Table of Contents

Kafka Broker Services

Kafka Features

Kafka Producer API

Kafka Consumer API

Kafka Architecture

Kafka Operations and Administration

Kafka Metrics

Kafka Tools

Kafka Configuration

Tips and Tricks

Kafka Connect

Appendix

Apache Kafka Notebook

Welcome to Apache Kafka Notebook!

I'm Jacek Laskowski, an **independent consultant** who is passionate about Apache Spark, **Apache Kafka**, Scala, sbt (with some flavour of Apache Mesos, Hadoop YARN, and DC/OS). I lead Warsaw Scala Enthusiasts and Warsaw Spark meetups in Warsaw, Poland.

Contact me at jacek@japila.pl or @jaceklaskowski to discuss Apache Kafka and Apache Spark opportunities, e.g. courses, workshops, mentoring or application development services.

If you like the Apache Kafka notes you should seriously consider participating in my own, very hands-on Spark Workshops.

This collections of notes (what some may rashly call a "book") serves as the ultimate place of mine to collect all the nuts and bolts of using Apache Kafka. The notes aim to help me designing and developing better products with Kafka. It is also a viable proof of my understanding of Apache Kafka. I do eventually want to reach the highest level of mastery in Apache Kafka.

Expect text and code snippets from a variety of public sources. Attribution follows.

Overview of Kafka

Apache Kafka is an open source project for a distributed publish-subscribe messaging system rethought as a distributed commit log.

Kafka stores messages in topics that are partitioned and replicated across multiple brokers in a cluster. Producers send messages to topics from which consumers read.

Language Agnostic — producers and consumers use binary protocol to talk to a Kafka cluster.

Messages are byte arrays (with String, JSON, and Avro being the most common formats). If a message has a key, Kafka makes sure that all messages of the same key are in the same partition.

Consumers may be grouped in a consumer group with multiple consumers. Each consumer in a consumer group will read messages from a unique subset of partitions in each topic they subscribe to. Each message is delivered to one consumer in the group, and all messages with the same key arrive at the same consumer.

Durability — Kafka does not track which messages were read by each consumer. Kafka keeps all messages for a finite amount of time, and it is consumers' responsibility to track their location per topic, i.e. offsets.

It is worth to note that Kafka is often compared to the following open source projects:

1. Apache ActiveMQ and RabbitMQ given they are message broker systems, too.
2. Apache Flume for its ingestion capabilities designed to send data to HDFS and Apache HBase.

AdminManager

`AdminManager` is…FIXME

`AdminManager` is created exclusively when `KafkaServer` is started.

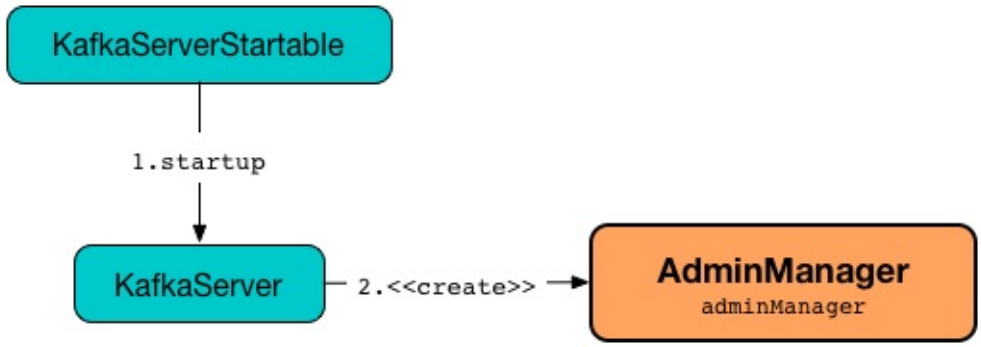

Figure 1. AdminManager

`logIdent` is **[Admin Manager on Broker [brokerId]]**.

Table 1. AdminManager's Internal Properties (e.g. Registries and Counters)

Name	Description
`alterConfigPolicy`	
`createTopicPolicy`	
`topicPurgatory`	

Tip	Enable `INFO`, `DEBUG`, `TRACE` logging levels for `kafka.server.AdminManager` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.server.AdminManager=TRACE, stdout` Refer to Logging.

Creating Topics — `createTopics` Method

```
createTopics(
  timeout: Int,
  validateOnly: Boolean,
  createInfo: Map[String, CreateTopicsRequest.TopicDetails],
  responseCallback: (Map[String, ApiError]) => Unit): Unit
```

`createTopics` …FIXME

Note	`createTopics` is used exclusively when `KafkaApis` handles a CREATE_TOPICS request.

Creating AdminManager Instance

`AdminManager` takes the following when created:

- KafkaConfig
- `Metrics`
- MetadataCache
- ZkUtils

`AdminManager` initializes the internal registries and counters.

Authorizer

`Authorizer` is…FIXME

`configure` Method

Caution	FIXME

Cluster

`Cluster` represents a subset of the nodes and topic partitions in a Kafka cluster.

Note	`org.apache.kafka.common.Cluster` is a `public final` class.

A special variant of a cluster is **boostrap cluster** that is made up of the bootstrap brokers that are mandatory (and specified explicitly) when Kafka clients are created, i.e. KafkaAdminClient, AdminClient, KafkaConsumer and KafkaProducer.

Note	A bootstrap cluster does not hold all information about the cluster.

Table 1. Cluster's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
`isBootstrapConfigured`	Flag…FIXME Used when…FIXME
`partitionsByTopic`	

`bootstrap` Method

```
static Cluster bootstrap(List<InetSocketAddress> addresses)
```

`bootstrap` …FIXME

Note	`bootstrap` is used when KafkaAdminClient, AdminClient, KafkaConsumer and KafkaProducer are created.

`isBootstrapConfigured` Method

```
boolean isBootstrapConfigured()
```

`isBootstrapConfigured` gives isBootstrapConfigured internal flag.

Note	`isBootstrapConfigured` is used when…FIXME

Getting Partitions for Topic — `partitionsForTopic` Method

```
List<PartitionInfo> partitionsForTopic(String topic)
```

`partitionsForTopic` returns a collection of zero or more partition of the input `topic` from partitionsByTopic internal lookup table.

Note	`partitionsForTopic` is used when: • `Metadata` getClusterForCurrentTopics • `KafkaAdminClient` describeTopics • `KafkaConsumer` partitionsFor • `KafkaProducer` requests partitions for a topic • `DefaultPartitioner` assigns the partition for a record

Cluster (deprecated)

Important	It *seems* that `Cluster` class is created using ZkUtils.getCluster that is used exclusively when `ZKRebalancerListener` does syncedRebalance (that in turn happens for the currently-deprecated `ZookeeperConsumerConnector`). In other words, `Cluster` class and the page are *soon* to be removed.

`kafka.cluster.Cluster` private class represents a set of active brokers in a Kafka cluster.

Note	There is also `org.apache.kafka.common.Cluster`.

`topics` Method

Caution	FIXME

`availablePartitionsForTopic` Method

Caution	FIXME

ClusterConnectionStates

`ClusterConnectionStates` is…FIXME

`connecting` Method

```
void connecting(String id, long now)
```

`connecting` …FIXME

Note	`connecting` is used when…FIXME

`disconnected` Method

```
void disconnected(String id, long now)
```

`disconnected` …FIXME

Note	`disconnected` is used when…FIXME

ClusterResourceListener (and ClusterResourceListeners Collection)

`ClusterResourceListener` is the contract for objects that want to be notified about changes in the cluster metadata.

```
package org.apache.kafka.common;

public interface ClusterResourceListener {
  void onUpdate(ClusterResource clusterResource);
}
```

You can register a `ClusterResourceListener` for the following Kafka services:

- `KafkaServer` and get notified when the server starts up
- `KafkaProducer` and get notified when it is created and…FIXME
- `KafkaConsumer` and get notified when it is created and…FIXME

ClusterResourceListeners Collection

`ClusterResourceListeners` collection holds zero or more `ClusterResourceListener` objects and uses them as if there were one.

`ClusterResourceListeners` is used when:

- `Metadata` notifies ClusterResourceListeners about every cluster metadata change
- `KafkaServer` starts up

DynamicConfigManager

`DynamicConfigManager` is…FIXME

`DynamicConfigManager` is created when…FIXME

`startup` Method

```
startup
```

`startup` …FIXME

Note	`startup` is used exclusively when `KafkaServer` starts up.

Creating DynamicConfigManager Instance

`DynamicConfigManager` takes the following when created:

- DynamicConfigManager

`DynamicConfigManager` initializes the internal registries and counters.

Fetcher

`Fetcher` is created exclusively when `KafkaConsumer` is created.

Figure 1. Fetcher and KafkaConsumer

Table 1. Fetcher's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
`client`	ConsumerNetworkClient that is given when `Fetcher` is created.

Creating Fetcher Instance

`Fetcher` takes the following when created:

- ConsumerNetworkClient
- Minimum number of bytes
- Maximum number of bytes
- Maximum wait time
- Fetch size
- How many records to poll
- Flag to check CRC or not
- Deserializer for keys
- Deserializer for values
- Metadata
- `SubscriptionState`
- `Metrics`
- `FetcherMetricsRegistry`
- `Time`
- Retry backoff in milliseconds

- `IsolationLevel`

`Fetcher` initializes the internal registries and counters.

`Fetcher` registers itself with SubscriptionState as a listener to receive notifications about… FIXME

`sendFetches` Method

Caution	FIXME

`sendMetadataRequest` Internal Method

```
RequestFuture<ClientResponse> sendMetadataRequest(MetadataRequest.Builder request)
```

Internally, `sendMetadataRequest` requests ConsumerNetworkClient for the least loaded node.

With the node, `sendMetadataRequest` requests ConsumerNetworkClient to send the request to the node.

When no node was found, `sendMetadataRequest` returns a `RequestFuture` with `NoAvailableBrokersException`.

Note	`sendMetadataRequest` is used exclusively when `Fetcher` is requested for topic metadata.

`beginningOffsets` Method

Caution	FIXME

`retrieveOffsetsByTimes` Method

Caution	FIXME

Getting Topic Metadata — `getTopicMetadata` Method

```
Map<String, List<PartitionInfo>> getTopicMetadata(MetadataRequest.Builder request, long
 timeout)
```

Internally, `getTopicMetadata` sends the metadata request and requests ConsumerNetworkClient to poll until it finishes successfully or `timeout` expires.

After `poll` finishes, `getTopicMetadata` takes the cluster information from `MetadataResponse`.

When `MetadataResponse` is successful, `getTopicMetadata` takes topics (from `Cluster`) and requests `Cluster` for available partitions for every topic.

In the end, `getTopicMetadata` creates a collection of topic and partitions pairs.

Caution	FIXME Describe the failure path

Note	`getTopicMetadata` is used when: • `Fetcher` is requested to find metadata for all topics • `KafkaConsumer` is requested to find partitions for a topic.

Finding Metadata for All Topics — `getAllTopicMetadata` Method

```
Map<String, List<PartitionInfo>> getAllTopicMetadata(long timeout)
```

`getAllTopicMetadata` gets topic metadata specifying no topics (which means all topics available).

Note	`getAllTopicMetadata` is used exclusively when `KafkaConsumer` requests metadata for all topics.

GroupCoordinator

`GroupCoordinator` is…FIXME

`GroupCoordinator` is created when…FIXME

Figure 1. GroupCoordinator's Startup

`logIdent` is **[GroupCoordinator [brokerId]]**.

`GroupCoordinator` uses `isActive` flag to…FIXME

Creating GroupCoordinator Instance — `apply` Factory Method

Caution	FIXME

Creating GroupCoordinator Instance

`GroupCoordinator` takes the following when created:

- Broker ID
- `GroupConfig`
- OffsetConfig
- `GroupMetadataManager`
- `DelayedOperationPurgatory[DelayedHeartbeat]`
- `DelayedOperationPurgatory[DelayedJoin]`
- `Time`

`GroupCoordinator` initializes the internal registries and counters.

Starting Up (and GroupMetadataManager) — `startup` Method

```
startup(enableMetadataExpiration: Boolean = true): Unit
```

`startup` first prints out the following INFO message to the logs:

```
INFO [GroupCoordinator [brokerId]]: Starting up. (kafka.coordinator.group.GroupCoordin
ator)
```

With `enableMetadataExpiration` input flag enabled, `startup` requests GroupMetadataManager to enableMetadataExpiration.

`startup` turns isActive flag on.

In the end, `startup` prints out the following INFO message to the logs:

```
INFO [GroupCoordinator [brokerId]]: Startup complete. (kafka.coordinator.group.GroupCo
ordinator)
```

Note	`startup` is used exclusively when `KafkaServer` starts up.

GroupMetadataManager

`GroupMetadataManager` is…FIXME

`GroupMetadataManager` is created exclusively when `GroupCoordinator` is created.

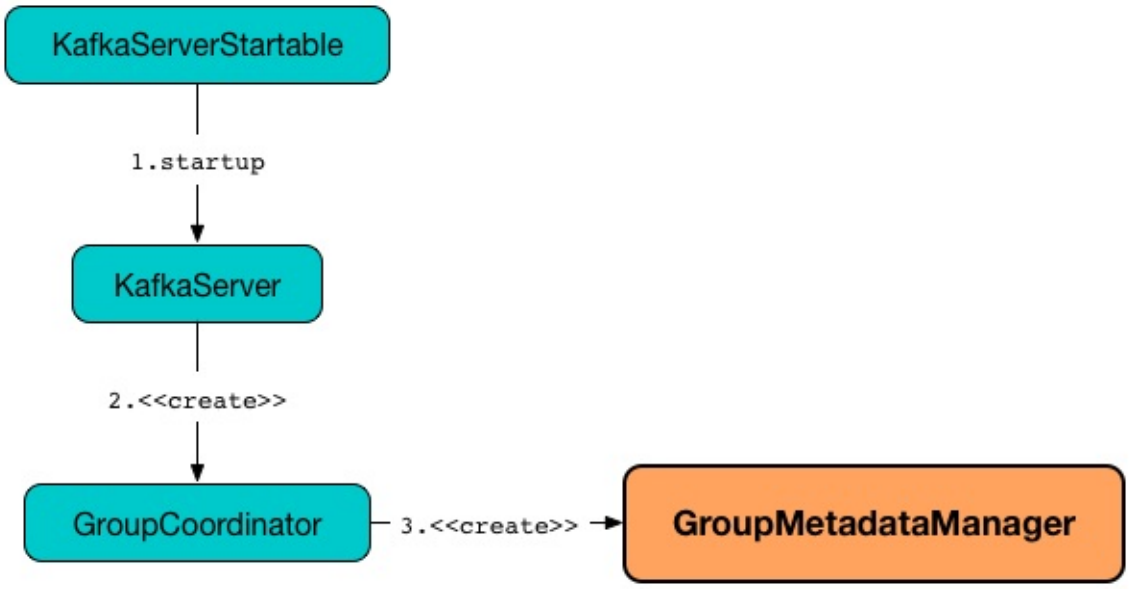

Figure 1. Creating GroupMetadataManager

Table 1. GroupMetadataManager's Internal Properties (e.g. Registries and Counters)

Name	Description
`groupMetadataCache`	Pool of `GroupMetadata` by group ID
`groupMetadataTopicPartitionCount`	
`scheduler`	KafkaScheduler

`enableMetadataExpiration` Method

```
enableMetadataExpiration(): Unit
```

`enableMetadataExpiration` requests KafkaScheduler to start.

`enableMetadataExpiration` schedules **delete-expired-group-metadata** task that cleanupGroupMetadata every `offsetsRetentionCheckIntervalMs` milliseconds.

Note	`enableMetadataExpiration` is used exclusively when `GroupCoordinator` is started.

Creating GroupMetadataManager Instance

`GroupMetadataManager` takes the following when created:

- Broker ID
- `ApiVersion`
- OffsetConfig
- ReplicaManager
- ZkUtils
- `Time`

`GroupMetadataManager` initializes the internal registries and counters.

`cleanupGroupMetadata` Internal Method

```
cleanupGroupMetadata(): Unit  (1)
cleanupGroupMetadata(deletedTopicPartitions: Option[Seq[TopicPartition]]): Unit
```

1. Calls the other `cleanupGroupMetadata` with empty `deletedTopicPartitions` collection

`cleanupGroupMetadata` takes the current time (using time) and for every `GroupMetadata` in cache does the following:

1. FIXME

In the end, `cleanupGroupMetadata` prints out the following INFO message to the logs:

```
Removed [offsetsRemoved] expired offsets in [duration] milliseconds
```

Note	`cleanupGroupMetadata` is used exclusively when `GroupMetadataManager` is requested to enableMetadataExpiration (as **delete-expired-group-metadata** task).

Getting Number of Partitions for `__consumer_offsets` Topic — `getGroupMetadataTopicPartitionCount` Internal Method

```
getGroupMetadataTopicPartitionCount: Int
```

`getGroupMetadataTopicPartitionCount` requests ZkUtils for getTopicPartitionCount of `__consumer_offsets` topic.

If not available, `getGroupMetadataTopicPartitionCount` requests OffsetConfig for offsetsTopicNumPartitions.

Note	`getGroupMetadataTopicPartitionCount` is used exclusively when `GroupMetadataManager` is requested for groupMetadataTopicPartitionCount.

InterBrokerSendThread

`InterBrokerSendThread` …FIXME

`doWork` Method

```
def doWork(): Unit
```

Note	`doWork` is a part of ShutdownableThread Contract.

`doWork` …FIXME

Note	`doWork` is used when…FIXME

Kafka — Standalone Command-Line Application

`kafka.Kafka` is a standalone command-line application that starts a Kafka broker.

`kafka.Kafka` is started using kafka-server-start.sh shell script.

```
// Using sh -xv to trace kafka-server-start.sh
$ sh -xv ./bin/kafka-server-start.sh config/server.properties
...
exec $base_dir/kafka-run-class.sh $EXTRA_ARGS kafka.Kafka "$@"
+ exec ./bin/kafka-run-class.sh -name kafkaServer -loggc kafka.Kafka config/server.pro
perties
...
```

`getPropsFromArgs` Method

Caution	FIXME

Starting Kafka Broker on Command Line — `main` Method

```
main(args: Array[String]): Unit
```

`main` merges properties and creates a `KafkaServerStartable`.

`main` registers a JVM shutdown hook to shut down `KafkaServerStartable`.

Note	`main` uses Java's Runtime.addShutdownHook to register the shutdown hook.

In the end, `main` starts the `KafkaServerStartable` and waits till it finishes.

`main` terminates the JVM with status `0` when `KafkaServerStartable` shuts down properly and with status `1` in case of any exception.

Note	`main` uses Java's System.exit to terminate a JVM.

Registering INFO Logging Signal Handlers (for TERM, INT and HUP Signals) — `registerLoggingSignalHandler` Internal Method

```
registerLoggingSignalHandler(): Unit
```

`registerLoggingSignalHandler` registers signal handlers for `TERM`, `INT` and `HUP` signals so that, once received, it prints out the following INFO message to the logs:

```
Terminating process due to signal [signal]
```

```
$ jps -lm | grep -i kafka
79965 kafka.Kafka config/server.properties

// You could use "pkill -TERM -nf kafka" instead
$ kill -TERM 79965

// In the Kafka server's console
INFO Terminating process due to signal SIGTERM (kafka.Kafka$)
```

Note	`registerLoggingSignalHandler` is used exclusively when a Kafka broker is started.

Note	`registerLoggingSignalHandler` was added to Kafka 1.0.0 in KAFKA-5679; Add logging for broker termination due to SIGTERM or SIGINT.

KafkaApis — API Request Handler

`KafkaApis` handles API requests (by means of handlers).

`KafkaApis` is created exclusively when `KafkaServer` is started (and creates KafkaRequestHandlerPool).

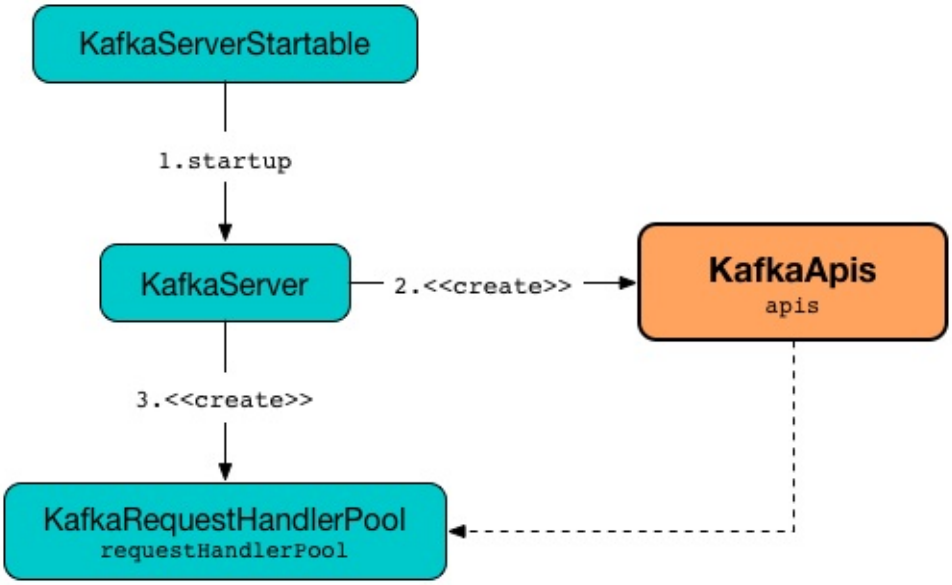

Figure 1. KafkaApis is Created for KafkaRequestHandlerPool when KafkaServer Starts Up

Table 1. KafkaApis's API Keys and Handlers (in alphabetical order)

Key	Handler
`AlterReplicaLogDirs`	handleLeaderAndIsrRequest
`ControlledShutdown`	handleControlledShutdownRequest
`CreatePartitions`	handleCreatePartitionsRequest
`CreateTopics`	handleCreateTopicsRequest
`DeleteTopics`	handleDeleteTopicsRequest
`Fetch`	handleFetchRequest
`LeaderAndIsr`	handleAlterReplicaLogDirsRequest
`Metadata`	handleTopicMetadataRequest
`OffsetFetch`	handleOffsetFetchRequest

Tip	Enable `INFO`, `DEBUG` or `TRACE` logging levels for `kafka.server.KafkaApis` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.server.KafkaApis=TRACE` Refer to Logging.

Routing API Requests — `handle` Method

```
handle(request: RequestChannel.Request): Unit
```

`handle` first prints out the following TRACE message to the logs:

```
Handling request:[request] from connection [id];securityProtocol:[protocol],principal:[principal]
```

`handle` then relays the input `request` to the corresponding handler per the apiKey (from the header of the input `request`).

Note	`handle` is used exclusively when `KafkaRequestHandler` thread is started.

Handling LeaderAndIsr Request — `handleLeaderAndIsrRequest` Handler

```
handleLeaderAndIsrRequest(request: RequestChannel.Request): Unit
```

`handleLeaderAndIsrRequest` …FIXME

Note	`handleLeaderAndIsrRequest` is used exclusively to handle LeaderAndIsr requests.

Handling AlterReplicaLogDirs Request — `handleAlterReplicaLogDirsRequest` Handler

```
handleAlterReplicaLogDirsRequest(request: RequestChannel.Request): Unit
```

`handleAlterReplicaLogDirsRequest` …FIXME

Note	`handleAlterReplicaLogDirsRequest` is used exclusively to handle AlterReplicaLogDirs requests.

Handling CreateTopics Request — `handleCreateTopicsRequest` Handler

```
handleCreateTopicsRequest(request: RequestChannel.Request): Unit
```

`handleCreateTopicsRequest` …FIXME

`handleCreateTopicsRequest` checks whether KafkaController is active…FIXME

`handleCreateTopicsRequest` authorizes the `Create` operation for `ClusterResource` …FIXME

In the end, `handleCreateTopicsRequest` requests AdminManager to create the topics.

Note	`handleCreateTopicsRequest` is used exclusively to handle CreateTopics requests.

Handling OffsetFetch Request — `handleOffsetFetchRequest` Handler

```
handleOffsetFetchRequest(request: RequestChannel.Request): Unit
```

`handleOffsetFetchRequest` …FIXME

Note	`handleOffsetFetchRequest` is used exclusively to handle OffsetFetch requests.

Handling Fetch Request — `handleFetchRequest` Handler

```
handleFetchRequest(request: RequestChannel.Request): Unit
```

`handleFetchRequest` …FIXME

Note	`handleFetchRequest` is used exclusively to handle Fetch requests.

Handling Metadata Request — `handleTopicMetadataRequest` Method

```
handleTopicMetadataRequest(request: RequestChannel.Request): Unit
```

`handleTopicMetadataRequest` takes the body (from the input `request`) as `MetadataRequest`.

Caution	FIXME

Note	`handleTopicMetadataRequest` is used exclusively to handle Metadata requests.

`authorize` Internal Method

Caution	FIXME

Handling CreatePartitions Request — `handleCreatePartitionsRequest` Handler

```
handleCreatePartitionsRequest(request: RequestChannel.Request): Unit
```

`handleCreatePartitionsRequest` …FIXME

Note	`handleCreatePartitionsRequest` is used when…FIXME

Handling DeleteTopics Request — `handleDeleteTopicsRequest` Handler

```
handleDeleteTopicsRequest(request: RequestChannel.Request): Unit
```

`handleDeleteTopicsRequest` …FIXME

Note	`handleDeleteTopicsRequest` is used when…FIXME

Handling ControlledShutdown Request — `handleControlledShutdownRequest` Handler

```
handleControlledShutdownRequest(request: RequestChannel.Request): Unit
```

`handleControlledShutdownRequest` …FIXME

Note	`handleControlledShutdownRequest` is used when…FIXME

Creating KafkaApis Instance

`KafkaApis` takes the following when created:

- `RequestChannel`
- ReplicaManager
- AdminManager
- GroupCoordinator
- `TransactionCoordinator`
- KafkaController
- ZkUtils
- Broker ID
- KafkaConfig
- MetadataCache
- `Metrics`
- Optional Authorizer
- `QuotaManagers`
- `BrokerTopicStats`
- Cluster ID
- `Time`

KafkaHealthcheck

`KafkaHealthcheck` registers the broker it runs on with Zookeeper (which in turn makes the broker visible to other brokers that together can form a Kafka cluster).

`KafkaHealthcheck` is created and started when `KafkaServer` is requested to start up.

