kafka-python Documentation

Release 1.3.4

Dana Powers

Aug 13, 2017

Contents

Python client for the Apache Kafka distributed stream processing system. kafka-python is designed to function much like the official java client, with a sprinkling of pythonic interfaces (e.g., consumer iterators).

kafka-python is best used with newer brokers (0.10 or 0.9), but is backwards-compatible with older versions (to 0.8.0). Some features will only be enabled on newer brokers, however; for example, fully coordinated consumer groups – i.e., dynamic partition assignment to multiple consumers in the same group – requires use of 0.9 kafka brokers. Supporting this feature for earlier broker releases would require writing and maintaining custom leadership election and membership / health check code (perhaps using zookeeper or consul). For older brokers, you can achieve something similar by manually assigning different partitions to each consumer instance with config management tools like chef, ansible, etc. This approach will work fine, though it does not support rebalancing on failures. See Compatibility for more details.

Please note that the master branch may contain unreleased features. For release documentation, please see readthedocs and/or python's inline help.

```
>>> pip install kafka-python
```

CHAPTER 1

KafkaConsumer

`KafkaConsumer` is a high-level message consumer, intended to operate as similarly as possible to the official java client. Full support for coordinated consumer groups requires use of kafka brokers that support the Group APIs: kafka v0.9+.

See KafkaConsumer for API and configuration details.

The consumer iterator returns ConsumerRecords, which are simple namedtuples that expose basic message attributes: topic, partition, offset, key, and value:

```
>>> from kafka import KafkaConsumer
>>> consumer = KafkaConsumer('my_favorite_topic')
>>> for msg in consumer:
...     print (msg)
```

```
>>> # join a consumer group for dynamic partition assignment and offset commits
>>> from kafka import KafkaConsumer
>>> consumer = KafkaConsumer('my_favorite_topic', group_id='my_favorite_group')
>>> for msg in consumer:
...     print (msg)
```

```
>>> # manually assign the partition list for the consumer
>>> from kafka import TopicPartition
>>> consumer = KafkaConsumer(bootstrap_servers='localhost:1234')
>>> consumer.assign([TopicPartition('foobar', 2)])
>>> msg = next(consumer)
```

```
>>> # Deserialize msgpack-encoded values
>>> consumer = KafkaConsumer(value_deserializer=msgpack.loads)
>>> consumer.subscribe(['msgpackfoo'])
>>> for msg in consumer:
...     assert isinstance(msg.value, dict)
```

CHAPTER 2

KafkaProducer

`KafkaProducer` is a high-level, asynchronous message producer. The class is intended to operate as similarly as possible to the official java client. See KafkaProducer for more details.

```
>>> from kafka import KafkaProducer
>>> producer = KafkaProducer(bootstrap_servers='localhost:1234')
>>> for _ in range(100):
...     producer.send('foobar', b'some_message_bytes')
```

```
>>> # Block until a single message is sent (or timeout)
>>> future = producer.send('foobar', b'another_message')
>>> result = future.get(timeout=60)
```

```
>>> # Block until all pending messages are at least put on the network
>>> # NOTE: This does not guarantee delivery or success! It is really
>>> # only useful if you configure internal batching using linger_ms
>>> producer.flush()
```

```
>>> # Use a key for hashed-partitioning
>>> producer.send('foobar', key=b'foo', value=b'bar')
```

```
>>> # Serialize json messages
>>> import json
>>> producer = KafkaProducer(value_serializer=lambda v: json.dumps(v).encode('utf-8'))
>>> producer.send('fizzbuzz', {'foo': 'bar'})
```

```
>>> # Serialize string keys
>>> producer = KafkaProducer(key_serializer=str.encode)
>>> producer.send('flipflap', key='ping', value=b'1234')
```

```
>>> # Compress messages
>>> producer = KafkaProducer(compression_type='gzip')
>>> for i in range(1000):
...     producer.send('foobar', b'msg %d' % i)
```

CHAPTER 3

Thread safety

The KafkaProducer can be used across threads without issue, unlike the KafkaConsumer which cannot.

While it is possible to use the KafkaConsumer in a thread-local manner, multiprocessing is recommended.

CHAPTER 4

Compression

kafka-python supports gzip compression/decompression natively. To produce or consume lz4 compressed messages, you should install python-lz4 (pip install lz4). To enable snappy, install python-snappy (also requires snappy library). See Installation for more information.

CHAPTER 5

Protocol

A secondary goal of kafka-python is to provide an easy-to-use protocol layer for interacting with kafka brokers via the python repl. This is useful for testing, probing, and general experimentation. The protocol support is leveraged to enable a `check_version()` method that probes a kafka broker and attempts to identify which version it is running (0.8.0 to 0.10).

CHAPTER 6

Low-level

Legacy support is maintained for low-level consumer and producer classes, SimpleConsumer and SimpleProducer.

Usage

KafkaConsumer

```
from kafka import KafkaConsumer

# To consume latest messages and auto-commit offsets
consumer = KafkaConsumer('my-topic',
                         group_id='my-group',
                         bootstrap_servers=['localhost:9092'])
for message in consumer:
    # message value and key are raw bytes -- decode if necessary!
    # e.g., for unicode: `message.value.decode('utf-8')`
    print ("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,
                                          message.offset, message.key,
                                          message.value))

# consume earliest available messages, don't commit offsets
KafkaConsumer(auto_offset_reset='earliest', enable_auto_commit=False)

# consume json messages
KafkaConsumer(value_deserializer=lambda m: json.loads(m.decode('ascii')))

# consume msgpack
KafkaConsumer(value_deserializer=msgpack.unpackb)

# StopIteration if no message after 1sec
KafkaConsumer(consumer_timeout_ms=1000)

# Subscribe to a regex topic pattern
```

```
consumer = KafkaConsumer()
consumer.subscribe(pattern='^awesome.*')

# Use multiple consumers in parallel w/ 0.9 kafka brokers
# typically you would run each on a different server / process / CPU
consumer1 = KafkaConsumer('my-topic',
                          group_id='my-group',
                          bootstrap_servers='my.server.com')
consumer2 = KafkaConsumer('my-topic',
                          group_id='my-group',
                          bootstrap_servers='my.server.com')
```

There are many configuration options for the consumer class. See `KafkaConsumer` API documentation for more details.

KafkaProducer

```
from kafka import KafkaProducer
from kafka.errors import KafkaError

producer = KafkaProducer(bootstrap_servers=['broker1:1234'])

# Asynchronous by default
future = producer.send('my-topic', b'raw_bytes')

# Block for 'synchronous' sends
try:
    record_metadata = future.get(timeout=10)
except KafkaError:
    # Decide what to do if produce request failed...
    log.exception()
    pass

# Successful result returns assigned partition and offset
print (record_metadata.topic)
print (record_metadata.partition)
print (record_metadata.offset)

# produce keyed messages to enable hashed partitioning
producer.send('my-topic', key=b'foo', value=b'bar')

# encode objects via msgpack
producer = KafkaProducer(value_serializer=msgpack.dumps)
producer.send('msgpack-topic', {'key': 'value'})

# produce json messages
producer = KafkaProducer(value_serializer=lambda m: json.dumps(m).encode('ascii'))
producer.send('json-topic', {'key': 'value'})

# produce asynchronously
for _ in range(100):
    producer.send('my-topic', b'msg')

# block until all async messages are sent
producer.flush()
```

```
# configure multiple retries
producer = KafkaProducer(retries=5)
```

kafka-python API

KafkaConsumer

class kafka.**KafkaConsumer**(**topics*, ***configs*)

Consume records from a Kafka cluster.

The consumer will transparently handle the failure of servers in the Kafka cluster, and adapt as topic-partitions are created or migrate between brokers. It also interacts with the assigned kafka Group Coordinator node to allow multiple consumers to load balance consumption of topics (requires kafka >= 0.9.0.0).

The consumer is not thread safe and should not be shared across threads.

Parameters ***topics** (*str*) – optional list of topics to subscribe to. If not set, call `subscribe()` or `assign()` before consuming records.

Keyword Arguments

- **bootstrap_servers** – 'host[:port]' string (or list of 'host[:port]' strings) that the consumer should contact to bootstrap initial cluster metadata. This does not have to be the full node list. It just needs to have at least one broker that will respond to a Metadata API Request. Default port is 9092. If no servers are specified, will default to localhost:9092.
- **client_id** (*str*) – A name for this client. This string is passed in each request to servers and can be used to identify specific server-side log entries that correspond to this client. Also submitted to GroupCoordinator for logging with respect to consumer group administration. Default: 'kafka-python-{version}'
- **group_id** (*str or None*) – The name of the consumer group to join for dynamic partition assignment (if enabled), and to use for fetching and committing offsets. If None, auto-partition assignment (via group coordinator) and offset commits are disabled. Default: None
- **key_deserializer** (*callable*) – Any callable that takes a raw message key and returns a deserialized key.
- **value_deserializer** (*callable*) – Any callable that takes a raw message value and returns a deserialized value.
- **fetch_min_bytes** (*int*) – Minimum amount of data the server should return for a fetch request, otherwise wait up to fetch_max_wait_ms for more data to accumulate. Default: 1.
- **fetch_max_wait_ms** (*int*) – The maximum amount of time in milliseconds the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement given by fetch_min_bytes. Default: 500.
- **fetch_max_bytes** (*int*) – The maximum amount of data the server should return for a fetch request. This is not an absolute maximum, if the first message in the first non-empty partition of the fetch is larger than this value, the message will still be returned to ensure that the consumer can make progress. NOTE: consumer performs fetches to multiple brokers in parallel so memory usage will depend on the number of brokers containing partitions for the topic. Supported Kafka version >= 0.10.1.0. Default: 52428800 (50 Mb).
- **max_partition_fetch_bytes** (*int*) – The maximum amount of data per-partition the server will return. The maximum total memory used for a request = #partitions *

max_partition_fetch_bytes. This size must be at least as large as the maximum message size the server allows or else it is possible for the producer to send messages larger than the consumer can fetch. If that happens, the consumer can get stuck trying to fetch a large message on a certain partition. Default: 1048576.

