KIP-2 - Refactor brokers to allow listening on multiple ports and IPs

Status

Current state: *[Accepted]*

Discussion thread: here

JIRA: *here*

Released: 0.8.3

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

Motivation

The goal is to eventually support different security mechanisms on different ports.
Currently brokers are defined as host+port pair, and this definition exists throughout the code-base, therefore some refactoring is needed to support multiple ports for a single broker.

Public Interfaces

- The following wire protocol APIs will change. We are bumping the protocol version to support backward compatibility:
  - UpdateMetadataRequest - will contain multiple host/port pairs, not just one. The new protocol is:

    [controllerId controllerEpoch partitionStateInfoCount partitionStateInfoCount [Topic Partition controllerEpoch leader leaderEpoch isr. size [isr] zkVersion] numAliveBrokers **[brokerId numEndpoints [host port securityProtocol]]**]
    The bold part is new. it used to be [brokerId host port].
    host => String
    port => Int32
    securityProtocol => Int16 (corresponding to SecurityProtocol ENUM)

- The following client libraries will change:

  - We are adding **org.apache.kafka.common.protocol.SecurityProtocol** - an ENUM describing the possible security protocols the broker may support and the client may try to connect to.

- Quite a few changes from user configuration perspective:
  - Producers and consumers need to specify **security.protocol** - which protocol they'll use to connect to the server. Current valid values are PLAINTEXT and TRACE (for testing). Future values should include TLS and SASL.
  - Brokers that wish to support multiple protocols and multiple host/port pairs will need to specify a new configuration parameter: **"listeners"** and/or **"advertised.listeners"** - replacing host/port and advertised.host/advertised.port. Listener format is a comma-separated list of **"protocol://host:port**
  - Brokers need to specify **replication.security.protocol** - which protocol / port will be used to replicate data between brokers? this must be identical in all brokers.
  - Following an upgrade, brokers need to specify **wire.protocol.version** = 0.8.3.0 (at least), to start using the new wire protocol and to support new security protocols.

- Broker registration in Zookeeper will change to include all host/port pairs the broker supports. The new json is:

```
{"version":2,
  "jmx_port":9999,
  "timestamp":"2233345666",
  "endpoints": [
    {"host":"myhost",
     "port":9092,
     "protocolType":"plain"},
   {"host":"myhost",
    "port":9093,
    "protocolType":"ssl"}]}
```

- The proposed client-server workflow is as follows:
  - Clients (producers and consumers) need to know about the broker ports in advance. They don't need to know about all brokers, but they need to know at least one host:port pair that speaks the protocol they

want to use.
The change from current behavior is that all host:port pairs in broker.list must be of the same protocol and match the security.protocol configuration parameter.
    - Client uses **security.protocol** configuration parameter to open a connection to one of the brokers and sends the good old MetadataRequest.
The broker knows which port it got the connection on, therefore it knows which security protocol is expected (it needs to use the same protocol to accept the connection and respond), and therefore it can send a response that contains only the host:port pairs that are relevant to that protocol.
    - From the client side the **MetadataResponse did not change** - it contains a list of brokerId,host,port that the client can connect to.
The fact that all those broker endpoints were chosen out of a larger collection to match the right protocol is irrelevant for the client.

- For testing purposes, I added a new protocol called TRACE. This is not meant for use by users (although nothing prevents them from using and and nothing is wrong if they do). Currently the only use is to validate the use of Kafka with two different listeners and non-default endpoint in unit and system tests. In terms of functionality, it is indistinguishable from PLAINTEXT at the moment, since we did not implement any protocol yet. Once we add SASL or TLS implementations we may remove TRACE, or we may continue using it to test a simple non-default channel implementation. One thought I had (and the reason for the name) is that we may use it to add extra instrumentation to the channel implementation in the future and use it for debugging - but this is an un-baked plan and not in scope for this KIP.

Proposed Changes

See detailed description here: Multiple Listeners for Kafka Brokers

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

  If a user is not interested in using non-default protocol (i.e they are using PlainText transmission and not TLS/SASL), there is no visible change

- *If we are changing behavior how will we phase out the older behavior?*

  The upgrade plan is:

  a. (if upgrading from 0.8.2.0 or earlier, add inter.broker.protocol.version=0.8.2.0 to configuration file, otherwise wire protocol will default to the latest version and we will not be able to do a rolling upgrade.)
  b. do a rolling upgrade on the brokers first
  c. bump version of inter.broker.protocol.version to current version (0.8.3.0 or 0.9.0.0 depending on release)
  d. If desired additional ports / security protocols can be added using the new "listeners" configuration parameter.
  e. Clients can be upgraded whenever is convenient.

     Only "plaintext" protocol is supported until broker upgrade is complete. Also, only upgraded clients will be able to use new security protocols.

- *When will we remove the existing behavior?*
  We plan to continue supporting existing behavior (i.e. a single "plaintext" host+port) indefinitely.

Rejected Alternatives

There was considerable discussion of the possibility of supporting all security protocols on the same port simultaneously. This was rejected for the following reasons:

1. This approach is vulnerable to "downgrade attack"
2. This requires a protocol-negotiation protocol, which can be rather complex
3. Most other applications and frameworks support multiple protocols on different ports. Especially SSL.
4. This approach allows cleanly disabling un-authenticated access (by removing "plaintext" ports after upgrade). This ability is desirable for auditing purposes.