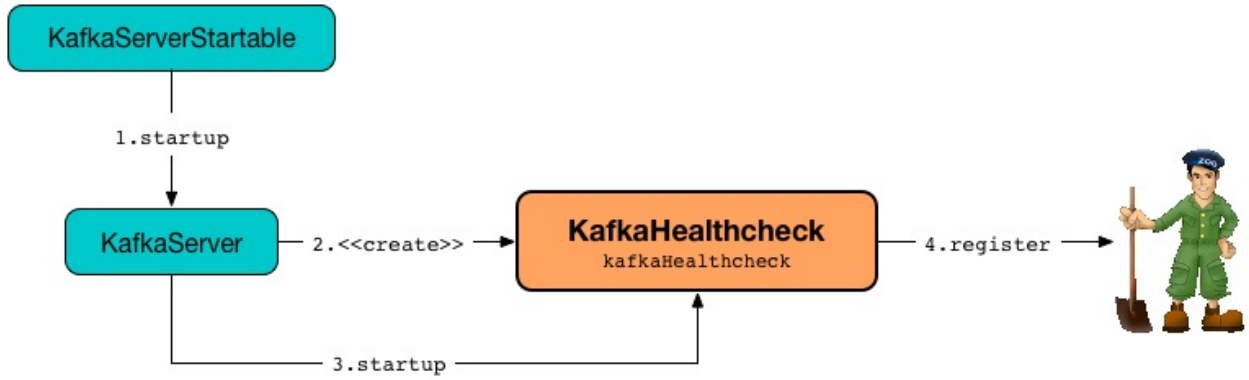

Figure 1. KafkaHealthcheck

Table 1. KafkaHealthcheck's Internal Properties (e.g. Registries and Counters)

Name	Description
`sessionExpireListener`	`SessionExpireListener`

Creating KafkaHealthcheck Instance

`KafkaHealthcheck` takes the following when created:

- Broker ID
- Advertised endpoints
- ZkUtils
- Optional rack name
- `ApiVersion`

`KafkaHealthcheck` initializes the internal registries and counters.

Starting Up — `startup` Method

```
startup
```

`startup` requests ZkUtils to subscribeStateChanges with sessionExpireListener.

In the end, `startup` registers the broker with Zookeeper.

Note	`startup` is used exclusively when `KafkaServer` starts up.

Registering Broker in Zookeeper — `register` Method

```
register(): Unit
```

`register` reads `com.sun.management.jmxremote.port` System property or defaults to `-1`.

For every `EndPoint` with no host assigned (in advertisedEndpoints), `register` assigns the fully-qualified domain name of the local host.

`register` then finds the first `EndPoint` with `PLAINTEXT` security protocol or creates an empty `EndPoint`.

Tip	Define `EndPoint` with `PLAINTEXT` security protocol for older clients to connect.

In the end, `register` requests ZkUtils to registerBrokerInZk for brokerId, the host and port of the `PLAINTEXT` endpoint, the updated endpoints, the JMX port, the optional rack and protocol version.

Note	`register` makes a broker visible for other brokers to form a Kafka cluster.

Note	`register` is used when `KafkaHealthcheck` starts up and handles a new session.

`handleNewSession` Method

Caution	FIXME

KafkaServerStartable — Thin Management Layer over KafkaServer

`KafkaServerStartable` is a thin management layer to manage a single KafkaServer instance.

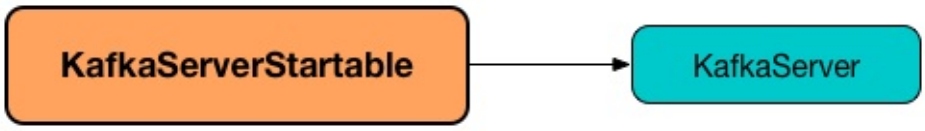

Figure 1. KafkaServerStartable manages KafkaServer

`KafkaServerStartable` allows for the KafkaServer instance to be started, shut down and waited for until shutdown.

Table 1. KafkaServerStartable's Internal Properties (e.g. Registries and Counters)

Name	Description
`server`	KafkaServer instance. Created when `KafkaServerStartable` is created.

`awaitShutdown` Method

Caution	FIXME

`shutdown` Method

Caution	FIXME

Creating KafkaServerStartable Instance

`KafkaServerStartable` takes the following when created:

1. KafkaConfig
2. Collection of KafkaMetricsReporters

`KafkaServerStartable` initializes the internal registries and counters.

Creating KafkaServerStartable From Properties — `fromProps` Method

```
fromProps(serverProps: Properties): KafkaServerStartable
```

`fromProps` creates a `KafkaServerStartable` with a custom `serverProps` properties file.

Caution	FIXME

Note	`fromProps` is used when `kafka.Kafka` runs as a standalone command-line application

`startup` Method

```
startup(): Unit
```

`startup` requests the managed KafkaServer to start.

In case of any exceptions, `startup` exits the JVM with status `1`. You should see the following FATAL message in the logs if that happens.

```
FATAL Exiting Kafka.
```

Note	`startup` uses Java's System.exit to terminate a JVM.

Note	`startup` is used when a Kafka Broker starts (on command line).

KafkaServer

`KafkaServer` is a Kafka broker that wires (creates and starts) Kafka services together.

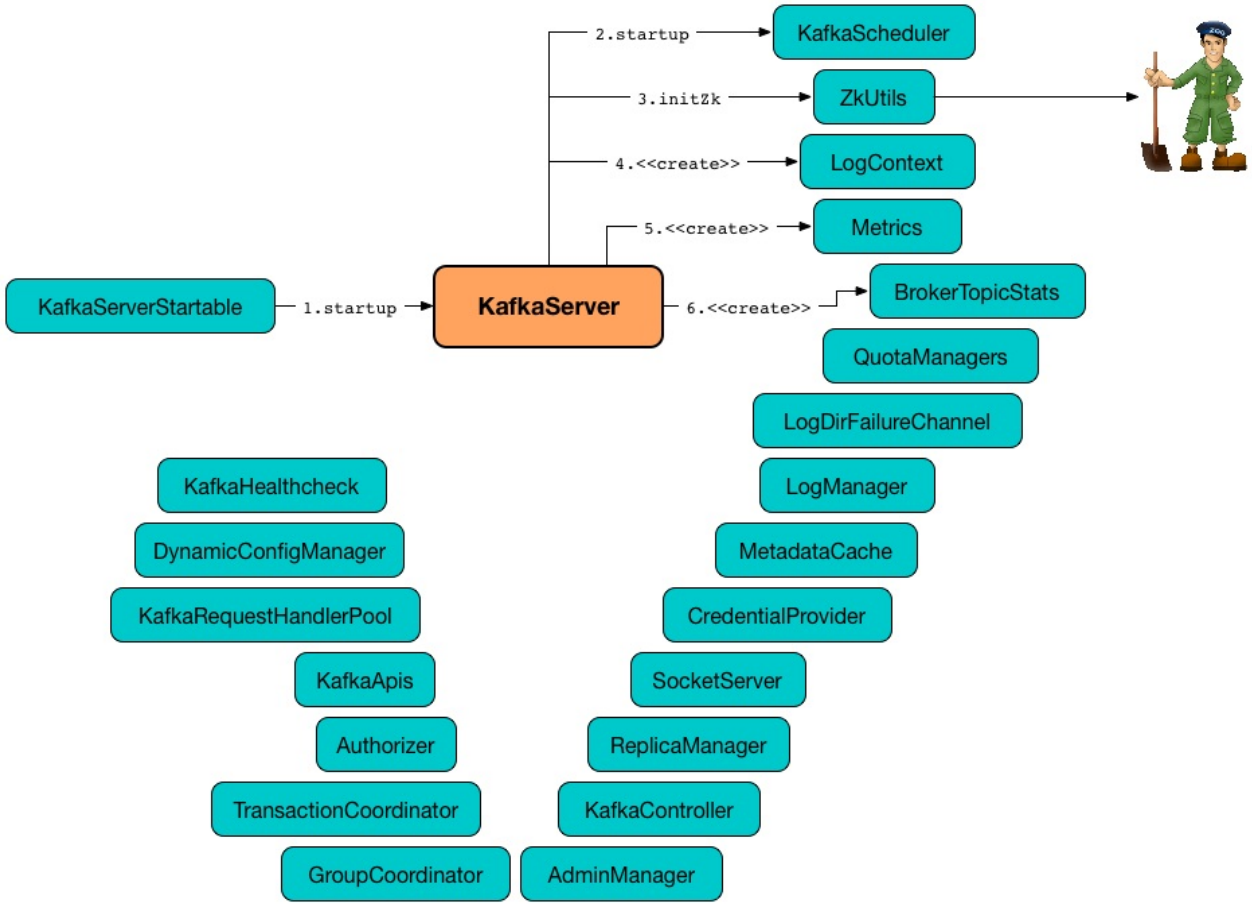

Figure 1. KafkaServer's Startup and Auxiliary Services

`KafkaServer` registers itself in the JMX system under **kafka.server**.

Table 1. KafkaServer's Internal Properties (e.g. Registries and Counters)

Name	Description
`adminManager`	AdminManager
`apis`	KafkaApis
`authorizer`	Optional Authorizer
`brokerState`	`BrokerState`
`_brokerTopicStats`	`BrokerTopicStats`
`_clusterId`	Cluster ID
`credentialProvider`	`CredentialProvider`

`dynamicConfigHandlers`	
`dynamicConfigManager`	DynamicConfigManager
`groupCoordinator`	GroupCoordinator
`isStartingUp`	Flag for…FIXME
`kafkaController`	KafkaController
`kafkaHealthcheck`	KafkaHealthcheck
`kafkaScheduler`	KafkaScheduler with…FIXME
`logContext`	`LogContext`
`logDirFailureChannel`	`LogDirFailureChannel`
`logManager`	LogManager
`metadataCache`	MetadataCache
`replicaManager`	ReplicaManager exclusively to create: • KafkaApis • GroupCoordinator • TransactionCoordinator --- • Created (and started immmediately) when `KafkaServer` starts up • Shut down when `KafkaServer` shuts down
`reporters`	Collection of MetricsReporter Used when…FIXME
`requestHandlerPool`	KafkaRequestHandlerPool
`socketServer`	SocketServer
`transactionCoordinator`	TransactionCoordinator
`quotaManagers`	`QuotaManagers`

`shutdownLatch`	Java's java.util.concurrentCountDownLatch
`startupComplete`	Flag for…FIXME
`zkUtils`	ZkUtils

Getting Broker ID and Initial Offline Directories — `getBrokerIdAndOfflineDirs` Internal Method

Caution	FIXME

`getOrGenerateClusterId` Internal Method

Caution	FIXME

Connecting to Zookeeper — `initZk` Internal Method

Caution	FIXME

`checkpointBrokerId` Internal Method

Caution	FIXME

Creating KafkaServer Instance

`KafkaServer` takes the following when created:

- KafkaConfig
- `Time` (defaults to `Time.SYSTEM`)
- Optional thread name prefix
- A collection of KafkaMetricsReporters (defaults to no reporters)

Caution	FIXME

Note	`KafkaServer` is created when `KafkaServerStartable` is created.

Starting Auxiliary Services — `startup` Method

```
startup(): Unit
```

`startup` starts a single Kafka server.

Internally, `startup` first prints out the following INFO message to the logs:

```
INFO starting (kafka.server.KafkaServer)
```

`startup` sets BrokerState as `Starting`.

`startup` requests KafkaScheduler to start.

`startup` connects to Zookeeper (and initializes ZkUtils).

`startup` getOrGenerateClusterId (that is recorded as cluster id).

You should see the following INFO message in the logs:

```
INFO Cluster ID = [clusterId] (kafka.server.KafkaServer)
```

`startup` gets broker id and initial offline directories.

`startup` creates the `LogContext` with **[KafkaServer id=[brokerId]]** prefix.

`startup` creates and configures metrics.

1. Requests KafkaConfig for configured instances of metric reporters
2. Adds a `JmxReporter` (with **kafka.server** prefix)
3. Creates the `MetricConfig`
4. Initializes Metrics internal registry

`startup` registers broker topic metrics (by initializing BrokerTopicStats).

`startup` initializes QuotaManagers.

`startup` notifies cluster resource listeners (i.e. KafkaMetricsReporters and the configured instances of metric reporters).

`startup` creates the LogDirFailureChannel

`startup` creates the LogManager and requests it to start up.

`startup` creates the MetadataCache (for the broker ID).

`startup` creates the CredentialProvider (per sasl.enabled.mechanisms property).

`startup` creates the SocketServer (for KafkaConfig, Metrics and CredentialProvider) and requests it to start up.

`startup` creates the ReplicaManager and requests it to start up.

`startup` creates the KafkaController (for KafkaConfig, ZkUtils, Metrics and the optional threadNamePrefix) and requests it to start up.

`startup` creates the AdminManager (for KafkaConfig, Metrics, MetadataCache and ZkUtils).

`startup` creates the GroupCoordinator (for KafkaConfig, ZkUtils and ReplicaManager) and requests it to start up.

`startup` creates the TransactionCoordinator (for KafkaConfig, ReplicaManager, a new dedicated KafkaScheduler with `transaction-log-manager-` thread name prefix, ZkUtils, Metrics and MetadataCache) and requests it to start up.

`startup` creates a Authorizer (if defined using authorizer.class.name property) and configures it.

`startup` creates the KafkaApis (for SocketServer, ReplicaManager, AdminManager, GroupCoordinator, TransactionCoordinator, KafkaController, ZkUtils, broker ID, KafkaConfig, MetadataCache, Metrics, Authorizer, QuotaManagers, BrokerTopicStats, cluster ID).

Note	At this point `KafkaServer` may start processing requests.

`startup` creates the KafkaRequestHandlerPool (for broker ID, SocketServer, KafkaApis and num.io.threads).

`startup` starts the HTTP interface of mx4j (if configured).

`startup` creates the DynamicConfigManager (for ZkUtils and dynamicConfigHandlers) and requests it to start up.

`startup` configures the advertised listeners (if defined).

`startup` creates the KafkaHealthcheck (for broker ID, the advertised listeners, ZkUtils, broker.rack and inter.broker.protocol.version Kafka properties) and requests it to start up.

`startup` checkpoints the broker ID.

`startup` sets BrokerState as `RunningAsBroker`, creates the CountDownLatch, enables the startupComplete flag, disables isStartingUp flag

`startup` registers `AppInfo` as an MBean with the MBean server as `kafka.server:type=app-info,id=[brokerId]`.

In the end, you should see the following INFO message in the logs:

```
INFO [Kafka Server [brokerId]], started (kafka.server.KafkaServer)
```

Note	The INFO message above uses so-called **log ident** with the value of `broker.id` property and is always in the format `[Kafka Server [brokerId]],` after a Kafka server has fully started.

Note	`startup` is used exclusively when `KafkaServerStartable` starts up.

Sending Updated Cluster Metadata to ClusterResourceListeners — `notifyClusterListeners` Internal Method

```
notifyClusterListeners(clusterListeners: Seq[AnyRef]): Unit
```

`notifyClusterListeners` creates a ClusterResourceListeners (with the objects from the input `clusterListeners` of type `ClusterResourceListener`) and sends the updated cluster metadata to them.

Note	`notifyClusterListeners` is used exclusively when `KafkaServer` starts up (with `clusterListeners` as kafkaMetricsReporters and the `MetricsReporter` reporters from metric.reporters Kafka property).

Creating ReplicaManager — `createReplicaManager` Internal Method

```
createReplicaManager(isShuttingDown: AtomicBoolean): ReplicaManager
```

`createReplicaManager` simply creates a `ReplicaManager`.

Note	`createReplicaManager` is used exclusively when `KafkaServer` starts up.

KafkaConfig

`KafkaConfig` is the configuration of a Kafka server and the services.

Table 1. KafkaConfig's Configuration Values (in alphabetical order)

Value	Kafka Property
`deleteTopicEnable`	delete.topic.enable
`hostName`	host.name
`listeners`	listeners (see getListeners)
`numNetworkThreads`	num.network.threads
`port`	port
`replicaLagTimeMaxMs`	

`getAdvertisedListeners` Internal Method

Caution	FIXME

`getConfiguredInstances` Method

Caution	FIXME

Creating Listeners — `getListeners` Internal Method

```
getListeners: Seq[EndPoint]
```

`getListeners` creates the `EndPoints` if defined using listeners Kafka property or defaults to `PLAINTEXT://[hostName]:[port]` (for hostName and port Kafka properties).

Note	`getListeners` is used when `KafkaConfig` is created and for getAdvertisedListeners.

KafkaController

`KafkaController` is a Kafka service responsible for:

- topic deletion
- …FIXME

`KafkaController` uses listeners as a notification system to monitor znodes in Zookeeper and react accordingly.

Quoting Kafka Controller Internals:

> In a Kafka cluster, one of the brokers serves as the controller, which is responsible for managing the states of partitions and replicas and for performing administrative tasks like reassigning partitions.

`KafkaController` is created and immediately started when `KafkaServer` starts up.

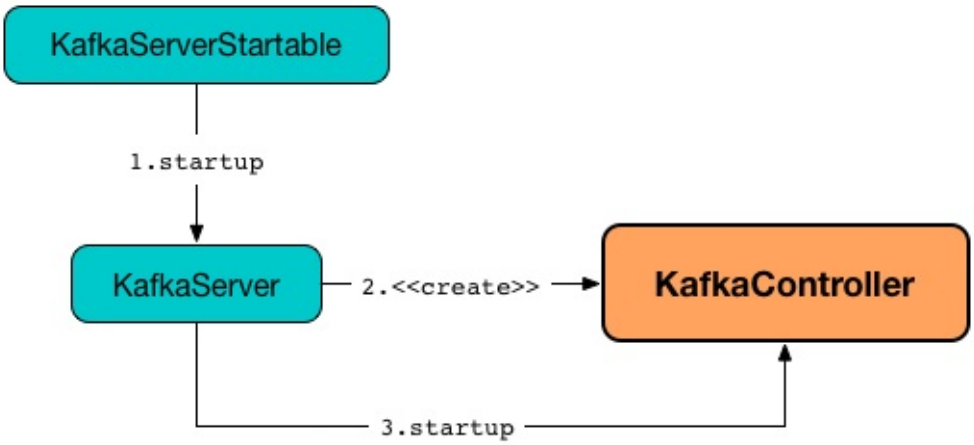

Figure 1. KafkaController

`KafkaController` is part of every Kafka broker, but only one `KafkaController` is active at all times.

`KafkaController` emulates a state machine using controller events.

Table 1. KafkaController's Controller Events

Event	ControllerState	process Handler
`BrokerChange`	`BrokerChange`	
`ControllerChange` • `newControllerId` : Int	`ControllerChange`	1. Assigns the current cont as the input `newControll` 2. (only when the broker is longer an active controlle Resigns as the active co Note: Similar to Reelect with the only differ that it does **not** trig election
`ControlledShutdown` • `ID` • `controlledShutdownCallback` : Try[Set[TopicAndPartition]] ⇒ Unit	`ControlledShutdown`	
`Reelect`	`ControllerChange`	1. Assigns the current cont as activeControllerId 2. (only when the broker is longer an active controlle Resigns as the active co 3. elect
`Startup`	`ControllerChange`	1. registerSessionExpiratio 2. registerControllerChange 3. elect

`logIdent` is **[Controller id=[brokerId]]**.

Table 2. KafkaController's Internal Properties (e.g. Registries and Counters)

Name	Description
`activeControllerId`	The ID of the active `KafkaController` • Initialized to `-1`
`controllerContext`	
`eventManager`	ControllerEventManager for `controllerContext.stats.rateAndTimeMetrics` and updateMetrics listener
`kafkaScheduler`	KafkaScheduler with a single daemon thread with prefix **kafka-scheduler**
`partitionStateMachine`	PartitionStateMachine
`replicaStateMachine`	ReplicaStateMachine
`stateChangeLogger`	`StateChangeLogger` with the broker ID and `inControllerContext` flag enabled
`topicDeletionManager`	TopicDeletionManager

Table 3. KafkaController's Listeners

Listener	Description
`brokerChangeListener`	`BrokerChangeListener` for this `KafkaController` and eventManager
`isrChangeNotificationListener`	`IsrChangeNotificationListener` for this `KafkaController` and eventManager Registered in registerIsrChangeNotificationListener when `KafkaController` does onControllerFailover. De-registered in deregisterIsrChangeNotificationListener when `KafkaController` resigns as the active controller.
`logDirEventNotificationListener`	`LogDirEventNotificationListener`
`partitionModificationsListeners`	`PartitionModificationsListener` by name
`partitionReassignmentListener`	`PartitionReassignmentListener` for this `KafkaController` and ControllerEventManager
`preferredReplicaElectionListener`	`PreferredReplicaElectionListener` for this `KafkaController` and ControllerEventManager
`topicDeletionListener`	`TopicDeletionListener` (for this `KafkaController` and ControllerEventManager) Registered in registerTopicDeletionListener when `KafkaController` does onControllerFailover. De-registered in deregisterTopicDeletionListener when `KafkaController` resigns as the active controller.

Tip	Enable `WARN`, `INFO` or `DEBUG` logging levels for `kafka.controller.KafkaController` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.controller.KafkaController=DEBUG, stdout` Refer to Logging.

`initiateReassignReplicasForTopicPartition` Method

```
initiateReassignReplicasForTopicPartition
```

`initiateReassignReplicasForTopicPartition` …FIXME

Note	`initiateReassignReplicasForTopicPartition` is used when…FIXME

`deregisterPartitionReassignmentIsrChangeListeners` Method

```
deregisterPartitionReassignmentIsrChangeListeners
```

`deregisterPartitionReassignmentIsrChangeListeners` …FIXME

Note	`deregisterPartitionReassignmentIsrChangeListeners` is used when…FIXME

`resetControllerContext` Method

```
resetControllerContext
```

`resetControllerContext` …FIXME

Note	`resetControllerContext` is used when…FIXME

`deregisterBrokerChangeListener` Method

```
deregisterBrokerChangeListener
```

`deregisterBrokerChangeListener` …FIXME

Note	`deregisterBrokerChangeListener` is used when…FIXME

`deregisterTopicChangeListener` Method

```
deregisterTopicChangeListener
```

`deregisterTopicChangeListener` …FIXME

Note	`deregisterTopicChangeListener` is used when…FIXME

Resigning As Active Controller — `onControllerResignation` Method

```
onControllerResignation(): Unit
```

`onControllerResignation` starts by printing out the following DEBUG message to the logs:

```
Resigning
```

`onControllerResignation` unsubscribes from intercepting Zookeeper events for the following znodes in order:

1. Child changes to /isr_change_notification znode
2. Data changes to /admin/reassign_partitions znode
3. Data changes to /admin/preferred_replica_election znode
4. Child changes to /log_dir_event_notification znode

`onControllerResignation` requests TopicDeletionManager to reset.

`onControllerResignation` requests KafkaScheduler to shutdown.

`onControllerResignation` resets the following internal counters:

- offlinePartitionCount
- preferredReplicaImbalanceCount
- globalTopicCount
- globalPartitionCount

`onControllerResignation` deregisterPartitionReassignmentIsrChangeListeners.

`onControllerResignation` requests PartitionStateMachine to shutdown.

`onControllerResignation` deregisterTopicChangeListener.

`onControllerResignation` deregisterPartitionModificationsListener every listener in partitionModificationsListeners.

`onControllerResignation` deregisterTopicDeletionListener.

`onControllerResignation` requests ReplicaStateMachine to shutdown.

`onControllerResignation` deregisterBrokerChangeListener.

`onControllerResignation` resetControllerContext.

In the end, `onControllerResignation` prints out the following DEBUG message to the logs:

```
Resigned
```

Note	`onControllerResignation` is used when: 1. `ControllerEventThread` processes ControllerChange and Reelect controller events 2. triggerControllerMove 3. `KafkaController` shuts down

Unsubscribing from Child Changes to /isr_change_notification ZNode — `deregisterIsrChangeNotificationListener` Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

`deregisterIsrChangeNotificationListener` prints out the following DEBUG message to the logs:

```
De-registering IsrChangeNotificationListener
```

`deregisterIsrChangeNotificationListener` requests ZkUtils to unsubscribe from intercepting changes to `/isr_change_notification` znode with IsrChangeNotificationListener.

Note	`deregisterIsrChangeNotificationListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Child Changes to /log_dir_event_notification ZNode — `deregisterLogDirEventNotificationListener` Internal Method

```
deregisterLogDirEventNotificationListener(): Unit
```

`deregisterLogDirEventNotificationListener` prints out the following DEBUG message to the logs:

```
De-registering logDirEventNotificationListener
```

`deregisterLogDirEventNotificationListener` requests ZkUtils to unsubscribe from intercepting changes to `/log_dir_event_notification` znode with LogDirEventNotificationListener.

Note	`deregisterLogDirEventNotificationListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Data Changes to /admin/preferred_replica_election ZNode — `deregisterPreferredReplicaElectionListener` Method

```
deregisterPreferredReplicaElectionListener(): Unit
```

`deregisterPreferredReplicaElectionListener` requests ZkUtils to unsubscribe from intercepting data changes to `/admin/preferred_replica_election` znode with PreferredReplicaElectionListener.

Note	`deregisterPreferredReplicaElectionListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Data Changes to /admin/reassign_partitions ZNode — `deregisterPartitionReassignmentListener` Method

```
deregisterPartitionReassignmentListener(): Unit
```

`deregisterPartitionReassignmentListener` requests ZkUtils to unsubscribe from intercepting data changes to `/admin/reassign_partitions` znode with PartitionReassignmentListener.

Note	`deregisterPartitionReassignmentListener` is used exclusively when `KafkaController` resigns as the active controller.

`triggerControllerMove` Internal Method

```
triggerControllerMove(): Unit
```

`triggerControllerMove` …FIXME

Note	`triggerControllerMove` is used when: 1. `KafkaController` handleIllegalState 2. `KafkaController` caught an exception while electing or becoming a controller

`handleIllegalState` Internal Method

```
handleIllegalState(e: IllegalStateException): Nothing
```

`handleIllegalState` …FIXME

Note	`handleIllegalState` is used when `KafkaController` catches a `IllegalStateException` in updateLeaderEpochAndSendRequest, sendUpdateMetadataRequest and ControlledShutdown event.

`sendUpdateMetadataRequest` Method

```
sendUpdateMetadataRequest(): Unit
```

`sendUpdateMetadataRequest` …FIXME

Note	`sendUpdateMetadataRequest` is used when…FIXME

`updateLeaderEpochAndSendRequest` Internal Method

```
updateLeaderEpochAndSendRequest(): Unit
```

`updateLeaderEpochAndSendRequest` …FIXME

Note	`updateLeaderEpochAndSendRequest` is used when…FIXME

`shutdown` Method

```
shutdown(): Unit
```

`shutdown` …FIXME

Note	`shutdown` is used when…FIXME

`updateMetrics` Internal Method

Caution	FIXME

`onBrokerStartup` Method

```
onBrokerStartup(newBrokers: Seq[Int]): Unit
```

`onBrokerStartup` …FIXME

Note	`onBrokerStartup` is used exclusively when `KafkaController` processes `BrokerChange` controller event.

`elect` Method

```
elect(): Unit
```

`elect` …FIXME

Note	`elect` is used when `KafkaController` enters `Startup` and `Reelect` states.

`onControllerFailover` Method

Caution	FIXME

Note	`onControllerFailover` is used exclusively when `KafkaController` is requested to elect.

`isActive` Method

```
isActive: Boolean
```

`isActive` says whether the activeControllerId equals the broker ID (from KafkaConfig).

Caution	FIXME When could they be different?

`registerIsrChangeNotificationListener` Internal Method

```
registerIsrChangeNotificationListener(): Option[Seq[String]]
```

`registerIsrChangeNotificationListener` …FIXME

Note	`registerIsrChangeNotificationListener` is used when…FIXME

`deregisterIsrChangeNotificationListener` Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

`deregisterIsrChangeNotificationListener` …FIXME

Note	`deregisterIsrChangeNotificationListener` is used when…FIXME

Creating KafkaController Instance

`KafkaController` takes the following when created:

- KafkaConfig
- ZkUtils
- `Time`
- `Metrics`
- Optional thread name prefix

`KafkaController` initializes the internal registries and counters.