- **request_timeout_ms** (*int*) – Client request timeout in milliseconds. Default: 40000.
- **retry_backoff_ms** (*int*) – Milliseconds to backoff when retrying on errors. Default: 100.
- **reconnect_backoff_ms** (*int*) – The amount of time in milliseconds to wait before attempting to reconnect to a given host. Default: 50.
- **reconnect_backoff_max_ms** (*int*) – The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. To avoid connection storms, a randomization factor of 0.2 will be applied to the backoff resulting in a random range between 20% below and 20% above the computed value. Default: 1000.
- **max_in_flight_requests_per_connection** (*int*) – Requests are pipelined to kafka brokers up to this number of maximum requests per broker connection. Default: 5.
- **auto_offset_reset** (*str*) – A policy for resetting offsets on OffsetOutOfRange errors: 'earliest' will move to the oldest available message, 'latest' will move to the most recent. Any other value will raise the exception. Default: 'latest'.
- **enable_auto_commit** (*bool*) – If True , the consumer's offset will be periodically committed in the background. Default: True.
- **auto_commit_interval_ms** (*int*) – Number of milliseconds between automatic offset commits, if enable_auto_commit is True. Default: 5000.
- **default_offset_commit_callback** (*callable*) – Called as callback(offsets, response) response will be either an Exception or an OffsetCommitResponse struct. This callback can be used to trigger custom actions when a commit request completes.
- **check_crcs** (*bool*) – Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk corruption to the messages occurred. This check adds some overhead, so it may be disabled in cases seeking extreme performance. Default: True
- **metadata_max_age_ms** (*int*) – The period of time in milliseconds after which we force a refresh of metadata, even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions. Default: 300000
- **partition_assignment_strategy** (*list*) – List of objects to use to distribute partition ownership amongst consumer instances when group management is used. Default: [RangePartitionAssignor, RoundRobinPartitionAssignor]
- **heartbeat_interval_ms** (*int*) – The expected time in milliseconds between heartbeats to the consumer coordinator when using Kafka's group management feature. Heartbeats are used to ensure that the consumer's session stays active and to facilitate rebalancing when new consumers join or leave the group. The value must be set lower than session_timeout_ms, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances. Default: 3000
- **session_timeout_ms** (*int*) – The timeout used to detect failures when using Kafka's group management facilities. Default: 30000
- **max_poll_records** (*int*) – The maximum number of records returned in a single call to `poll()`. Default: 500

- **receive_buffer_bytes** (*int*) – The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. Default: None (relies on system defaults). The java client defaults to 32768.
- **send_buffer_bytes** (*int*) – The size of the TCP send buffer (SO_SNDBUF) to use when sending data. Default: None (relies on system defaults). The java client defaults to 131072.
- **socket_options** (*list*) – List of tuple-arguments to socket.setsockopt to apply to broker connection sockets. Default: [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)]
- **consumer_timeout_ms** (*int*) – number of milliseconds to block during message iteration before raising StopIteration (i.e., ending the iterator). Default block forever [float('inf')].
- **skip_double_compressed_messages** (*bool*) – A bug in KafkaProducer <= 1.2.4 caused some messages to be corrupted via double-compression. By default, the fetcher will return these messages as a compressed blob of bytes with a single offset, i.e. how the message was actually published to the cluster. If you prefer to have the fetcher automatically detect corrupt messages and skip them, set this option to True. Default: False.
- **security_protocol** (*str*) – Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL. Default: PLAINTEXT.
- **ssl_context** (*ssl.SSLContext*) – Pre-configured SSLContext for wrapping socket connections. If provided, all other ssl_* configurations will be ignored. Default: None.
- **ssl_check_hostname** (*bool*) – Flag to configure whether ssl handshake should verify that the certificate matches the brokers hostname. Default: True.
- **ssl_cafile** (*str*) – Optional filename of ca file to use in certificate verification. Default: None.
- **ssl_certfile** (*str*) – Optional filename of file in pem format containing the client certificate, as well as any ca certificates needed to establish the certificate's authenticity. Default: None.
- **ssl_keyfile** (*str*) – Optional filename containing the client private key. Default: None.
- **ssl_password** (*str*) – Optional password to be used when loading the certificate chain. Default: None.
- **ssl_crlfile** (*str*) – Optional filename containing the CRL to check for certificate expiration. By default, no CRL check is done. When providing a file, only the leaf certificate will be checked against this CRL. The CRL can only be checked with Python 3.4+ or 2.7.9+. Default: None.
- **api_version** (*tuple*) – Specify which Kafka API version to use. If set to None, the client will attempt to infer the broker version by probing various APIs. Different versions enable different functionality.

Examples

(0, 9) enables full group coordination features with automatic partition assignment and rebalancing,

(0, 8, 2) enables kafka-storage offset commits with manual partition assignment only,

(0, 8, 1) enables zookeeper-storage offset commits with manual partition assignment only,

(0, 8, 0) enables basic functionality but requires manual partition assignment and offset management.

For the full list of supported versions, see KafkaClient.API_VERSIONS. Default: None

- **api_version_auto_timeout_ms** (*int*) – number of milliseconds to throw a timeout exception from the constructor when checking the broker api version. Only applies if api_version set to 'auto'
- **metric_reporters** (*list*) – A list of classes to use as metrics reporters. Implementing the AbstractMetricsReporter interface allows plugging in classes that will be notified of new metric creation. Default: []
- **metrics_num_samples** (*int*) – The number of samples maintained to compute metrics. Default: 2
- **metrics_sample_window_ms** (*int*) – The maximum age in milliseconds of samples used to compute metrics. Default: 30000
- **selector** (*selectors.BaseSelector*) – Provide a specific selector implementation to use for I/O multiplexing. Default: selectors.DefaultSelector
- **exclude_internal_topics** (*bool*) – Whether records from internal topics (such as offsets) should be exposed to the consumer. If set to True the only way to receive records from an internal topic is subscribing to it. Requires 0.10+ Default: True
- **sasl_mechanism** (*str*) – String picking sasl mechanism when security_protocol is SASL_PLAINTEXT or SASL_SSL. Currently only PLAIN is supported. Default: None
- **sasl_plain_username** (*str*) – Username for sasl PLAIN authentication. Default: None
- **sasl_plain_password** (*str*) – Password for sasl PLAIN authentication. Default: None

---

Note: Configuration parameters are described in more detail at https://kafka.apache.org/documentation/#newconsumerconfigs

---

assign(*partitions*)

Manually assign a list of TopicPartitions to this consumer.

Parameters **partitions** (*list of TopicPartition*) – Assignment for this instance.

Raises

- `IllegalStateError` – If consumer has already called
- `subscribe()`.

Warning: It is not possible to use both manual partition assignment with `assign()` and group assignment with `subscribe()`.

Note: This interface does not support incremental assignment and will replace the previous assignment (if there was one).

Note: Manual topic assignment through this method does not use the consumer's group management functionality. As such, there will be no rebalance operation triggered when group membership or cluster and topic metadata change.

assignment()
Get the TopicPartitions currently assigned to this consumer.

If partitions were directly assigned using `assign()`, then this will simply return the same partitions that were previously assigned. If topics were subscribed using `subscribe()`, then this will give the set of topic partitions currently assigned to the consumer (which may be None if the assignment hasn't happened yet, or if the partitions are in the process of being reassigned).

Returns {TopicPartition, ...}

Return type set

beginning_offsets(*partitions*)
Get the first offset for the given partitions.

This method does not change the current consumer position of the partitions.

Note: This method may block indefinitely if the partition does not exist.

Parameters **partitions** (*list*) – List of TopicPartition instances to fetch offsets for.

Returns int}'': The earliest available offsets for the given partitions.

Return type ''{TopicPartition

Raises

- `UnsupportedVersionError` – If the broker does not support looking up the offsets by timestamp.
- `KafkaTimeoutError` – If fetch failed in request_timeout_ms.

close(*autocommit=True*)
Close the consumer, waiting indefinitely for any needed cleanup.

Keyword Arguments **autocommit** (*bool*) – If auto-commit is configured for this consumer, this optional flag causes the consumer to attempt to commit any pending consumed offsets prior to close. Default: True

commit(*offsets=None*)
Commit offsets to kafka, blocking until success or error.

This commits offsets only to Kafka. The offsets committed using this API will be used on the first fetch after every rebalance and also on startup. As such, if you need to store offsets in anything other than Kafka, this API should not be used. To avoid re-processing the last message read if a consumer is restarted, the committed offset should be the next message your application should consume, i.e.: last_offset + 1.

Blocks until either the commit succeeds or an unrecoverable error is encountered (in which case it is thrown to the caller).

Currently only supports kafka-topic offset storage (not zookeeper).

Parameters **offsets** (*dict, optional*) – {TopicPartition: OffsetAndMetadata} dict to commit with the configured group_id. Defaults to currently consumed offsets for all subscribed partitions.

commit_async(*offsets=None*, *callback=None*)
Commit offsets to kafka asynchronously, optionally firing callback.

This commits offsets only to Kafka. The offsets committed using this API will be used on the first fetch after every rebalance and also on startup. As such, if you need to store offsets in anything other than Kafka, this API should not be used. To avoid re-processing the last message read if a consumer is restarted, the committed offset should be the next message your application should consume, i.e.: last_offset + 1.

This is an asynchronous call and will not block. Any errors encountered are either passed to the callback (if provided) or discarded.

Parameters

- **offsets** (*dict, optional*) – {TopicPartition: OffsetAndMetadata} dict to commit with the configured group_id. Defaults to currently consumed offsets for all subscribed partitions.
- **callback** (*callable, optional*) – Called as callback(offsets, response) with response as either an Exception or an OffsetCommitResponse struct. This callback can be used to trigger custom actions when a commit request completes.

Returns kafka.future.Future

committed(*partition*)
Get the last committed offset for the given partition.

This offset will be used as the position for the consumer in the event of a failure.

This call may block to do a remote call if the partition in question isn't assigned to this consumer or if the consumer hasn't yet initialized its cache of committed offsets.

Parameters **partition** (*TopicPartition*) – The partition to check.

Returns The last committed offset, or None if there was no prior commit.

end_offsets(*partitions*)
Get the last offset for the given partitions. The last offset of a partition is the offset of the upcoming message, i.e. the offset of the last available message + 1.

This method does not change the current consumer position of the partitions.

Note: This method may block indefinitely if the partition does not exist.

Parameters **partitions** (*list*) – List of TopicPartition instances to fetch offsets for.

Returns int}‘‘: The end offsets for the given partitions.

Return type ‘‘{TopicPartition

Raises

- UnsupportedVersionError – If the broker does not support looking up the offsets by timestamp.
- KafkaTimeoutError – If fetch failed in request_timeout_ms

highwater(*partition*)
Last known highwater offset for a partition.

A highwater offset is the offset that will be assigned to the next message that is produced. It may be useful for calculating lag, by comparing with the reported position. Note that both position and highwater refer to the *next* offset – i.e., highwater offset is one greater than the newest available message.

Highwater offsets are returned in FetchResponse messages, so will not be available if no FetchRequests have been sent for this partition yet.

Parameters **partition** (*TopicPartition*) – Partition to check

Returns Offset if available

Return type int or None

metrics(*raw=False*)

Warning: this is an unstable interface. It may change in future releases without warning

offsets_for_times(*timestamps*)

Look up the offsets for the given partitions by timestamp. The returned offset for each partition is the earliest offset whose timestamp is greater than or equal to the given timestamp in the corresponding partition.

This is a blocking call. The consumer does not have to be assigned the partitions.

If the message format version in a partition is before 0.10.0, i.e. the messages do not have timestamps, `None` will be returned for that partition. `None` will also be returned for the partition if there are no messages in it.

Note: This method may block indefinitely if the partition does not exist.

Parameters **timestamps** (*dict*) – `{TopicPartition: int}` mapping from partition to the timestamp to look up. Unit should be milliseconds since beginning of the epoch (midnight Jan 1, 1970 (UTC))

Returns OffsetAndTimestamp}‘‘: mapping from partition to the timestamp and offset of the first message with timestamp greater than or equal to the target timestamp.

Return type ‘‘{TopicPartition

Raises

- `ValueError` – If the target timestamp is negative
- `UnsupportedVersionError` – If the broker does not support looking up the offsets by timestamp.
- `KafkaTimeoutError` – If fetch failed in request_timeout_ms

partitions_for_topic(*topic*)

Get metadata about the partitions for a given topic.

Parameters **topic** (*str*) – Topic to check.

Returns Partition ids

Return type set

pause(**partitions*)

Suspend fetching from the requested partitions.

Future calls to `poll()` will not return any records from these partitions until they have been resumed using `resume()`.

Note: This method does not affect partition subscription. In particular, it does not cause a group rebalance when automatic assignment is used.

Parameters ***partitions** (*TopicPartition*) – Partitions to pause.

paused()
Get the partitions that were previously paused using `pause()`.

Returns {partition (TopicPartition), ...}

Return type set

poll(*timeout_ms=0*, *max_records=None*)
Fetch data from assigned topics / partitions.

Records are fetched and returned in batches by topic-partition. On each poll, consumer will try to use the last consumed offset as the starting offset and fetch sequentially. The last consumed offset can be manually set through `seek()` or automatically set as the last committed offset for the subscribed list of partitions.

