pykafka

Release 1.1.1

Sep 18, 2017

Contents

PyKafka is a cluster-aware Kafka protocol client for python. It includes python implementations of Kafka producers and consumers.

PyKafka's primary goal is to provide a similar level of abstraction to the JVM Kafka client using idioms familiar to python programmers and exposing the most pythonic API possible.

You can install PyKafka from PyPI with

```
$ pip install pykafka
```

Full documentation for PyKafka can be found on readthedocs.

CHAPTER 1

Getting Started

Assuming you have a Kafka instance running on localhost, you can use PyKafka to connect to it.

```
>>> from pykafka import KafkaClient
>>> client = KafkaClient(hosts="127.0.0.1:9092")
```

If the cluster you've connected to has any topics defined on it, you can list them with:

```
>>> client.topics
{'my.test': <pykafka.topic.Topic at 0x19bc8c0 (name=my.test)>}
>>> topic = client.topics['my.test']
```

Once you've got a *Topic*, you can create a *Producer* for it and start producing messages.

```
>>> producer = topic.get_producer()
>>> producer.produce(['test message ' + i ** 2 for i in range(4)])
```

You can also consume messages from this topic using a *Consumer* instance.

```
>>> consumer = topic.get_simple_consumer()
>>> for message in consumer:
    if message is not None:
        print message.offset, message.value
0 test message 0
1 test message 1
2 test message 4
3 test message 9
```

This *SimpleConsumer* doesn't scale - if you have two *SimpleConsumers* consuming the same topic, they will receive duplicate messages. To get around this, you can use the *BalancedConsumer*.

```
>>> balanced_consumer = topic.get_balanced_consumer(
    consumer_group='testgroup',
    auto_commit_enable=True,
    zookeeper_connect='myZkClusterNode1.com:2181,myZkClusterNode2.com:2181/myZkChroot'
)
```

You can have as many *BalancedConsumer* instances consuming a topic as that topic has partitions. If they are all connected to the same zookeeper instance, they will communicate with it to automatically balance the partitions between themselves.

CHAPTER 2

What happened to Samsa?

This project used to be called samsa. It has been renamed PyKafka and has been fully overhauled to support Kafka 0.8.2. We chose to target 0.8.2 because it's currently the latest stable version, and the Offset Commit/Fetch API is stabilized.

The Samsa PyPI package will stay up for the foreseeable future and tags for previous versions will always be available in this repo.

CHAPTER 3

Support

If you need help using PyKafka or have found a bug, please open a github issue or use the Google Group.

API Documentation

pykafka.balancedconsumer

class pykafka.balancedconsumer.**BalancedConsumer**(*topic*, *cluster*, *consumer_group*, *fetch_message_max_bytes=1048576*, *num_consumer_fetchers=1*, *auto_commit_enable=False*, *auto_commit_interval_ms=60000*, *queued_max_messages=2000*, *fetch_min_bytes=1*, *fetch_wait_max_ms=100*, *offsets_channel_backoff_ms=1000*, *offsets_commit_max_retries=5*, *auto_offset_reset=-1*, *consumer_timeout_ms=-1*, *rebalance_max_retries=5*, *rebalance_backoff_ms=2000*, *zookeeper_connection_timeout_ms=6000*, *zookeeper_connect='127.0.0.1:2181'*, *zookeeper=None*, *auto_start=True*, *reset_offset_on_start=False*)

A self-balancing consumer for Kafka that uses ZooKeeper to communicate with other balancing consumers.

Maintains a single instance of SimpleConsumer, periodically using the consumer rebalancing algorithm to reassign partitions to this SimpleConsumer.

__init__(*topic*, *cluster*, *consumer_group*, *fetch_message_max_bytes=1048576*, *num_consumer_fetchers=1*, *auto_commit_enable=False*, *auto_commit_interval_ms=60000*, *queued_max_messages=2000*, *fetch_min_bytes=1*, *fetch_wait_max_ms=100*, *offsets_channel_backoff_ms=1000*, *offsets_commit_max_retries=5*, *auto_offset_reset=-1*, *consumer_timeout_ms=-1*, *rebalance_max_retries=5*, *rebalance_backoff_ms=2000*, *zookeeper_connection_timeout_ms=6000*, *zookeeper_connect='127.0.0.1:2181'*, *zookeeper=None*, *auto_start=True*, *reset_offset_on_start=False*)