Starting ControllerEventManager (and Putting Startup Event in Event Queue) — `startup` Method

```
startup(): Unit
```

`startup` puts `Startup` event at the end of the event queue of ControllerEventManager and requests it to start.

Note	`startup` is used exclusively when `KafkaServer` is started up.

Registering SessionExpirationListener To Control Session Recreation — `registerSessionExpirationListener` Internal Method

```
registerSessionExpirationListener(): Unit
```

`registerSessionExpirationListener` requests ZkUtils to subscribe to state changes with a `SessionExpirationListener` (with the `KafkaController` and ControllerEventManager).

Note	`SessionExpirationListener` puts Reelect event on the event queue of `ControllerEventManager` every time the Zookeeper session has expired and a new session has been created.

Note	`registerSessionExpirationListener` is used exclusively when Startup event is processed (after `ControllerEventThread` is started).

Registering ControllerChangeListener for /controller ZNode Changes — `registerControllerChangeListener` Internal Method

```
registerControllerChangeListener(): Unit
```

`registerControllerChangeListener` requests ZkUtils to subscribe to data changes for `/controller` znode with a `ControllerChangeListener` (with the `KafkaController` and ControllerEventManager).

Note	`ControllerChangeListener` emits: 1. ControllerChange event with the current controller ID (on the event queue of `ControllerEventManager`) every time the data of a znode changes 2. Reelect event when the data associated with a znode has been deleted

Note	`registerControllerChangeListener` is used exclusively when Startup event is processed (after `ControllerEventThread` is started).

`registerBrokerChangeListener` Internal Method

```
registerBrokerChangeListener(): Option[Seq[String]]
```

`registerBrokerChangeListener` requests ZkUtils to subscribeChildChanges for `/brokers/ids` path with BrokerChangeListener.

Note	`registerBrokerChangeListener` is used exclusively when `KafkaController` does onControllerFailover.

Getting Active Controller ID (from JSON under /controller znode) — `getControllerID` Method

```
getControllerID(): Int
```

`getControllerID` returns the ID of the active Kafka controller that is associated with `/controller` znode in JSON format or `-1` otherwise.

Internally, `getControllerID` requests ZkUtils for data associated with `/controller` znode.

If available, `getControllerID` parses the data (being the current controller info in JSON format) to extract `brokerid` field.

```
$ ./bin/zookeeper-shell.sh 0.0.0.0:2181
Connecting to 0.0.0.0:2181
Welcome to ZooKeeper!
...
get /controller
{"version":1,"brokerid":100,"timestamp":"1506197069724"}
cZxid = 0xf9
ctime = Sat Sep 23 22:04:29 CEST 2017
mZxid = 0xf9
mtime = Sat Sep 23 22:04:29 CEST 2017
pZxid = 0xf9
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15eaa3a4fdd000d
dataLength = 56
numChildren = 0
```

Otherwise, when no `/controller` znode is available, `getControllerID` returns `-1`.

Note	`getControllerID` is used when: 1. Processing `Reelect` controller event 2. elect

Registering TopicDeletionListener for Child Changes to /admin/delete_topics ZNode — `registerTopicDeletionListener` Internal Method

```
registerTopicDeletionListener(): Option[Seq[String]]
```

`registerTopicDeletionListener` requests ZkUtils to subscribeChildChanges to `/admin/delete_topics` znode with TopicDeletionListener.

Note	`registerTopicDeletionListener` is used exclusively when `KafkaController` does onControllerFailover.

De-Registering TopicDeletionListener for Child Changes to /admin/delete_topics ZNode — `deregisterTopicDeletionListener` Internal Method

```
deregisterTopicDeletionListener(): Unit
```

`deregisterTopicDeletionListener` requests ZkUtils to unsubscribeChildChanges to `/admin/delete_topics` znode with TopicDeletionListener.

Note	`deregisterTopicDeletionListener` is used exclusively when `KafkaController` resigns as the active controller.

ControllerEventManager

`ControllerEventManager` is…FIXME

`ControllerEventManager` is created when `KafkaController` is created.

`ControllerEventManager` is started when `KafkaController` is started up.

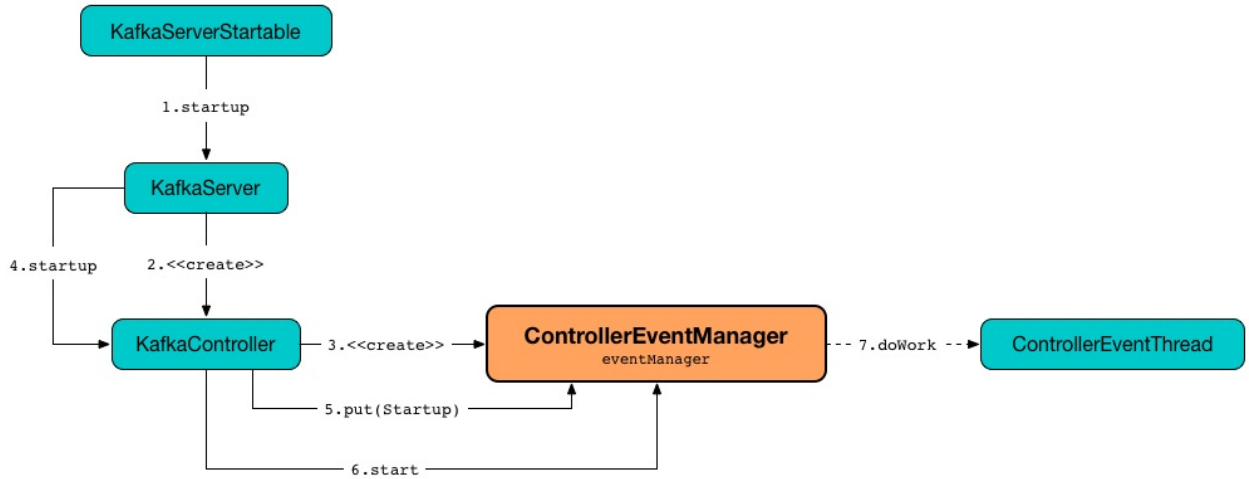

Figure 1. ControllerEventManager is Created and Started Alongside KafkaController

Table 1. ControllerEventManager's Internal Properties (e.g. Registries and Counters)

Name	Description
`queue`	Java's java.util.concurrent.LinkedBlockingQueue (i.e. an optionally-bounded blocking queue based on linked nodes that orders elements in first-in-first-out fashion) of ControllerEvents.
`_state`	`ControllerState` with `Idle` being the initial state
`thread`	ControllerEventThread with **controller-event-thread** thread name

Creating ControllerEventManager Instance

`ControllerEventManager` takes the following when created:

- `rateAndTimeMetrics` collection of `ControllerState` and `KafkaTimer`
- `eventProcessedListener` Procedure of `ControllerEvent`

`ControllerEventManager` initializes the internal registries and counters.

Inserting Controller Event to Event Queue — `put` Method

```
put(event: ControllerEvent): Unit
```

`put` inserts `event` at the tail of event queue.

Note	`put` is used when…FIXME

Starting ControllerEventManager (and ControllerEventThread) — `start` Method

```
start(): Unit
```

`start` requests ControllerEventThread to do the work.

Note	`ControllerEventThread` is a `ShutdownableThread` that, once started, triggers `doWork()` method.

Note	`start` is used exclusively when `KafkaController` is started up.

ControllerEventThread

`ControllerEventThread` is a ShutdownableThread that is started when `ControllerEventManager` is started

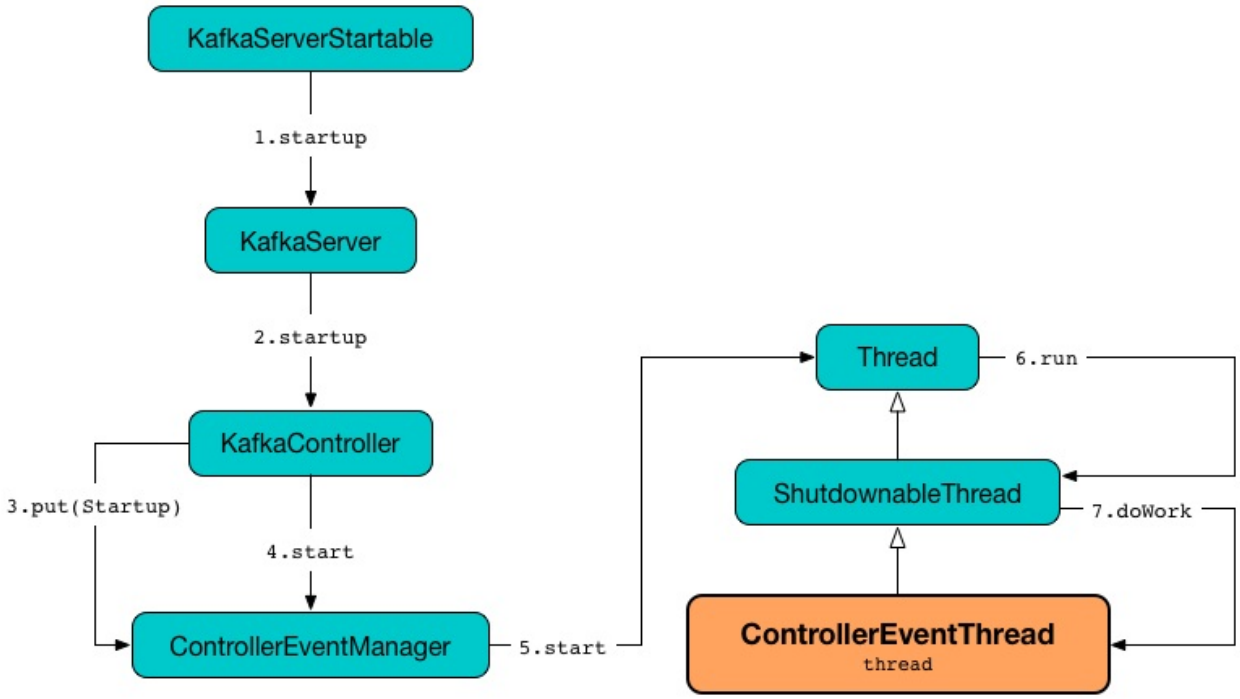

Figure 1. ControllerEventThread is Started Alongside ControllerEventManager

When created, `ControllerEventThread` takes the name of the thread (which `ControllerEventManager` sets as **controller-event-thread**).

```
"controller-event-thread" #44 prio=5 os_prio=31 tid=0x00007fac45730800 nid=0xad03 wait
ing on condition [0x0000000178b30000]
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000007bcb03938> (a java.util.concurrent.locks.Abst
ractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await
(AbstractQueuedSynchronizer.java:2039)
        at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
        at kafka.controller.ControllerEventManager$ControllerEventThread.doWork(Contro
llerEventManager.scala:48)
        at kafka.utils.ShutdownableThread.run(ShutdownableThread.scala:64)
```

Processing ControllerEvents — `doWork` Method

```
doWork(): Unit
```

`doWork` takes and removes the head of event queue (waiting if necessary until an element becomes available).

Note	The very first event in the event queue is `Startup` that `KafkaController` puts when it is started.

`doWork` sets _state (of `ControllerEventManager`) as the state of the event.

`doWork` finds the `KafkaTimer` for the state in rateAndTimeMetrics lookup table (of `ControllerEventManager`).

`doWork` processes the event (i.e. calls `ControllerEvent.process` method).

In the end, `doWork` passes the event to eventProcessedListener (of `ControllerEventManager`) and sets _state (of `ControllerEventManager`) as `Idle`.

ControllerEvent

`ControllerEvent` is the contract of events in the lifecycle of KafkaController state machine that, once emitted, triggers state change and the corresponding process action.

```
package kafka.controller

sealed trait ControllerEvent {
  def state: ControllerState
  def process(): Unit
}
```

Note	`ControllerEvent` is a Scala sealed trait and so all the available events are in a single compilation unit (i.e. a file).

Table 1. ControllerEvent Contract

Method	Description
`state`	`ControllerState` of the ControllerEventManager Used when `ControllerEventThread` does the work
`process`	Used when `ControllerEventThread` does the work to trigger an action associated with state change.

Table 2. Known ControllerEvents

ControllerEvent	ControllerState
TopicDeletion	`TopicDeletion`

TopicDeletion Controller Event

`TopicDeletion` is a ControllerEvent that is executed on the active `KafkaController` (and does nothing otherwise).

Note	`TopicDeletion` uses delete.topic.enable Kafka property.

Note	Topics to be deleted are created in `/admin/delete_topics` path in Zookeeper.

`state` is `TopicDeletion`.

`process` Method

```
process(): Unit
```

Note	`process` is a part of ControllerEvent Contract.

Note	`process` is executed on the active controller only (and does nothing otherwise).

`process` prints out the following DEBUG message to the logs:

```
Delete topics listener fired for topics [topicsToBeDeleted] to be deleted
```

`process` requests ControllerContext for `allTopics` and finds topics that are supposed to be deleted, but are not available in the Kafka cluster.

If there are any non-existent topics, `process` prints out the following WARN message to the logs and requests ZkUtils to deletePathRecursive `/admin/delete_topics/[topicName]` znode for every topic in the list.

```
Ignoring request to delete non-existing topics [nonExistentTopics]
```

`process` branches off per delete.topic.enable Kafka property.

`process` with `delete.topic.enable` Enabled

With delete.topic.enable enabled (i.e. `true`), `process` prints out the following INFO message to the logs:

```
Starting topic deletion for topics [topicsToBeDeleted]
```

`process` requests TopicDeletionManager to markTopicIneligibleForDeletion for topics to be deleted with partitions in `controllerContext.partitionsBeingReassigned` list.

`process` requests TopicDeletionManager to enqueueTopicsForDeletion.

`process` with `delete.topic.enable` Disabled

With delete.topic.enable disabled (i.e. `false`), `process` prints out the following INFO message to the logs (for every topic):

```
Removing /admin/delete_topics/[topicName] since delete topic is disabled
```

`process` requests ZkUtils to deletePath `/admin/delete_topics/[topicName]` znode (for every topic).

ControllerBrokerRequestBatch

`ControllerBrokerRequestBatch` is...FIXME

`sendRequestsToBrokers` Method

```
sendRequestsToBrokers(controllerEpoch: Int): Unit
```

`sendRequestsToBrokers` ...FIXME

Note	`sendRequestsToBrokers` is used when:

- `KafkaController` updateLeaderEpochAndSendRequest, sendUpdateMetadataRequest and `ControlledShutdown` is processed.
- `PartitionStateMachine` handleStateChanges and triggerOnlinePartitionStateChange
- `ReplicaStateMachine` handleStateChanges

KafkaMetricsReporter

Caution	FIXME

KafkaRequestHandler

`KafkaRequestHandler` is a thread of execution (i.e. Java's Runnable) that is responsible for relaying client requests (from RequestChannel) to KafkaApis (except `ShutdownRequest` requests that are handled directly).

`KafkaRequestHandler` is created exclusively when `KafkaRequestHandlerPool` is created (and starts the internal `runnables` threads).

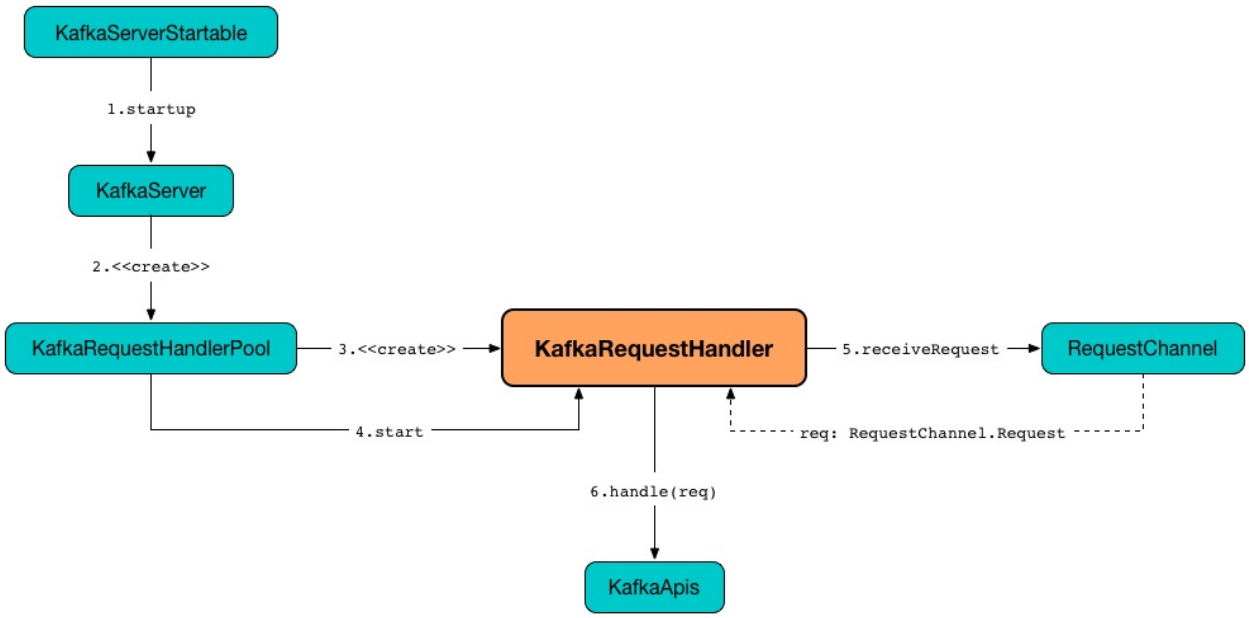

Figure 1. KafkaRequestHandler's Startup and Request Relay

`logIdent` is **[Kafka Request Handler [id] on Broker [brokerId]]**.

Tip	Enable `DEBUG` or `TRACE` logging levels for `kafka.server.KafkaRequestHandler` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.server.KafkaRequestHandler=TRACE` Refer to Logging.

Starting Thread — `run` Method

```
run(): Unit
```

Caution	FIXME

Note	`run` is used when `KafkaRequestHandlerPool` is created.

Creating KafkaRequestHandler Instance

`KafkaRequestHandler` takes the following when created:

- ID
- Broker ID
- Aggregate Idle `Meter`
- Total number of handler threads
- `RequestChannel`
- KafkaApis
- `Time`

`KafkaRequestHandler` initializes the internal registries and counters.

KafkaRequestHandlerPool — Pool of Daemon KafkaRequestHandler Threads

`KafkaRequestHandlerPool` is a pool of daemon **kafka-request-handler** threads that are started immediately when `KafkaRequestHandlerPool` is created.

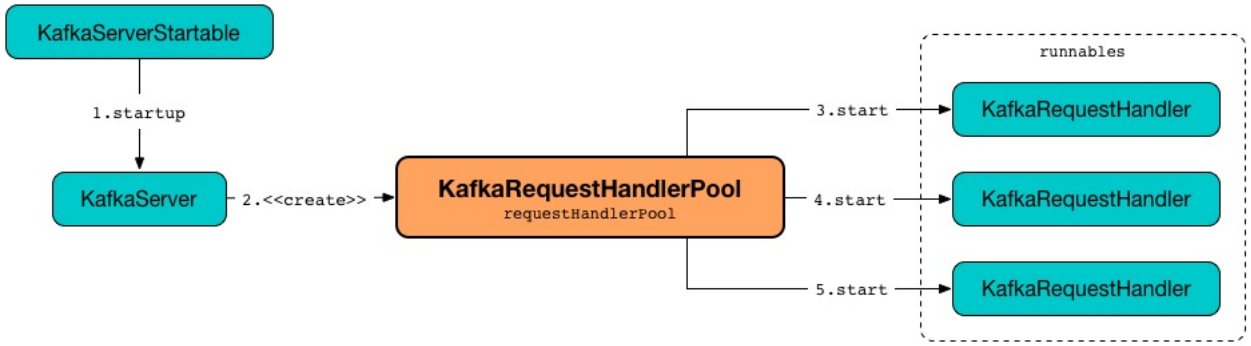

Figure 1. KafkaRequestHandlerPool and KafkaRequestHandler Threads

Note	The number of **kafka-request-handler** threads is controlled by num.network.threads Kafka property and defaults to `3`.

`KafkaRequestHandlerPool` is created exclusively when `KafkaServer` is started.

`logIdent` is **[Kafka Request Handler on Broker [brokerId]]**.

Table 1. KafkaRequestHandlerPool's Internal Properties (e.g. Registries and Counters)

Name	Description
`aggregateIdleMeter`	
`runnables`	Collection of KafkaRequestHandler

`shutdown` Method

Caution	FIXME

Creating KafkaRequestHandlerPool Instance

`KafkaRequestHandlerPool` takes the following when created:

- Broker ID
- `RequestChannel`
- KafkaApis

- `Time`
- Number of threads (i.e. instances of KafkaRequestHandlers as defined by numNetworkThreads property)

`KafkaRequestHandlerPool` initializes the internal registries and counters and starts numThreads daemon **kafka-request-handler** threads (as registered in runnables).

KafkaScheduler

`KafkaScheduler` is a Scheduler to schedule tasks in Kafka.

Table 1. KafkaScheduler's Internal Properties (e.g. Registries and Counters)

Name	Description
`executor`	Java's ScheduledThreadPoolExecutor used to schedule tasks. Initialized when `KafkaScheduler` starts up and shut down when `KafkaScheduler` shuts down.

Tip	Enable `INFO` or `DEBUG` logging levels for `kafka.utils.KafkaScheduler` logger to see what happens in `KafkaScheduler`. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.utils.KafkaScheduler=DEBUG, stdout` Refer to Logging.

Starting Up — `startup` Method

```
startup(): Unit
```

Note	`startup` is a part of Scheduler contract.

When `startup` is executed, you should see the following DEBUG message in the logs:

```
DEBUG Initializing task scheduler. (kafka.utils.KafkaScheduler)
```

`startup` initializes executor with `threads` threads. The name of the threads is in format of `threadNamePrefix` followed by `schedulerThreadId`, e.g. `kafka-scheduler-0`

Note	`threads` and `threadNamePrefix` are defined when `KafkaScheduler` is created.

If `KafkaScheduler` is already started, `startup` throws a `IllegalStateException` with the message:

```
This scheduler has already been started!
```

Creating KafkaScheduler Instance

Caution	FIXME

`shutdown` Method

Caution	FIXME

`ensureRunning` Internal Method

Caution	FIXME

Scheduling Tasks — `schedule` Method

```
schedule(name: String, fun: () => Unit, delay: Long, period: Long, unit: TimeUnit): Un
it
```

Note	`schedule` is a part of Scheduler contract to schedule tasks.

When `schedule` is executed, you should see the following DEBUG message in the logs:

```
DEBUG Scheduling task [name] with initial delay [delay] ms and period [period] ms. (ka
fka.utils.KafkaScheduler)
```

Note	`schedule` uses Java's java.util.concurrent.TimeUnit to convert `delay` and `period` to milliseconds.

`schedule` first makes sure that `KafkaScheduler` is running (which simply means that the internal executor has been initialized).

`schedule` creates an execution thread for the input `fun`.

For positive `period`, `schedule` schedules the thread every `period` after the initial `delay`. Otherwise, `schedule` schedules the thread once.

Note	`schedule` uses the internal executor to schedule `fun` using ScheduledThreadPoolExecutor.scheduleAtFixedRate and ScheduledThreadPoolExecutor.schedule for periodic and one-off executions, respectively.

Whenever the thread is executed, and before `fun` gets triggerred, you should see the following TRACE message in the logs:

```
Beginning execution of scheduled task '[name]'.
```

After the execution thread is finished, you should see the following TRACE message in the logs:

```
Completed execution of scheduled task '[name]'.
```

In case of any exceptions, the execution thread catches them and you should see the following ERROR message in the logs:

```
Uncaught exception in scheduled task '[name]'
```

Scheduler Contract

```
trait Scheduler {
  def startup(): Unit
  def shutdown(): Unit
  def isStarted: Boolean
  def schedule(name: String, fun: () => Unit, delay: Long = 0, period: Long = -1, unit
: TimeUnit = TimeUnit.MILLISECONDS)
}
```

Table 2. Scheduler Contract

Method	Description
`schedule`	Schedules a task

LogDirFailureHandler

`LogDirFailureHandler` is…FIXME

`start` Method

Caution	FIXME

LogManager

Caution	FIXME

`startup` Method

Caution	FIXME

Metadata

`Metadata` describes a Kafka cluster…FIXME

`Metadata` is created right when KafkaConsumer, KafkaProducer, KafkaAdminClient and AdminClient are created.

Table 1. Metadata's Properties When Created by Clients

Client	refreshBackoffMs	metadataExpireMs	allowAutoTopicCr
`KafkaConsumer`	retry.backoff.ms	metadata.max.age.ms	enabled
`KafkaProducer`	retry.backoff.ms	metadata.max.age.ms	enabled
`KafkaAdminClient`	retry.backoff.ms	metadata.max.age.ms	
`AdminClient`			

A (*seemingly*) common usage pattern is as follows:

1. Request `Metadata` for a update (that simply turns the `needUpdate` flag on)
2. Request (indirectly) `KafkaClient` to wake up if blocked on I/O
3. Request `Metadata` to wait for metadata change (i.e. until the metadata version has changed)

Table 2. Metadata's Internal Properties (e.g. Registries and Counters)

Name	Description
`cluster`	Cluster with a subset of the nodes and topic partitions in a Kafka cluster. • Empty (with no nodes and no topic partitions) when `Metadata` is created • Updated when: ◦ `DefaultMetadataUpdater` handles MetadataResponse ◦ `KafkaConsumer`, `KafkaProducer`, `KafkaAdminClient` and `AdminClient` are created and update the cluster with a "bootstrap" cluster with bootstrap brokers • Can be accessed using fetch
`listeners`	

`lastRefreshMs`	The time (in millis) of the last successful update (and failed update) • Used in timeToNextUpdate • Starts `0` when `Metadata` is created • Reset (to `0`) in requestUpdateForNewTopics
`lastSuccessfulRefreshMs`	
`needMetadataForAllTopics`	Flag…FIXME • Disabled (i.e. `false`) when `Metadata` is created • Updated when `Metadata` is requested to set state to indicate that metadata for all topics in Kafka cluster is required
`needUpdate`	Flag that controls whether a metadata update has been requested (enabled) or not (disabled). • Starts turned off when `Metadata` is created • Turned on exclusively when `Metadata` is requested for an update • Turned off when `Metadata` is updated Use updateRequested to know the current value.
`version`	Metadata version • `0` when `Metadata` is created • Incremented every update

Tip	Enable `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.Metadata` logger to see what happens inside. Add the following line to `config/tools-log4j.properties`: `log4j.logger.org.apache.kafka.clients.Metadata=TRACE` Refer to Logging.

