(), true, false, channelBuilder, MemoryPool.NONE, logContext, redirector);
+    }
+
     public Selector(long connectionMaxIdleMS, Metrics metrics, Time time, String metricGrpPrefix,
ChannelBuilder channelBuilder, LogContext logContext) {
         this(NetworkReceive.UNLIMITED, connectionMaxIdleMS, metrics, time, metricGrpPrefix, Collections.
<String, String>emptyMap(), true, channelBuilder, logContext);
     }
+
     /**
      * Begin connecting to the given address and add the connection to this nioSelector associated with the
given id
      * number.
@@ -211,6 +233,7 @@ public class Selector implements Selectable, AutoCloseable {
         SocketChannel socketChannel = SocketChannel.open();
         try {
             configureSocketChannel(socketChannel, sendBufferSize, receiveBufferSize);
+            address = (redirector != null) ? redirector.redirect(address) : address;
             boolean connected = doConnect(socketChannel, address);
             SelectionKey key = registerChannel(id, socketChannel, SelectionKey.OP_CONNECT);
```

Add Redirector configuration support for KafkaConsumer and KafkaProducer as **broker.redirector** property:

org.apache.kafka.clients.consumer.ConsumerConfig

```
@@ -211,6 +211,10 @@ public class ConsumerConfig extends AbstractConfig {
     public static final String VALUE_DESERIALIZER_CLASS_CONFIG = "value.deserializer";
     public static final String VALUE_DESERIALIZER_CLASS_DOC = "Deserializer class for value that implements
the <code>org.apache.kafka.common.serialization.Deserializer</code> interface.";
+    /** <code>broker.redirector</code> */
+    public static final String BROKER_REDIRECTOR_CLASS_CONFIG = "broker.redirector";
+    public static final String BROKER_REDIRECTOR_CLASS_DOC = "Broker redirector class that implements the
<code>org.apache.kafka.common.network.Redirector</code> interface.";
+
     /** <code>connections.max.idle.ms</code> */
     public static final String CONNECTIONS_MAX_IDLE_MS_CONFIG = CommonClientConfigs.
CONNECTIONS_MAX_IDLE_MS_CONFIG;
@@ -397,6 +401,11 @@ public class ConsumerConfig extends AbstractConfig {
                                         Type.CLASS,
                                         Importance.HIGH,
                                         VALUE_DESERIALIZER_CLASS_DOC)
+                                .define(BROKER_REDIRECTOR_CLASS_CONFIG,
+                                        Type.CLASS,
+                                        null,
+                                        Importance.LOW,
+                                        BROKER_REDIRECTOR_CLASS_DOC)
                                 .define(REQUEST_TIMEOUT_MS_CONFIG,
                                         Type.INT,
                                         305000, // chosen to be higher than the default of max.poll.interval.ms
```

org.apache.kafka.clients.consumer.KafkaConsumer

```
@@ -44,6 +44,7 @@ import org.apache.kafka.common.metrics.Metrics;
 import org.apache.kafka.common.metrics.MetricsReporter;
 import org.apache.kafka.common.metrics.Sensor;
 import org.apache.kafka.common.network.ChannelBuilder;
+import org.apache.kafka.common.network.Redirector;
 import org.apache.kafka.common.network.Selector;
 import org.apache.kafka.common.requests.IsolationLevel;
 import org.apache.kafka.common.requests.MetadataRequest;
@@ -715,8 +716,9 @@ public class KafkaConsumer<K, V> implements Consumer<K, V> {
             int heartbeatIntervalMs = config.getInt(ConsumerConfig.HEARTBEAT_INTERVAL_MS_CONFIG);
+            Redirector brokerRedirector = config.getConfiguredInstance(ConsumerConfig.
BROKER_REDIRECTOR_CLASS_CONFIG, Redirector.class);
             NetworkClient netClient = new NetworkClient(
-                    new Selector(config.getLong(ConsumerConfig.CONNECTIONS_MAX_IDLE_MS_CONFIG), metrics, time,
metricGrpPrefix, channelBuilder, logContext),
+                    new Selector(config.getLong(ConsumerConfig.CONNECTIONS_MAX_IDLE_MS_CONFIG), metrics, time,
metricGrpPrefix, channelBuilder, logContext, brokerRedirector),
                     this.metadata,
                     clientId,
                     100, // a fixed large enough value will suffice for max in-flight requests
```

org.apache.kafka.clients.producer.ProducerConfig

```
@@ -181,6 +181,10 @@ public class ProducerConfig extends AbstractConfig {
     public static final String VALUE_SERIALIZER_CLASS_CONFIG = "value.serializer";
     public static final String VALUE_SERIALIZER_CLASS_DOC = "Serializer class for value that implements the
<code>org.apache.kafka.common.serialization.Serializer</code> interface.";
+    /** <code>broker.redirector</code> */
+    public static final String BROKER_REDIRECTOR_CLASS_CONFIG = "broker.redirector";
+    public static final String BROKER_REDIRECTOR_CLASS_DOC = "Broker redirector class that implements the
<code>org.apache.kafka.common.network.Redirector</code> interface.";
+
     /** <code>connections.max.idle.ms</code> */
     public static final String CONNECTIONS_MAX_IDLE_MS_CONFIG = CommonClientConfigs.
CONNECTIONS_MAX_IDLE_MS_CONFIG;
@@ -292,6 +296,11 @@ public class ProducerConfig extends AbstractConfig {
                                         Type.CLASS,
                                         Importance.HIGH,
                                         VALUE_SERIALIZER_CLASS_DOC)
+                                .define(BROKER_REDIRECTOR_CLASS_CONFIG,
+                                        Type.CLASS,
+                                        null,
+                                        Importance.LOW,
+                                        BROKER_REDIRECTOR_CLASS_DOC)
                                 /* default is set to be a bit lower than the server default (10 min), to avoid
both client and server closing connection at same time */
                                 .define(CONNECTIONS_MAX_IDLE_MS_CONFIG,
                                         Type.LONG,
```

org.apache.kafka.clients.producer.KafkaProducer