Incompatible with iterator interface – use one or the other, not both.

Parameters

- **timeout_ms** (*int, optional*) – Milliseconds spent waiting in poll if data is not available in the buffer. If 0, returns immediately with any records that are available currently in the buffer, else returns empty. Must not be negative. Default: 0
- **max_records** (*int, optional*) – The maximum number of records returned in a single call to `poll()`. Default: Inherit value from max_poll_records.

Returns

Topic to list of records since the last fetch for the subscribed list of topics and partitions.

Return type dict

position(*partition*)
Get the offset of the next record that will be fetched

Parameters **partition** (*TopicPartition*) – Partition to check

Returns Offset

Return type int

resume(**partitions*)
Resume fetching from the specified (paused) partitions.

Parameters ***partitions** (*TopicPartition*) – Partitions to resume.

seek(*partition*, *offset*)
Manually specify the fetch offset for a TopicPartition.

Overrides the fetch offsets that the consumer will use on the next `poll()`. If this API is invoked for the same partition more than once, the latest offset will be used on the next `poll()`.

Note: You may lose data if this API is arbitrarily used in the middle of consumption to reset the fetch offsets.

Parameters

- **partition** (*TopicPartition*) – Partition for seek operation
- **offset** (*int*) – Message offset in partition

Raises `AssertionError` – If offset is not an int >= 0; or if partition is not currently assigned.

seek_to_beginning(**partitions*)
Seek to the oldest available offset for partitions.

Parameters ***partitions** – Optionally provide specific TopicPartitions, otherwise default to all assigned partitions.

Raises `AssertionError` – If any partition is not currently assigned, or if no partitions are assigned.

seek_to_end(**partitions*)
Seek to the most recent available offset for partitions.

Parameters ***partitions** – Optionally provide specific TopicPartitions, otherwise default to all assigned partitions.

Raises `AssertionError` – If any partition is not currently assigned, or if no partitions are assigned.

subscribe(*topics=()*, *pattern=None*, *listener=None*)
Subscribe to a list of topics, or a topic regex pattern.

Partitions will be dynamically assigned via a group coordinator. Topic subscriptions are not incremental: this list will replace the current assignment (if there is one).

This method is incompatible with `assign()`.

Parameters

- **topics** (*list*) – List of topics for subscription.
- **pattern** (*str*) – Pattern to match available topics. You must provide either topics or pattern, but not both.
- **listener** (*ConsumerRebalanceListener*) – Optionally include listener callback, which will be called before and after each rebalance operation.

  As part of group management, the consumer will keep track of the list of consumers that belong to a particular group and will trigger a rebalance operation if one of the following events trigger:

  – Number of partitions change for any of the subscribed topics

  – Topic is created or deleted

  – An existing member of the consumer group dies

  – A new member is added to the consumer group

  When any of these events are triggered, the provided listener will be invoked first to indicate that the consumer's assignment has been revoked, and then again when the new assignment has been received. Note that this listener will immediately override any listener set in a previous call to subscribe. It is guaranteed, however, that the partitions revoked/assigned through this interface are from topics subscribed in this call.

Raises

- `IllegalStateError` – If called after previously calling `assign()`.
- `AssertionError` – If neither topics or pattern is provided.
- `TypeError` – If listener is not a ConsumerRebalanceListener.

subscription()
Get the current topic subscription.

Returns {topic, ...}

Return type set

topics()
Get all topics the user is authorized to view.

Returns topics

Return type set

unsubscribe()
Unsubscribe from all topics and clear all assigned partitions.

KafkaProducer

class kafka.**KafkaProducer**(***configs*)
A Kafka client that publishes records to the Kafka cluster.

The producer is thread safe and sharing a single producer instance across threads will generally be faster than having multiple instances.

The producer consists of a pool of buffer space that holds records that haven't yet been transmitted to the server as well as a background I/O thread that is responsible for turning these records into requests and transmitting them to the cluster.

send() is asynchronous. When called it adds the record to a buffer of pending record sends and immediately returns. This allows the producer to batch together individual records for efficiency.

The 'acks' config controls the criteria under which requests are considered complete. The "all" setting will result in blocking on the full commit of the record, the slowest but most durable setting.

If the request fails, the producer can automatically retry, unless 'retries' is configured to 0. Enabling retries also opens up the possibility of duplicates (see the documentation on message delivery semantics for details: http://kafka.apache.org/documentation.html#semantics).

The producer maintains buffers of unsent records for each partition. These buffers are of a size specified by the 'batch_size' config. Making this larger can result in more batching, but requires more memory (since we will generally have one of these buffers for each active partition).

By default a buffer is available to send immediately even if there is additional unused space in the buffer. However if you want to reduce the number of requests you can set 'linger_ms' to something greater than 0. This will instruct the producer to wait up to that number of milliseconds before sending a request in hope that more records will arrive to fill up the same batch. This is analogous to Nagle's algorithm in TCP. Note that records that arrive close together in time will generally batch together even with linger_ms=0 so under heavy load batching will occur regardless of the linger configuration; however setting this to something larger than 0 can lead to fewer, more efficient requests when not under maximal load at the cost of a small amount of latency.

The buffer_memory controls the total amount of memory available to the producer for buffering. If records are sent faster than they can be transmitted to the server then this buffer space will be exhausted. When the buffer space is exhausted additional send calls will block.

The key_serializer and value_serializer instruct how to turn the key and value objects the user provides into bytes.

Keyword Arguments

- **bootstrap_servers** – 'host[:port]' string (or list of 'host[:port]' strings) that the producer should contact to bootstrap initial cluster metadata. This does not have to be the full node list. It just needs to have at least one broker that will respond to a Metadata API Request. Default port is 9092. If no servers are specified, will default to localhost:9092.
- **client_id** (*str*) – a name for this client. This string is passed in each request to servers and can be used to identify specific server-side log entries that correspond to this client. Default: 'kafka-python-producer-#' (appended with a unique number per instance)
- **key_serializer** (*callable*) – used to convert user-supplied keys to bytes If not None, called as f(key), should return bytes. Default: None.

- **value_serializer** (*callable*) – used to convert user-supplied message values to bytes. If not None, called as f(value), should return bytes. Default: None.
- **acks** (*0, 1, 'all'*) – The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are common:

  0: Producer will not wait for any acknowledgment from the server. The message will immediately be added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1.

  1: Wait for leader to write the record to its local log only. Broker will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost.

  all: Wait for the full set of in-sync replicas to write the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.

  If unset, defaults to acks=1.
- **compression_type** (*str*) – The compression type for all data generated by the producer. Valid values are 'gzip', 'snappy', 'lz4', or None. Compression is of full batches of data, so the efficacy of batching will also impact the compression ratio (more batching means better compression). Default: None.
- **retries** (*int*) – Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries without setting max_in_flight_requests_per_connection to 1 will potentially change the ordering of records because if two batches are sent to a single partition, and the first fails and is retried but the second succeeds, then the records in the second batch may appear first. Default: 0.
- **batch_size** (*int*) – Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent. A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). Default: 16384
- **linger_ms** (*int*) – The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs only under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay; that is, rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get batch_size worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up. This setting defaults to 0 (i.e. no delay). Setting linger_ms=5 would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absense of load. Default: 0.
- **partitioner** (*callable*) – Callable used to determine which partition each message is assigned to. Called (after key serialization): partitioner(key_bytes, all_partitions, available_partitions). The default partitioner implementation hashes each non-None key using

the same murmur2 algorithm as the java client so that messages with the same key are assigned to the same partition. When a key is None, the message is delivered to a random partition (filtered to partitions with available leaders only, if possible).

- **buffer_memory** (*int*) – The total bytes of memory the producer should use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the server the producer will block up to max_block_ms, raising an exception on timeout. In the current implementation, this setting is an approximation. Default: 33554432 (32MB)
- **max_block_ms** (*int*) – Number of milliseconds to block during `send()` and `partitions_for()`. These methods can be blocked either because the buffer is full or metadata unavailable. Blocking in the user-supplied serializers or partitioner will not be counted against this timeout. Default: 60000.
- **max_request_size** (*int*) – The maximum size of a request. This is also effectively a cap on the maximum record size. Note that the server has its own cap on record size which may be different from this. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests. Default: 1048576.
- **metadata_max_age_ms** (*int*) – The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions. Default: 300000
- **retry_backoff_ms** (*int*) – Milliseconds to backoff when retrying on errors. Default: 100.
- **request_timeout_ms** (*int*) – Client request timeout in milliseconds. Default: 30000.
- **receive_buffer_bytes** (*int*) – The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. Default: None (relies on system defaults). Java client defaults to 32768.
- **send_buffer_bytes** (*int*) – The size of the TCP send buffer (SO_SNDBUF) to use when sending data. Default: None (relies on system defaults). Java client defaults to 131072.
- **socket_options** (*list*) – List of tuple-arguments to socket.setsockopt to apply to broker connection sockets. Default: [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)]
- **reconnect_backoff_ms** (*int*) – The amount of time in milliseconds to wait before attempting to reconnect to a given host. Default: 50.
- **reconnect_backoff_max_ms** (*int*) – The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. To avoid connection storms, a randomization factor of 0.2 will be applied to the backoff resulting in a random range between 20% below and 20% above the computed value. Default: 1000.
- **max_in_flight_requests_per_connection** (*int*) – Requests are pipelined to kafka brokers up to this number of maximum requests per broker connection. Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e., if retries are enabled). Default: 5.
- **security_protocol** (*str*) – Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAINTEXT, SASL_SSL. Default: PLAINTEXT.
- **ssl_context** (*ssl.SSLContext*) – pre-configured SSLContext for wrapping socket connections. If provided, all other ssl_* configurations will be ignored. Default: None.

- **ssl_check_hostname** (*bool*) – flag to configure whether ssl handshake should verify that the certificate matches the brokers hostname. default: true.
- **ssl_cafile** (*str*) – optional filename of ca file to use in certificate veriication. default: none.
- **ssl_certfile** (*str*) – optional filename of file in pem format containing the client certificate, as well as any ca certificates needed to establish the certificate's authenticity. default: none.
- **ssl_keyfile** (*str*) – optional filename containing the client private key. default: none.
- **ssl_password** (*str*) – optional password to be used when loading the certificate chain. default: none.
- **ssl_crlfile** (*str*) – optional filename containing the CRL to check for certificate expiration. By default, no CRL check is done. When providing a file, only the leaf certificate will be checked against this CRL. The CRL can only be checked with Python 3.4+ or 2.7.9+. default: none.
- **api_version** (*tuple*) – Specify which Kafka API version to use. If set to None, the client will attempt to infer the broker version by probing various APIs. For a full list of supported versions, see KafkaClient.API_VERSIONS. Default: None
- **api_version_auto_timeout_ms** (*int*) – number of milliseconds to throw a timeout exception from the constructor when checking the broker api version. Only applies if api_version set to 'auto'
- **metric_reporters** (*list*) – A list of classes to use as metrics reporters. Implementing the AbstractMetricsReporter interface allows plugging in classes that will be notified of new metric creation. Default: []
- **metrics_num_samples** (*int*) – The number of samples maintained to compute metrics. Default: 2
- **metrics_sample_window_ms** (*int*) – The maximum age in milliseconds of samples used to compute metrics. Default: 30000
- **selector** (*selectors.BaseSelector*) – Provide a specific selector implementation to use for I/O multiplexing. Default: selectors.DefaultSelector
- **sasl_mechanism** (*str*) – string picking sasl mechanism when security_protocol is SASL_PLAINTEXT or SASL_SSL. Currently only PLAIN is supported. Default: None
- **sasl_plain_username** (*str*) – username for sasl PLAIN authentication. Default: None
- **sasl_plain_password** (*str*) – password for sasl PLAIN authentication. Default: None

---

Note: Configuration parameters are described in more detail at https://kafka.apache.org/0100/configuration.html#producerconfigs

---

close(*timeout=None*)

Close this producer.