Checking if Metadata Update was Requested — `updateRequested` Method

```
synchronized boolean updateRequested()
```

`updateRequested` …FIXME

Note	`updateRequested` is used when: • `DefaultMetadataUpdater` handles an authentication failure • `ConsumerCoordinator` polls for coordinator events • `ConsumerNetworkClient` makes sure that the metadata is fresh

Recording Update Request Failure — `failedUpdate` Method

```
synchronized void failedUpdate(long now, AuthenticationException authenticationExcepti
on)
```

`failedUpdate` …FIXME

Note	`failedUpdate` is used when…FIXME

`getClusterForCurrentTopics` Internal Method

```
Cluster getClusterForCurrentTopics(Cluster cluster)
```

`getClusterForCurrentTopics` …FIXME

Note	`getClusterForCurrentTopics` is used when…FIXME

`timeToNextUpdate` Method

```
synchronized long timeToNextUpdate(long nowMs)
```

`timeToNextUpdate` …FIXME

Note	`timeToNextUpdate` is used when: • `ConsumerNetworkClient` ensureFreshMetadata • `DefaultMetadataUpdater` (of `NetworkClient`) isUpdateDue and maybeUpdate

`add` Method

```
synchronized void add(String topic)
```

`add` …FIXME

Note	`add` is used when…FIXME

`requestUpdate` Method

```
synchronized int requestUpdate()
```

`requestUpdate` …FIXME

Note	`requestUpdate` is used when…FIXME

Waiting for Metadata Update (i.e. Metadata Version Change) — `awaitUpdate` Method

```
synchronized void awaitUpdate(
  final int lastVersion,
  final long maxWaitMs) throws InterruptedException
```

`awaitUpdate` …FIXME

Note	`awaitUpdate` is used when…FIXME

Getting Current Cluster Information — `fetch` Method

```
synchronized Cluster fetch()
```

`fetch` returns current cluster information.

Note	`fetch` is used when…FIXME

Setting Topics to Maintain — `setTopics` Method

Caution	FIXME

Updating Cluster Metadata — `update` Method

```
synchronized void update(Cluster cluster, Set<String> unavailableTopics, long now)
```

`update` turns needUpdate flag off and increments version.

`update` sets lastRefreshMs and lastSuccessfulRefreshMs internal registries to the input `now`.

(only when topicExpiryEnabled is enabled, e.g. KafkaProducer) `update` …FIXME

`update` notifies listeners that the metadata has been updated.

`update` does getClusterForCurrentTopics for the `cluster` when needMetadataForAllTopics flag is on and turns needUpdate flag off (that may have been turned on…FIXME).

`update` sets the cluster to the input `cluster`.

`update` prints out the cluster ID and notifies clusterResourceListeners that cluster has changed (only for a non-bootstrap cluster).

```
Cluster ID: [clusterId]
```

Note	`update` is used when: • `DefaultMetadataUpdater` handles MetadataResponse • `KafkaConsumer` is created (and updates the cluster with a "bootstrap" cluster with bootstrap servers) • `KafkaProducer` is created (and updates the cluster with a "bootstrap" cluster with bootstrap servers) • `KafkaAdminClient` is created (and updates the cluster with a "bootstrap" cluster with bootstrap brokers) • `AdminClient` is created (and updates the cluster with a "bootstrap" cluster with bootstrap brokers)

Creating Metadata Instance

`Metadata` takes the following when created:

- `refreshBackoffMs`
- `metadataExpireMs`
- `allowAutoTopicCreation` flag

- `topicExpiryEnabled` flag
- ClusterResourceListeners

`Metadata` initializes the internal registries and counters.

Conditionally Requesting Update For New Topics (for KafkaConsumer) — `needMetadataForAllTopics` Method

```
synchronized void needMetadataForAllTopics(boolean needMetadataForAllTopics)
```

`needMetadataForAllTopics` requestUpdateForNewTopics when the input `needMetadataForAllTopics` flag is enabled (i.e. `true`) and the current needMetadataForAllTopics is disabled (i.e. `false`).

`needMetadataForAllTopics` sets needMetadataForAllTopics to be the input `needMetadataForAllTopics`.

Note	`needMetadataForAllTopics` is used when `KafkaConsumer`: • Subscribes to topics matching specified pattern (and `needMetadataForAllTopics` flag is then enabled) • Unsubscribes from topics (and `needMetadataForAllTopics` flag is then disabled)

`requestUpdateForNewTopics` Internal Method

```
synchronized void requestUpdateForNewTopics()
```

`requestUpdateForNewTopics` sets lastRefreshMs to `0` and requests update.

Note	`requestUpdateForNewTopics` is used when `Metadata`: • add • needMetadataForAllTopics • setTopics

Metadata Update Listener

`Listener` is the contract of…FIXME

```
package org.apache.kafka.clients;

public final class Metadata {
  public interface Listener {
    void onMetadataUpdate(Cluster cluster, Set<String> unavailableTopics);
  }
}
```

MetadataCache

`MetadataCache` is…FIXME

MetadataResponse

`MetadataResponse` holds information about a Kafka cluster, i.e. the broker nodes, the controller and the topics.

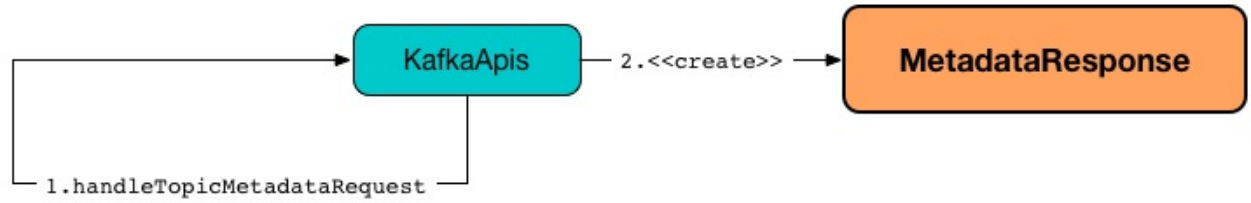

Figure 1. MetadataResponse

`MetadataResponse` is created mainly when `KafkaApis` handles a Metadata request.

`cluster` Method

```
Cluster cluster()
```

`cluster` …FIXME

Note	`cluster` is used when…FIXME

Creating MetadataResponse Instance

`MetadataResponse` takes the following when created:

- `throttleTimeMs`
- Broker nodes
- cluster ID
- controller ID
- Collection of `TopicMetadata`

`MetadataResponse` initializes the internal registries and counters.

MetadataUpdater

MetadataUpdater Contract

```
package org.apache.kafka.clients;

interface MetadataUpdater {
  List<Node> fetchNodes();
  void handleDisconnection(String destination);
  void handleAuthenticationFailure(AuthenticationException exception);
  void handleCompletedMetadataResponse(RequestHeader requestHeader, long now, Metadata
Response metadataResponse);
  boolean isUpdateDue(long now);
  long maybeUpdate(long now);
  void requestUpdate();
}
```

Table 1. MetadataUpdater Contract (in alphabetical order)

Method	Description
`maybeUpdate`	Starts a cluster metadata update if needed and possible. Used exclusively when `NetworkClient` is requested to read and write to sockets.
`requestUpdate`	
`handleAuthenticationFailure`	
`handleCompletedMetadataResponse`	Used exclusively when `NetworkClient` handles completed receives

DefaultMetadataUpdater

`DefaultMetadataUpdater` is a MetadataUpdater that `NetworkClient` uses to…FIXME

`DefaultMetadataUpdater` is created when…FIXME

Table 1. DefaultMetadataUpdater's Internal Properties (e.g. Registries and Counters)

Name	Description
`metadata`	Metadata
`metadataFetchInProgress`	Flag to control whether a cluster metadata update is in progress, i.e. FIXME • Disabled when `DefaultMetadataUpdater` is created • Turned on exclusively when `DefaultMetadataUpdater` does maybeUpdate (with a timestamp and a broker node) • Turned off when `DefaultMetadataUpdater` handles completed metadata response, disconnection or authentication failure

Tip	Enable `WARN`, `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.NetworkClient` logger to see what happens inside.
	Add the following line to `config/tools-log4j.properties` (for Kafka tools):
	`log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG`
	Add the following line to `config/log4j.properties`:
	`log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG, stdout`
	Refer to Logging.

Creating DefaultMetadataUpdater Instance

`DefaultMetadataUpdater` takes the following when created:

- FIXME

`DefaultMetadataUpdater` initializes the internal registries and counters.

`isUpdateDue` Method

Caution	FIXME

`maybeUpdate` Internal Method (with timestamp only)

```
maybeUpdate(long now)
```

Note	`maybeUpdate` is a part of MetadataUpdater Contract.

`maybeUpdate` requests Metadata for timeToNextUpdate (with the input `now`).

`maybeUpdate` takes requestTimeoutMs for the time to wait till metadata fetch in progress finishes if metadataFetchInProgress flag is turned on or `0` otherwise.

`maybeUpdate` takes the maximum of the two values above to check if the current cluster metadata has expired.

If not, `maybeUpdate` gives the maximum value (that says how long to wait till the current cluster metadata expires).

Otherwise, `maybeUpdate` selects the node to request a cluster metadata from and maybeUpdate (with the input `now` timestamp and the node).

If no node was found, `maybeUpdate` prints out the following DEBUG message to the logs and gives reconnectBackoffMs.

```
Give up sending metadata request since no node is available
```

`maybeUpdate` Internal Method (with timestamp and node)

```
long maybeUpdate(long now, Node node)
```

`maybeUpdate` …FIXME

Note	`maybeUpdate` is used exclusively when `DefaultMetadataUpdater` is requested to maybeUpdate (with the timestamp only).

`handleAuthenticationFailure` Callback Method

```
void handleAuthenticationFailure(AuthenticationException exception)
```

Note	`handleAuthenticationFailure` is a part of MetadataUpdater Contract.

`handleCompletedMetadataResponse` turns metadataFetchInProgress flag off.

`handleCompletedMetadataResponse` asks Metadata whether metadata update was requested and if so requests it to record a failure (passing on the `exception`).

`handleCompletedMetadataResponse` Callback Method

```
void handleCompletedMetadataResponse(RequestHeader requestHeader, long now, MetadataResponse response)
```

Note	`handleCompletedMetadataResponse` is a part of MetadataUpdater Contract.

`handleCompletedMetadataResponse` turns metadataFetchInProgress flag off.

`handleCompletedMetadataResponse` takes the cluster from the `response`.

`handleCompletedMetadataResponse` requests Metadata to update (with the cluster and unavailable topics) when there is at least one node in the cluster.

When there are no nodes in the cluster, `handleCompletedMetadataResponse` prints out the following TRACE message to the logs and requests Metadata to record a failure (with no exception).

```
Ignoring empty metadata response with correlation id [correlationId].
```

In case `response` has errors, `handleCompletedMetadataResponse` prints out the following WARN message to the logs:

```
Error while fetching metadata with correlation id [correlationId] : [errors]"
```

NetworkClient — Non-Blocking KafkaClient

`NetworkClient` is a non-blocking KafkaClient that uses Selectable for network communication (i.e. sending and receiving messages).

Note	Selector is the one and only Selectable that uses Java's selectable channels for stream-oriented connecting sockets (i.e. Java's java.nio.channels.SocketChannel).

`NetworkClient` does the actual reads and writes (to sockets) every poll.

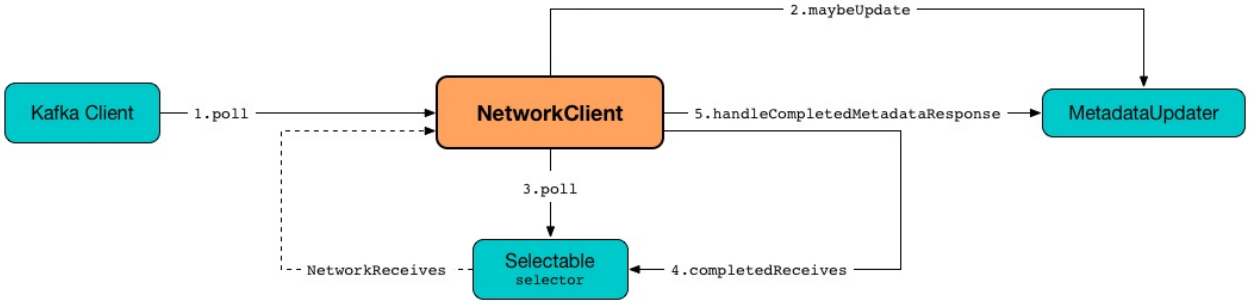

Figure 1. NetworkClient

`NetworkClient` is created when:

- `KafkaConsumer` is created (with `ConsumerNetworkClient`)
- `KafkaProducer` is created (with `Sender`)
- `KafkaAdminClient` is created (using `createInternal`)
- `AdminClient` is created (with `ConsumerNetworkClient`)
- `ControllerChannelManager` does `addNewBroker` (and creates a `RequestSendThread` daemon thread and a `ControllerBrokerStateInfo`)
- `TransactionMarkerChannelManager` is created
- `KafkaServer` does `doControlledShutdown`
- `ReplicaFetcherBlockingSend` is created

Table 1. NetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
`connectionStates`	`ClusterConnectionStates` Used when…FIXME

Tip	Enable `DEBUG` logging level for `org.apache.kafka.clients.NetworkClient` logger to see what happens inside.
	
	Add the following line to `config/tools-log4j.properties` (for Kafka tools):
	
	`log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG`
	
	Add the following line to `config/log4j.properties`:
	
	`log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG, stdout`
	
	Refer to Logging.

Creating ClientRequest — `newClientRequest` Method

```
ClientRequest newClientRequest(
  String nodeId
  AbstractRequest.Builder<?> requestBuilder
  long createdTimeMs,
  boolean expectResponse)
```

`newClientRequest` …FIXME

Note	`newClientRequest` is used when…FIXME

Establishing Connection to Broker Node — `initiateConnect` Internal Method

```
void initiateConnect(Node node, long now)
```

`initiateConnect` prints out the following DEBUG message to the logs:

```
Initiating connection to node [node]
```

`initiateConnect` requests ClusterConnectionStates to enter the connecting state for the connection to the broker `node`.

`initiateConnect` requests Selectable to connect to the broker `node` (at a given host and port).

Note	`initiateConnect` passes the sizes of send and receive buffers for the socket connection.

In case of an IO failure, `initiateConnect` requests ClusterConnectionStates to enter the disconnected state for the connection to the broker `node`.

`initiateConnect` requests MetadataUpdater for update.

You should see the following DEBUG message in the logs:

```
Error connecting to node [node]
```

Note	`initiateConnect` is used when: • `NetworkClient` attempts to connect to a broker node • `DefaultMetadataUpdater` maybeUpdate

`ready` Method

```
boolean ready(Node node, long now)
```

Note	`ready` is a part of KafkaClient Contract.

`ready` …FIXME

Note	`ready` is used when…FIXME

`wakeup` Method

```
void wakeup()
```

Note	`wakeup` is a part of KafkaClient Contract.

`wakeup` simply requests the internal Selectable to wakeup

Note	`wakeup` is used when…FIXME

Reading and Writing to Sockets — `poll` Method

```
List<ClientResponse> poll(long timeout, long now)
```

Note	`poll` is a part of KafkaClient Contract.

`poll` requests MetadataUpdater for cluster metadata update (if needed and possible).

`poll` then requests Selectable to poll.

In the end, `poll` handles completed request sends, receives, disconnected connections, records any connections to new brokers, initiates API version requests, expire in-flight requests, and finally triggers their `RequestCompletionHandlers`.

In case abortedSends is not empty, `poll` creates a collection of `ClientResponse` with abortedSends, triggers their `RequestCompletionHandlers` and returns them.

`handleCompletedReceives` Method

```
void handleCompletedReceives(List<ClientResponse> responses, long now)
```

`handleCompletedReceives` …FIXME

Note	`handleCompletedReceives` is used exclusively when `NetworkClient` polls.

Creating NetworkClient Instance

`NetworkClient` takes the following when created:

Arguments	Description
MetadataUpdater	
Metadata	
Selectable	
Client ID	
`maxInFlightRequestsPerConnection`	
`reconnectBackoffMs`	
`reconnectBackoffMax`	
`socketSendBuffer`	Size of the TCP send buffer (SO_SNDBUF) for socket connection (in bytes) Note: Use send.buffer.bytes property to configure it. Used when `NetworkClient` establishes connection to a broker node.
`socketReceiveBuffer`	Size of the TCP receive buffer (SO_RCVBUF) for socket connection (in bytes) Note: Use receive.buffer.bytes property to configure it. Used when `NetworkClient` establishes connection to a broker node
`requestTimeoutMs`	
`Time`	
`discoverBrokerVersions`	Flag…
`ApiVersions`	
Sensor	
`LogContext`	

`NetworkClient` initializes the internal registries and counters.

Informing ClientResponse about Response Being Completed — `completeResponses` Internal Method

```
void completeResponses(List<ClientResponse> responses)
```

`completeResponses` informs every `ClientResponse` (in the input `responses`) that a response has been completed.

In case of any exception, `completeResponses` prints out the following ERROR message to the logs:

```
Uncaught error in request completion: [exception]
```

Note	`completeResponses` is used when `NetworkClient` poll (for both abortedSends and completed actions).

KafkaClient

`KafkaClient` is the contract for…FIXME

Note	NetworkClient is the one and only implementation.

KafkaClient Contract

```
package org.apache.kafka.clients;

public interface KafkaClient extends Closeable {
  void close(String nodeId);
  long connectionDelay(Node node, long now);
  boolean connectionFailed(Node node);
  void disconnect(String nodeId);
  boolean hasInFlightRequests();
  boolean hasInFlightRequests(String nodeId);
  boolean hasReadyNodes();
  int inFlightRequestCount();
  int inFlightRequestCount(String nodeId);
  boolean isReady(Node node, long now);
  Node leastLoadedNode(long now);
  ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuil
der,
                                 long createdTimeMs, boolean expectResponse);
  ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuil
der, long createdTimeMs,
                                 boolean expectResponse, RequestCompletionHandler call
back);
  boolean ready(Node node, long now);
  List<ClientResponse> poll(long timeout, long now);
  void send(ClientRequest request, long now);
  void wakeup();
}
```

Table 1. KafkaClient Contract

Method	Description
`newClientRequest`	Used when: • …FIXME
`wakeup`	Used when: • …FIXME
`poll`	Used when: • `ConsumerNetworkClient` polls • (Blocking) `NetworkClientUtils` does `awaitReady`, `isReady` or `sendAndReceive` • `AdminClientRunnable` is started (and `run` is executed) • `Sender` is requested to run once • `InterBrokerSendThread` does its work

NetworkClientUtils

`NetworkClientUtils` is…FIXME

`sendAndReceive` Method

```
static ClientResponse sendAndReceive(
  KafkaClient client
  ClientRequest request
  Time time) throws IOException
```

`sendAndReceive` …FIXME

Note	`sendAndReceive` is used when…FIXME

Waiting Until Connection to Broker Node is Ready — `awaitReady` Method

```
static boolean awaitReady(
  KafkaClient client
  Node node
  Time time
  long timeoutMs) throws IOException
```

`awaitReady` …FIXME

Note	`awaitReady` is used when…FIXME

OffsetConfig

`OffsetConfig` is…FIXME

Table 1. OffsetConfig's Properties (in alphabetical order)

Property	Default Value
`offsetsTopicNumPartitions`	`50`

Partition

A Kafka topic is spread across a Kafka cluster as a virtual group of one or more **partitions**.

A single partition of a topic (**topic partition**) can be replicated across a Kafka cluster to one or more Kafka brokers.

A topic partition has one partition leader node and zero or more replicas.

Kafka producers publish messages to topic leaders as do Kafka consumers consume them from.

In-Sync Replicas are brokers that…FIXME

Offline Replicas are…FIXME

`Partition` is…FIXME

Table 1. Partition's Internal Properties (e.g. Registries and Counters)

Name	Description
`leaderReplicaIdOpt`	Optional leader replica ID

`maybeExpandIsr` Method

```
FIXME
```

`maybeExpandIsr` …FIXME

Note	`maybeExpandIsr` is used exclusively when `Partition` does updateReplicaLogReadResult.

`maybeShrinkIsr` Method

```
maybeShrinkIsr(replicaMaxLagTimeMs: Long): Unit
```

`maybeShrinkIsr` …FIXME

Note	`maybeShrinkIsr` is used exclusively when `ReplicaManager` maybeShrinkIsr.

`updateReplicaLogReadResult` Method

```
updateReplicaLogReadResult(replica: Replica, logReadResult: LogReadResult): Boolean
```

`updateReplicaLogReadResult` ...FIXME

Note	`updateReplicaLogReadResult` is used exclusively when `ReplicaManager` updateFollowerLogReadResults.

`updateIsr` Internal Method

```
updateIsr(newIsr: Set[Replica]): Unit
```

`updateIsr` ...FIXME

Note	`updateIsr` is used when `Partition` is requested to expand or shrink the ISR.

`makeLeader` Method

```
makeLeader(
  controllerId: Int,
  partitionStateInfo: LeaderAndIsrRequest.PartitionState,
  correlationId: Int): Boolean
```

`makeLeader` ...FIXME

Note	`makeLeader` is used...FIXME

`makeFollower` Method

```
makeFollower(
  controllerId: Int,
  partitionStateInfo: LeaderAndIsrRequest.PartitionState,
  correlationId: Int): Boolean
```

`makeFollower` ...FIXME

Note	`makeFollower` is used...FIXME

`leaderReplicaIfLocal` Method

```
leaderReplicaIfLocal: Option[Replica]
```

`leaderReplicaIfLocal` gives…FIXME

Note	`leaderReplicaIfLocal` is used…FIXME

`maybeShrinkIsr` Method

Caution	FIXME

Creating Partition Instance

`Partition` takes the following when created:

- Topic name
- Partition ID
- `Time`
- ReplicaManager
- `isOffline` flag (disabled by default)

`Partition` initializes the internal registries and counters.

PartitionStateMachine

`PartitionStateMachine` is…FIXME

`triggerOnlinePartitionStateChange` Method

```
triggerOnlinePartitionStateChange(): Unit
```

`triggerOnlinePartitionStateChange` …FIXME

Note	`triggerOnlinePartitionStateChange` is used when…FIXME

`handleStateChanges` Method

```
handleStateChanges(
  partitions: Set[TopicAndPartition],
  targetState: PartitionState,
  leaderSelector: PartitionLeaderSelector = noOpPartitionLeaderS...,
  callbacks: Callbacks): Unit
```

`handleStateChanges` …FIXME

Note	`handleStateChanges` is used when…FIXME

`shutdown` Method

```
FIXME
```

`shutdown` …FIXME

Note	`shutdown` is used when…FIXME

ReplicaManager

`ReplicaManager` is created and started when `KafkaServer` starts up.

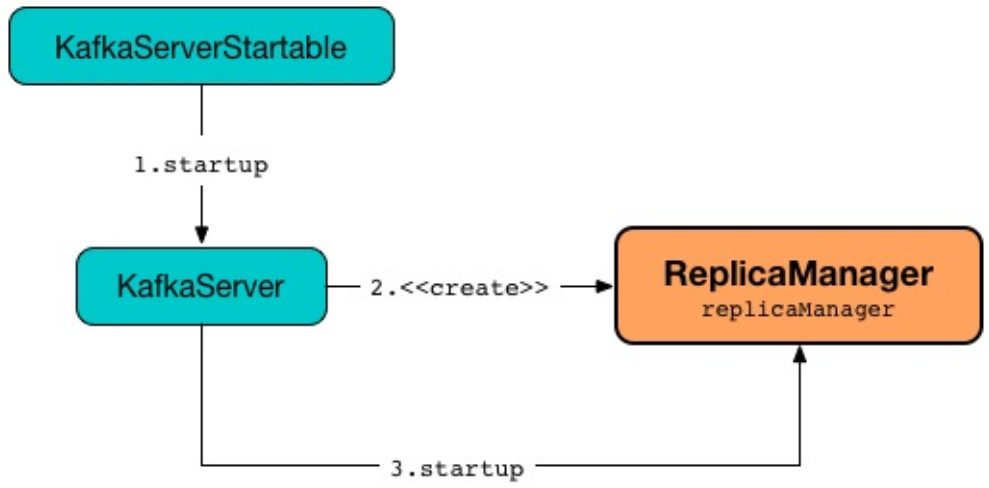

Figure 1. ReplicaManager and KafkaServer

When started, `ReplicaManager` schedules isr-expiration and isr-change-propagation recurring tasks (every half of replica.lag.time.max.ms property and 2500 ms, respectively).

`ReplicaManager` is a `KafkaMetricsGroup`.