Create a BalancedConsumer instance

Parameters

- **topic** (*pykafka.topic.Topic*) – The topic this consumer should consume
- **cluster** (*pykafka.cluster.Cluster*) – The cluster to which this consumer should connect
- **consumer_group** (*str*) – The name of the consumer group this consumer should join.
- **fetch_message_max_bytes** (*int*) – The number of bytes of messages to attempt to fetch with each fetch request
- **num_consumer_fetchers** (*int*) – The number of workers used to make FetchRequests
- **auto_commit_enable** (*bool*) – If true, periodically commit to kafka the offset of messages already fetched by this consumer. This also requires that *consumer_group* is not *None*.
- **auto_commit_interval_ms** (*int*) – The frequency (in milliseconds) at which the consumer's offsets are committed to kafka. This setting is ignored if *auto_commit_enable* is *False*.
- **queued_max_messages** (*int*) – The maximum number of messages buffered for consumption in the internal *pykafka.simpleconsumer.SimpleConsumer*
- **fetch_min_bytes** (*int*) – The minimum amount of data (in bytes) that the server should return for a fetch request. If insufficient data is available, the request will block until sufficient data is available.
- **fetch_wait_max_ms** (*int*) – The maximum amount of time (in milliseconds) that the server will block before answering a fetch request if there isn't sufficient data to immediately satisfy *fetch_min_bytes*.
- **offsets_channel_backoff_ms** (*int*) – Backoff time to retry failed offset commits and fetches.
- **offsets_commit_max_retries** (*int*) – The number of times the offset commit worker should retry before raising an error.
- **auto_offset_reset** (*pykafka.common.OffsetType*) – What to do if an offset is out of range. This setting indicates how to reset the consumer's internal offset counter when an *OffsetOutOfRangeError* is encountered.
- **consumer_timeout_ms** (*int*) – Amount of time (in milliseconds) the consumer may spend without messages available for consumption before returning None.
- **rebalance_max_retries** (*int*) – The number of times the rebalance should retry before raising an error.
- **rebalance_backoff_ms** (*int*) – Backoff time (in milliseconds) between retries during rebalance.

- **zookeeper_connection_timeout_ms** (*int*) – The maximum time (in milliseconds) that the consumer waits while establishing a connection to zookeeper.
- **zookeeper_connect** (*str*) – Comma-separated (ip1:port1,ip2:port2) strings indicating the zookeeper nodes to which to connect.
- **zookeeper** (kazoo.client.KazooClient) – A KazooClient connected to a Zookeeper instance. If provided, *zookeeper_connect* is ignored.
- **auto_start** (*bool*) – Whether the consumer should begin communicating with zookeeper after __init__ is complete. If false, communication can be started with *start()*.
- **reset_offset_on_start** (*bool*) – Whether the consumer should reset its internal offset counter to *self._auto_offset_reset* and commit that offset immediately upon starting up

_add_partitions(*partitions*)
Add partitions to the zookeeper registry for this consumer.

Also add these partitions to the consumer's internal partition registry.

Parameters **partitions** (Iterable of *pykafka.partition.Partition*) – The partitions to add.

_add_self()
Register this consumer in zookeeper.

This method ensures that the number of participants is at most the number of partitions.

_check_held_partitions()
Double-check held partitions against zookeeper

Ensure that the partitions held by this consumer are the ones that zookeeper thinks it's holding. If not, rebalance.

_decide_partitions(*participants*)
Decide which partitions belong to this consumer.

Uses the consumer rebalancing algorithm described here http://kafka.apache.org/documentation.html

It is very important that the participants array is sorted, since this algorithm runs on each consumer and indexes into the same array. The same array index operation must return the same result on each consumer.

Parameters **participants** (*Iterable of str*) – Sorted list of ids of all other consumers in this consumer group.

_get_participants()
Use zookeeper to get the other consumers of this topic.

Returns A sorted list of the ids of the other consumers of this consumer's topic

_path_from_partition(*p*)
Given a partition, return its path in zookeeper.

_rebalance()
Claim partitions for this consumer.

This method is called whenever a zookeeper watch is triggered.

_remove_partitions(*partitions*)
Remove partitions from the zookeeper registry for this consumer.

Also remove these partitions from the consumer's internal partition registry.

Parameters **partitions** (Iterable of *pykafka.partition.Partition*) – The partitions to remove.

_set_watches()
Set watches in zookeeper that will trigger rebalances.