Parameters **timeout** (*float, optional*) – timeout in seconds to wait for completion.

flush(*timeout=None*)

Invoking this method makes all buffered records immediately available to send (even if linger_ms is greater than 0) and blocks on the completion of the requests associated with these records. The post-condition

of `flush()` is that any previously sent record will have completed (e.g. Future.is_done() == True). A request is considered completed when either it is successfully acknowledged according to the 'acks' configuration for the producer, or it results in an error.

Other threads can continue sending messages while one thread is blocked waiting for a flush call to complete; however, no guarantee is made about the completion of messages sent after the flush call begins.

Parameters **timeout** (*float, optional*) – timeout in seconds to wait for completion.

Raises `KafkaTimeoutError` – failure to flush buffered records within the provided timeout

metrics(*raw=False*)
Warning: this is an unstable interface. It may change in future releases without warning

partitions_for(*topic*)
Returns set of all known partitions for the topic.

send(*topic*, *value=None*, *key=None*, *partition=None*, *timestamp_ms=None*)
Publish a message to a topic.

Parameters

- **topic** (*str*) – topic where the message will be published
- **value** (*optional*) – message value. Must be type bytes, or be serializable to bytes via configured value_serializer. If value is None, key is required and message acts as a 'delete'. See kafka compaction documentation for more details: http://kafka.apache.org/documentation.html#compaction (compaction requires kafka >= 0.8.1)
- **partition** (*int, optional*) – optionally specify a partition. If not set, the partition will be selected using the configured 'partitioner'.
- **key** (*optional*) – a key to associate with the message. Can be used to determine which partition to send the message to. If partition is None (and producer's partitioner config is left as default), then messages with the same key will be delivered to the same partition (but if key is None, partition is chosen randomly). Must be type bytes, or be serializable to bytes via configured key_serializer.
- **timestamp_ms** (*int, optional*) – epoch milliseconds (from Jan 1 1970 UTC) to use as the message timestamp. Defaults to current time.

Returns resolves to RecordMetadata

Return type FutureRecordMetadata

Raises `KafkaTimeoutError` – if unable to fetch topic metadata, or unable to obtain memory buffer prior to configured max_block_ms

KafkaClient

class `kafka.client.`**KafkaClient**(***configs*)
A network client for asynchronous request/response network I/O.

This is an internal class used to implement the user-facing producer and consumer clients.

This class is not thread-safe!

cluster
`ClusterMetadata` – Local cache of cluster metadata, retrieved via MetadataRequests during `poll()`.

Keyword Arguments

- **bootstrap_servers** – 'host[:port]' string (or list of 'host[:port]' strings) that the consumer should contact to bootstrap initial cluster metadata. This does not have to be the full node list. It just needs to have at least one broker that will respond to a Metadata API Request. Default port is 9092. If no servers are specified, will default to localhost:9092.
- **client_id** (*str*) – a name for this client. This string is passed in each request to servers and can be used to identify specific server-side log entries that correspond to this client. Also submitted to GroupCoordinator for logging with respect to consumer group administration. Default: 'kafka-python-{version}'
- **reconnect_backoff_ms** (*int*) – The amount of time in milliseconds to wait before attempting to reconnect to a given host. Default: 50.
- **reconnect_backoff_max_ms** (*int*) – The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. To avoid connection storms, a randomization factor of 0.2 will be applied to the backoff resulting in a random range between 20% below and 20% above the computed value. Default: 1000.
- **request_timeout_ms** (*int*) – Client request timeout in milliseconds. Default: 40000.
- **retry_backoff_ms** (*int*) – Milliseconds to backoff when retrying on errors. Default: 100.
- **max_in_flight_requests_per_connection** (*int*) – Requests are pipelined to kafka brokers up to this number of maximum requests per broker connection. Default: 5.
- **receive_buffer_bytes** (*int*) – The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. Default: None (relies on system defaults). Java client defaults to 32768.
- **send_buffer_bytes** (*int*) – The size of the TCP send buffer (SO_SNDBUF) to use when sending data. Default: None (relies on system defaults). Java client defaults to 131072.
- **socket_options** (*list*) – List of tuple-arguments to socket.setsockopt to apply to broker connection sockets. Default: [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)]
- **metadata_max_age_ms** (*int*) – The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions. Default: 300000
- **security_protocol** (*str*) – Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL. Default: PLAINTEXT.
- **ssl_context** (*ssl.SSLContext*) – pre-configured SSLContext for wrapping socket connections. If provided, all other ssl_* configurations will be ignored. Default: None.
- **ssl_check_hostname** (*bool*) – flag to configure whether ssl handshake should verify that the certificate matches the brokers hostname. default: true.
- **ssl_cafile** (*str*) – optional filename of ca file to use in certificate veriication. default: none.
- **ssl_certfile** (*str*) – optional filename of file in pem format containing the client certificate, as well as any ca certificates needed to establish the certificate's authenticity. default: none.
- **ssl_keyfile** (*str*) – optional filename containing the client private key. default: none.

- **ssl_password** (*str*) – optional password to be used when loading the certificate chain. default: none.
- **ssl_crlfile** (*str*) – optional filename containing the CRL to check for certificate expiration. By default, no CRL check is done. When providing a file, only the leaf certificate will be checked against this CRL. The CRL can only be checked with Python 3.4+ or 2.7.9+. default: none.
- **api_version** (*tuple*) – Specify which Kafka API version to use. If set to None, KafkaClient will attempt to infer the broker version by probing various APIs. For the full list of supported versions, see KafkaClient.API_VERSIONS. Default: None
- **api_version_auto_timeout_ms** (*int*) – number of milliseconds to throw a timeout exception from the constructor when checking the broker api version. Only applies if api_version is None
- **selector** (*selectors.BaseSelector*) – Provide a specific selector implementation to use for I/O multiplexing. Default: selectors.DefaultSelector
- **metrics** (*kafka.metrics.Metrics*) – Optionally provide a metrics instance for capturing network IO stats. Default: None.
- **metric_group_prefix** (*str*) – Prefix for metric names. Default: ''
- **sasl_mechanism** (*str*) – string picking sasl mechanism when security_protocol is SASL_PLAINTEXT or SASL_SSL. Currently only PLAIN is supported. Default: None
- **sasl_plain_username** (*str*) – username for sasl PLAIN authentication. Default: None
- **sasl_plain_password** (*str*) – password for sasl PLAIN authentication. Default: None

add_topic(*topic*)

Add a topic to the list of topics tracked via metadata.

Parameters **topic** (*str*) – topic to track

Returns resolves after metadata request/response

Return type Future

check_version(*node_id=None*, *timeout=2*, *strict=False*)

Attempt to guess the version of a Kafka broker.

Note: It is possible that this method blocks longer than the specified timeout. This can happen if the entire cluster is down and the client enters a bootstrap backoff sleep. This is only possible if node_id is None.

Returns: version tuple, i.e. (0, 10), (0, 9), (0, 8, 2), ...

Raises

- NodeNotReadyError (if node_id is provided)
- NoBrokersAvailable (if node_id is None)
- UnrecognizedBrokerVersion – please file bug if seen!
- *AssertionError (if strict=True)* – please file bug if seen!

close(*node_id=None*)

Close one or all broker connections.

Parameters **node_id** (*int, optional*) – the id of the node to close

connected(*node_id*)
Return True iff the node_id is connected.

connection_delay(*node_id*)
Return the number of milliseconds to wait, based on the connection state, before attempting to send data. When disconnected, this respects the reconnect backoff time. When connecting, returns 0 to allow non-blocking connect to finish. When connected, returns a very large number to handle slow/stalled connections.

Parameters **node_id** (*int*) – The id of the node to check

Returns The number of milliseconds to wait.

Return type int

in_flight_request_count(*node_id=None*)
Get the number of in-flight requests for a node or all nodes.

Parameters **node_id** (*int, optional*) – a specific node to check. If unspecified, return the total for all nodes

Returns pending in-flight requests for the node, or all nodes if None

Return type int

is_disconnected(*node_id*)
Check whether the node connection has been disconnected or failed.

A disconnected node has either been closed or has failed. Connection failures are usually transient and can be resumed in the next ready() call, but there are cases where transient failures need to be caught and re-acted upon.

Parameters **node_id** (*int*) – the id of the node to check

Returns True iff the node exists and is disconnected

Return type bool

is_ready(*node_id*, *metadata_priority=True*)
Check whether a node is ready to send more requests.

In addition to connection-level checks, this method also is used to block additional requests from being sent during a metadata refresh.

Parameters

- **node_id** (*int*) – id of the node to check
- **metadata_priority** (*bool*) – Mark node as not-ready if a metadata refresh is required. Default: True

Returns True if the node is ready and metadata is not refreshing

Return type bool

least_loaded_node()
Choose the node with fewest outstanding requests, with fallbacks.

This method will prefer a node with an existing connection and no in-flight-requests. If no such node is found, a node will be chosen randomly from disconnected nodes that are not "blacked out" (i.e., are not subject to a reconnect backoff).

Returns node_id or None if no suitable node was found

poll(*timeout_ms=None*, *future=None*, *sleep=True*, *delayed_tasks=True*)
Try to read and write to sockets.

This method will also attempt to complete node connections, refresh stale metadata, and run previously-scheduled tasks.

Parameters

- **timeout_ms** (*int, optional*) – maximum amount of time to wait (in ms) for at least one response. Must be non-negative. The actual timeout will be the minimum of timeout, request timeout and metadata timeout. Default: request_timeout_ms
- **future** (*Future, optional*) – if provided, blocks until future.is_done
- **sleep** (*bool*) – if True and there is nothing to do (no connections or requests in flight), will sleep for duration timeout before returning empty results. Default: False.

Returns responses received (can be empty)

Return type list

ready(*node_id*, *metadata_priority=True*)
Check whether a node is connected and ok to send more requests.

Parameters

- **node_id** (*int*) – the id of the node to check
- **metadata_priority** (*bool*) – Mark node as not-ready if a metadata refresh is required. Default: True

Returns True if we are ready to send to the given node

Return type bool

schedule(*task*, *at*)
Schedule a new task to be executed at the given time.

This is "best-effort" scheduling and should only be used for coarse synchronization. A task cannot be scheduled for multiple times simultaneously; any previously scheduled instance of the same task will be cancelled.

Parameters

- **task** (*callable*) – task to be scheduled
- **at** (*float or int*) – epoch seconds when task should run

Returns resolves to result of task call, or exception if raised

Return type Future

send(*node_id*, *request*)
Send a request to a specific node.

Parameters

- **node_id** (*int*) – destination node
- **request** (*Struct*) – request object (not-encoded)

Raises AssertionError – if node_id is not in current cluster metadata

Returns resolves to Response struct or Error

Return type Future

set_topics(*topics*)
Set specific topics to track for metadata.

Parameters **topics** (*list of str*) – topics to check for metadata

Returns resolves after metadata request/response

Return type Future

unschedule(*task*)
Unschedule a task.

This will remove all instances of the task from the task queue. This is a no-op if the task is not scheduled.