Table 1. ReplicaManager's Internal Properties (e.g. Registries and Counters)

Name	Description
`replicaFetcherManager`	ReplicaFetcherManager
`allPartitions`	Pool of `TopicPartition` and Partitions
`isrChangeSet`	Collection of `TopicPartition` that…FIXME
`lastIsrChangeMs`	Time when isrChangeSet has a new `TopicPartition` added.
`logDirFailureHandler`	LogDirFailureHandler
`OfflinePartition`	

`createReplicaFetcherManager` Internal Method

```
createReplicaFetcherManager(
  metrics: Metrics
  time: Time
  threadNamePrefix: Option[String]
  quotaManager: ReplicationQuotaManager): ReplicaFetcherManager
```

`createReplicaFetcherManager` ...FIXME

Note	`createReplicaFetcherManager` is used when...FIXME

`shutdown` Method

```
shutdown(checkpointHW: Boolean = true): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

`alterReplicaLogDirs` Method

```
alterReplicaLogDirs(partitionDirs: Map[TopicPartition, String]): Map[TopicPartition, E
rrors]
```

`alterReplicaLogDirs` ...FIXME

Note	`alterReplicaLogDirs` is used exclusively when `KafkaApis` handles AlterReplicaLogDirs request.

`becomeLeaderOrFollower` Method

```
becomeLeaderOrFollower(
  correlationId: Int,
  leaderAndISRRequest: LeaderAndIsrRequest,
  onLeadershipChange: (Iterable[Partition], Iterable[Partition]) => Unit): BecomeLeade
rOrFollowerResult
```

`becomeLeaderOrFollower` ...FIXME

Note	`becomeLeaderOrFollower` is used exclusively when `KafkaApis` handles LeaderAndIsr request.

`makeFollowers` Internal Method

```
makeFollowers(
  controllerId: Int,
  epoch: Int,
  partitionState: Map[Partition, LeaderAndIsrRequest.PartitionState],
  correlationId: Int,
  responseMap: mutable.Map[TopicPartition, Errors]) : Set[Partition]
```

`makeFollowers` ...FIXME

Note	`makeFollowers` is used exclusively when `ReplicaManager` becomeLeaderOrFollower.

`recordIsrChange` Method

```
recordIsrChange(topicPartition: TopicPartition): Unit
```

`recordIsrChange` adds the input `topicPartition` to isrChangeSet internal registry and sets lastIsrChangeMs to the current time.

Note	`recordIsrChange` is used exclusively when `Partition` does updateIsr

`updateFollowerLogReadResults` Internal Method

```
updateFollowerLogReadResults(
  replicaId: Int,
  readResults: Seq[(TopicPartition, LogReadResult)]): Seq[(TopicPartition, LogReadResu
lt)]
```

`updateFollowerLogReadResults` ...FIXME

Note	`updateFollowerLogReadResults` is used exclusively when `ReplicaManager` fetches messages from the leader replica.

`fetchMessages` Method

```
fetchMessages(
  timeout: Long,
  replicaId: Int,
  fetchMinBytes: Int,
  fetchMaxBytes: Int,
  hardMaxBytesLimit: Boolean,
  fetchInfos: Seq[(TopicPartition, FetchRequest.PartitionData)],
  quota: ReplicaQuota = UnboundedQuota,
  responseCallback: Seq[(TopicPartition, FetchPartitionData)] => Unit,
  isolationLevel: IsolationLevel): Unit
```

`fetchMessages` ...FIXME

Note	`fetchMessages` is used exclusively when `KafkaApis` handles a Fetch request.

`getLeaderPartitions` Internal Method

```
getLeaderPartitions: List[Partition]
```

`getLeaderPartitions` gives the partitions from allPartitions that are not offline and their leaderReplicaIfLocal property is defined.

Note	`getLeaderPartitions` is used when...FIXME

`isr-expiration` Task

Caution	FIXME

`isr-change-propagation` Task

Caution	FIXME

`maybePropagateIsrChanges` Method

```
maybePropagateIsrChanges(): Unit
```

`maybePropagateIsrChanges` ...FIXME

Note	`maybePropagateIsrChanges` is used exclusively when isr-change-propagation task is executed (every 2500 milliseconds).

Creating ReplicaManager Instance

`ReplicaManager` takes the following when created:

- KafkaConfig
- `Metrics`
- `Time`
- ZkUtils
- `Scheduler`
- LogManager
- `isShuttingDown` flag
- ReplicationQuotaManager
- `BrokerTopicStats`
- MetadataCache
- `LogDirFailureChannel`
- `DelayedOperationPurgatory[DelayedProduce]`
- `DelayedOperationPurgatory[DelayedFetch]`
- `DelayedOperationPurgatory[DelayedDeleteRecords]`
- Optional thread name prefix

`ReplicaManager` initializes the internal registries and counters.

Starting ReplicaManager (and Scheduling ISR-Related Tasks) — `startup` Method

```
startup(): Unit
```

`startup` requests Scheduler to schedule the ISR-related tasks:

1. isr-expiration
2. isr-change-propagation

`startup` then creates a LogDirFailureHandler and requests it to start.

Note	`startup` uses `Scheduler` that was specified when `ReplicaManager` was created.

Note	`startup` is used exclusively when `KafkaServer` starts up.

`maybeShrinkIsr` Internal Method

```
maybeShrinkIsr(): Unit
```

`maybeShrinkIsr` prints out the following TRACE message to the logs:

```
TRACE Evaluating ISR list of partitions to see which replicas can be removed from the ISR
```

`maybeShrinkIsr` requests the partitions (from allPartitions pool that are not offline partitions) to maybeShrinkIsr (with replicaLagTimeMaxMs property).

Note	`maybeShrinkIsr` is used exclusively to schedule `isr-expiration` recurring task when `ReplicaManager` starts up.

ReplicaFetcherManager

`ReplicaFetcherManager` is a AbstractFetcherManager that…FIXME (describe properties)

`ReplicaFetcherManager` is created exclusively when `ReplicaManager` is requested to create one (which is when `ReplicaManager` is created).

`createFetcherThread` Method

```
createFetcherThread(fetcherId: Int, sourceBroker: BrokerEndPoint): AbstractFetcherThre
ad
```

`createFetcherThread` …FIXME

Note	`createFetcherThread` is used exclusively when `AbstractFetcherManager` addFetcherForPartitions

Creating ReplicaFetcherManager Instance

`ReplicaFetcherManager` takes the following when created:

- KafkaConfig
- ReplicaManager
- `Metrics`
- `Time`
- Optional thread name prefix (undefined by default)
- ReplicationQuotaManager

`ReplicaFetcherManager` initializes the internal registries and counters.

AbstractFetcherManager

`AbstractFetcherManager` is...FIXME

`addFetcherForPartitions` Method

```
addFetcherForPartitions(partitionAndOffsets: Map[TopicPartition, BrokerAndInitialOffset
]): Unit
```

`addFetcherForPartitions` ...FIXME

Note	`addFetcherForPartitions` is used when: • `ReplicaManager` alterReplicaLogDirs, becomeLeaderOrFollower, makeFollowers • `LeaderFinderThread` (of the currently-deprecated `ConsumerFetcherManager`) does `doWork`

ReplicaFetcherThread

`ReplicaFetcherThread` is a AbstractFetcherThread…FIXME

`ReplicaFetcherThread` is created exclusively when `ReplicaFetcherManager` is requested to create one (when…FIXME).

Creating ReplicaFetcherThread Instance

`ReplicaFetcherThread` takes the following when created:

- Name
- Fetcher ID
- Source `BrokerEndPoint`
- KafkaConfig
- ReplicaManager
- `Metrics`
- `Time`
- ReplicationQuotaManager
- Optional `BlockingSend` (undefined by default)

`ReplicaFetcherThread` initializes the internal registries and counters.

`earliestOrLatestOffset` Internal Method

```
earliestOrLatestOffset(topicPartition: TopicPartition, earliestOrLatest: Long): Long
```

`earliestOrLatestOffset`…FIXME

Note	`earliestOrLatestOffset` is used when…FIXME

`fetchEpochsFromLeader` Method

```
fetchEpochsFromLeader(partitions: Map[TopicPartition, Int]): Map[TopicPartition, Epoch
EndOffset]
```

Note	`fetchEpochsFromLeader` is a part of AbstractFetcherThread Contract.

`fetchEpochsFromLeader` …FIXME

Note	`fetchEpochsFromLeader` is used when…FIXME

AbstractFetcherThread

`AbstractFetcherThread` is…FIXME

ReplicaFetcherBlockingSend

`ReplicaFetcherBlockingSend` is…FIXME

`ReplicaFetcherBlockingSend` is created exclusively when `ReplicaFetcherThread` is created.

`ReplicaFetcherBlockingSend` uses NetworkClient to send requests for…FIXME

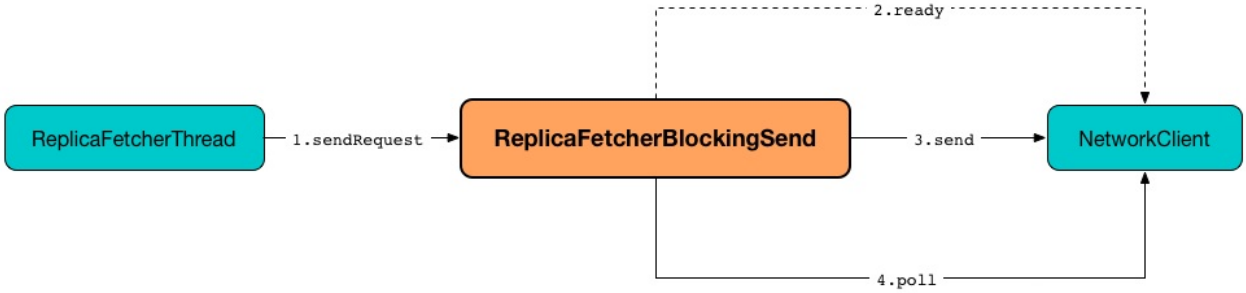

Figure 1. ReplicaFetcherBlockingSend's Sending Client Request and Waiting for Response

`ReplicaFetcherBlockingSend` uses replica.socket.timeout.ms Kafka property for…FIXME

Table 1. ReplicaFetcherBlockingSend's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
`networkClient`	NetworkClient Used when…FIXME
`sourceNode`	`Node` Used when…FIXME

Creating ReplicaFetcherBlockingSend Instance

`ReplicaFetcherBlockingSend` takes the following when created:

- `BrokerEndPoint`
- KafkaConfig
- `Metrics`
- `Time`
- Fetcher ID
- Client ID

- `LogContext`

`ReplicaFetcherBlockingSend` initializes the internal registries and counters.

Sending Client Request and Waiting for Response — `sendRequest` Method

```
sendRequest(requestBuilder: Builder[_ <: AbstractRequest]): ClientResponse
```

`sendRequest` requests `NetworkClientUtils` to wait until the connection is ready to the source broker node (in replica.socket.timeout.ms).

`sendRequest` requests NetworkClient to create a new client request to the source broker.

`sendRequest` requests `NetworkClientUtils` to send the client request and wait for a response.

Note	`sendRequest` is a blocking operation (i.e. blocks the current thread) and polls for responses until the one arrives or a disconnection or a version mismatch happens.

In case `NetworkClientUtils` found the broker node unavailable, `sendRequest` reports a `SocketTimeoutException`:

```
Failed to connect to [sourceNode] within [socketTimeout] ms
```

Note	`sendRequest` is used when `ReplicaFetcherThread` earliestOrLatestOffset and fetchEpochsFromLeader.

`close` Method

```
close(): Unit
```

`close` …FIXME

Note	`close` is used when…FIXME

ReplicationQuotaManager

`ReplicationQuotaManager` is…FIXME

ReplicationUtils

`ReplicationUtils` ...FIXME

`propagateIsrChanges` Method

```
propagateIsrChanges(zkUtils: ZkUtils, isrChangeSet: Set[TopicPartition]): Unit
```

`propagateIsrChanges` ...FIXME

Note	`propagateIsrChanges` is used exclusively when maybePropagateIsrChanges.

ReplicaStateMachine

`ReplicaStateMachine` is…FIXME

`handleStateChanges` Method

```
handleStateChanges(
  replicas: Set[PartitionAndReplica],
  targetState: ReplicaState,
  callbacks: Callbacks): Unit
```

`handleStateChanges` …FIXME

Note	`handleStateChanges` is used when…FIXME

`shutdown` Method

```
FIXME
```

`shutdown` …FIXME

Note	`shutdown` is used when…FIXME

Selector — Selectable on Socket Channels (from Java's New IO API)

`Selector` is the one and only Selectable that uses Java's selectable channels for stream-oriented connecting sockets (i.e. Java's java.nio.channels.SocketChannel).

`Selector` is used by Kafka services to create a `NetworkClient`.

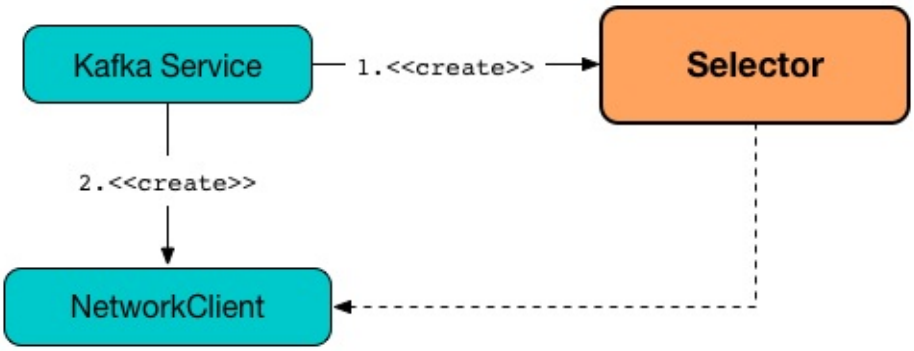

Figure 1. Selector is Created for Kafka Services For NetworkClient

`Selector` is created when:

- `KafkaAdminClient` is created (using `createInternal`)
- `KafkaConsumer` is created
- `KafkaProducer` is created
- `AdminClient` is created
- `ControllerChannelManager` does `addNewBroker`
- `TransactionMarkerChannelManager` is created
- `Processor` is created
- `KafkaServer` does `doControlledShutdown`
- `ReplicaFetcherBlockingSend` is created

`connect` Method

```
void connect(
  String id,
  InetSocketAddress address,
  int sendBufferSize,
  int receiveBufferSize) throws IOException
```

Note	`connect` is a part of Selectable Contract that `NetworkClient` uses when requested to establish a connection to a broker.

`connect` …FIXME

Selectable

`Selectable` is the contract for asynchronous, multi-channel network I/O.

Note	Selector is the one and only `Selectable`.

```
package org.apache.kafka.common.network;

public interface Selectable {
  void connect(String id, InetSocketAddress address, int sendBufferSize, int receiveBu
fferSize) throws IOException;
  void close();
  void close(String id);

  void send(Send send);
  void poll(long timeout) throws IOException;
  void wakeup();

  List<Send> completedSends();
  List<NetworkReceive> completedReceives();

  Map<String, ChannelState> disconnected();
  List<String> connected();

  void mute(String id);
  void unmute(String id);
  void muteAll();
  void unmuteAll();

  boolean isChannelReady(String id);
}
```

Table 1. Selectable Contract (in alphabetical order)

Method	Description
`connect`	Used exclusively when `NetworkClient` is requested to establish a connection to a broker
`poll`	

ShutdownableThread

`ShutdownableThread` is the contract for non-daemon threads of execution.

`ShutdownableThread` contract expects that the objects implement doWork method.

```
def doWork(): Unit
```

Table 1. ShutdownableThread's Internal Properties (e.g. Registries and Counters)

Name	Description
`isRunning`	Flag that controls how long to execute run method.
`shutdownLatch`	Java's java.util.concurrentCountDownLatch with the number of passes being `1`

`run` Method

```
run(): Unit
```

Note	`run` is a part of java.lang.Runnable that is executed when the thread is started.

`run` first prints out the following INFO message to the logs:

```
Starting
```

`run` then executes doWork method until isRunning flag is disabled.

In the end, `run` decrements the count of shutdownLatch and prints out the following INFO message to the logs:

```
Stopped
```

SocketServer

`SocketServer` is a NIO socket server.

`SocketServer` is created exclusively when `KafkaServer` is started.

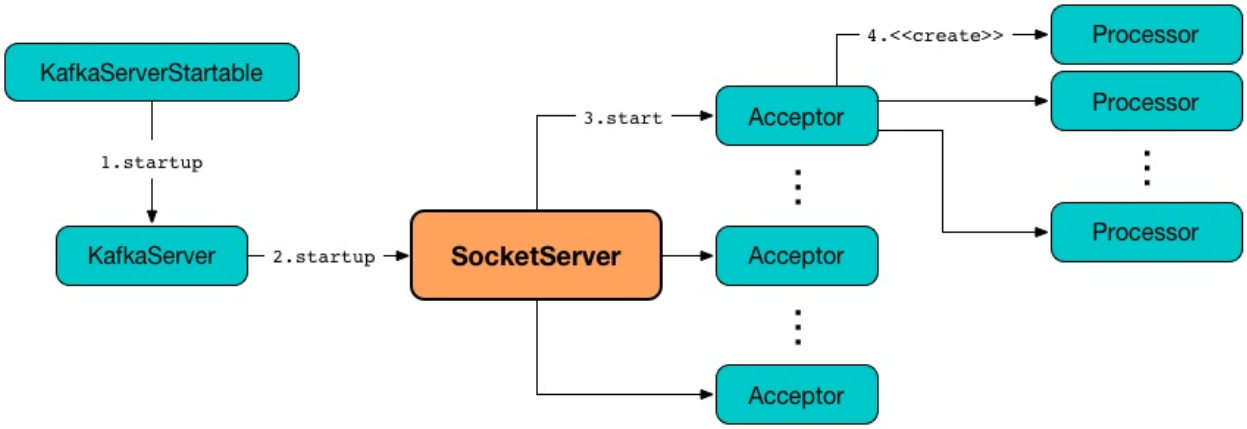

Figure 1. SocketServer's Startup

Table 1. SocketServer's Metrics (in kafka.network group)

Name	Description
`NetworkProcessorAvgIdlePercent`	
`MemoryPoolAvailable`	
`MemoryPoolUsed`	

Table 2. SocketServer's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
`acceptors`	`Acceptor` threads per `EndPoint`
`connectionQuotas`	`ConnectionQuotas`
`endpoints`	`EndPoints` (aka *listeners*) per name (as configured using listeners Kafka property)
`maxQueuedRequests`	
`maxConnectionsPerIp`	
`maxConnectionsPerIpOverrides`	
`memoryPool`	
`numProcessorThreads`	The number of processors per endpoint (as configured using num.network.threads Kafka property)
`processors`	`Processor` threads (initially totalProcessorThreads)
`requestChannel`	
`totalProcessorThreads`	Total number of processors, i.e. numProcessorThreads for every endpoint

Creating Processor — `newProcessor` Internal Method

Caution	FIXME

Starting Up (and Auxiliary Services) — `startup` Method

```
startup(): Unit
```

Internally, `startup` creates the ConnectionQuotas (with maxConnectionsPerIp and maxConnectionsPerIpOverrides).

For every endpoint (in endpoints registry) `startup` does the following:

1. Creates up to numProcessorThreads number of Processors (for ConnectionQuotas and MemoryPool)

2. Creates a `Acceptor` for the endpoint and processors
3. Records the `Acceptor` in acceptors internal registry
4. Starts a non-daemon thread for the `Acceptor` with the name as `kafka-socket-acceptor-[listenerName]-[securityProtocol]-[port]` (e.g. `kafka-socket-acceptor-ListenerName(PLAINTEXT)-PLAINTEXT-9092`) and waits until it has started fully

`startup` then registers metrics.

In the end, `startup` prints out the following INFO message to the logs:

```
INFO [SocketServer brokerId=[brokerID]] Started [number] acceptor threads
```

Note	`startup` is used exclusively when `KafkaServer` starts up.

Creating SocketServer Instance

`SocketServer` takes the following when created:

- KafkaConfig
- `Metrics`
- `Time`
- `CredentialProvider`

`SocketServer` initializes the internal registries and counters.

TopicDeletionManager

`TopicDeletionManager` is…FIXME

`TopicDeletionManager` is created exclusively when `KafkaController` is created.

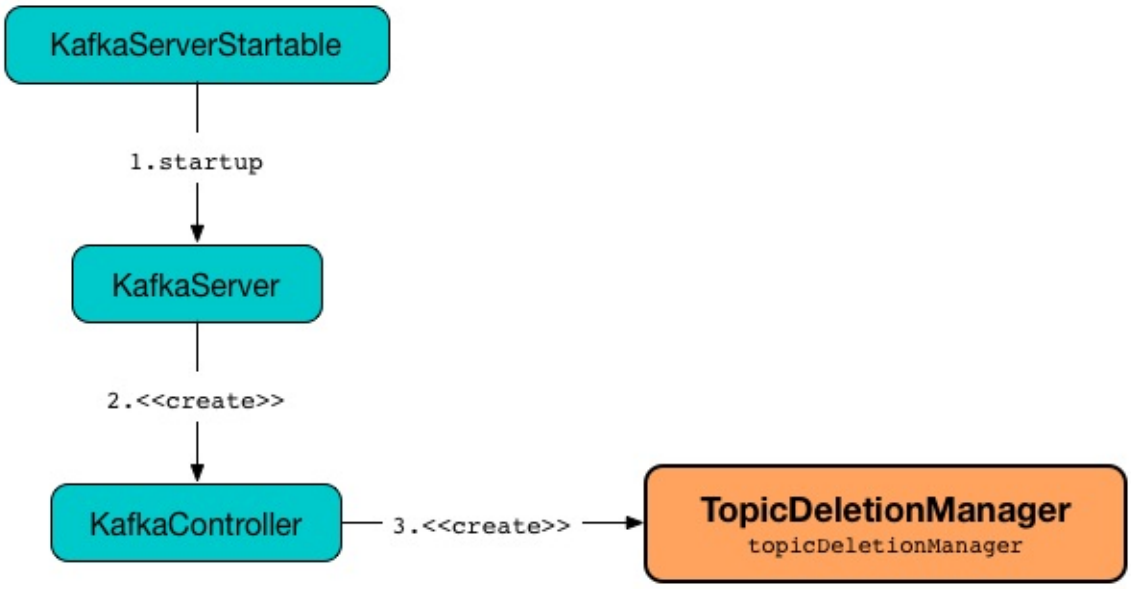

Figure 1. TopicDeletionManager is Created Alongside KafkaController

`TopicDeletionManager` is controlled by delete.topic.enable Kafka property and does nothing when it is turned off (i.e. `false`).

`logIdent` is **[Topic Deletion Manager [brokerId]]**.

Table 1. TopicDeletionManager's Internal Properties (e.g. Registries and Counters)

Name	Description
`partitionsToBeDeleted`	`TopicAndPartitions` to be deleted
`topicsToBeDeleted`	The names of the topics to be deleted
`topicsIneligibleForDeletion`	The names of the topics that must not be deleted (i.e. are ineligible for deletion)

Tip	Enable `INFO` logging level for `kafka.controller.TopicDeletionManager` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.controller.TopicDeletionManager=INFO, stdout` Refer to Logging.

`startReplicaDeletion` Internal Method

Caution	FIXME

`enqueueTopicsForDeletion` Method

Caution	FIXME

`failReplicaDeletion` Method

Caution	FIXME

Creating TopicDeletionManager Instance

`TopicDeletionManager` takes the following when created:

- KafkaController
- ControllerEventManager

`TopicDeletionManager` initializes the internal registries and counters.

`markTopicIneligibleForDeletion` Method

```
markTopicIneligibleForDeletion(topics: Set[String]): Unit
```

(only with delete.topic.enable Kafka property enabled) `markTopicIneligibleForDeletion` computes the intersection between topicsToBeDeleted and the input `topics` sets and adds the intersection to topicsIneligibleForDeletion set.

If there are any topics in the intersection, `markTopicIneligibleForDeletion` prints out the following INFO message to the logs:

```
Halted deletion of topics [newTopicsToHaltDeletion]
```

Note	`markTopicIneligibleForDeletion` is used when: • `KafkaController` initiateReassignReplicasForTopicPartition • `TopicDeletion` controller event is processed (for topics to be deleted with partitions in `controllerContext.partitionsBeingReassigned` list) • `TopicDeletionManager` does startReplicaDeletion and failReplicaDeletion

Reseting — `reset` Method

```
reset(): Unit
```

(only with delete.topic.enable Kafka property enabled) `reset` removes all elements from the following internal registries:

- topicsToBeDeleted
- partitionsToBeDeleted
- topicsIneligibleForDeletion

Note	`reset` does nothing when delete.topic.enable Kafka property is `false`.

Note	`reset` is used exclusively when `KafkaController` resigns as the active controller.

TransactionCoordinator

`TransactionCoordinator` is…FIXME

`TransactionCoordinator` is created when…FIXME

`startup` Method

```
startup
```

`startup` …FIXME

Note	`startup` is used exclusively when `KafkaServer` starts up.

Creating TransactionCoordinator Instance

`TransactionCoordinator` takes the following when created:

- TransactionCoordinator

`TransactionCoordinator` initializes the internal registries and counters.

TransactionStateManager

`TransactionStateManager` is…FIXME

`getTransactionTopicPartitionCount` Method

```
getTransactionTopicPartitionCount
```

`getTransactionTopicPartitionCount` …FIXME

Note	`getTransactionTopicPartitionCount` is used when…FIXME

ZkUtils

`ZkUtils` is…FIXME

`ZkUtils` is created when…FIXME

Note	Quoting Nodes and ephemeral nodes from the Zookeeper documentation: > Unlike standard file systems, each node in a ZooKeeper namespace can have data associated with it as well as children. It is like having a file-system that allows a file to also be a directory. (ZooKeeper was designed to store coordination data: status information, configuration, location information, etc., so the data stored at each node is usually small, in the byte to kilobyte range.) We use the term znode to make it clear that we are talking about ZooKeeper data nodes. > Znodes maintain a stat structure that includes version numbers for data changes, ACL changes, and timestamps, to allow cache validations and coordinated updates. Each time a znode's data changes, the version number increases. For instance, whenever a client retrieves data it also receives the version of the data. > The data stored at each znode in a namespace is read and written atomically. Reads get all the data bytes associated with a znode and a write replaces all the data. Each node has an Access Control List (ACL) that restricts who can do what.

Table 1. ZkUtils's ZNodes in Zookeeper

ZNode	Description
`ControllerPath`	`/controller`

Table 2. ZkUtils's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
`persistentZkPaths`	
`zkPath`	

`getCluster` Method

```
getCluster(): Cluster
```

`getCluster` gets the children znodes of `/brokers/ids` znode and reads their data (as a JSON blob).

`getCluster` then adds creates a `Broker` from the znode id and the JSON blob (with a host, a port and endpoints).

Note	`getCluster` is used exclusively when `ZKRebalancerListener` does syncedRebalance (that happens for the currently-deprecated `ZookeeperConsumerConnector`).

`deletePathRecursive` Method

Caution	FIXME

`deletePath` Method

Caution	FIXME

Creating ZkUtils Instance — `apply` Factory Method

```
apply(
  zkUrl: String,
  sessionTimeout: Int,
  connectionTimeout: Int,
  isZkSecurityEnabled: Boolean): ZkUtils
```

`apply` ...FIXME

Note	`apply` is used when: 1. `KafkaServer` connects to Zookeeper 2. FIXME

Registering Listener for State Changes — `subscribeStateChanges` Method

```
subscribeStateChanges(listener: IZkStateListener): Unit
```

`subscribeStateChanges` requests ZkClient to `subscribeStateChanges` with the `listener`.

Note	`subscribeStateChanges` is used when: 1. `KafkaController` is requested to register a SessionExpirationListener 2. FIXME

Registering Listener for Child Changes — `subscribeChildChanges` Method

```
subscribeChildChanges(path: String, listener: IZkChildListener): Option[Seq[String]]
```

`subscribeChildChanges` ...FIXME

Note	`subscribeChildChanges` is used...FIXME

De-Registering Listener for Child Changes — `unsubscribeChildChanges` Method

```
unsubscribeChildChanges(path: String, childListener: IZkChildListener): Unit
```

`unsubscribeChildChanges` requests ZkClient to `unsubscribeChildChanges` for the input `path` and `childListener`.

Note	`unsubscribeChildChanges` is used when...FIXME

De-Registering Listener for Data Changes — `unsubscribeDataChanges` Method

```
unsubscribeDataChanges(path: String, dataListener: IZkDataListener): Unit
```

`unsubscribeDataChanges` requests ZkClient to `unsubscribeDataChanges` for the input `path` and `dataListener`.

Note	`unsubscribeDataChanges` is used when...FIXME

`registerBrokerInZk` Method

```
registerBrokerInZk(
  id: Int,
  host: String,
  port: Int,
  advertisedEndpoints: Seq[EndPoint],
  jmxPort: Int,
  rack: Option[String],
  apiVersion: ApiVersion): Unit
```

`registerBrokerInZk` ...FIXME

Note	`registerBrokerInZk` is used exclusively when `KafkaHealthcheck` is requested to register.

`getTopicPartitionCount` Method

```
getTopicPartitionCount(topic: String): Option[Int]
```

`getTopicPartitionCount` ...FIXME

Note	`getTopicPartitionCount` is used when: 1. `GroupMetadataManager` is requested for getGroupMetadataTopicPartitionCount of `__consumer_offsets` topic 2. `TransactionStateManager` is requested for getTransactionTopicPartitionCount of `__transaction_state` topic

Creating JSON with Broker ID — `controllerZkData` Method

```
controllerZkData(brokerId: Int, timestamp: Long): String
```

`controllerZkData` creates a JSON with the following fields:

- `"version":1`
- `"brokerid":[brokerId]`
- `"timestamp":[timestamp]`

```
import kafka.utils._
scala> ZkUtils.controllerZkData(1, System.currentTimeMillis())
res0: String = {"version":1,"brokerid":1,"timestamp":"1506161225262"}
```

Note	`controllerZkData` is used exclusively when `KafkaController` is requested for elect.

Creating ZkUtils Instance

`ZkUtils` takes the following when created:

- `ZkClient`
- `ZkConnection`
- `isSecure` flag

`ZkUtils` initializes the internal registries and counters.

Reading Data Associated with ZNode — `readDataMaybeNull` Method

```
readDataMaybeNull(path: String): (Option[String], Stat)
```

`readDataMaybeNull` requests ZkClient to `readData` from `path` znode.

`readDataMaybeNull` returns `None` (for `Option[String]`) when `path` znode is not available.

ZKRebalancerListener

`ZKRebalancerListener` is…FIXME

`syncedRebalance` Method

```
syncedRebalance(): Unit
```

`syncedRebalance` …FIXME

Note	`syncedRebalance` is used when: • `ZKSessionExpireListener` does `handleNewSession` • That happens exclusively in (deprecated) `ZookeeperConsumerConnector` • `ZKRebalancerListener` is created (and creates `watcherExecutorThread`) • That happens exclusively in (deprecated) `ZookeeperConsumerConnector` • (deprecated) `ZookeeperConsumerConnector` does `reinitializeConsumer`

Topic Replication

Topic Replication is the process to offer fail-over capability for a topic.

Replication factor defines the number of copies of a topic in a Kafka cluster.

Replication factor can be defined at topic level.