Rebalances should be triggered whenever a broker, topic, or consumer znode is changed in zookeeper. This ensures that the balance of the consumer group remains up-to-date with the current state of the cluster.

_setup_checker_worker()
Start the zookeeper partition checker thread

_setup_internal_consumer(*start=True*)
Instantiate an internal SimpleConsumer.

If there is already a SimpleConsumer instance held by this object, disable its workers and mark it for garbage collection before creating a new one.

_setup_zookeeper(*zookeeper_connect*, *timeout*)
Open a connection to a ZooKeeper host.

Parameters

- **zookeeper_connect** (*str*) – The 'ip:port' address of the zookeeper node to which to connect.
- **timeout** (*int*) – Connection timeout (in milliseconds)

commit_offsets()
Commit offsets for this consumer's partitions

Uses the offset commit/fetch API

consume(*block=True*)
Get one message from the consumer

Parameters **block** (*bool*) – Whether to block while waiting for a message

held_offsets
Return a map from partition id to held offset for each partition

reset_offsets(*partition_offsets=None*)
Reset offsets for the specified partitions

Issue an OffsetRequest for each partition and set the appropriate returned offset in the OwnedPartition

Parameters **partition_offsets** (Iterable of (*pykafka.partition.Partition*, int)) – (*partition*, *offset*) pairs to reset where *partition* is the partition for which to reset the offset and *offset* is the new offset the partition should have

start()
Open connections and join a cluster.

stop()
Close the zookeeper connection and stop consuming.

This method should be called as part of a graceful shutdown process.

pykafka.broker

Author: Keith Bourgoin, Emmett Butler

class pykafka.broker.**Broker**(*id_*, *host*, *port*, *handler*, *socket_timeout_ms*, *offsets_channel_socket_timeout_ms*, *buffer_size=1048576*)

A Broker is an abstraction over a real kafka server instance. It is used to perform requests to these servers.

__init__(*id_*, *host*, *port*, *handler*, *socket_timeout_ms*, *offsets_channel_socket_timeout_ms*, *buffer_size=1048576*)

Create a Broker instance.

Parameters

- **id** (*int*) – The id number of this broker
- **host** (*str*) – The host address to which to connect. An IP address or a DNS name
- **port** (*int*) – The port on which to connect
- **handler** (*pykafka.handlers.Handler*) – A Handler instance that will be used to service requests and responses
- **socket_timeout_ms** (*int*) – The socket timeout for network requests
- **offsets_channel_socket_timeout_ms** (*int*) – The socket timeout for network requests on the offsets channel
- **buffer_size** (*int*) – The size (bytes) of the internal buffer used to receive network responses

commit_consumer_group_offsets(*consumer_group*, *consumer_group_generation_id*, *consumer_id*, *preqs*)

Commit offsets to Kafka using the Offset Commit/Fetch API

Commit the offsets of all messages consumed so far by this consumer group with the Offset Commit/Fetch API

Based on Step 2 here https://cwiki.apache.org/confluence/display/KAFKA/Committing+and+fetching+consumer+offsets+in+Kafka

Parameters

- **consumer_group** (*str*) – the name of the consumer group for which to commit offsets
- **consumer_group_generation_id** (*int*) – The generation ID for this consumer group
- **consumer_id** (*str*) – The identifier for this consumer group
- **preqs** (Iterable of *pykafka.protocol.PartitionOffsetCommitRequest*) – Requests indicating the partitions for which offsets should be committed

connect()

Establish a connection to the broker server.

Creates a new *pykafka.connection.BrokerConnection* and a new *pykafka.handlers.RequestHandler* for this broker

connect_offsets_channel()

Establish a connection to the Broker for the offsets channel

Creates a new *pykafka.connection.BrokerConnection* and a new *pykafka.handlers.RequestHandler* for this broker's offsets channel

connected

Returns True if this object's main connection to the Kafka broker is active

fetch_consumer_group_offsets (*consumer_group*, *preqs*)
Fetch the offsets stored in Kafka with the Offset Commit/Fetch API

Based on Step 2 here https://cwiki.apache.org/confluence/display/KAFKA/Committing+and+fetching+consumer+offsets+in+Kafka

Parameters

- **consumer_group** (*str*) – the name of the consumer group for which to fetch offsets
- **preqs** (Iterable of *pykafka.protocol.PartitionOffsetFetchRequest*) – Requests indicating the partitions for which offsets should be fetched

fetch_messages (*partition_requests*, *timeout=30000*, *min_bytes=1*)
Fetch messages from a set of partitions.

