Table of Contents

Part II — Build Tips

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.

Broker — Kafka Server

Note	Given the scaladoc of `KafkaServer` a **Kafka server**, a **Kafka broker** and a **Kafka node** all refer to the same concept and are hence considered synonyms.

A **Kafka broker** is a Kafka server that hosts topics.

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]*
```

`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.

Topics

Topics are virtual groups of partitions that a Kafka broker uses as a set of logs to store messages.

A 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.

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
```

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.

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 own consumption strategy

Using Kafka Consumer API requires the following dependency in your project (with `0.10.0.1` being the latest Kafka release):

```
libraryDependencies += "org.apache.kafka" % "kafka-clients" % 0.10.0.1
```

Consumer Contract

```
public interface Consumer<K, V> extends Closeable {
  // FIXME more method...
  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

KafkaConsumer

`KafkaConsumer` is…FIXME

```
// sandbox/kafka-sandbox
val props = new Properties()
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
props.put(ConsumerConfig.GROUP_ID_CONFIG, "my-kafka-consumer")
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, classOf[StringDeserializer].ge
tName)
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, classOf[StringDeserializer].
getName)

val consumer = new KafkaConsumer[String, String](props)
```

Note	`KafkaConsumer` is not safe for multi-threaded access and so you should use a single thread per instance.

Table 1. KafkaConsumer's Internal Registries and Counters

Name	Description
`client`	`ConsumerNetworkClient` Used when…FIXME
`fetcher`	`Fetcher` Used when…FIXME
`interceptors`	`ConsumerInterceptors` that holds ConsumerInterceptor instances (defined using interceptor.classes setting). Used when…FIXME
`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.

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

Caution	FIXME

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

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

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

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

Internally, `poll` polls once (for ConsumerRecords per TopicPartitions).

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

Caution	FIXME Make the above more user-friendly

`poll` returns the fetched `ConsumerRecords` when no `interceptors` are defined or passes them on to `ConsumerInterceptors` using `onConsume`

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

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

Creating KafkaConsumer Instance

`KafkaConsumer` takes the following when created:

- ConsumerConfig
- `Deserializer` for the keys
- `Deserializer` for the values

`KafkaConsumer` initializes the internal registries and counters.

Note	`KafkaConsumer` offers other methods to create itself, but they all eventually use the 3-argument constructor.

When created, you should see the following messages in the logs:

```
DEBUG Starting the Kafka consumer (org.apache.kafka.clients.consumer.KafkaConsumer)
```

A `KafkaConsumer` sets the internal `requestTimeoutMs` to request.timeout.ms that has to be greater than session.timeout.ms and fetch.max.wait.ms (you get `ConfigException` otherwise).

`clientId` property is set to client.id if defined or auto-generated. It is used for metrics with the tag `client-id` being `clientId`.

`metrics` property is set to the configured metrics reporters.

`retryBackoffMs` is set to retry.backoff.ms.

Caution	FIXME

When successfully created, you should see the following DEBUG in the logs:

```
DEBUG Kafka consumer created (org.apache.kafka.clients.consumer.KafkaConsumer)
```

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

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

ConsumerConfig

Caution	FIXME

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)")
  }
}
```

Clusters

A Kafka **cluster** is the central data exchange backbone for an organization.

MetricsReporter

JmxReporter

`JmxReporter` is a metrics reporter that is always included in metric.reporters setting with `kafka.consumer` metrics prefix.