```
./bin/kafka-topics.sh --create \
  --topic my-topic \
  --replication-factor 1 \  // <-- define replication factor
  --partitions 1 \
  --zookeeper localhost:2181
```

Replicas are distributed evenly among Kafka brokers in a cluster.

Leader replica is…FIXME

Follower replica is…FIXME

Producers always send requests to the broker that is the current leader replica for a topic partition.

Data from producers is first saved to a commit log before consumers can find out that it is available. It will only be visible to consumers when the followers acknowledge that they have got the data and stored in their local logs.

Topic Deletion

Topic Deletion is a feature of Kafka that allows for deleting topics.

TopicDeletionManager is responsible for topic deletion.

Topic deletion is controlled by delete.topic.enable Kafka property that turns it on when `true`.

Start a Kafka broker with broker ID `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=100 \
  --override log.dirs=/tmp/kafka-logs-100 \
  --override port=9192
```

Create **remove-me** topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 \
  --create \
  --topic remove-me \
  --partitions 1 \
  --replication-factor 1
Created topic "remove-me".
```

Use `kafka-topics.sh --list` to list available topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me
```

Use `kafka-topics.sh --describe` to list details for `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic remove-me
Topic:remove-me PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: remove-me        Partition: 0    Leader: 100     Replicas: 100   Isr: 1
00
```

Note that the broker `100` is the leader for `remove-me` topic.

Stop the broker `100` and start another with broker ID `200`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=200 \
  --override log.dirs=/tmp/kafka-logs-200 \
  --override port=9292
```

Use `kafka-topics.sh --delete` to delete `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic remove-me
Topic remove-me is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

List the topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me - marked for deletion
```

As you may have noticed, `kafka-topics.sh --delete` will only delete a topic if the topic's leader broker is available (and can acknowledge the removal). Since the broker 100 is down and currently unavailable the topic deletion has only been recorded in Zookeeper.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 0] ls /admin/delete_topics
[remove-me]
```

As long as the leader broker `100` is not available, the topic to be deleted remains marked for deletion.

Start the broker `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=100 \
  --override log.dirs=/tmp/kafka-logs-100 \
  --override port=9192
```

With kafka.controller.KafkaController logger at `DEBUG` level, you should see the following messages in the logs:

```
DEBUG [Controller id=100] Delete topics listener fired for topics remove-me to be dele
ted (kafka.controller.KafkaController)
INFO [Controller id=100] Starting topic deletion for topics remove-me (kafka.controlle
r.KafkaController)
INFO [GroupMetadataManager brokerId=100] Removed 0 expired offsets in 0 milliseconds.
(kafka.coordinator.group.GroupMetadataManager)
DEBUG [Controller id=100] Removing replica 100 from ISR 100 for partition remove-me-0.
 (kafka.controller.KafkaController)
INFO [Controller id=100] Retaining last ISR 100 of partition remove-me-0 since unclean
 leader election is disabled (kafka.controller.KafkaController)
INFO [Controller id=100] New leader and ISR for partition remove-me-0 is {"leader":-1,
"leader_epoch":1,"isr":[100]} (kafka.controller.KafkaController)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions remove-me-0
(kafka.server.ReplicaFetcherManager)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions  (kafka.serv
er.ReplicaFetcherManager)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions remove-me-0
(kafka.server.ReplicaFetcherManager)
INFO Log for partition remove-me-0 is renamed to /tmp/kafka-logs-100/remove-me-0.fe6d0
39ff884498b9d6113fb22a75264-delete and is scheduled for deletion (kafka.log.LogManager
)
DEBUG [Controller id=100] Delete topic callback invoked for org.apache.kafka.common.re
quests.StopReplicaResponse@8c0f4f0 (kafka.controller.KafkaController)
INFO [Controller id=100] New topics: [Set()], deleted topics: [Set()], new partition r
eplica assignment [Map()] (kafka.controller.KafkaController)
DEBUG [Controller id=100] Delete topics listener fired for topics  to be deleted (kafk
a.controller.KafkaController)
```

The topic is now deleted. Use Zookeeper CLI tool to confirm it.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 1] ls /admin/delete_topics
[]
```

Kafka Controller Election

Use the following setup with one Zookeeper server and two Kafka brokers to observe the Kafka controller election.

Start Zookeeper server.

```
$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
...
INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFa
ctory)
```

Add the following line to `config/log4j.properties` to enable `DEBUG` logging level for `kafka.controller.KafkaController` logger.

```
log4j.logger.kafka.controller.KafkaController=DEBUG, stdout
```

Start a Kafka broker.

```
$ ./bin/kafka-server-start.sh config/server.properties \
    --override broker.id=100 \
    --override log.dirs=/tmp/kafka-logs-100 \
    --override port=9192
...
INFO Registered broker 100 at path /brokers/ids/100 with addresses: EndPoint(192.168.1
.4,9192,ListenerName(PLAINTEXT),PLAINTEXT) (kafka.utils.ZkUtils)
INFO Kafka version : 1.0.0-SNAPSHOT (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka commitId : 852297efd99af04d (org.apache.kafka.common.utils.AppInfoParser)
INFO [KafkaServer id=100] started (kafka.server.KafkaServer)
```

Start another Kafka broker.

```
# Note the different properties
$ ./bin/kafka-server-start.sh config/server.properties \
    --override broker.id=200 \
    --override log.dirs=/tmp/kafka-logs-200 \
    --override port=9292
...
INFO Registered broker 200 at path /brokers/ids/200 with addresses: EndPoint(192.168.1
.4,9292,ListenerName(PLAINTEXT),PLAINTEXT) (kafka.utils.ZkUtils)
INFO Kafka version : 1.0.0-SNAPSHOT (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka commitId : 852297efd99af04d (org.apache.kafka.common.utils.AppInfoParser)
INFO [KafkaServer id=200] started (kafka.server.KafkaServer)
```

Connect to Zookeeper using Zookeeper CLI (command-line interface). Use the official distribution of Apache Zookeeper as described in Zookeeper Tips.

```
$ ./bin/zkCli.sh -server localhost:2181
```

Once connected, execute `get /controller` to get the data associated with `/controller` znode where the active Kafka controller stores the controller ID.

```
[zk: localhost:2181(CONNECTED) 0] get /controller
{"version":1,"brokerid":100,"timestamp":"1506423376977"}
cZxid = 0x191
ctime = Tue Sep 26 12:56:16 CEST 2017
mZxid = 0x191
mtime = Tue Sep 26 12:56:16 CEST 2017
pZxid = 0x191
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15ebdd241840002
dataLength = 56
numChildren = 0
```

(optional) Clear the consoles of the two Kafka brokers so you have the election logs only.

Delete `/controller` znode and observe the controller election.

```
[zk: localhost:2181(CONNECTED) 2] delete /controller
```

You should see the following in the logs in the consoles of the two Kafka brokers.

```
DEBUG [Controller id=100] Resigning (kafka.controller.KafkaController)
DEBUG [Controller id=100] De-registering IsrChangeNotificationListener (kafka.controll
er.KafkaController)
DEBUG [Controller id=100] De-registering logDirEventNotificationListener (kafka.contro
ller.KafkaController)
INFO [Controller id=100] Resigned (kafka.controller.KafkaController)
DEBUG [Controller id=100] Broker 200 has been elected as the controller, so stopping t
he election process. (kafka.controller.KafkaController)
```

and

```
INFO Creating /controller (is it secure? false) (kafka.utils.ZKCheckedEphemeral)
INFO Result of znode creation is: OK (kafka.utils.ZKCheckedEphemeral)
INFO [Controller id=200] 200 successfully elected as the controller (kafka.controller.
KafkaController)
INFO [Controller id=200] Starting become controller state transition (kafka.controller
.KafkaController)
INFO [Controller id=200] Initialized controller epoch to 39 and zk version 38 (kafka.c
ontroller.KafkaController)
INFO [Controller id=200] Incremented epoch to 40 (kafka.controller.KafkaController)
DEBUG [Controller id=200] Registering IsrChangeNotificationListener (kafka.controller.
KafkaController)
DEBUG [Controller id=200] Registering logDirEventNotificationListener (kafka.controlle
r.KafkaController)
INFO [Controller id=200] Partitions being reassigned: Map() (kafka.controller.KafkaCon
troller)
INFO [Controller id=200] Partitions already reassigned: Set() (kafka.controller.KafkaC
ontroller)
INFO [Controller id=200] Resuming reassignment of partitions: Map() (kafka.controller.
KafkaController)
INFO [Controller id=200] Currently active brokers in the cluster: Set(100, 200) (kafka
.controller.KafkaController)
INFO [Controller id=200] Currently shutting brokers in the cluster: Set() (kafka.contr
oller.KafkaController)
INFO [Controller id=200] Current list of topics in the cluster: Set(my-topic2, NEW, my
-topic, my-topic1) (kafka.controller.KafkaController)
INFO [Controller id=200] List of topics to be deleted:  (kafka.controller.KafkaControl
ler)
INFO [Controller id=200] List of topics ineligible for deletion:  (kafka.controller.Ka
fkaController)
INFO [Controller id=200] Ready to serve as the new controller with epoch 40 (kafka.con
troller.KafkaController)
INFO [Controller id=200] Partitions undergoing preferred replica election:  (kafka.con
troller.KafkaController)
INFO [Controller id=200] Partitions that completed preferred replica election:  (kafka
.controller.KafkaController)
INFO [Controller id=200] Skipping preferred replica election for partitions due to top
ic deletion:  (kafka.controller.KafkaController)
INFO [Controller id=200] Resuming preferred replica election for partitions:  (kafka.c
ontroller.KafkaController)
INFO [Controller id=200] Starting preferred replica leader election for partitions  (k
afka.controller.KafkaController)
INFO [Controller id=200] Starting the controller scheduler (kafka.controller.KafkaCont
roller)
```

KafkaProducer — Main Class For Kafka Producers

`KafkaProducer` is the main public class that you and other Kafka developers use to write Kafka producers that publish records to a Kafka cluster.

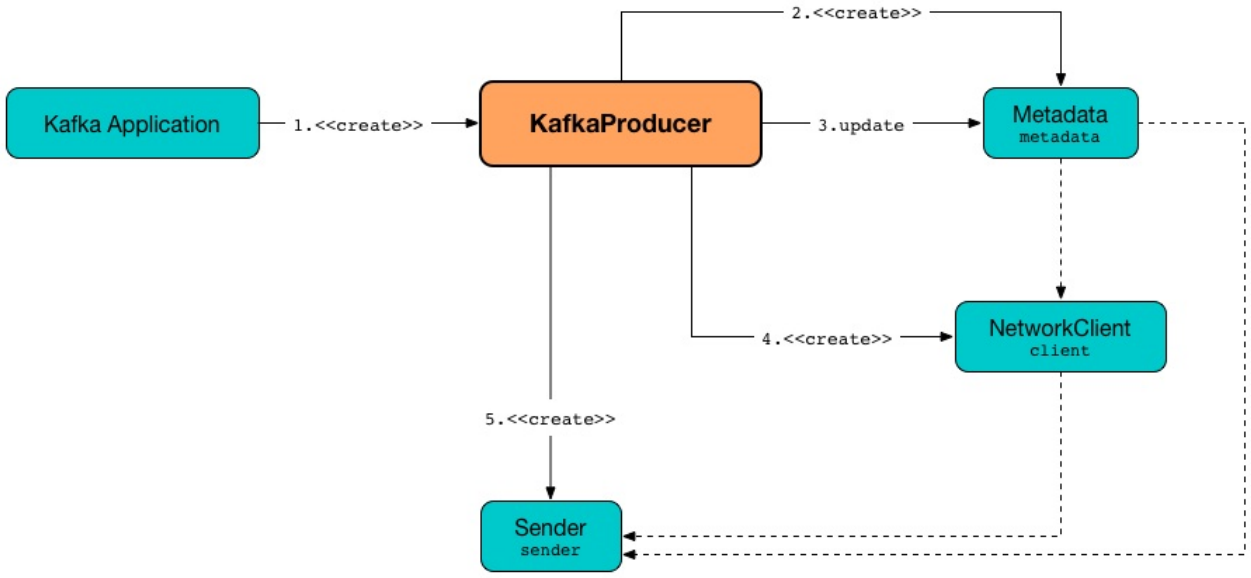

Figure 1. KafkaProducer

`KafkaProducer` is a part of the public API and is created with properties and (key and value) serializers as configuration.

`KafkaProducer` follows the Producer contract.

Table 1. KafkaProducer's Internal Properties (e.g. Registries and Counters)

Name	Description
`maxBlockTimeMs`	Time `KafkaProducer` uses to block when requesting partitions for a topic. Note: Use max.block.ms Kafka property to set the value.
`metadata`	Metadata • Created when `KafkaProducer` is created with the following properties: ◦ retry.backoff.ms for refreshBackoffMs ◦ metadata.max.age.ms for metadataExpireMs ◦ allowAutoTopicCreation flag enabled ◦ topicExpiryEnabled flag enabled • Updated with a bootstrap cluster when `KafkaProducer` is created • Used in waitOnMetadata
`sender`	Sender with the following: • …FIXME
`ioThread`	Thread of execution…FIXME

Tip: Enable `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.producer.KafkaProducer` logger to see what happens inside.

Add the following line to `config/tools-log4j.properties`:

```
log4j.logger.org.apache.kafka.clients.producer.KafkaProducer=TRACE
```

Refer to Logging.

Asynchronously Sending Record to Topic and Notifying Producer Interceptors — `send` Method

```
Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback)
```

Note	`send` is a part of Producer Contract.

`send` …FIXME

Asynchronously Sending Record to Topic — `doSend` Internal Method

```
Future<RecordMetadata> doSend(ProducerRecord<K, V> record, Callback callback)
```

`doSend` …FIXME

Note	`doSend` is used exclusively when `KafkaProducer` asynchronously sends a record to a topic and notifies producer interceptors.

Creating KafkaProducer Instance

`KafkaProducer` takes the following when created:

- FIXME

`KafkaProducer` initializes the internal registries and counters.

Configuring ClusterResourceListeners — `configureClusterResourceListeners` Internal Method

```
ClusterResourceListeners configureClusterResourceListeners(
  Serializer<K> keySerializer,
  Serializer<V> valueSerializer,
  List<?>... candidateLists)
```

`configureClusterResourceListeners` creates a ClusterResourceListeners and registers `ClusterResourceListener` instances from the input `candidateLists`, `keySerializer` and `valueSerializer`.

Note	`configureClusterResourceListeners` is used exclusively when `KafkaProducer` is created (to create the Metadata) with the following input arguments: • key and value serializers (defined when `KafkaProducer` is created) • ProducerInterceptors from interceptor.classes Kafka property • MetricsReporters from metric.reporters Kafka property

Requesting Partitions for Topic — `partitionsFor` Method

```
List<PartitionInfo> partitionsFor(String topic)
```

Note	`partitionsFor` is a part of Producer Contract.

`partitionsFor` waits on cluster metadata for the input `topic` and max.block.ms time. Once retrieved, `partitionsFor` requests `Cluster` for the partitions.

Waiting for Cluster Metadata (with Partitions for Topic) — `waitOnMetadata` Internal Recursive Method

```
ClusterAndWaitTime waitOnMetadata(
  String topic,
  Integer partition,
  long maxWaitMs) throws InterruptedException
```

`waitOnMetadata` adds the input `topic` to Metadata.

`waitOnMetadata` first checks if the available cluster metadata could be current enough.

`waitOnMetadata` requests Metadata for the current cluster information and then requests the cluster for the number of partitions of the input `topic`.

If the cluster metadata is not current enough (i.e. the number of partitions is unavailable or the `partition` is above the current count), `waitOnMetadata` prints out the following TRACE message to the logs:

```
Requesting metadata update for topic [topic].
```

`waitOnMetadata` requests Metadata for update and requests Sender to wake up.

`waitOnMetadata` then requests Metadata to wait for a metadata update and then Metadata for the current cluster information.

`waitOnMetadata` keeps doing it until the number of partitions of the input `topic` is available.

`waitOnMetadata` reports a `TimeoutException` when `maxWaitMs` has elapsed.

```
Failed to update metadata after [maxWaitMs] ms.
```

`waitOnMetadata` reports a `TopicAuthorizationException` when the access to the `topic` is unauthorized.

`waitOnMetadata` reports a `KafkaException` when the `partition` is above the number of available partitions.

```
Invalid partition given with record: [partition] is not in the range [0...[partitionsC
ount]).
```

Note	`waitOnMetadata` is used when `KafkaProducer` requests partitions for a topic and asynchronously sends a record to a topic.

Producer

`Producer` is...FIXME

DefaultPartitioner

`DefaultPartitioner` is...FIXME

Partitioner

`Partitioner` is…FIXME

ProducerInterceptor

`ProducerInterceptor` is...FIXME

Sender

`Sender` is thread of execution that handles the sending of produce requests to a Kafka cluster.

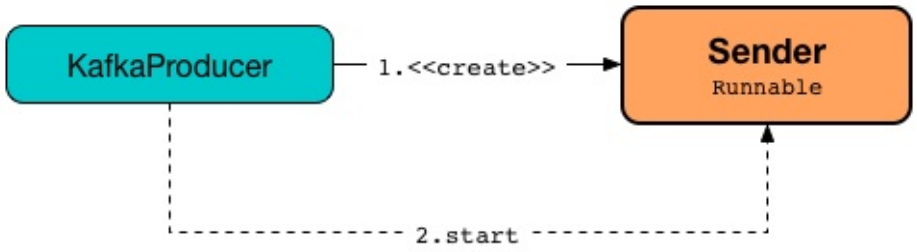

Figure 1. Sender and KafkaProducer

`Sender` is created exclusively when `KafkaProducer` is created.

`Sender` is started under the name **kafka-producer-network-thread | [clientId]** when `KafkaProducer` is created.

Table 1. Sender's Internal Properties (e.g. Registries and Counters)

Name	Description
`running`	Flag that controls whether run should finish or not

`sendProduceRequests` Internal Method

```
void sendProduceRequests(Map<Integer, List<ProducerBatch>> collated, long now)
```

`sendProduceRequests` …FIXME

Note	`sendProduceRequests` is used exclusively when `Sender` sendProducerData.

`sendProduceRequest` Internal Method

```
void sendProduceRequest(
  long now,
  int destination,
  short acks,
  int timeout,
  List<ProducerBatch> batches)
```

`sendProduceRequest` …FIXME

Note	`sendProduceRequest` is used exclusively when `Sender` sendProduceRequests.

`sendProducerData` Internal Method

```
long sendProducerData(long now)
```

`sendProducerData` …FIXME

Note	`sendProducerData` is used exclusively when `Sender` runs a single iteration of sending.

Running Single Iteration of Sending — `run` Method

```
void run(long now)
```

`run` …FIXME

Note	`run` is used exclusively when `Sender` is started (as a thread of execution).

Starting Thread of Execution — `run` Method (of Java's `Runnable`)

```
void run()
```

Note	`run` is a part of java.lang.Runnable that is executed when the thread is started.

`run` first prints out the following DEBUG message to the logs:

```
Starting Kafka producer I/O thread.
```

`run` keeps running (with the current time in milliseconds) until running flag is turned off.

`run` …FIXME

Creating Sender Instance

`Sender` takes the following when created:

- `LogContext`

- KafkaClient
- Metadata
- `RecordAccumulator`
- `guaranteeMessageOrder` flag
- `maxRequestSize`
- `acks`
- The number of retries
- `SenderMetricsRegistry`
- `Time`
- `requestTimeout`
- `retryBackoffMs`
- `TransactionManager`
- `ApiVersions`

`Sender` initializes the internal registries and counters.

Serializer

`Serializer` is…FIXME

KafkaConsumer — Main Class For Kafka Consumers

`KafkaConsumer` is the main public class that you and other Kafka developers use to write Kafka consumers that consume records from a Kafka cluster.

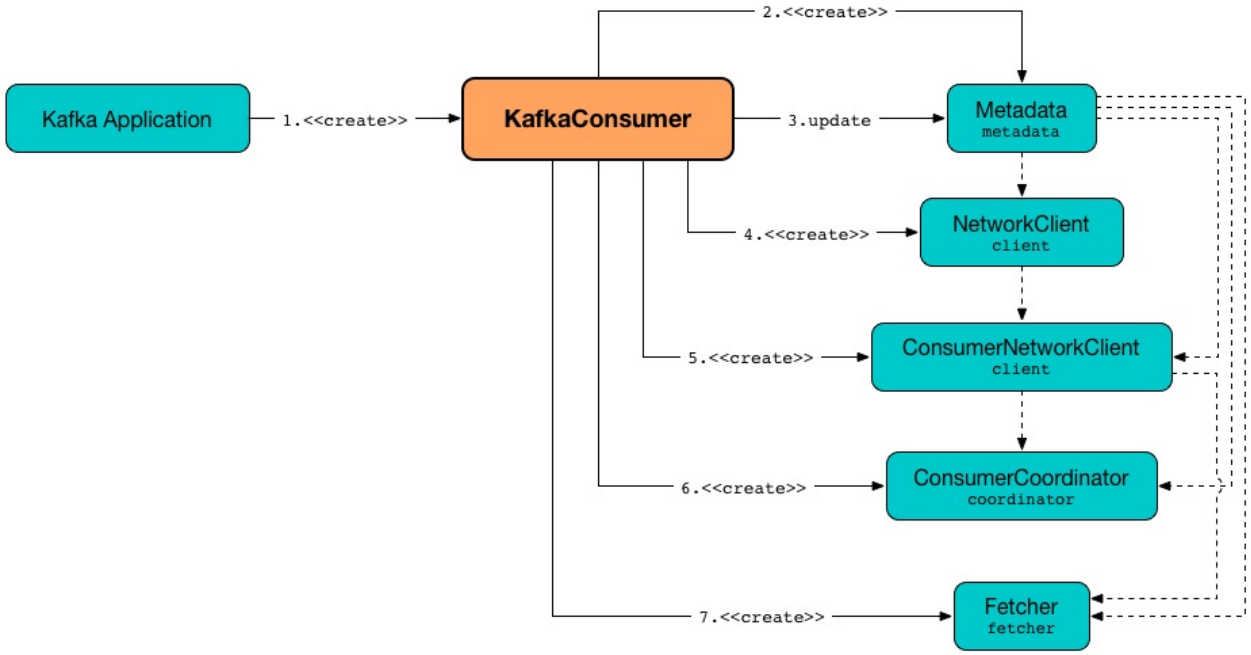

Figure 1. KafkaConsumer

`KafkaConsumer` is a part of the public API and is created with properties and (key and value) deserializers as configuration.

Note	bootstrap.servers and group.id properties are mandatory. They usually appear in the code as `ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG` and `ConsumerConfig.GROUP_ID_CONFIG` values, respectively.

`KafkaConsumer` follows the Consumer contract.

```
// sandbox/kafka-sandbox
val bootstrapServers = "localhost:9092"
val groupId = "kafka-sandbox"
import org.apache.kafka.clients.consumer.ConsumerConfig
val configs: Map[String, Object] = Map(
  // required properties
  ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> bootstrapServers,
  ConsumerConfig.GROUP_ID_CONFIG -> groupId
)
import org.apache.kafka.common.serialization.StringDeserializer
val keyDeserializer = new StringDeserializer
val valueDeserializer = new StringDeserializer

import scala.collection.JavaConverters._
import org.apache.kafka.clients.consumer.KafkaConsumer
val consumer = new KafkaConsumer[String, String](
  configs.asJava,
  keyDeserializer,
  valueDeserializer)
```

`KafkaConsumer` registers itself in JMX with **kafka.consumer** prefix.

Caution	FIXME How does the JMX registration happen?

Important	`KafkaConsumer` does not support multi-threaded access. You should use only one thread per `KafkaConsumer` instance. `KafkaConsumer` uses light locks protecting itself from multi-threaded access and reports `ConcurrentModificationException` when it happens. `KafkaConsumer is not safe for multi-threaded access`

Table 1. KafkaConsumer's Internal Properties (e.g. Registries and Counters)

Name	Description
`client`	ConsumerNetworkClient Used mainly *(?)* to create the Fetcher and ConsumerCoordinator Used also in poll, pollOnce and wakeup (but *I* think the usage should be limited to create Fetcher and ConsumerCoordinator)
`clientId`	
`coordinator`	ConsumerCoordinator
`fetcher`	Fetcher Created right when `KafkaConsumer` is created. Used when…FIXME
`interceptors`	`ConsumerInterceptors` that holds ConsumerInterceptor instances (defined using interceptor.classes setting). Used when…FIXME
`metadata`	Metadata Created right when `KafkaConsumer` is created. Used when…FIXME
`metrics`	`Metrics`
`retryBackoffMs`	retry.backoff.ms property or a user-defined value
`requestTimeoutMs`	Corresponds to request.timeout.ms property `KafkaConsumer` reports `ConfigException` when smaller or equal than session.timeout.ms and fetch.max.wait.ms properties.
`subscriptions`	`SubscriptionState` for auto.offset.reset setting. Created when `KafkaConsumer` is created.

Tip	Enable `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.consumer.KafkaConsumer` logger to see what happens inside. Add the following line to `config/tools-log4j.properties`: `log4j.logger.org.apache.kafka.clients.consumer.KafkaConsumer=TRACE` Refer to Logging.

`unsubscribe` Method

Caution	FIXME

Subscribing to Topics Matching Pattern — `subscribe` Method

```
void subscribe(Collection<String> topics)
```

Note	`subscribe` is a part of Consumer Contract.