Parameters

- **partition_requests** (Iterable of *pykafka.protocol.PartitionFetchRequest*) – Requests of messages to fetch.
- **timeout** (*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 min_bytes
- **min_bytes** (*int*) – the minimum amount of data (in bytes) the server should return. If insufficient data is available the request will block for up to *timeout* milliseconds.

classmethod from_metadata (*metadata*, *handler*, *socket_timeout_ms*, *offsets_channel_socket_timeout_ms*, *buffer_size=65536*)
Create a Broker using BrokerMetadata

Parameters

- **metadata** (pykafka.protocol.BrokerMetadata.) – Metadata that describes the broker.
- **handler** (*pykafka.handlers.Handler*) – A Handler instance that will be used to service requests and responses
- **socket_timeout_ms** (*int*) – The socket timeout for network requests
- **offsets_channel_socket_timeout_ms** (*int*) – The socket timeout for network requests on the offsets channel
- **buffer_size** (*int*) – The size (bytes) of the internal buffer used to receive network responses

handler
The primary *pykafka.handlers.RequestHandler* for this broker

This handler handles all requests outside of the commit/fetch api

host
The host to which this broker is connected

id
The broker's ID within the Kafka cluster

offsets_channel_connected
Returns True if this object's offsets channel connection to the Kafka broker is active

offsets_channel_handler

The offset channel *pykafka.handlers.RequestHandler* for this broker

This handler handles all requests that use the commit/fetch api

port
The port where the broker is available

produce_messages(*produce_request*)
Produce messages to a set of partitions.

Parameters **produce_request** (*pykafka.protocol.ProduceRequest*) – a request object indicating the messages to produce

request_metadata(*topics=None*)
Request cluster metadata

Parameters **topics** (*Iterable of int*) – The topic ids for which to request metadata

request_offset_limits(*partition_requests*)
Request offset information for a set of topic/partitions

Parameters **partition_requests** (Iterable of *pykafka.protocol.PartitionOffsetRequest*) – requests specifying the partitions for which to fetch offsets

pykafka.client

Author: Keith Bourgoin, Emmett Butler

class pykafka.client.**KafkaClient**(*hosts='127.0.0.1:9092'*, *use_greenlets=False*, *socket_timeout_ms=30000*, *offsets_channel_socket_timeout_ms=10000*, *ignore_rdkafka=False*, *exclude_internal_topics=True*)

Bases: object

A high-level pythonic client for Kafka

__init__(*hosts='127.0.0.1:9092'*, *use_greenlets=False*, *socket_timeout_ms=30000*, *offsets_channel_socket_timeout_ms=10000*, *ignore_rdkafka=False*, *exclude_internal_topics=True*)
Create a connection to a Kafka cluster.

Parameters

- **hosts** (*str*) – Comma-separated list of kafka hosts to used to connect.
- **use_greenlets** (*bool*) – If True, use gevent instead of threading.
- **socket_timeout_ms** (*int*) – The socket timeout (in milliseconds) for network requests
- **offsets_channel_socket_timeout_ms** (*int*) – The socket timeout (in milliseconds) when reading responses for offset commit and offset fetch requests.
- **ignore_rdkafka** (*bool*) – Don't use rdkafka, even if installed.
- **exclude_internal_topics** (*bool*) – Whether messages from internal topics (specifically, the offsets topic) should be exposed to the consumer.

__weakref__
list of weak references to the object (if defined)

update_cluster()
Update known brokers and topics.

Updates each Topic and Broker, adding new ones as found, with current metadata from the cluster.

pykafka.cluster

class pykafka.cluster.**Cluster**(*hosts*, *handler*, *socket_timeout_ms=30000*, *offsets_channel_socket_timeout_ms=10000*, *exclude_internal_topics=True*)

Bases: object

A Cluster is a high-level abstraction of the collection of brokers and topics that makes up a real kafka cluster.

__init__(*hosts*, *handler*, *socket_timeout_ms=30000*, *offsets_channel_socket_timeout_ms=10000*, *exclude_internal_topics=True*)

Create a new Cluster instance.

Parameters

- **hosts** (*str*) – Comma-separated list of kafka hosts to used to connect.
- **handler** (*pykafka.handlers.Handler*) – The concurrency handler for network requests.
- **socket_timeout_ms** (*int*) – The socket timeout (in milliseconds) for network requests
- **offsets_channel_socket_timeout_ms** (*int*) – The socket timeout (in milliseconds) when reading responses for offset commit and offset fetch requests.
- **exclude_internal_topics** (*bool*) – Whether messages from internal topics (specifically, the offsets topic) should be exposed to consumers.