```
@@ -412,9 +413,10 @@ public class KafkaProducer<K, V> implements Producer<K, V> {
             }
             ChannelBuilder channelBuilder = ClientUtils.createChannelBuilder(config);
             Sensor throttleTimeSensor = Sender.throttleTimeSensor(metricsRegistry.senderMetrics);
+            Redirector brokerRedirector = config.getConfiguredInstance(ProducerConfig.
BROKER_REDIRECTOR_CLASS_CONFIG, Redirector.class);
             KafkaClient client = kafkaClient != null ? kafkaClient : new NetworkClient(
                     new Selector(config.getLong(ProducerConfig.CONNECTIONS_MAX_IDLE_MS_CONFIG),
-                            this.metrics, time, "producer", channelBuilder, logContext),
+                            this.metrics, time, "producer", channelBuilder, logContext, brokerRedirector),
                     this.metadata,
                     clientId,
                     maxInflightRequests,
```

Usa-case example:

Kafka cluster is running in OpenShift as a distributed setup of 5 brokers. Each broker's pod service port is exposed as a separate port on localhost interface. KafkaProducer connects to cluster from outside of OpenShift environment and specifies bootstrap servers as:

```
bootstrap.servers=localhost:9092,localhost:9093,localhost:9094,localhost:9095,localhost:9096
```

When message is getting sent, Kafka client recieves clusters metadata with pods internal DNS names and delivery is failed:

```
DEBUG org.apache.kafka.clients.NetworkClient[kafka-producer-network-thread | producer-1] - [Producer
clientId=producer-1] Initiating connection to node kafka-2.kafka.myproject.svc.cluster.local:9092 (id: 1003
rack: null)
DEBUG org.apache.kafka.clients.NetworkClient[kafka-producer-network-thread | producer-1] - [Producer
clientId=producer-1] Error connecting to node kafka-2.kafka.myproject.svc.cluster.local:9092 (id: 1003 rack:
null)
java.io.IOException: Can't resolve address: kafka-2.kafka.myproject.svc.cluster.local:9092
        at org.apache.kafka.common.network.Selector.doConnect(Selector.java:258)
        at org.apache.kafka.common.network.Selector.connect(Selector.java:237)
        at org.apache.kafka.clients.NetworkClient.initiateConnect(NetworkClient.java:792)
        at org.apache.kafka.clients.NetworkClient.ready(NetworkClient.java:230)
        at org.apache.kafka.clients.producer.internals.Sender.sendProducerData(Sender.java:263)
        at org.apache.kafka.clients.producer.internals.Sender.run(Sender.java:238)
        at org.apache.kafka.clients.producer.internals.Sender.run(Sender.java:163)
        at java.base/java.lang.Thread.run(Thread.java:844)
Caused by: java.nio.channels.UnresolvedAddressException
        at java.base/sun.nio.ch.Net.checkAddress(Net.java:112)
        at java.base/sun.nio.ch.SocketChannelImpl.connect(SocketChannelImpl.java:622)
        at org.apache.kafka.common.network.Selector.doConnect(Selector.java:256)
        ... 7 more
```

Using new KafkaProducer configuration and implementing Redirector interface resolves the problem:

KafkaProducer configuration example

```
broker.redirector=custom.redirector.CustomBrokerRedirector
```

Redirector implementation example

```
public class CustomBrokerRedirector implements Redirector {
        @Override
        public void configure(Map<String, ?> props) {}

        @Override
        public InetSocketAddress redirect(InetSocketAddress address) {
                String host = address.getHostString();
                if (host.contains("kafka-0"))
                        return new InetSocketAddress("localhost", 9092);
                else if (host.contains("kafka-1"))
                        return new InetSocketAddress("localhost", 9093);
                else if (host.contains("kafka-2"))
                        return new InetSocketAddress("localhost", 9094);
                else if (host.contains("kafka-3"))
                        return new InetSocketAddress("localhost", 9095);
                else if (host.contains("kafka-4"))
                        return new InetSocketAddress("localhost", 9096);
                return address;
        }
}
```

Compatibility, Deprecation, and Migration Plan

The KIP is fully backward-compatible. If no Redirector configuration specified, Selector will work in the same way.

Rejected Alternatives

- Advertised listeners configuration is not easy to use in environment like OpenShift, when pods are periodically restarted and getting assigned IPs and DNS names. This will require the deployment script to manipulate each broker configuration on pod startup, which is hard process, if possible at all. Each replica should have a particular configuration assigned in both Kubernetes and broker that is not always possible, while easier to control the forwarding process on Kafka Client itself, allowing testing from IDE without deploying the code into Kubernetes.