Parameters **task** (*callable*) – task to be unscheduled

BrokerConnection

class kafka.**BrokerConnection**(*host*, *port*, *afi*, ***configs*)
Initialize a Kafka broker connection

Keyword Arguments

- **client_id** (*str*) – a name for this client. This string is passed in each request to servers and can be used to identify specific server-side log entries that correspond to this client. Also submitted to GroupCoordinator for logging with respect to consumer group administration. Default: 'kafka-python-{version}'
- **reconnect_backoff_ms** (*int*) – The amount of time in milliseconds to wait before attempting to reconnect to a given host. Default: 50.
- **reconnect_backoff_max_ms** (*int*) – The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. To avoid connection storms, a randomization factor of 0.2 will be applied to the backoff resulting in a random range between 20% below and 20% above the computed value. Default: 1000.
- **request_timeout_ms** (*int*) – Client request timeout in milliseconds. Default: 40000.
- **max_in_flight_requests_per_connection** (*int*) – Requests are pipelined to kafka brokers up to this number of maximum requests per broker connection. Default: 5.
- **receive_buffer_bytes** (*int*) – The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. Default: None (relies on system defaults). Java client defaults to 32768.
- **send_buffer_bytes** (*int*) – The size of the TCP send buffer (SO_SNDBUF) to use when sending data. Default: None (relies on system defaults). Java client defaults to 131072.
- **socket_options** (*list*) – List of tuple-arguments to socket.setsockopt to apply to broker connection sockets. Default: [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)]
- **security_protocol** (*str*) – Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL. Default: PLAINTEXT.
- **ssl_context** (*ssl.SSLContext*) – pre-configured SSLContext for wrapping socket connections. If provided, all other ssl_* configurations will be ignored. Default: None.
- **ssl_check_hostname** (*bool*) – flag to configure whether ssl handshake should verify that the certificate matches the brokers hostname. default: True.

- **ssl_cafile** (*str*) – optional filename of ca file to use in certificate veriication. default: None.
- **ssl_certfile** (*str*) – optional filename of file in pem format containing the client certificate, as well as any ca certificates needed to establish the certificate's authenticity. default: None.
- **ssl_keyfile** (*str*) – optional filename containing the client private key. default: None.
- **ssl_password** (*callable, str, bytes, bytearray*) – optional password or callable function that returns a password, for decrypting the client private key. Default: None.
- **ssl_crlfile** (*str*) – optional filename containing the CRL to check for certificate expiration. By default, no CRL check is done. When providing a file, only the leaf certificate will be checked against this CRL. The CRL can only be checked with Python 3.4+ or 2.7.9+. default: None.
- **api_version** (*tuple*) – Specify which Kafka API version to use. Accepted values are: (0, 8, 0), (0, 8, 1), (0, 8, 2), (0, 9), (0, 10). Default: (0, 8, 2)
- **api_version_auto_timeout_ms** (*int*) – number of milliseconds to throw a timeout exception from the constructor when checking the broker api version. Only applies if api_version is None
- **state_change_callback** (*callable*) – function to be called when the connection state changes from CONNECTING to CONNECTED etc.
- **metrics** (*kafka.metrics.Metrics*) – Optionally provide a metrics instance for capturing network IO stats. Default: None.
- **metric_group_prefix** (*str*) – Prefix for metric names. Default: ''
- **sasl_mechanism** (*str*) – string picking sasl mechanism when security_protocol is SASL_PLAINTEXT or SASL_SSL. Currently only PLAIN is supported. Default: None
- **sasl_plain_username** (*str*) – username for sasl PLAIN authentication. Default: None
- **sasl_plain_password** (*str*) – password for sasl PLAIN authentication. Default: None

blacked_out()
Return true if we are disconnected from the given node and can't re-establish a connection yet

can_send_more()
Return True unless there are max_in_flight_requests_per_connection.

check_version(*timeout=2*, *strict=False*)
Attempt to guess the broker version.

Note: This is a blocking call.

Returns: version tuple, i.e. (0, 10), (0, 9), (0, 8, 2), ...

close(*error=None*)
Close socket and fail all in-flight-requests.

Parameters **error** (*Exception, optional*) – pending in-flight-requests will be failed with this exception. Default: kafka.errors.ConnectionError.

connect()
Attempt to connect and return ConnectionState

connected()
Return True iff socket is connected.

connecting()
Returns True if still connecting (this may encompass several different states, such as SSL handshake, authorization, etc).

disconnected()
Return True iff socket is closed

recv()
Non-blocking network receive.

Return response if available

send(*request*)
send request, return Future()

Can block on network if request is larger than send_buffer_bytes

ClusterMetadata

class kafka.cluster.**ClusterMetadata**(***configs*)
A class to manage kafka cluster metadata.

This class does not perform any IO. It simply updates internal state given API responses (MetadataResponse, GroupCoordinatorResponse).

Keyword Arguments

- **retry_backoff_ms** (*int*) – Milliseconds to backoff when retrying on errors. Default: 100.
- **metadata_max_age_ms** (*int*) – The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions. Default: 300000

add_group_coordinator(*group*, *response*)
Update with metadata for a group coordinator

Parameters

- **group** (*str*) – name of group from GroupCoordinatorRequest
- **response** (*GroupCoordinatorResponse*) – broker response

Returns True if metadata is updated, False on error

Return type bool

add_listener(*listener*)
Add a callback function to be called on each metadata update

available_partitions_for_topic(*topic*)
Return set of partitions with known leaders

Parameters **topic** (*str*) – topic to check for partitions

Returns {partition (int), ...}

Return type set

broker_metadata(*broker_id*)
Get BrokerMetadata

Parameters **broker_id** (*int*) – node_id for a broker to check

Returns BrokerMetadata or None if not found

brokers()
Get all BrokerMetadata

Returns {BrokerMetadata, ...}

Return type set

coordinator_for_group(*group*)
Return node_id of group coordinator.

Parameters **group** (*str*) – name of consumer group

Returns node_id for group coordinator

Return type int

failed_update(*exception*)
Update cluster state given a failed MetadataRequest.

leader_for_partition(*partition*)
Return node_id of leader, -1 unavailable, None if unknown.

partitions_for_broker(*broker_id*)
Return TopicPartitions for which the broker is a leader.

Parameters **broker_id** (*int*) – node id for a broker

Returns {TopicPartition, ...}

Return type set

partitions_for_topic(*topic*)
Return set of all partitions for topic (whether available or not)

Parameters **topic** (*str*) – topic to check for partitions

Returns {partition (int), ...}

Return type set

refresh_backoff()
Return milliseconds to wait before attempting to retry after failure

remove_listener(*listener*)
Remove a previously added listener callback

request_update()
Flags metadata for update, return Future()

Actual update must be handled separately. This method will only change the reported ttl()

Returns kafka.future.Future (value will be the cluster object after update)

topics(*exclude_internal_topics=True*)
Get set of known topics.

Parameters **exclude_internal_topics** (*bool*) – Whether records from internal topics (such as offsets) should be exposed to the consumer. If set to True the only way to receive records from an internal topic is subscribing to it. Default True

Returns {topic (str), ...}

Return type set

ttl()
Milliseconds until metadata should be refreshed

update_metadata(*metadata*)
Update cluster state given a MetadataResponse.

Parameters **metadata** (*MetadataResponse*) – broker response to a metadata request

Returns: None

with_partitions(*partitions_to_add*)
Returns a copy of cluster metadata with partitions added

Simple APIs (DEPRECATED)

SimpleConsumer (DEPRECATED)

```
from kafka import SimpleProducer, SimpleClient

# To consume messages
client = SimpleClient('localhost:9092')
consumer = SimpleConsumer(client, "my-group", "my-topic")
for message in consumer:
    # message is raw byte string -- decode if necessary!
    # e.g., for unicode: `message.decode('utf-8')`
    print(message)


# Use multiprocessing for parallel consumers
from kafka import MultiProcessConsumer

# This will split the number of partitions among two processes
consumer = MultiProcessConsumer(client, "my-group", "my-topic", num_procs=2)

# This will spawn processes such that each handles 2 partitions max
consumer = MultiProcessConsumer(client, "my-group", "my-topic",
                                partitions_per_proc=2)

for message in consumer:
    print(message)

for message in consumer.get_messages(count=5, block=True, timeout=4):
    print(message)

client.close()
```

SimpleProducer (DEPRECATED)

Asynchronous Mode

```
from kafka import SimpleProducer, SimpleClient

# To send messages asynchronously
client = SimpleClient('localhost:9092')
```

```
producer = SimpleProducer(client, async=True)
producer.send_messages('my-topic', b'async message')

# To send messages in batch. You can use any of the available
# producers for doing this. The following producer will collect
# messages in batch and send them to Kafka after 20 messages are
# collected or every 60 seconds
# Notes:
# * If the producer dies before the messages are sent, there will be losses
# * Call producer.stop() to send the messages and cleanup
producer = SimpleProducer(client,
                          async=True,
                          batch_send_every_n=20,
                          batch_send_every_t=60)
```

Synchronous Mode

```
from kafka import SimpleProducer, SimpleClient

# To send messages synchronously
client = SimpleClient('localhost:9092')
producer = SimpleProducer(client, async=False)

# Note that the application is responsible for encoding messages to type bytes
producer.send_messages('my-topic', b'some message')
producer.send_messages('my-topic', b'this method', b'is variadic')

# Send unicode message
producer.send_messages('my-topic', u'?'.encode('utf-8'))

# To wait for acknowledgements
# ACK_AFTER_LOCAL_WRITE : server will wait till the data is written to
#                         a local log before sending response
# ACK_AFTER_CLUSTER_COMMIT : server will block until the message is committed
#                            by all in sync replicas before sending a response
producer = SimpleProducer(client,
                          async=False,
                          req_acks=SimpleProducer.ACK_AFTER_LOCAL_WRITE,
                          ack_timeout=2000,
                          sync_fail_on_error=False)

responses = producer.send_messages('my-topic', b'another message')
for r in responses:
    logging.info(r.offset)
```

KeyedProducer (DEPRECATED)

```
from kafka import (
    SimpleClient, KeyedProducer,
    Murmur2Partitioner, RoundRobinPartitioner)

kafka = SimpleClient('localhost:9092')

# HashedPartitioner is default (currently uses python hash())
```

```
producer = KeyedProducer(kafka)
producer.send_messages(b'my-topic', b'key1', b'some message')
producer.send_messages(b'my-topic', b'key2', b'this methode')

# Murmur2Partitioner attempts to mirror the java client hashing
producer = KeyedProducer(kafka, partitioner=Murmur2Partitioner)

# Or just produce round-robin (or just use SimpleProducer)
producer = KeyedProducer(kafka, partitioner=RoundRobinPartitioner)
```

SimpleClient (DEPRECATED)

```
import time
from kafka import SimpleClient
from kafka.errors import LeaderNotAvailableError, NotLeaderForPartitionError
from kafka.protocol import create_message
from kafka.structs import ProduceRequestPayload

kafka = SimpleClient('localhost:9092')
payload = ProduceRequestPayload(topic='my-topic', partition=0,
                                messages=[create_message("some message")])

retries = 5
resps = []
while retries and not resps:
    retries -= 1
    try:
        resps = kafka.send_produce_request(
            payloads=[payload], fail_on_error=True)
    except LeaderNotAvailableError, NotLeaderForPartitionError:
        kafka.load_metadata_for_topics()
        time.sleep(1)

    # Other exceptions you might consider handling:
    # UnknownTopicOrPartitionError, TopicAuthorizationFailedError,
    # RequestTimedOutError, MessageSizeTooLargeError, InvalidTopicError,
    # RecordListTooLargeError, InvalidRequiredAcksError,
    # NotEnoughReplicasError, NotEnoughReplicasAfterAppendError

kafka.close()

resps[0].topic      # 'my-topic'
resps[0].partition  # 0
resps[0].error      # 0
resps[0].offset     # offset of the first message sent in this request
```

Install

Install with your favorite package manager

Latest Release

Pip:

```
pip install kafka-python
```

Releases are also listed at https://github.com/dpkp/kafka-python/releases

Bleeding-Edge

```
git clone https://github.com/dpkp/kafka-python
pip install ./kafka-python
```

Optional LZ4 install

To enable LZ4 compression/decompression, install python-lz4:

```
>>> pip install lz4
```

Optional Snappy install

Install Development Libraries

Download and build Snappy from http://code.google.com/p/snappy/downloads/list

Ubuntu:

```
apt-get install libsnappy-dev
```

OSX:

```
brew install snappy
```

From Source:

```
wget http://snappy.googlecode.com/files/snappy-1.0.5.tar.gz
tar xzvf snappy-1.0.5.tar.gz
cd snappy-1.0.5
./configure
make
sudo make install
```

Install Python Module

Install the *python-snappy* module

```
pip install python-snappy
```

Tests

Test environments are managed via tox. The test suite is run via pytest. Individual tests are written using unittest, pytest, and in some cases, doctest.