`subscribe` …FIXME

Note	`subscribe` is used when…FIXME

Subscribing to Topics — `subscribe` Method

```
void subscribe(Collection<String> topics) (1)
void subscribe(Collection<String> topics, ConsumerRebalanceListener listener)
```

1. A short-hand for the other subscribe with `NoOpConsumerRebalanceListener` as `ConsumerRebalanceListener`

`subscribe` subscribes `KafkaConsumer` to the given topics.

```
val topics = Seq("topic1")
println(s"Subscribing to ${topics.mkString(", ")}")

import scala.collection.JavaConverters._
consumer.subscribe(topics.asJava)
```

Internally, `subscribe` prints out the following DEBUG message to the logs:

```
DEBUG Subscribed to topic(s): [comma-separated topics]
```

`subscribe` then requests SubscriptionState to `subscribe` for the `topics` and `listener`.

In the end, `subscribe` requests SubscriptionState for `groupSubscription` that it then passes along to Metadata to set the topics to track.

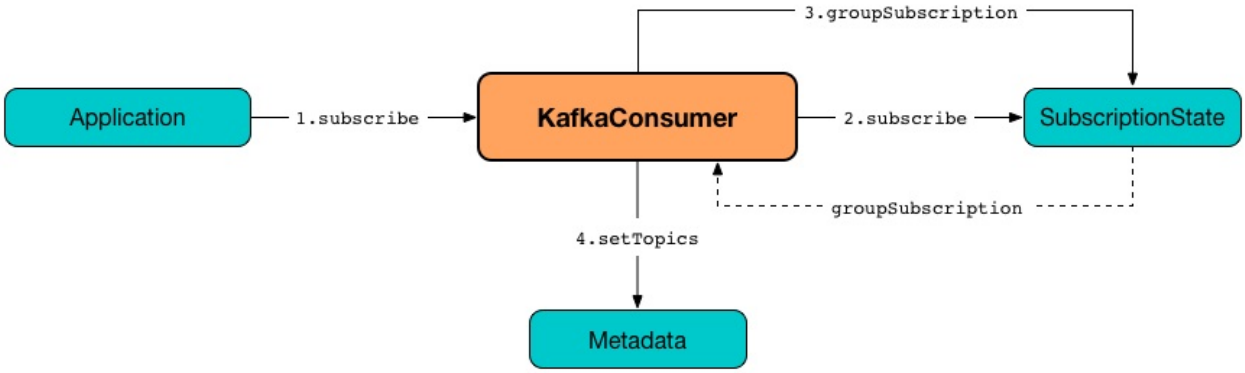

Figure 2. KafkaConsumer subscribes to topics

Poll Specified Milliseconds For ConsumerRecords per TopicPartitions — `poll` Method

```
ConsumerRecords<K, V> poll(long timeout)
```

`poll` polls for new records until `timeout` expires.

Note	`KafkaConsumer` has to be subscribed to some topics or assigned partitions before calling poll.

Note	The input `timeout` should be `0` or greater and represents the milliseconds to poll for records.

```
val seconds = 10
while (true) {
  println(s"Polling for records for $seconds secs")
  val records = consumer.poll(seconds * 1000)
  // do something with the records here
}
```

Internally, `poll` starts by polling once (for `timeout` milliseconds).

If there are records available, `poll` checks Fetcher for sendFetches and ConsumerNetworkClient for pendingRequestCount flag. If either is positive, `poll` requests ConsumerNetworkClient to pollNoWakeup.

Caution	FIXME Make the above more user-friendly

`poll` returns the available `ConsumerRecords` directly when no ConsumerInterceptors are defined or passes them through ConsumerInterceptors using onConsume.

Caution	FIXME Make the above more user-friendly, e.g. when could `interceptors` be empty?

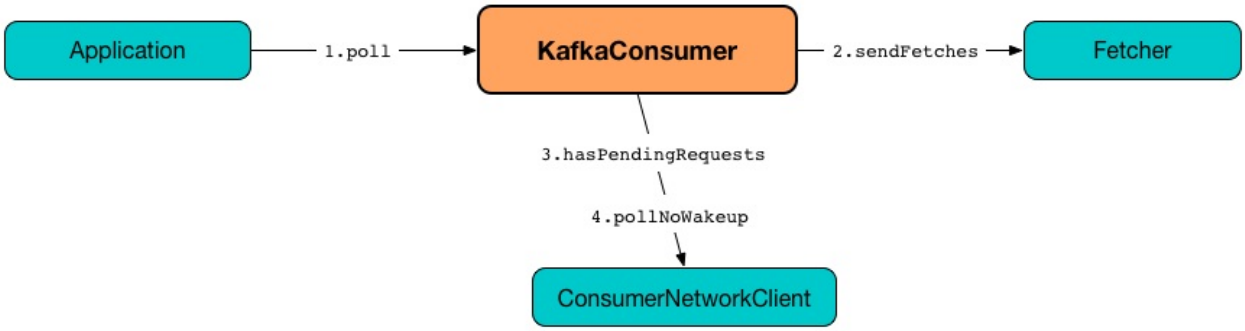

Figure 3. KafkaConsumer polls topics

Note	`poll` is a part of Consumer contract to…FIXME

Getting Partitions For Topic — `partitionsFor` Method

Caution	FIXME

`endOffsets` Method

Caution	FIXME

`offsetsForTimes` Method

Caution	FIXME

`updateFetchPositions` Method

Caution	FIXME

Polling Once for ConsumerRecords per TopicPartition — `pollOnce` Internal Method

Caution	FIXME

Requesting Metadata for All Topics (From Brokers) — `listTopics` Method

```
Map<String, List<PartitionInfo>> listTopics()
```

Internally, `listTopics` simply requests Fetcher for metadata for all topics and returns it.

```
consumer.listTopics().asScala.foreach { case (name, partitions) =>
  println(s"topic: $name (partitions: ${partitions.size()})")
}
```

Note	`listTopics` uses requestTimeoutMs that corresponds to request.timeout.ms property.

`beginningOffsets` Method

```
Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions)
```

`beginningOffsets` requests Fetcher for beginningOffsets and returns it.

Creating KafkaConsumer Instance

`KafkaConsumer` takes the following when created:

- Consumer configuration (that is converted internally to ConsumerConfig)
- Deserializer for keys
- Deserializer for values

`KafkaConsumer` initializes the internal registries and counters.

Note	`KafkaConsumer` API offers other constructors that in the end use the public 3-argument constructor that in turn passes the call on to the private internal constructor.

KafkaConsumer Public Constructor

```
// Public API
KafkaConsumer(
  Map<String, Object> configs,
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer)
```

When created, `KafkaConsumer` adds the keyDeserializer and valueDeserializer to configs (as key.deserializer and value.deserializer properties respectively) and creates a ConsumerConfig.

`KafkaConsumer` passes the call on to the internal constructor.

KafkaConsumer Internal Constructor

```
KafkaConsumer(
  ConsumerConfig config,
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer)
```

When called, the internal `KafkaConsumer` constructor prints out the following DEBUG message to the logs:

```
DEBUG Starting the Kafka consumer
```

`KafkaConsumer` sets the internal requestTimeoutMs to request.timeout.ms property.

`KafkaConsumer` sets the internal clientId to client.id or generates one with prefix **consumer-** (starting from 1) if not set.

`KafkaConsumer` sets the internal Metrics (and `JmxReporter` with **kafka.consumer** prefix).

`KafkaConsumer` sets the internal retryBackoffMs to retry.backoff.ms property.

Caution	FIXME Finish me!

`KafkaConsumer` creates the internal Metadata with the following arguments:

1. retryBackoffMs
2. metadata.max.age.ms
3. `allowAutoTopicCreation` enabled
4. `topicExpiryEnabled` disabled

5. ClusterResourceListeners with user-defined list of ConsumerInterceptors in interceptor.classes property

`KafkaConsumer` updates `metadata` with bootstrap.servers.

Caution	FIXME Finish me!

`KafkaConsumer` creates a NetworkClient with…FIXME

Caution	FIXME Finish me!

`KafkaConsumer` creates Fetcher with the following properties:

- fetch.min.bytes
- fetch.max.bytes
- fetch.max.wait.ms
- max.partition.fetch.bytes
- max.poll.records
- check.crcs

In the end, `KafkaConsumer` prints out the following DEBUG message to the logs:

```
DEBUG Kafka consumer created
```

Any issues while creating a `KafkaConsumer` are reported as `KafkaException`.

```
org.apache.kafka.common.KafkaException: Failed to construct kafka consumer
```

`wakeup` Method

```
void wakeup()
```

Note	`wakeup` is a part of Consumer Contract.

`wakeup` simply requests ConsumerNetworkClient to wakeup.

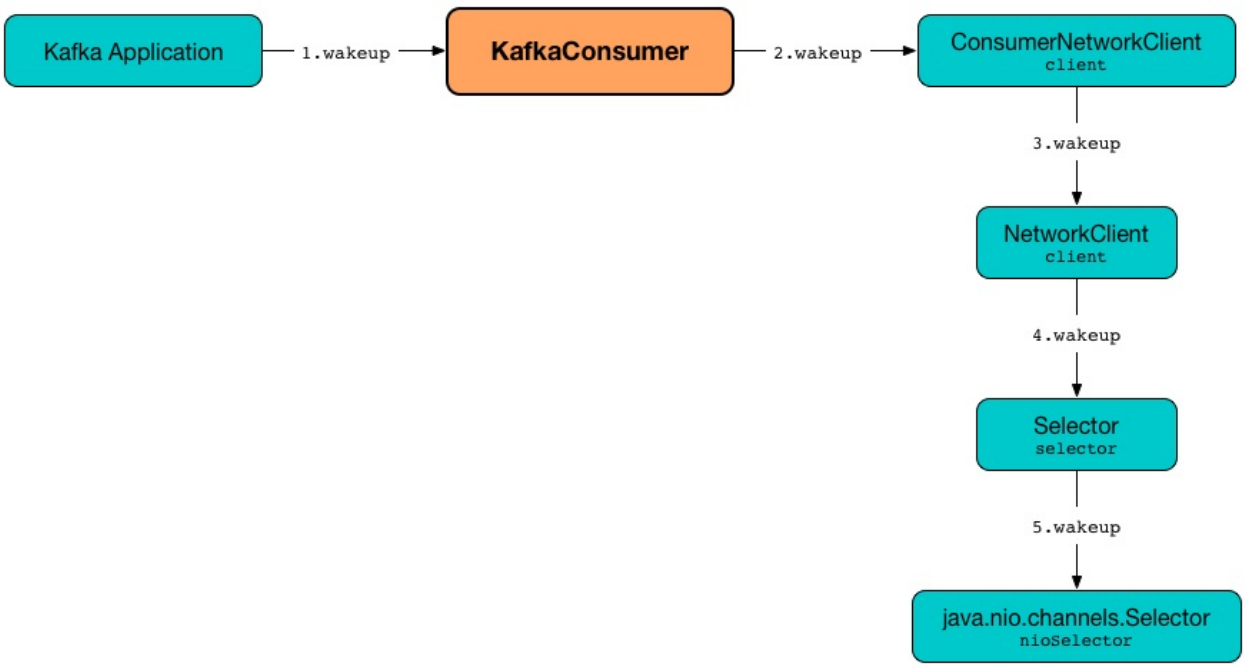

Figure 4. KafkaConsumer's wakeup Method

Note	Quoting `wakeup` of Java's java.nio.channels.Selector given `wakeup` simply passes through the intermediaries and in the end triggers it. > Causes the first selection operation that has not yet returned to return immediately. Read about Selection in java.nio.channels.Selector's javadoc.

Note	`wakeup` is used when…FIXME

Configuring ClusterResourceListeners — `configureClusterResourceListeners` Internal Method

```
ClusterResourceListeners configureClusterResourceListeners(
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer,
  List<?>... candidateLists)
```

`configureClusterResourceListeners` creates a ClusterResourceListeners and registers `ClusterResourceListener` instances from the input `candidateLists`, `keyDeserializer` and `valueDeserializer`.

Note	`configureClusterResourceListeners` is used exclusively when `KafkaConsumer` is created (to create the Metadata) with the following input arguments: • key and value deserializers (defined when `KafkaConsumer` is created) • ConsumerInterceptors from interceptor.classes Kafka property • MetricsReporters from metric.reporters Kafka property

Consumer

`Consumer` is the contract for Kafka consumers.

Note	KafkaConsumer is the main public class that Kafka developers use to write Kafka consumers.

Consumer Contract

```
package org.apache.kafka.clients.consumer;

public interface Consumer<K, V> extends Closeable {
  void assign(Collection<TopicPartition> partitions);
  Set<TopicPartition> assignment();

  void subscribe(Collection<String> topics);
  void subscribe(Collection<String> topics, ConsumerRebalanceListener callback);
  void subscribe(Pattern pattern, ConsumerRebalanceListener callback);
  void subscribe(Pattern pattern);
  Set<String> subscription();
  void unsubscribe();

  ConsumerRecords<K, V> poll(long timeout);

  void commitSync();
  void commitSync(Map<TopicPartition, OffsetAndMetadata> offsets);
  void commitAsync();
  void commitAsync(OffsetCommitCallback callback);
  void commitAsync(Map<TopicPartition, OffsetAndMetadata> offsets, OffsetCommitCallbac
k callback);

  void seek(TopicPartition partition, long offset);
  void seekToBeginning(Collection<TopicPartition> partitions);
  void seekToEnd(Collection<TopicPartition> partitions);

  long position(TopicPartition partition);
  OffsetAndMetadata committed(TopicPartition partition);
  Map<MetricName, ? extends Metric> metrics();
  List<PartitionInfo> partitionsFor(String topic);
  Map<String, List<PartitionInfo>> listTopics();
  Set<TopicPartition> paused();
  void pause(Collection<TopicPartition> partitions);
  void resume(Collection<TopicPartition> partitions);
  Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> ti
mestampsToSearch);
  Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions);
  Map<TopicPartition, Long> endOffsets(Collection<TopicPartition> partitions);

  void close();
  void close(long timeout, TimeUnit unit);

  void wakeup();
}
```

Table 1. Consumer Contract

Method	Description
`wakeup`	

Deserializer

Caution	FIXME

ConsumerConfig

Caution	FIXME

Caution	FIXME Describe `static` block when `ConsumerConfig` is created.

ConsumerCoordinator

`ConsumerCoordinator` is a `AbstractCoordinator` that…FIXME

`ConsumerCoordinator` is created exclusively when `KafkaConsumer` is created.

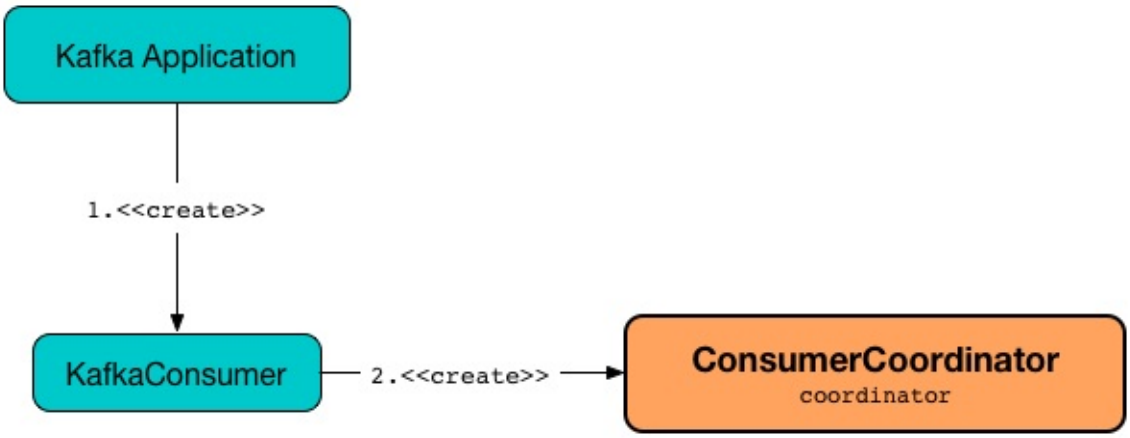

Figure 1. ConsumerCoordinator and KafkaConsumer

Table 1. ConsumerCoordinator's Internal Properties (e.g. Registries and Counters)

Name	Description

`refreshCommittedOffsetsIfNeeded` Method

```
FIXME
```

`refreshCommittedOffsetsIfNeeded` …FIXME

Note	`refreshCommittedOffsetsIfNeeded` is used when…FIXME

`onJoinComplete` Method

```
FIXME
```

`onJoinComplete` …FIXME

Note	`onJoinComplete` is used when…FIXME

`performAssignment` Method

```
FIXME
```

`performAssignment` …FIXME

Note	`performAssignment` is used when…FIXME

`maybeLeaveGroup` Method

```
FIXME
```

`maybeLeaveGroup` …FIXME

Note	`maybeLeaveGroup` is used when…FIXME

`updatePatternSubscription` Method

```
FIXME
```

`updatePatternSubscription` …FIXME

Note	`updatePatternSubscription` is used when…FIXME

`needRejoin` Method

```
FIXME
```

`needRejoin` …FIXME

Note	`needRejoin` is used when…FIXME

`timeToNextPoll` Method

```
FIXME
```

`timeToNextPoll` …FIXME

Note	`timeToNextPoll` is used when…FIXME

Polling for Coordinator Events — `poll` Method

```
FIXME
```

`poll` …FIXME

Note	`poll` is used when…FIXME

`commitOffsetsSync` Method

```
FIXME
```

`commitOffsetsSync` …FIXME

Note	`commitOffsetsSync` is used when…FIXME

`commitOffsetsAsync` Method

```
FIXME
```

`commitOffsetsAsync` …FIXME

Note	`commitOffsetsAsync` is used when…FIXME

`maybeAutoCommitOffsetsNow` Method

```
FIXME
```

`maybeAutoCommitOffsetsNow` …FIXME

Note	`maybeAutoCommitOffsetsNow` is used when…FIXME

`addMetadataListener` Method

```
FIXME
```

`addMetadataListener` …FIXME

Note	`addMetadataListener` is used when…FIXME

`commitOffsetsSync` Method

```
FIXME
```

`commitOffsetsSync` …FIXME

Note	`commitOffsetsSync` is used when…FIXME

`fetchCommittedOffsets` Method

```
FIXME
```

`fetchCommittedOffsets` …FIXME

Note	`fetchCommittedOffsets` is used when…FIXME

Creating ConsumerCoordinator Instance

`ConsumerCoordinator` takes the following when created:

- `LogContext`
- ConsumerNetworkClient
- Group ID
- `rebalanceTimeoutMs`
- `sessionTimeoutMs`
- `heartbeatIntervalMs`
- Collection of `PartitionAssignors`
- Metadata
- `SubscriptionState`
- `Metrics`
- Prefix of the metric group

- `Time`
- `retryBackoffMs`
- `autoCommitEnabled` flag
- `autoCommitIntervalMs`
- `ConsumerInterceptors`
- `excludeInternalTopics` flag
- `leaveGroupOnClose` flag

`ConsumerCoordinator` initializes the internal registries and counters.

In the end, `ConsumerCoordinator` requests Metadata to update and addMetadataListener.

ConsumerInterceptor

Example

```
package pl.jaceklaskowski.kafka

import java.util

import org.apache.kafka.clients.consumer.{ConsumerInterceptor, ConsumerRecords, Offset
AndMetadata}
import org.apache.kafka.common.TopicPartition

class KafkaInterceptor extends ConsumerInterceptor[String, String] {
  override def onConsume(records: ConsumerRecords[String, String]):
      ConsumerRecords[String, String] = {
    println(s"KafkaInterceptor.onConsume")
    import scala.collection.JavaConverters._
    records.asScala.foreach { r =>
      println(s"=> $r")
    }
    records
  }

  override def close(): Unit = {
    println("KafkaInterceptor.close")
  }

  override def onCommit(offsets: util.Map[TopicPartition, OffsetAndMetadata]): Unit =
{
    println("KafkaInterceptor.onCommit")
    println(s"$offsets")
  }

  override def configure(configs: util.Map[String, _]): Unit = {
    println(s"KafkaInterceptor.configure($configs)")
  }
}
```

`onConsume` Method

Caution	FIXME

ConsumerNetworkClient

`ConsumerNetworkClient` is a high-level Kafka consumer that…FIXME

`ConsumerNetworkClient` is created for KafkaConsumer and AdminClient.

Table 1. ConsumerNetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
`pendingCompletion`	
`unsent`	`UnsentRequests` with pending requests per node that have not been sent yet (i.e. awaiting transmission).

Tip	Enable `DEBUG` logging level for `org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient` logger to see w inside. Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient=D` Refer to Logging.

`checkDisconnects` Internal Method

```
void checkDisconnects(long now)
```

`checkDisconnects` …FIXME

Note	`checkDisconnects` is used exclusively when `ConsumerNetworkClient` is requested to poll.

"Sending" Request to Broker — `send` Method

```
RequestFuture<ClientResponse> send(Node node, AbstractRequest.Builder<?> requestBuilde
r)
```

`send` creates a `RequestFutureCompletionHandler` and requests the KafkaClient for a new ClientRequest (with the `RequestFutureCompletionHandler` and expecting a response).

`send` records the new `ClientRequest` with the `node` in unsent internal registry.

`send` requests the KafkaClient to wake up.

Note	`send` is used…FIXME

`wakeup` Method

```
void wakeup()
```

`wakeup` prints out the following DEBUG message to the logs:

```
Received user wakeup
```

`wakeup` turns the internal wakeup flag on and requests KafkaClient to wakeup

Note	`wakeup` is used when…FIXME

Ensuring Fresh Metadata (and Possibly Blocking Until It Refreshes) — `ensureFreshMetadata` Method

```
void ensureFreshMetadata()
```

`ensureFreshMetadata` waits for metadata update when Metadata was requested for update or time to the next update is now.

Note	`ensureFreshMetadata` is used when `ConsumerCoordinator` does onJoinComplete, performAssignment and poll.

`pendingRequestCount` Method

Caution	FIXME

`leastLoadedNode` Method

Caution	FIXME

`poll` Method

```
void poll(long timeout)
void poll(long timeout, long now, PollCondition pollCondition)
void poll(long timeout, long now, PollCondition pollCondition, boolean disableWakeup)
void poll(RequestFuture<?> future)
boolean poll(RequestFuture<?> future, long timeout)
```

`poll` …FIXME

Note	`poll` is used when: • `KafkaConsumer` polls once • `ConsumerNetworkClient` does pollNoWakeup, awaitMetadataUpdate, awaitPendingRequests • `AbstractCoordinator` does `ensureCoordinatorReady` and `joinGroupIfNeeded` • `ConsumerCoordinator` commitOffsetsSync and fetchCommittedOffsets • `Fetcher` is requested for topic metadata or retrieveOffsetsByTimes

Waiting for Metadata Update — `awaitMetadataUpdate` Method

```
boolean awaitMetadataUpdate(long timeout)
```

`awaitMetadataUpdate` …FIXME

Note	`awaitMetadataUpdate` is used when…FIXME

`awaitPendingRequests` Method

Caution	FIXME

`pollNoWakeup` Method

```
void pollNoWakeup()
```

`pollNoWakeup` …FIXME

Note	`pollNoWakeup` is used when: • `KafkaConsumer` polls • `AbstractCoordinator` does `maybeLeaveGroup` • `HeartbeatThread` runs • `ConsumerCoordinator` does `commitOffsetsAsync`

Creating ConsumerNetworkClient Instance

`ConsumerNetworkClient` takes the following when created:

- `LogContext`
- KafkaClient
- Metadata
- `Time`
- `retryBackoffMs`
- `requestTimeoutMs`

`ConsumerNetworkClient` initializes the internal registries and counters.

Broker Nodes — Kafka Servers

Note	A **Kafka server**, a **Kafka broker** and a **Kafka node** all refer to the same concept and are synonyms (see the scaladoc of KafkaServer).

A **Kafka broker** is modelled as KafkaServer that hosts topics.

Note	Given topics are always partitioned across brokers in a cluster a single broker hosts topic partitions of one or more topics actually (even when a topic is only partitioned to just a single partition).

Quoting Broker article (from Wikipedia, the free encyclopedia):

> A broker is an individual person who arranges transactions between a buyer and a seller for a commission when the deal is executed.

> A broker's prime responsibility is to bring sellers and buyers together and thus a broker is the third-person facilitator between a buyer and a seller.

A Kafka broker receives messages from producers and stores them on disk keyed by unique **offset**.

A Kafka broker allows consumers to fetch messages by topic, partition and offset.

Kafka brokers can create a Kafka cluster by sharing information between each other directly or indirectly using Zookeeper.

A Kafka cluster has exactly one broker that acts as the Controller.

You can start a single Kafka broker using `kafka-server-start.sh` script.

Starting Kafka Broker

Start Zookeeper.

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Only when Zookeeper is up and running you can start a Kafka server (that will connect to Zookeeper).

```
./bin/kafka-server-start.sh config/server.properties
```

Tip	Read kafka-server-start.sh script.

`kafka-server-start.sh` script

`kafka-server-start.sh` starts a Kafka broker.

```
$ ./bin/kafka-server-start.sh
USAGE: ./bin/kafka-server-start.sh [-daemon] server.properties [--override property=va
lue]*
```

Note	Before you run `kafka-server-start.sh` make sure that Zookeeper is up and running. Use `zookeeper-server-start` shell script.

`kafka-server-start.sh` uses `config/log4j.properties` for logging configuration that you can override using `KAFKA_LOG4J_OPTS` environment variable.

```
KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:config/log4j.properties"
```

`kafka-server-start.sh` accepts `KAFKA_HEAP_OPTS` and `EXTRA_ARGS` environment variables.

Command-line options:

1. `-name` — defaults to `kafkaServer` when in daemon mode.
2. `-loggc` — enabled when in daemon mode.
3. `-daemon` — enables daemon mode.
4. `--override property=value` — `value` that should override the value set for `property` in `server.properties` file.

```
$ ./bin/kafka-server-start.sh config/server.properties --override broker.id=100
...
INFO [KafkaServer id=100] started (kafka.server.KafkaServer)
```

Broker

`Broker` represents a Kafka broker that has an id, a host, a port, communication endpoints (and few other properties).

`createBroker` Method

```
createBroker(id: Int, brokerInfoString: String): Broker
```

`createBroker` …FIXME

Note	`createBroker` is used when…FIXME

Topics

Topics are virtual groups of one or many partitions across Kafka brokers in a Kafka cluster.

A single Kafka broker stores messages in a partition in an ordered fashion, i.e. appends them one message after another and creates a log file.

Producers write messages to the tail of these logs that consumers read at their own pace.

Kafka scales topic consumption by distributing partitions among a consumer group, which is a set of consumers sharing a common group identifier.

Partitions

Partitions with messages — topics can be partitioned to improve read/write performance and resiliency. You can lay out a topic (as partitions) across a cluster of machines to allow data streams larger than the capability of a single machine. Partitions are log files on disk with sequential write only. Kafka guarantees message ordering in a partition.

The **log end offset** is the offset of the last message written to a log.

The **high watermark offset** is the offset of the last message that was successfully copied to all of the log's replicas.

Note	A consumer can only read up to the high watermark offset to prevent reading unreplicated messages.

Messages

Messages are the data that brokers store in the partitions of a topic.

Messages are sequentially appended to the end of the partition log file and numbered by unique offsets. They are persisted on disk (aka *disk-based persistence*) and replicated within the cluster to prevent data loss. It has an in-memory page cache to improve data reads. Messages are in partitions until deleted when **TTL** occurs or after **compaction**.