Kafka Tools

`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
```

`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 Settings

Table 1. Settings

Setting	Default Value	Importance	
`auto.offset.reset`	`latest`	No	Reset policy — what to do w any more on the server (e.g • **earliest** — automaticall • **latest** — automatically • **none** — throw an excep group • anything else: throw an
`bootstrap.servers`	(empty)	Yes	A comma-separated list of `localhost:9092` or `localho` The client will make use of a bootstrapping and only impa cluster. Since these servers are just (which may change dynami want more than one, though
`client.id`	(random-generated)		A Consumer identifier string The purpose of this is to be logical application name to
`fetch.max.wait.ms`			
`group.id`			A unique string that identifie
`heartbeat.interval.ms`			The expected time between management facilities.
`interceptor.classes`	(empty)		Comma-separated list of Co `props.put(ConsumerConfig.`
`key.deserializer`			How to deserialize message
`metric.reporters`	JmxReporter		The list of fully-qualified clas

`metrics.num.samples`			Number of samples to comp
`metrics.sample.window.ms`			Time window (in millisecond
`rebalance.timeout.ms`			The maximum allowed time
`retry.backoff.ms`			Time to wait before attempt This avoids repeatedly send
`request.timeout.ms`			
`session.timeout.ms`	10000	High	The timeout used to detect
`value.deserializer`			How to deserialize message

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?

Logging

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_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`.

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.

KafkaServerStartable

`KafkaServerStartable` is a thin management layer to manage a single `KafkaServer` instance, i.e. to start and shut it down.

Table 1. KafkaServerStartable's Internal 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` creates a KafkaServer.

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` starts the managed `KafkaServer` (using `server` handler).

If starting `KafkaServer` throws an exception, `startup` terminates the JVM with status `1`. You should see the following FATAL message in the logs if that happens.

```
FATAL Fatal error during KafkaServerStartable startup. Prepare to shutdown
```

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

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

KafkaMetricsReporter

Caution	FIXME

KafkaServer

Caution	FIXME

`KafkaServer` acts as a Kafka broker.

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

Table 1. KafkaServer's Internal Registries and Counters

Name	Description
`reporters`	Collection of `MetricsReporter` Used when…FIXME

`startup` Method

```
startup(): Unit
```

`startup` starts a single Kafka server.

When `startup` starts, you should see the following INFO message in the logs:

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

`startup` starts `KafkaScheduler`.

You should see the following INFO message in the logs:

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

`startup` notifies cluster change listeners.

`startup` creates a `ReplicaManager` and starts it right afterwards.

`startup` creates a `KafkaController` and starts it.

`startup` creates a `AdminManager`.

`startup` creates a `KafkaApis`.

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

```
INFO [Kafka Server 0], 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.

Caution	FIXME

Note	`startup` is used exclusively when `KafkaServerStartable` is started.

`notifyClusterListeners` Method

Caution	FIXME

Creating KafkaServer Instance

`KafkaServer` takes the following when created:

1. KafkaConfig
2. `Time` (defaults to `Time.SYSTEM`)
3. `threadNamePrefix` (defaults to no prefix)
4. `kafkaMetricsReporters` — a collection of KafkaMetricsReporter (defaults to no reporters)

Caution	FIXME

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

KafkaConfig

Caution	FIXME

Kafka — Standalone Command-Line Application

`kafka.Kafka` is a standalone command-line application to start a Kafka broker.

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

`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.

KafkaScheduler

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

Table 1. KafkaScheduler's Internal 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` Refer to Logging.

Starting KafkaScheduler — `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

ReplicaManager

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

When started, `ReplicaManager` schedules isr-expiration and isr-change-propagation tasks.

Table 1. ReplicaManager's Internal Registries and Counters

Name	Description
`FIXME`	Internal cache with…FIXME Used when…FIXME

`maybeShrinkIsr` Internal Method

Caution	FIXME

`maybePropagateIsrChanges` Method

Caution	FIXME

`isr-expiration` Task

Caution	FIXME

`isr-change-propagation` Task

Caution	FIXME

Creating ReplicaManager Instance

`ReplicaManager` takes the following when created:

1. `config` — KafkaConfig
2. `Metrics`
3. `Time`
4. `ZkUtils`
5. `scheduler` — `Scheduler`

6. `logManager` — LogManager
7. `isShuttingDown` flag
8. `quotaManager` — `ReplicationQuotaManager`,
9. Optional `threadNamePrefix` (empty by default)

`ReplicaManager` initializes the internal registries and counters.

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

```
startup(): Unit
```

`startup` schedules the ISR-related tasks:

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

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

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

KafkaController

Caution	FIXME

`startup` Method

Caution	FIXME

Creating KafkaController Instance

`KafkaController` takes the following when created:

1. FIXME

KafkaApis

Caution	FIXME

Creating KafkaApis Instance

`KafkaApis` takes the following when created:

1. FIXME

AdminManager

Caution	FIXME

Creating AdminManager Instance

`AdminManager` takes the following when created:

1. FIXME

LogManager

Caution	FIXME

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"
```

Further Reading or Watching

Articles

1. Apache Kafka for Beginners - an excellent article that you should start your Kafka journey with.