Linting is run via pylint, but is generally skipped on pypy due to pylint compatibility / performance issues.

For test coverage details, see https://coveralls.io/github/dpkp/kafka-python

The test suite includes unit tests that mock network interfaces, as well as integration tests that setup and teardown kafka broker (and zookeeper) fixtures for client / consumer / producer testing.

Unit tests

To run the tests locally, install tox – *pip install tox* See https://tox.readthedocs.io/en/latest/install.html

Then simply run tox, optionally setting the python environment. If unset, tox will loop through all environments.

```
tox -e py27
tox -e py35

# run protocol tests only
tox -- -v test.test_protocol

# re-run the last failing test, dropping into pdb
tox -e py27 -- --lf --pdb

# see available (pytest) options
tox -e py27 -- --help
```

Integration tests

```
KAFKA_VERSION=0.10.1.1 tox -e py27
KAFKA_VERSION=0.8.2.2 tox -e py35
```

Integration tests start Kafka and Zookeeper fixtures. This requires downloading kafka server binaries:

```
./build_integration.sh
```

By default, this will install 0.8.2.2, 0.9.0.1, 0.10.1.1, and 0.10.2.1 brokers into the servers/ directory. To install a specific version,

e.g., set *KAFKA_VERSION=0.10.2.1*:

```
KAFKA_VERSION=0.10.2.1 ./build_integration.sh
```

Then run the tests against supported Kafka versions, simply set the *KAFKA_VERSION* env variable to the server build you want to use for testing:

```
KAFKA_VERSION=0.10.2.1 tox -e py27
```

To test against the kafka source tree, set KAFKA_VERSION=trunk [optionally set SCALA_VERSION (defaults to 2.10)]

```
SCALA_VERSION=2.11 KAFKA_VERSION=trunk ./build_integration.sh
KAFKA_VERSION=trunk tox -e py35
```

Compatibility

kafka-python is compatible with (and tested against) broker versions 0.10 through 0.8.0 . kafka-python is not compatible with the 0.8.2-beta release.

kafka-python is tested on python 2.7, 3.3, 3.4, 3.5, and pypy.

Builds and tests via Travis-CI. See https://travis-ci.org/dpkp/kafka-python

Support

For support, see github issues at https://github.com/dpkp/kafka-python

Limited IRC chat at #kafka-python on freenode (general chat is #apache-kafka).

For information about Apache Kafka generally, see https://kafka.apache.org/

For general discussion of kafka-client design and implementation (not python specific), see https://groups.google.com/forum/m/#!forum/kafka-clients

License

Apache License, v2.0. See LICENSE.

Copyright 2016, Dana Powers, David Arthur, and Contributors (See AUTHORS).

Changelog

1.3.3 (Mar 14, 2017)

Core / Protocol

- Derive all api classes from Request / Response base classes (dpkp 1030)
- Prefer python-lz4 if available (dpkp 1024)
- Fix kwarg handing in kafka.protocol.struct.Struct (dpkp 1025)
- Fixed couple of “leaks” when gc is disabled (Mephius 979)
- Added *max_bytes* option and FetchRequest_v3 usage. (Drizzt1991 962)
- CreateTopicsRequest / Response v1 (dpkp 1012)
- Add MetadataRequest_v2 and MetadataResponse_v2 structures for KIP-78 (Drizzt1991 974)
- KIP-88 / KAFKA-3853: OffsetFetch v2 structs (jeffwidman 971)
- DRY-up the MetadataRequest_v1 struct (jeffwidman 966)

- Add JoinGroup v1 structs (jeffwidman 965)
- DRY-up the OffsetCommitResponse Structs (jeffwidman 970)
- DRY-up the OffsetFetch structs (jeffwidman 964)
- time –> timestamp to match Java API (jeffwidman 969)
- Add support for offsetRequestV1 messages (jlafaye 951)
- Add FetchRequest/Response_v3 structs (jeffwidman 943)
- Add CreateTopics / DeleteTopics Structs (jeffwidman 944)

Test Infrastructure

- Add python3.6 to travis test suite, drop python3.3 (exponea 992)
- Update to 0.10.1.1 for integration testing (dpkp 953)
- Update vendored berkerpeksag/selectors34 to ff61b82 (Mephius 979)
- Remove dead code (jeffwidman 967)
- Update pytest fixtures to new yield syntax (jeffwidman 919)

Consumer

- Avoid re-encoding message for crc check (dpkp 1027)
- Optionally skip auto-commit during consumer.close (dpkp 1031)
- Return copy of consumer subscription set (dpkp 1029)
- Short-circuit group coordinator requests when NodeNotReady (dpkp 995)
- Avoid unknown coordinator after client poll (dpkp 1023)
- No longer configure a default consumer group (dpkp 1016)
- Dont refresh metadata on failed group coordinator request unless needed (dpkp 1006)
- Fail-fast on timeout constraint violations during KafkaConsumer creation (harelba 986)
- Default max_poll_records to Java default of 500 (jeffwidman 947)
- For 0.8.2, only attempt connection to coordinator if least_loaded_node succeeds (dpkp)

Producer

- change default timeout of KafkaProducer.close() to threading.TIMEOUT_MAX on py3 (mmyjona 991)

Client

- Add optional kwarg to ready/is_ready to disable metadata-priority logic (dpkp 1017)
- When closing a broker connection without error, fail in-flight-requests with Cancelled (dpkp 1010)
- Catch socket errors during ssl handshake (dpkp 1007)
- Drop old brokers when rebuilding broker metadata (dpkp 1005)
- Drop bad disconnect test – just use the mocked-socket test (dpkp 982)

- Add support for Python built without ssl (minagawa-sho 954)
- Do not re-close a disconnected connection (dpkp)
- Drop unused last_failure time from BrokerConnection (dpkp)
- Use connection state functions where possible (dpkp)
- Pass error to BrokerConnection.close() (dpkp)

Bugfixes

- Free lz4 decompression context to avoid leak (dpkp 1024)
- Fix sasl reconnect bug: auth future must be reset on close (dpkp 1003)
- Fix raise exception from SubscriptionState.assign_from_subscribed (qntln 960)
- Fix blackout calculation: mark last_attempt time during connection close (dpkp 1008)
- Fix buffer pool reallocation after raising timeout (dpkp 999)

Logging / Error Messages

- Add client info logging re bootstrap; log connection attempts to balance with close (dpkp)
- Minor additional logging for consumer coordinator (dpkp)
- Add more debug-level connection logging (dpkp)
- Do not need str(self) when formatting to %s (dpkp)
- Add new broker response errors (dpkp)
- Small style fixes in kafka.errors (dpkp)
- Include the node id in BrokerConnection logging (dpkp 1009)
- Replace %s with %r in producer debug log message (chekunkov 973)

Documentation

- Sphinx documentation updates (jeffwidman 1019)
- Add sphinx formatting to hyperlink methods (jeffwidman 898)
- Fix BrokerConnection api_version docs default (jeffwidman 909)
- PEP-8: Spacing & removed unused imports (jeffwidman 899)
- Move BrokerConnection docstring to class (jeffwidman 968)
- Move docstring so it shows up in Sphinx/RTD (jeffwidman 952)
- Remove non-pip install instructions (jeffwidman 940)
- Spelling and grammar changes (melissacrawford396 923)
- Fix typo: coorelation –> correlation (jeffwidman 929)
- Make SSL warning list the correct Python versions (jeffwidman 924)
- Fixup comment reference to _maybe_connect (dpkp)
- Add ClusterMetadata sphinx documentation (dpkp)

Legacy Client

- Add send_list_offset_request for searching offset by timestamp (charsyam 1001)
- Use select to poll sockets for read to reduce CPU usage (jianbin-wei 958)
- Use select.select without instance bounding (adamwen829 949)

1.3.2 (Dec 28, 2016)

Core

- Add kafka.serializer interfaces (dpkp 912)
- from kafka import ConsumerRebalanceListener, OffsetAndMetadata
- Use 0.10.0.1 for integration tests (dpkp 803)

Consumer

- KAFKA-3007: KafkaConsumer max_poll_records (dpkp 831)
- Raise exception if given a non-str topic (ssaamm 824)
- Immediately update metadata for pattern subscription (laz2 915)

Producer

- Update Partitioners for use with KafkaProducer (barrotsteindev 827)
- Sort partitions before calling partitioner (ms7s 905)
- Added ssl_password config option to KafkaProducer class (kierkegaard13 830)

Client

- Always check for request timeouts (dpkp 887)
- When hostname lookup is necessary, do every connect (benauthor 812)

Bugfixes

- Fix errorcode check when socket.connect_ex raises an exception (guojh 907)
- Fix fetcher bug when processing offset out of range (sibiryakov 860)
- Fix possible request draining in ensure_active_group (dpkp 896)
- Fix metadata refresh handling with 0.10+ brokers when topic list is empty (sibiryakov 867)
- KafkaProducer should set timestamp in Message if provided (Drizzt1991 875)
- Fix murmur2 bug handling python2 bytes that do not ascii encode (dpkp 815)
- Monkeypatch max_in_flight_requests_per_connection when checking broker version (dpkp 834)
- Fix message timestamp_type (qix 828)

Logging / Error Messages

- Always include an error for logging when the coordinator is marked dead (dpkp 890)
- Only string-ify BrokerResponseError args if provided (dpkp 889)
- Update warning re advertised.listeners / advertised.host.name (jeffwidman 878)
- Fix unrecognized sasl_mechanism error message (sharego 883)

Documentation

- Add docstring for max_records (jeffwidman 897)
- Fixup doc references to max_in_flight_requests_per_connection
- Fix typo: passowrd –> password (jeffwidman 901)
- Fix documentation typo 'Defualt' -> 'Default'. (rolando 895)
- Added doc for *max_poll_records* option (Drizzt1991 881)
- Remove old design notes from Kafka 8 era (jeffwidman 876)
- Fix documentation typos (jeffwidman 874)
- Fix quota violation exception message (dpkp 809)
- Add comment for round robin partitioner with different subscriptions
- Improve KafkaProducer docstring for retries configuration

1.3.1 (Aug 8, 2016)

Bugfixes

- Fix AttributeError in BrokerConnectionMetrics after reconnecting

1.3.0 (Aug 4, 2016)

Incompatible Changes

- Delete KafkaConnection class (dpkp 769)
- Rename partition_assignment -> assignment in MemberMetadata for consistency
- Move selectors34 and socketpair to kafka.vendor (dpkp 785)
- Change api_version config to tuple; deprecate str with warning (dpkp 761)
- Rename _DEFAULT_CONFIG -> DEFAULT_CONFIG in KafkaProducer (dpkp 788)

Improvements

- Vendor six 1.10.0 to eliminate runtime dependency (dpkp 785)
- Add KafkaProducer and KafkaConsumer.metrics() with instrumentation similar to java client (dpkp 754 / 772 / 794)