__weakref__

list of weak references to the object (if defined)

_get_metadata()

Get fresh cluster metadata from a broker.

_should_exclude_topic(*topic_name*)

Should this topic be excluded from the list shown to the client?

_update_brokers(*broker_metadata*)

Update brokers with fresh metadata.

Parameters **broker_metadata** (Dict of *{name: metadata}* where *metadata* is pykafka.protocol.BrokerMetadata and *name* is str.) – Metadata for all brokers.

_update_topics(*metadata*)

Update topics with fresh metadata.

Parameters **metadata** (Dict of *{name, metadata}* where *metadata* is pykafka.protocol.TopicMetadata and *name* is str.) – Metadata for all topics.

brokers

The dict of known brokers for this cluster

get_offset_manager(*consumer_group*)

Get the broker designated as the offset manager for this consumer group.

Based on Step 1 at https://cwiki.apache.org/confluence/display/KAFKA/Committing+and+fetching+consumer+offsets+in+Kafka

Parameters **consumer_group** (*str*) – The name of the consumer group for which to find the offset manager.

handler
The concurrency handler for network requests

topics
The dict of known topics for this cluster

update()
Update known brokers and topics.

pykafka.common

Author: Keith Bourgoin

class pykafka.common.**Message**
Bases: object

Message class.

Variables

- **response_code** – Response code from Kafka
- ***topic*** – Originating topic
- **payload** – Message payload
- **key** – (optional) Message key
- **offset** – Message offset

__weakref__
list of weak references to the object (if defined)

class pykafka.common.**CompressionType**
Bases: object

Enum for the various compressions supported.

Variables

- **NONE** – Indicates no compression in use
- **GZIP** – Indicates gzip compression in use
- **SNAPPY** – Indicates snappy compression in use

__weakref__
list of weak references to the object (if defined)

class pykafka.common.**OffsetType**
Bases: object

Enum for special values for earliest/latest offsets.

Variables

- **EARLIEST** – Indicates the earliest offset available for a partition
- **LATEST** – Indicates the latest offset available for a partition

__weakref__
list of weak references to the object (if defined)

pykafka.connection

class pykafka.connection.**BrokerConnection**(*host*, *port*, *buffer_size=1048576*)
Bases: object

BrokerConnection thinly wraps a *socket.create_connection* call and handles the sending and receiving of data that conform to the kafka binary protocol over that socket.

__del__()
Close this connection when the object is deleted.

__init__(*host*, *port*, *buffer_size=1048576*)
Initialize a socket connection to Kafka.

Parameters

- **host** (*str*) – The host to which to connect
- **port** (*int*) – The port on the host to which to connect
- **buffer_size** (*int*) – The size (in bytes) of the buffer in which to hold response data.

__weakref__
list of weak references to the object (if defined)

connect(*timeout*)
Connect to the broker.

connected
Returns true if the socket connection is open.

disconnect()
Disconnect from the broker.

reconnect()
Disconnect from the broker, then reconnect

request(*request*)
Send a request over the socket connection

response()
Wait for a response from the broker

pykafka.exceptions

Author: Keith Bourgoin, Emmett Butler

exception pykafka.exceptions.**ConsumerCoordinatorNotAvailable**
Bases: *pykafka.exceptions.ProtocolClientError*

The broker returns this error code for consumer metadata requests or offset commit requests if the offsets topic has not yet been created.

exception pykafka.exceptions.**ConsumerStoppedException**
Bases: *pykafka.exceptions.KafkaException*

Indicates that the consumer was stopped when an operation was attempted that required it to be running

exception pykafka.exceptions.**InvalidMessageError**
Bases: *pykafka.exceptions.ProtocolClientError*

This indicates that a message contents does not match its CRC

exception pykafka.exceptions.**InvalidMessageSize**
Bases: *pykafka.exceptions.ProtocolClientError*

The message has a negative size

exception pykafka.exceptions.**KafkaException**
Bases: exceptions.Exception

Generic exception type. The base of all pykafka exception types.