Offsets

Offsets are message positions in a topic.

Kafka Clients

Producers

Multiple concurrent **producers** that send (aka *push*) messages to topics which is appending the messages to the end of partitions. They can batch messages before they are sent over the wire to a topic. Producers support message compression. Producers can send messages in synchronous (with acknowledgement) or asynchronous mode.

```
import collection.JavaConversions._
import org.apache.kafka.common.serialization._
import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerRecord

val cfg = Map(
  "bootstrap.servers" -> "localhost:9092",
  "key.serializer" -> classOf[IntegerSerializer],
  "value.serializer" -> classOf[StringSerializer])
val producer = new KafkaProducer[Int, String](cfg)
val msg = new ProducerRecord(topic = "my-topic", key = 1, value = "hello")

scala> val f = producer.send(msg)
f: java.util.concurrent.Future[org.apache.kafka.clients.producer.RecordMetadata] = org
.apache.kafka.clients.producer.internals.FutureRecordMetadata@2e9e8fe

scala> f.get
res7: org.apache.kafka.clients.producer.RecordMetadata = my-topic-0@1

producer.close
```

Kafka Consumers

Multiple concurrent **consumers** read (aka *pull*) messages from topics however they want using offsets. Unlike typical messaging systems, Kafka consumers pull messages from a topic using offsets.

A **Kafka consumer** consume records from a Kafka cluster.

Note	Kafka 0.9.0.0 was about introducing a brand new Consumer API *aka* **New Consumer**.

When a consumer is created, it requires bootstrap.servers which is the initial list of brokers to discover the full set of alive brokers in a cluster from.

A consumer has to subscribe to the topics it wants to read messages from called topic subscription.

Caution	FIXME Building a custom consumption strategy

Using Kafka Consumer API requires **kafka-clients** dependency in your project.

```
val kafka = "0.10.2.1"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafka

// You should also define the logging binding for slf4j
// Kafka uses slf4j for logging
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

Consumer Contract

```
public interface Consumer<K, V> extends Closeable {
  // FIXME more methods...
  ConsumerRecords<K, V> poll(long timeout)
}
```

Table 1. Consumer Contract

Method	Description
`poll`	Used to…

Topic Subscription

Topic Subscription is the process of announcing the topics a consumer wants to read messages from.

```
void subscribe(Collection<String> topics)
void subscribe(Collection<String> topics, ConsumerRebalanceListener callback)
void subscribe(Pattern pattern, ConsumerRebalanceListener callback)
```

Note	`subscribe` method is not incremental and you always must include the full list of topics that you want to consume from.

You can change the set of topics a consumer is subscrib to at any time and (given the note above) any topics previously subscribed to will be replaced by the new list after `subscribe`.

Automatic and Manual Partition Assignment

Caution	FIXME

Consumer Groups

A **consumer group** is a set of Kafka consumers that share a common link:a set of consumers sharing a common group identifier#group.id[group identifier].

Partitions in a topic are assigned to exactly one member in a consumer group.

Group Coordination Protocol

Caution	FIXME

- the new consumer uses a group coordination protocol built into Kafka
- For each group, one of the brokers is selected as the group coordinator. The coordinator is responsible for managing the state of the group. Its main job is to mediate partition assignment when new members arrive, old members depart, and when topic metadata changes. The act of reassigning partitions is known as rebalancing the group.
- When a group is first initialized, the consumers typically begin reading from either the earliest or latest offset in each partition. The messages in each partition log are then read sequentially. As the consumer makes progress, it commits the offsets of messages it has successfully processed.

- When a partition gets reassigned to another consumer in the group, the initial position is set to the last committed offset. If a consumer suddenly crashed, then the group member taking over the partition would begin consumption from the last committed offset (possibly reprocessing messages that the failed consumer would have processed already but not committed yet).

Further reading or watching

1. Introducing the Kafka Consumer: Getting Started with the New Apache Kafka 0.9 Consumer Client

RequestCompletionHandler

`RequestCompletionHandler` is the contract to attach an action that is executed when a request is complete, i.e. the corresponding response has been received or there was a disconnection while handling the request.

```
package org.apache.kafka.clients;

public interface RequestCompletionHandler {
    public void onComplete(ClientResponse response);
}
```

Table 1. RequestCompletionHandler's Known Implementations

Name	Description
`RequestFutureCompletionHandler`	
`TxnRequestHandler`	
`TransactionMarkerRequestCompletionHandler`	

ClientResponse

`ClientResponse` is…FIXME

`onComplete` Method

```
void onComplete()
```

`onComplete` triggers RequestCompletionHandler if defined.

Note	`onComplete` is used exclusively when `NetworkClient` completeResponses

Clusters

A Kafka **cluster** is the central data exchange backbone for an organization.

kafka-consumer-groups.sh Shell Script

`kafka-consumer-groups.sh` is a shell script that…FIXME

ConsumerGroupCommand

`ConsumerGroupCommand` is a standalone command-line application that is used for the following actions:

- Listing all consumer groups (--list option)
- Describing a consumer group (`--describe` option)
- Resetting consumer group offsets (`--reset-offsets` option)
- (only for the old Zookeeper-based consumer API) Deleting consumer group info (`--delete` option)

`ConsumerGroupCommand` can be executed as kafka-consumer-groups.sh shell script.

Note	Quoting Kafka 0.9.0.0: > The `kafka-consumer-offset-checker.sh` (`kafka.tools.ConsumerOffsetChecker`) has been deprecated. Going forward, please use `kafka-consumer-groups.sh` (`kafka.admin.ConsumerGroupCommand`) for this functionality.

`main` Method

```
main(args: Array[String]): Unit
```

`main` parses and checks the command-line arguments (using `ConsumerGroupCommandOptions`).

`main` creates a KafkaConsumerGroupService.

Note	`main` creates the old `ZkConsumerGroupService` when `--zookeeper` option for the old client API is used.

`main` branches per option used.

- --list option
- Describing a consumer group (`--describe` option)
- Resetting consumer group offsets (`--reset-offsets` option)

Caution	FIXME

list Option

`main` simply request groups from the consumer group service (e.g. `KafkaConsumerGroupService` for the new consumer API) and prints them out to the console.

KafkaConsumerGroupService

`KafkaConsumerGroupService` is a ConsumerGroupService that ConsumerGroupCommand uses for listing, describing and resetting offsets of consumer groups.

`KafkaConsumerGroupService` is created exclusively when `ConsumerGroupCommand` is executed (as a standalone command-line application, i.e. using kafka-consumer-groups.sh shell script).

`KafkaConsumerGroupService` is used by ConsumerGroupCommand for consumer groups that use the new Java consumer API (and hence do not use Zookeeper to store information).

`KafkaConsumerGroupService` uses AdminClient for the actions.

Listing All Consumer Groups — `listGroups` Method

```
listGroups(): List[String]
```

Note	`listGroups` is a part of ConsumerGroupService Contract.

`listGroups` requests AdminClient for all consumer groups and takes their group ids.

Note	`listGroups` is used exclusively when `ConsumerGroupCommand` is requested for all consumer groups using `--list` option.

`describeGroup` Method

Caution	FIXME

`resetOffsets` Method

Caution	FIXME

Creating KafkaConsumerGroupService Instance

`KafkaConsumerGroupService` takes the following when created:

- `ConsumerGroupCommandOptions`

`KafkaConsumerGroupService` initializes the internal registries and counters.

`prepareOffsetsToReset` Method

```
FIXME
```

`prepareOffsetsToReset` …FIXME

Note	`prepareOffsetsToReset` is used when…FIXME

`getPartitionsToReset` Method

```
FIXME
```

`getPartitionsToReset` …FIXME

Note	`getPartitionsToReset` is used when…FIXME

`collectGroupAssignment` Method

```
FIXME
```

`collectGroupAssignment` …FIXME

Note	`collectGroupAssignment` is used when…FIXME

ConsumerGroupService

`ConsumerGroupService` is…FIXME

KafkaAdminClient

`KafkaAdminClient` is a AdminClient that…FIXME

`KafkaAdminClient` is created when…FIXME

`describeTopics` Method

```
DescribeTopicsResult describeTopics(final Collection<String> topicNames, DescribeTopic
sOptions options)
```

`describeTopics` …FIXME

Note	`describeTopics` is used when…FIXME

`alterReplicaLogDirs` Method

```
AlterReplicaLogDirsResult alterReplicaLogDirs(
  Map<TopicPartitionReplica, String> replicaAssignment,
  final AlterReplicaLogDirsOptions options)
```

`alterReplicaLogDirs` …FIXME

Note	`alterReplicaLogDirs` is used when…FIXME

Creating KafkaAdminClient Instance

`KafkaAdminClient` takes the following when created:

- `AdminClientConfig`
- client ID
- sanitized client ID
- `Time`
- Metadata
- `Metrics`
- KafkaClient

- `TimeoutProcessorFactory`
- `LogContext`

`KafkaAdminClient` initializes the internal registries and counters.

AdminClient

`AdminClient` …FIXME

`AdminClient` uses the admin-client-network-thread to poll continuously (using ConsumerNetworkClient).

Note	`AdminClient` is deprecated and is going to be replaced by `KafkaAdminClient` (after the public API becomes stable).

Table 1. ConsumerNetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
`networkThread`	**admin-client-network-thread**

Tip	Enable `DEBUG` logging level for `kafka.admin.AdminClient` logger to see what happens inside. Add the following line to `config/tools-log4j.properties`: `log4j.logger.kafka.admin.AdminClient=DEBUG` Refer to Logging.

Creating AdminClient — `create` Method

```
create(config: AdminConfig): AdminClient
```

`create` …FIXME

Note	`create` is used when…FIXME

Sending Request (Using ConsumerNetworkClient) — `send` Internal Method

```
send(
  target: Node,
  api: ApiKeys,
  request: AbstractRequest.Builder[_ <: AbstractRequest]): AbstractResponse
```

`send` requests ConsumerNetworkClient to send the input `request` to the `target`.

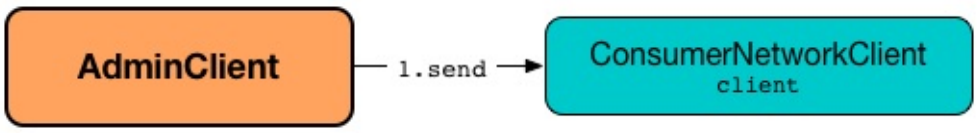

Figure 1. AdminClient Sends Requests Using ConsumerNetworkClient

`send` records the future result in pendingFutures registry.

`send` waits until the future result has come after which it is removed from pendingFutures registry.

Caution	FIXME Why is the future result registed in `pendingFutures` ?

When the future result has completed, `send` takes the response body for a successful result or reports a `RuntimeException`.

Note	`send` is used when `AdminClient` does sendAnyNode, listGroups, getApiVersions, listGroupOffsets and describeConsumerGroupHandler.

`findAllBrokers` Method

```
findAllBrokers(): List[Node]
```

`findAllBrokers` creates a `Metadata` API request and sends it to one of the bootstrap brokers.

`findAllBrokers` returns the nodes from the cluster metadata of the `MetadataResponse`.

Note	`findAllBrokers` is used when `AdminClient` does awaitBrokers, lists all groups per broker and listAllBrokerVersionInfo.

Sending API Request to Bootstrap Broker — `sendAnyNode` Internal Method

```
sendAnyNode(api: ApiKeys, request: AbstractRequest.Builder[_ <: AbstractRequest]): Abs
tractResponse
```

`sendAnyNode` walks over bootstrapBrokers and sends the input `request`.

`sendAnyNode` exits in case of `AuthenticationException`.

In case of any other `Exceptions` (but `AuthenticationException`) `sendAnyNode` prints DEBUG message to the logs and tries the remaining brokers.

```
Request [api] failed against node [broker]
```

When no brokers succeeded, `sendAnyNode` reports a `RuntimeException` with the following message:

```
Request [api] failed on brokers [bootstrapBrokers]
```

Note	`sendAnyNode` is used when `AdminClient` is requested to find the coordinator, finds all brokers and deleteRecordsBefore.

`findCoordinator` Method

```
FIXME
```

`findCoordinator` ...FIXME

Note	`findCoordinator` is used when `KafkaConsumerGroupService` ...FIXME

`deleteRecordsBefore` Method

```
FIXME
```

`deleteRecordsBefore` ...FIXME

Note	`deleteRecordsBefore` is used when `KafkaConsumerGroupService` ...FIXME

`listGroups` Method

```
FIXME
```

`listGroups` ...FIXME

Note	`listGroups` is used when `KafkaConsumerGroupService` ...FIXME

`listAllBrokerVersionInfo` Method

```
FIXME
```

`listAllBrokerVersionInfo` ...FIXME

Note	`listAllBrokerVersionInfo` is used when `KafkaConsumerGroupService` ...FIXME

`awaitBrokers` Method

```
FIXME
```

`awaitBrokers` ...FIXME

Note	`awaitBrokers` is used when `KafkaConsumerGroupService` ...FIXME

`listAllConsumerGroups` Method

```
FIXME
```

`listAllConsumerGroups` ...FIXME

Note	`listAllConsumerGroups` is used when `KafkaConsumerGroupService` ...FIXME

Listing All Groups per Broker — `listAllGroups` Method

```
listAllGroups(): Map[Node, List[GroupOverview]]
```

`listAllGroups` finds all brokers (in a cluster) and collects their groups.

Note	`listAllGroups` is used when `AdminClient` does listAllConsumerGroups and listAllGroupsFlattened.

`listAllGroupsFlattened` Method

```
listAllGroupsFlattened(): List[GroupOverview]
```

`listAllGroupsFlattened` simply takes all groups (across all brokers in a cluster).

Note	`listAllGroupsFlattened` is used excusively when `AdminClient` lists all consumer groups.

Listing All Consumer Groups — `listAllConsumerGroupsFlattened` Method

```
listAllConsumerGroupsFlattened(): List[GroupOverview]
```

`listAllConsumerGroupsFlattened` takes all groups with `consumer` protocol type.

Note	`listAllConsumerGroupsFlattened` is used exclusively when `KafkaConsumerGroupService` is requested for all consumer groups.

`listGroupOffsets` Method

```
FIXME
```

`listGroupOffsets` …FIXME

Note	`listGroupOffsets` is used when `KafkaConsumerGroupService` …FIXME

ReassignPartitionsCommand

`ReassignPartitionsCommand` is…FIXME

Starting ReassignPartitionsCommand on Command Line — `main` Entry Method

```
main(args: Array[String]): Unit
```

`main` …FIXME

`executeAssignment` Method

```
executeAssignment(
  zkUtils: ZkUtils,
  adminClientOpt: Option[AdminClient],
  opts: ReassignPartitionsCommand.ReassignPartitionsCommandOptions): Unit

executeAssignment(
  zkUtils: ZkUtils,
  adminClientOpt: Option[AdminClient],
  reassignmentJsonString: String,
  throttle: ReassignPartitionsCommand.Throttle,
  timeoutMs: Long = 10000L): Unit
```

`executeAssignment` …FIXME

Note	`executeAssignment` is used exclusively when `ReassignPartitionsCommand` is started on command line with `execute` option.

`reassignPartitions` Method

```
reassignPartitions(throttle: Throttle = NoThrottle, timeoutMs: Long = 10000L): Boolean
```

`reassignPartitions` …FIXME

Note	`reassignPartitions` is used exclusively when `ReassignPartitionsCommand` executeAssignment.

`alterReplicaLogDirsIgnoreReplicaNotAvailable` Internal Method

```
alterReplicaLogDirsIgnoreReplicaNotAvailable(
  replicaAssignment: Map[TopicPartitionReplica, String],
  adminClient: JAdminClient,
  timeoutMs: Long): Set[TopicPartitionReplica]
```

`alterReplicaLogDirsIgnoreReplicaNotAvailable` ...FIXME

Note	`alterReplicaLogDirsIgnoreReplicaNotAvailable` is used exclusively when `ReassignPartitionsCommand` reassignPartitions

TopicCommand

`kafka.admin.TopicCommand`

```
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --par
titions 1 --topic my-topic

./bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic my-topic
```

Sensor

`Sensor` is…FIXME

MetricsReporter

JmxReporter

`JmxReporter` is a metrics reporter that is always included in metric.reporters setting with `kafka.consumer` metrics prefix.

ProducerMetrics

`ProducerMetrics` is...FIXME

```
// Generate HTML with the metrics
$ ./bin/kafka-run-class.sh org.apache.kafka.clients.producer.internals.ProducerMetrics
 > metrics.html
```

SenderMetrics

`SenderMetrics` is…FIXME

Kafka Tools

`ConsoleProducer`

`kafka.tools.ConsoleProducer`

```
./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-topic
```

`ConsoleConsumer`

`kafka.tools.ConsoleConsumer`

```
./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my-topic
```

kafka-configs.sh Shell Script

`kafka-configs.sh` is a shell script that…FIXME

```
./bin/kafka-configs.sh \
  --zookeeper localhost:2181 \
  --alter \
  --entity-type topics \
  --entity-name test \
  --add-config retention.ms=5000
```

kafka-topics.sh Shell Script

`kafka-topics.sh` is a shell script that…FIXME

Kafka Properties

Table 1. Proper

Name	Default Value	Importance	
`auto.offset.reset`	`latest`	No	Reset policy — what to any more on the serve • **earliest** — automa • **latest** — automati • **none** — throw an group • anything else: thro
`authorizer.class.name`			
bootstrap.servers	(empty)	Yes	A comma-separated lis cluster, e.g. `localhost`
`broker.rack`			
`check.crcs`			
client.id	(random-generated)		
`delete.topic.enable`	`true`	High	Enables topic deletion Note | Deleting to
enable.auto.commit			
`fetch.min.bytes`			
`fetch.max.bytes`			
`fetch.max.wait.ms`			
group.id			The name of the consu
`heartbeat.interval.ms`			The expected time bet management facilities.
`host.name`		(empty)	The hostname the defa

`inter.broker.protocol.version`			
`interceptor.classes`	(empty)		Comma-separated list `props.put(ConsumerCo`
`listeners`		(empty)	The addresses the soc
`key.deserializer`			How to deserialize me
`max.block.ms`			
`max.partition.fetch.bytes`			How to deserialize me
`max.poll.records`			
`metadata.max.age.ms`			
`metric.reporters`	JmxReporter		The list of fully-qualifie
`metrics.num.samples`			Number of samples to
`metrics.sample.window.ms`			Time window (in millise
`num.io.threads`	`8`		The number of threads
`num.network.threads`	`3`		The number of threads
`port`		(empty)	The port the default en
`rebalance.timeout.ms`			The maximum allowed
`receive.buffer.bytes`			The hint about the size reading data. If the val
`replica.lag.time.max.ms`			
`replica.socket.timeout.ms`			
retry.backoff.ms			Time to wait before att This avoids repeatedly
`request.timeout.ms`			

`sasl.enabled.mechanisms`			
`send.buffer.bytes`			The hint about the size sending data. If the va
`session.timeout.ms`	10000	High	The timeout used to de
`value.deserializer`			How to deserialize me

Caution	FIXME What's worker?

```
// requires org.apache.kafka:connect-runtime:0.10.0.1 dependency

import org.apache.kafka.connect.runtime.distributed.DistributedConfig
DistributedConfig.SESSION_TIMEOUT_MS_CONFIG
```

Caution	FIXME How to know the current value of a setting on a producer's and a consumer's side?

bootstrap.servers Property

`bootstrap.servers` is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers in a "bootstrap" Kafka cluster that a Kafka client connects to initially to bootstrap itself.

A host and port pair uses `:` as the separator.

```
localhost:9092
localhost:9092,another.host:9092
```

`bootstrap.servers` provides the initial hosts that act as the starting point for a Kafka client to discover the full set of alive servers in the cluster.

Note	Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list does not have to contain the full set of servers (you may want more than one, though, in case a server is down).

Note	Clients (producers or consumers) make use of all servers irrespective of which servers are specified in `bootstrap.servers` for bootstrapping.

Tip	Use `org.apache.kafka.clients.CommonClientConfigs.BOOTSTRAP_SERVERS_CONFIG` public value to refer to the property.

client.id Property

An optional identifier of a Kafka consumer (in a consumer group) that is passed to a Kafka broker with every request.

The sole purpose of this is to be able to track the source of requests beyond just ip and port by allowing a logical application name to be included in Kafka logs and monitoring aggregates.

enable.auto.commit Property

`enable.auto.commit` …FIXME

By default, as the consumer reads messages from Kafka, it will periodically commit its current offset (defined as the offset of the next message to be read) for the partitions it is reading from back to Kafka. Often you would like more control over exactly when offsets are committed. In this case you can set `enable.auto.commit` to `false` and call the `commit` method on the consumer.

group.id Property

`group.id` specifies the name of the consumer group a Kafka consumer belongs to.

When the Kafka consumer is constructed and `group.id` does not exist yet (i.e. there are no existing consumers that are part of the group), the consumer group will be created automatically.

Note	If all consumers in a group leave the group, the group is automatically destroyed.

retry.backoff.ms Property

`retry.backoff.ms` is the time to wait before attempting to retry a failed request to a given topic partition.

This avoids repeatedly sending requests in a tight loop under some failure scenarios.

Logging

Kafka Broker

A Kafka broker (started using `kafka-server-start.sh`) uses `config/log4j.properties` for logging configuration.

`config/log4j.properties` defaults to INFO logging level with `stdout` and `kafkaAppender` (that spit messages to standard output and `logs/server.log`, respectively).

```
log4j.rootLogger=INFO, stdout, kafkaAppender

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.appender.kafkaAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.kafkaAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.kafkaAppender.File=${kafka.logs.dir}/server.log
log4j.appender.kafkaAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.kafkaAppender.layout.ConversionPattern=[%d] %p %m (%c)%n
```

If you want to change the logging level of a logger, e.g. `kafka.controller.KafkaController`, to `DEBUG` add the following line to `config/log4j.properties` and restart the server.

```
log4j.logger.kafka.controller.KafkaController=DEBUG, stdout
```

Kafka Tools

Kafka tools like `kafka-console-consumer.sh` (that uses KafkaConsumer under the covers) use `config/tools-log4j.properties` file.

Note	Kafka tools use `bin/kafka-run-class.sh` to execute their implementations.

Kafka uses Simple Logging Facade for Java (SLF4J) for logging.

Tip

Use `slf4j-simple` library dependency in Scala applications (in `build.sbt`) for basic logging where messages of level `INFO` and higher are printed to `System.err`.

build.sbt

```
libraryDependencies += "org.slf4j" % "slf4j-simple" % "1.8.0-alpha2"
```

Replace slf4j's simple binding to switch between logging frameworks (e.g. `slf4j-log4j12` for log4j).

build.sbt

```
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

Note

With logback's configuration (as described in the above tip) you may see the follow

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/org.slf4j/slf4j-log4
SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/ch.qos.logback/logba
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanatior
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

Commenting out `logback-classic` in `build.sbt` resolves it.

build.sbt

```
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
//libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

FIXME: Explain why the commenting out is required?

`log4j.properties` Logging Configuration File

log4j.properties

```
log4j.rootLogger=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.org.apache.kafka.clients.consumer.ConsumerConfig=DEBUG
```

Tip

Save `log4j.properties` in `src/main/resources` in your Kafka application's project.

KAFKA_LOG4J_OPTS Environment Variable

You can use `KAFKA_LOG4J_OPTS` environment variable to specify the log4j configuration to use.

```
KAFKA_LOG4J_OPTS=-Dlog4j.configuration=file:[your-log4j-configuration-here]
```

Note	Unless defined, `kafka-run-class.sh` sets it to `config/tools-log4j.properties`.

Gradle Tips

Building Kafka Distribution

```
gradle -PscalaVersion=2.11.8 clean releaseTarGz install
```

It takes around 2 minutes (after all the dependencies were downloaded once).

After the command, you can unpack the release as follows:

```
tar -zxvf core/build/distributions/kafka_2.11-0.10.1.0-SNAPSHOT.tgz
```

Executing Single Test

```
gradle -PscalaVersion=2.11.8 :core:test --no-rebuild --tests "*PlaintextProducerSendTe
st"
```

Zookeeper Tips

The zookeeper shell shipped with Kafka works with no support for command line history because jline jar is missing (see KAFKA-2385).

A solution is to use the official distribution of Apache Zookeeper (**3.4.10** as of this writing) from Apache ZooKeeper Releases.

Once downloaded, use `./bin/zkCli.sh` to connect to Zookeeper that is used for Kafka.

```
$ ./bin/zkCli.sh -server localhost:2181
...
JLine support is enabled

[zk: localhost:2181(CONNECTED) 0] ls /
[cluster, controller_epoch, controller, brokers, zookeeper, admin, isr_change_notifica
tion, consumers, log_dir_event_notification, latest_producer_id_block, config]

[zk: localhost:2181(CONNECTED) 5] get /controller
{"version":1,"brokerid":200,"timestamp":"1506417983145"}
cZxid = 0x11b
ctime = Tue Sep 26 11:26:23 CEST 2017
mZxid = 0x11b
mtime = Tue Sep 26 11:26:23 CEST 2017
pZxid = 0x11b
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15ebd811a0e0001
dataLength = 56
numChildren = 0
```

Kafka in Scala REPL for Interactive Exploration

Use the following `build.sbt` and execute `sbt` followed by `console` command (while inside the sbt shell).

Note	The reason for executing `console` command after sbt has started up is that command history did not work using the key-up and key-down keys. *YMMV*

build.sbt

```
name := "kafka-sandbox"

version := "1.0"

scalaVersion := "2.12.3"

//val kafkaVer = "0.11.0.1"
resolvers += Resolver.mavenLocal
val kafkaVer = "1.0.0-SNAPSHOT"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafkaVer
libraryDependencies += "org.apache.kafka" %% "kafka" % kafkaVer

val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

sbt with console command

```
➜  kafka-sandbox sbt
[info] Loading settings from plugins.sbt ...
[info] Loading project definition from /Users/jacek/dev/sandbox/kafka-sandbox/project
[info] Loading settings from build.sbt ...
[info] Set current project to kafka-sandbox (in build file:/Users/jacek/dev/sandbox/ka
fka-sandbox/)
[info] sbt server started at 127.0.0.1:4408
sbt:kafka-sandbox> console
[info] Starting scala interpreter...
Welcome to Scala 2.12.3 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144).
Type in expressions for evaluation. Or try :help.

scala> import kafka.utils._
import kafka.utils._

scala> ZkUtils.controllerZkData(1, System.currentTimeMillis())
res0: String = {"version":1,"brokerid":1,"timestamp":"1506162122097"}
```

WorkerGroupMember

Caution	FIXME WorkerCoordinator? DistributedHerder?

ConnectDistributed

`ConnectDistributed` is a command-line utility that runs Kafka Connect in distributed mode.

Caution	FIXME Doh, I'd rather not enter Kafka Connect yet. Not interested in it yet.

Further Reading or Watching

Videos

1. Apache Kafka Core Internals: A Deep Dive by Jun Rao, Confluent at Kafka Summit 2017 in New York City

Articles

1. Apache Kafka for Beginners - an excellent article that you should start your Kafka journey with.
2. Introduction to Apache Kafka™ for Python Programmers - using Python as the programming language, but offers enough exposure to the topics of the Kafka architecture.