- Support Sasl PLAIN authentication (larsjsol PR 779)
- Add checksum and size to RecordMetadata and ConsumerRecord (KAFKA-3196 / 770 / 594)
- Use MetadataRequest v1 for 0.10+ api_version (dpkp 762)
- Fix KafkaConsumer autocommit for 0.8 brokers (dpkp 756 / 706)
- Improve error logging (dpkp 760 / 759)
- Adapt benchmark scripts from https://github.com/mrafayaleem/kafka-jython (dpkp 754)
- Add api_version config to KafkaClient (dpkp 761)
- New Metadata method with_partitions() (dpkp 787)
- Use socket_options configuration to setsockopts(). Default TCP_NODELAY (dpkp 783)
- Expose selector type as config option (dpkp 764)
- Drain pending requests to the coordinator before initiating group rejoin (dpkp 798)
- Send combined size and payload bytes to socket to avoid potentially split packets with TCP_NODELAY (dpkp 797)

Bugfixes

- Ignore socket.error when checking for protocol out of sync prior to socket close (dpkp 792)
- Fix offset fetch when partitions are manually assigned (KAFKA-3960 / 786)
- Change pickle_method to use python3 special attributes (jpaulodit 777)
- Fix ProduceResponse v2 throttle_time_ms
- Always encode size with MessageSet (#771)
- Avoid buffer overread when compressing messageset in KafkaProducer
- Explicit format string argument indices for python 2.6 compatibility
- Simplify RecordMetadata; short circuit callbacks (#768)
- Fix autocommit when partitions assigned manually (KAFKA-3486 / #767 / #626)
- Handle metadata updates during consumer rebalance (KAFKA-3117 / #766 / #701)
- Add a consumer config option to exclude internal topics (KAFKA-2832 / #765)
- Protect writes to wakeup socket with threading lock (#763 / #709)
- Fetcher spending unnecessary time during metrics recording (KAFKA-3785)
- Always use absolute_import (dpkp)

Test / Fixtures

- Catch select errors while capturing test fixture logs
- Fix consumer group test race condition (dpkp 795)
- Retry fixture failures on a different port (dpkp 796)
- Dump fixture logs on failure

Documentation

- Fix misspelling of password (ssaamm 793)
- Document the ssl_password config option (ssaamm 780)
- Fix typo in KafkaConsumer documentation (ssaamm 775)
- Expand consumer.fetcher inline comments
- Update kafka configuration links -> 0.10.0.0 docs
- Fixup metrics_sample_window_ms docstring in consumer

1.2.5 (July 15, 2016)

Bugfixes

- Fix bug causing KafkaProducer to double-compress message batches on retry
- Check for double-compressed messages in KafkaConsumer, log warning and optionally skip
- Drop recursion in _unpack_message_set; only decompress once

1.2.4 (July 8, 2016)

Bugfixes

- Update consumer_timeout_ms docstring - KafkaConsumer raises StopIteration, no longer ConsumerTimeout
- Use explicit subscription state flag to handle seek() during message iteration
- Fix consumer iteration on compacted topics (dpkp PR 752)
- Support ssl_password config when loading cert chains (amckemie PR 750)

1.2.3 (July 2, 2016)

Patch Improvements

- Fix gc error log: avoid AttributeError in _unregister_cleanup (dpkp PR 747)
- Wakeup socket optimizations (dpkp PR 740)
- Assert will be disabled by "python -O" (tyronecai PR 736)
- Randomize order of topics/partitions processed by fetcher to improve balance (dpkp PR 732)
- Allow client.check_version timeout to be set in Producer and Consumer constructors (eastlondoner PR 647)

1.2.2 (June 21, 2016)

Bugfixes

- Clarify timeout unit in KafkaProducer close and flush (ms7s PR 734)
- Avoid busy poll during metadata refresh failure with retry_backoff_ms (dpkp PR 733)

- Check_version should scan nodes until version found or timeout (dpkp PR 731)
- Fix bug which could cause least_loaded_node to always return the same unavailable node (dpkp PR 730)
- Fix producer garbage collection with weakref in atexit handler (dpkp PR 728)
- Close client selector to fix fd leak (msmith PR 729)
- Tweak spelling mistake in error const (steve8918 PR 719)
- Rearrange connection tests to separate legacy KafkaConnection

1.2.1 (June 1, 2016)

Bugfixes

- Fix regression in MessageSet decoding wrt PartialMessages (#716)
- Catch response decode errors and log details (#715)
- Fix Legacy support url (#712 - JonasGroeger)
- Update sphinx docs re 0.10 broker support

1.2.0 (May 24, 2016)

Support Kafka 0.10 Features

- Add protocol support for ApiVersionRequest (dpkp PR 678)
- KAFKA-3025: Message v1 – add timetamp and relative offsets (dpkp PR 693)
- Use Fetch/Produce API v2 for brokers >= 0.10 (uses message format v1) (dpkp PR 694)
- Use standard LZ4 framing for v1 messages / kafka 0.10 (dpkp PR 695)

Consumers

- Update SimpleConsumer / legacy protocol to handle compressed messages (paulcavallaro PR 684)

Producers

- KAFKA-3388: Fix expiration of batches sitting in the accumulator (dpkp PR 699)
- KAFKA-3197: when max.in.flight.request.per.connection = 1, attempt to guarantee ordering (dpkp PR 698)
- Don't use soon-to-be-reserved keyword await as function name (FutureProduceResult) (dpkp PR 697)

Clients

- Fix socket leaks in KafkaClient (dpkp PR 696)

Documentation

<none>

Internals

- Support SSL CRL [requires python 2.7.9+ / 3.4+] (vincentbernat PR 683)
- Use original hostname for SSL checks (vincentbernat PR 682)
- Always pass encoded message bytes to MessageSet.encode()
- Raise ValueError on protocol encode/decode errors
- Supplement socket.gaierror exception in BrokerConnection.connect() (erikbeebe PR 687)
- BrokerConnection check_version: expect 0.9 to fail with CorrelationIdError
- Fix small bug in Sensor (zackdever PR 679)

1.1.1 (Apr 26, 2016)

Bugfixes

- Fix throttle_time_ms sensor handling (zackdever PR 667)
- Improve handling of disconnected sockets (EasyPost PR 666 / dpkp)
- Disable standard metadata refresh triggers during bootstrap (dpkp)
- More predictable Future callback/errback exceptions (zackdever PR 670)
- Avoid some exceptions in Coordinator.__del__ (dpkp PR 668)

1.1.0 (Apr 25, 2016)

Consumers

- Avoid resending FetchRequests that are pending on internal queue
- Log debug messages when skipping fetched messages due to offset checks
- KAFKA-3013: Include topic-partition in exception for expired batches
- KAFKA-3318: clean up consumer logging and error messages
- Improve unknown coordinator error handling
- Improve auto-commit error handling when group_id is None
- Add paused() API (zackdever PR 602)
- Add default_offset_commit_callback to KafkaConsumer DEFAULT_CONFIGS

Producers

<none>

Clients

- Support SSL connections
- Use selectors module for non-blocking IO
- Refactor KafkaClient connection management
- Fix AttributeError in __del__
- SimpleClient: catch errors thrown by _get_leader_for_partition (zackdever PR 606)

Documentation

- Fix serializer/deserializer examples in README
- Update max.block.ms docstring
- Remove errant next(consumer) from consumer documentation
- Add producer.flush() to usage docs

Internals

- Add initial metrics implementation (zackdever PR 637)
- KAFKA-2136: support Fetch and Produce v1 (throttle_time_ms)
- Use version-indexed lists for request/response protocol structs (dpkp PR 630)
- Split kafka.common into kafka.structs and kafka.errors
- Handle partial socket send() (dpkp PR 611)
- Fix windows support (dpkp PR 603)
- IPv6 support (TimEvens PR 615; Roguelazer PR 642)

1.0.2 (Mar 14, 2016)

Consumers

- Improve KafkaConsumer Heartbeat handling (dpkp PR 583)
- Fix KafkaConsumer.position bug (stefanth PR 578)
- Raise TypeError when partition is not a TopicPartition (dpkp PR 587)
- KafkaConsumer.poll should sleep to prevent tight-loops (dpkp PR 597)

Producers

- Fix producer threading bug that can crash sender (dpkp PR 590)
- Fix bug in producer buffer pool reallocation (dpkp PR 585)
- Remove spurious warnings when closing sync SimpleProducer (twm PR 567)
- Fix FutureProduceResult.await() on python2.6 (dpkp)
- Add optional timeout parameter to KafkaProducer.flush() (dpkp)

- KafkaProducer optimizations (zackdever PR 598)

Clients

- Improve error handling in SimpleClient.load_metadata_for_topics (dpkp)
- Improve handling of KafkaClient.least_loaded_node failure (dpkp PR 588)

Documentation

- Fix KafkaError import error in docs (shichao-an PR 564)
- Fix serializer / deserializer examples (scribu PR 573)

Internals

- Update to Kafka 0.9.0.1 for integration testing
- Fix ifr.future.failure in conn.py (mortenlj PR 566)
- Improve Zookeeper / Kafka Fixture management (dpkp)

1.0.1 (Feb 19, 2016)

Consumers

- Add RangePartitionAssignor (and use as default); add assignor tests (dpkp PR 550)
- Make sure all consumers are in same generation before stopping group test
- Verify node ready before sending offset fetch request from coordinator
- Improve warning when offset fetch request returns unknown topic / partition

Producers

- Warn if pending batches failed during flush
- Fix concurrency bug in RecordAccumulator.ready()
- Fix bug in SimpleBufferPool memory condition waiting / timeout
- Support batch_size = 0 in producer buffers (dpkp PR 558)
- Catch duplicate batch.done() calls [e.g., maybe_expire then a response errback]

Clients

Documentation

- Improve kafka.cluster docstrings
- Migrate load_example.py to KafkaProducer / KafkaConsumer

Internals

- Don't override system rcvbuf or sndbuf unless configured explicitly (dpkp PR 557)
- Some attributes may not exist in __del__ if we failed assertions
- Break up some circular references and close client wake pipes on __del__ (aisch PR 554)

1.0.0 (Feb 15, 2016)

This release includes significant code changes. Users of older kafka-python versions are encouraged to test upgrades before deploying to production as some interfaces and configuration options have changed.

Users of SimpleConsumer / SimpleProducer / SimpleClient (formerly KafkaClient) from prior releases should migrate to KafkaConsumer / KafkaProducer. Low-level APIs (Simple*) are no longer being actively maintained and will be removed in a future release.

For comprehensive API documentation, please see python help() / docstrings, kafka-python.readthedocs.org, or run 'tox -e docs' from source to build documentation locally.

Consumers

- KafkaConsumer re-written to emulate the new 0.9 kafka consumer (java client) and support coordinated consumer groups (feature requires >= 0.9.0.0 brokers)
  - Methods no longer available:
    - configure [initialize a new consumer instead]
    - set_topic_partitions [use subscribe() or assign()]
    - fetch_messages [use poll() or iterator interface]
    - get_partition_offsets
    - offsets [use committed(partition)]
    - task_done [handled internally by auto-commit; or commit offsets manually]
  - Configuration changes (consistent with updated java client):
    - lots of new configuration parameters – see docs for details
    - auto_offset_reset: previously values were 'smallest' or 'largest', now values are 'earliest' or 'latest'
    - fetch_wait_max_ms is now fetch_max_wait_ms
    - max_partition_fetch_bytes is now max_partition_fetch_bytes
    - deserializer_class is now value_deserializer and key_deserializer
    - auto_commit_enable is now enable_auto_commit
    - auto_commit_interval_messages was removed
    - socket_timeout_ms was removed
    - refresh_leader_backoff_ms was removed
- SimpleConsumer and MultiProcessConsumer are now deprecated and will be removed in a future release. Users are encouraged to migrate to KafkaConsumer.

Producers

- new producer class: KafkaProducer. Exposes the same interface as official java client. Async by default; returned future.get() can be called for synchronous blocking
- SimpleProducer is now deprecated and will be removed in a future release. Users are encouraged to migrate to KafkaProducer.

Clients

- synchronous KafkaClient renamed to SimpleClient. For backwards compatibility, you will get a SimpleClient via 'from kafka import KafkaClient'. This will change in a future release.
- All client calls use non-blocking IO under the hood.
- Add probe method check_version() to infer broker versions.

Documentation

- Updated README and sphinx documentation to address new classes.
- Docstring improvements to make python help() easier to use.

Internals

- Old protocol stack is deprecated. It has been moved to kafka.protocol.legacy and may be removed in a future release.
- Protocol layer re-written using Type classes, Schemas and Structs (modeled on the java client).
- Add support for LZ4 compression (including broken framing header checksum).

0.9.5 (Dec 6, 2015)

Consumers

- Initial support for consumer coordinator: offsets only (toddpalino PR 420)
- Allow blocking until some messages are received in SimpleConsumer (saaros PR 457)
- Support subclass config changes in KafkaConsumer (zackdever PR 446)
- Support retry semantics in MultiProcessConsumer (barricadeio PR 456)
- Support partition_info in MultiProcessConsumer (scrapinghub PR 418)
- Enable seek() to an absolute offset in SimpleConsumer (haosdent PR 412)
- Add KafkaConsumer.close() (ucarion PR 426)

Producers

- Catch client.reinit() exceptions in async producer (dpkp)
- Producer.stop() now blocks until async thread completes (dpkp PR 485)
- Catch errors during load_metadata_for_topics in async producer (bschopman PR 467)

- Add compression-level support for codecs that support it (trbs PR 454)
- Fix translation of Java murmur2 code, fix byte encoding for Python 3 (chrischamberlin PR 439)
- Only call stop() on not-stopped producer objects (docker-hub PR 435)
- Allow null payload for deletion feature (scrapinghub PR 409)

Clients

- Use non-blocking io for broker aware requests (ecanzonieri PR 473)
- Use debug logging level for metadata request (ecanzonieri PR 415)
- Catch KafkaUnavailableError in _send_broker_aware_request (mutability PR 436)
- Lower logging level on replica not available and commit (ecanzonieri PR 415)

Documentation

- Update docs and links wrt maintainer change (mumrah -> dpkp)

Internals

- Add py35 to tox testing
- Update travis config to use container infrastructure
- Add 0.8.2.2 and 0.9.0.0 resources for integration tests; update default official releases
- new pylint disables for pylint 1.5.1 (zackdever PR 481)
- Fix python3 / python2 comments re queue/Queue (dpkp)
- Add Murmur2Partitioner to kafka __all__ imports (dpkp Issue 471)
- Include LICENSE in PyPI sdist (koobs PR 441)

0.9.4 (June 11, 2015)

Consumers

- Refactor SimpleConsumer internal fetch handling (dpkp PR 399)
- Handle exceptions in SimpleConsumer commit() and reset_partition_offset() (dpkp PR 404)
- Improve FailedPayloadsError handling in KafkaConsumer (dpkp PR 398)
- KafkaConsumer: avoid raising KeyError in task_done (dpkp PR 389)
- MultiProcessConsumer – support configured partitions list (dpkp PR 380)
- Fix SimpleConsumer leadership change handling (dpkp PR 393)
- Fix SimpleConsumer connection error handling (reAsOn2010 PR 392)
- Improve Consumer handling of 'falsy' partition values (wting PR 342)
- Fix _offsets call error in KafkaConsumer (hellais PR 376)
- Fix str/bytes bug in KafkaConsumer (dpkp PR 365)

- Register atexit handlers for consumer and producer thread/multiprocess cleanup (dpkp PR 360)
- Always fetch commit offsets in base consumer unless group is None (dpkp PR 356)
- Stop consumer threads on delete (dpkp PR 357)
- Deprecate metadata_broker_list in favor of bootstrap_servers in KafkaConsumer (dpkp PR 340)
- Support pass-through parameters in multiprocess consumer (scrapinghub PR 336)
- Enable offset commit on SimpleConsumer.seek (ecanzonieri PR 350)
- Improve multiprocess consumer partition distribution (scrapinghub PR 335)
- Ignore messages with offset less than requested (wkiser PR 328)
- Handle OffsetOutOfRange in SimpleConsumer (ecanzonieri PR 296)

Producers

- Add Murmur2Partitioner (dpkp PR 378)
- Log error types in SimpleProducer and SimpleConsumer (dpkp PR 405)
- SimpleProducer support configuration of fail_on_error (dpkp PR 396)
- Deprecate KeyedProducer.send() (dpkp PR 379)
- Further improvements to async producer code (dpkp PR 388)
- Add more configuration parameters for async producer (dpkp)
- Deprecate SimpleProducer batch_send=True in favor of async (dpkp)
- Improve async producer error handling and retry logic (vshlapakov PR 331)
- Support message keys in async producer (vshlapakov PR 329)
- Use threading instead of multiprocessing for Async Producer (vshlapakov PR 330)
- Stop threads on __del__ (chmduquesne PR 324)
- Fix leadership failover handling in KeyedProducer (dpkp PR 314)

KafkaClient

- Add .topics property for list of known topics (dpkp)
- Fix request / response order guarantee bug in KafkaClient (dpkp PR 403)
- Improve KafkaClient handling of connection failures in _get_conn (dpkp)
- Client clears local metadata cache before updating from server (dpkp PR 367)
- KafkaClient should return a response or error for each request - enable better retry handling (dpkp PR 366)
- Improve str/bytes conversion in KafkaClient and KafkaConsumer (dpkp PR 332)
- Always return sorted partition ids in client.get_partition_ids_for_topic() (dpkp PR 315)

Documentation

- Cleanup Usage Documentation
- Improve KafkaConsumer documentation (dpkp PR 341)
- Update consumer documentation (sontek PR 317)
- Add doc configuration for tox (sontek PR 316)
- Switch to .rst doc format (sontek PR 321)
- Fixup google groups link in README (sontek PR 320)
- Automate documentation at kafka-python.readthedocs.org

Internals

- Switch integration testing from 0.8.2.0 to 0.8.2.1 (dpkp PR 402)
- Fix most flaky tests, improve debug logging, improve fixture handling (dpkp)
- General style cleanups (dpkp PR 394)
- Raise error on duplicate topic-partition payloads in protocol grouping (dpkp)
- Use module-level loggers instead of simply 'kafka' (dpkp)
- Remove pkg_resources check for __version__ at runtime (dpkp PR 387)
- Make external API consistently support python3 strings for topic (kecaps PR 361)
- Fix correlation id overflow (dpkp PR 355)
- Cleanup kafka/common structs (dpkp PR 338)
- Use context managers in gzip_encode / gzip_decode (dpkp PR 337)
- Save failed request as FailedPayloadsError attribute (jobevers PR 302)
- Remove unused kafka.queue (mumrah)

0.9.3 (Feb 3, 2015)

- Add coveralls.io support (sontek PR 307)
- Fix python2.6 threading.Event bug in ReentrantTimer (dpkp PR 312)
- Add kafka 0.8.2.0 to travis integration tests (dpkp PR 310)
- Auto-convert topics to utf-8 bytes in Producer (sontek PR 306)
- Fix reference cycle between SimpleConsumer and ReentrantTimer (zhaopengzp PR 309)
- Add Sphinx API docs (wedaly PR 282)
- Handle additional error cases exposed by 0.8.2.0 kafka server (dpkp PR 295)
- Refactor error class management (alexcb PR 289)
- Expose KafkaConsumer in __all__ for easy imports (Dinoshauer PR 286)
- SimpleProducer starts on random partition by default (alexcb PR 288)
- Add keys to compressed messages (meandthewallaby PR 281)

- Add new high-level KafkaConsumer class based on java client api (dpkp PR 234)
- Add KeyedProducer.send_messages api (pubnub PR 277)
- Fix consumer pending() method (jettify PR 276)
- Update low-level demo in README (sunisdown PR 274)
- Include key in KeyedProducer messages (se7entyse7en PR 268)
- Fix SimpleConsumer timeout behavior in get_messages (dpkp PR 238)
- Fix error in consumer.py test against max_buffer_size (rthille/wizzat PR 225/242)
- Improve string concat performance on pypy / py3 (dpkp PR 233)
- Reorg directory layout for consumer/producer/partitioners (dpkp/wizzat PR 232/243)
- Add OffsetCommitContext (locationlabs PR 217)
- Metadata Refactor (dpkp PR 223)
- Add Python 3 support (brutasse/wizzat - PR 227)
- Minor cleanups - imports / README / PyPI classifiers (dpkp - PR 221)
- Fix socket test (dpkp - PR 222)
- Fix exception catching bug in test_failover_integration (zever - PR 216)

0.9.2 (Aug 26, 2014)

- Warn users that async producer does not reliably handle failures (dpkp - PR 213)
- Fix spurious ConsumerFetchSizeTooSmall error in consumer (DataDog - PR 136)
- Use PyLint for static error checking (dpkp - PR 208)
- Strictly enforce str message type in producer.send_messages (dpkp - PR 211)
- Add test timers via nose-timer plugin; list 10 slowest timings by default (dpkp)
- Move fetching last known offset logic to a stand alone function (zever - PR 177)
- Improve KafkaConnection and add more tests (dpkp - PR 196)
- Raise TypeError if necessary when encoding strings (mdaniel - PR 204)
- Use Travis-CI to publish tagged releases to pypi (tkuhlman / mumrah)
- Use official binary tarballs for integration tests and parallelize travis tests (dpkp - PR 193)
- Improve new-topic creation handling (wizzat - PR 174)

0.9.1 (Aug 10, 2014)

- Add codec parameter to Producers to enable compression (patricklucas - PR 166)
- Support IPv6 hosts and network (snaury - PR 169)
- Remove dependency on distribute (patricklucas - PR 163)
- Fix connection error timeout and improve tests (wizzat - PR 158)
- SimpleProducer randomization of initial round robin ordering (alexcb - PR 139)
- Fix connection timeout in KafkaClient and KafkaConnection (maciejkula - PR 161)

- Fix seek + commit behavior (wizzat - PR 148)

0.9.0 (Mar 21, 2014)

- Connection refactor and test fixes (wizzat - PR 134)
- Fix when partition has no leader (mrtheb - PR 109)
- Change Producer API to take topic as send argument, not as instance variable (rdiomar - PR 111)
- Substantial refactor and Test Fixing (rdiomar - PR 88)
- Fix Multiprocess Consumer on windows (mahendra - PR 62)
- Improve fault tolerance; add integration tests (jimjh)
- PEP8 / Flakes / Style cleanups (Vetoshkin Nikita; mrtheb - PR 59)
- Setup Travis CI (jimjh - PR 53/54)
- Fix import of BufferUnderflowError (jimjh - PR 49)
- Fix code examples in README (StevenLeRoux - PR 47/48)

0.8.0

- Changing auto_commit to False in [SimpleConsumer](kafka/consumer.py), until 0.8.1 is release offset commits are unsupported
- Adding fetch_size_bytes to SimpleConsumer constructor to allow for user-configurable fetch sizes
- Allow SimpleConsumer to automatically increase the fetch size if a partial message is read and no other messages were read during that fetch request. The increase factor is 1.5
- Exception classes moved to kafka.common

Index

O

P

R

S

T

U

W