__weakref__
list of weak references to the object (if defined)

exception pykafka.exceptions.**LeaderNotAvailable**
Bases: *pykafka.exceptions.ProtocolClientError*

This error is thrown if we are in the middle of a leadership election and there is currently no leader for this partition and hence it is unavailable for writes.

exception pykafka.exceptions.**MessageSizeTooLarge**
Bases: *pykafka.exceptions.ProtocolClientError*

The server has a configurable maximum message size to avoid unbounded memory allocation. This error is thrown if the client attempts to produce a message larger than this maximum.

exception pykafka.exceptions.**NoMessagesConsumedError**
Bases: *pykafka.exceptions.KafkaException*

Indicates that no messages were returned from a MessageSet

exception pykafka.exceptions.**NotCoordinatorForConsumer**
Bases: *pykafka.exceptions.ProtocolClientError*

The broker returns this error code if it receives an offset fetch or commit request for a consumer group that it is not a coordinator for.

exception pykafka.exceptions.**NotLeaderForPartition**
Bases: *pykafka.exceptions.ProtocolClientError*

This error is thrown if the client attempts to send messages to a replica that is not the leader for some partition. It indicates that the client's metadata is out of date.

exception pykafka.exceptions.**OffsetMetadataTooLarge**
Bases: *pykafka.exceptions.ProtocolClientError*

If you specify a string larger than configured maximum for offset metadata

exception pykafka.exceptions.**OffsetOutOfRangeError**
Bases: *pykafka.exceptions.ProtocolClientError*

The requested offset is outside the range of offsets maintained by the server for the given topic/partition.

exception pykafka.exceptions.**OffsetRequestFailedError**
Bases: *pykafka.exceptions.KafkaException*

Indicates that OffsetRequests for offset resetting failed more times than the configured maximum

exception pykafka.exceptions.**OffsetsLoadInProgress**
Bases: *pykafka.exceptions.ProtocolClientError*

The broker returns this error code for an offset fetch request if it is still loading offsets (after a leader change for that offsets topic partition).

exception pykafka.exceptions.**PartitionOwnedError**(*partition*, **args*, ***kwargs*)
Bases: *pykafka.exceptions.KafkaException*

Indicates a given partition is still owned in Zookeeper.

exception pykafka.exceptions.**ProduceFailureError**
Bases: *pykafka.exceptions.KafkaException*

Indicates a generic failure in the producer

exception pykafka.exceptions.**ProtocolClientError**
Bases: *pykafka.exceptions.KafkaException*

Base class for protocol errors

exception pykafka.exceptions.**RequestTimedOut**
Bases: *pykafka.exceptions.ProtocolClientError*

This error is thrown if the request exceeds the user-specified time limit in the request.

exception pykafka.exceptions.**SocketDisconnectedError**
Bases: *pykafka.exceptions.KafkaException*

Indicates that the socket connecting this client to a kafka broker has become disconnected

exception pykafka.exceptions.**UnknownError**
Bases: *pykafka.exceptions.ProtocolClientError*

An unexpected server erro

exception pykafka.exceptions.**UnknownTopicOrPartition**
Bases: *pykafka.exceptions.ProtocolClientError*

This request is for a topic or partition that does not exist on this broker.

pykafka.handlers

Author: Keith Bourgoin, Emmett Butler

class pykafka.handlers.**ResponseFuture**(*handler*)
Bases: object

A response which may have a value at some point.

__init__(*handler*)

__weakref__
list of weak references to the object (if defined)

get(*response_cls=None*, *timeout=None*)
Block until data is ready and return.

Raises an exception if there was an error.

set_error(*error*)
Set error and trigger get method.

set_response(*response*)
Set response data and trigger get method.

class pykafka.handlers.**Handler**
Bases: object

Base class for Handler classes

__weakref__
list of weak references to the object (if defined)

spawn(*target*, **args*, ***kwargs*)
Create the worker that will process the work to be handled

class pykafka.handlers.**ThreadingHandler**
Bases: *pykafka.handlers.Handler*

A handler. that uses a `threading.Thread` to perform its work

Event(**args*, ***kwargs*)
A factory function that returns a new event.

Events manage a flag that can be set to true with the set() method and reset to false with the clear() method. The wait() method blocks until the flag is true.

Lock()
allocate_lock() -> lock object (allocate() is an obsolete synonym)

Create a new lock object. See help(LockType) for information about locks.

class Queue(*maxsize=0*)
Create a queue object with a given maximum size.

If maxsize is <= 0, the queue size is infinite.

empty()
Return True if the queue is empty, False otherwise (not reliable!).

full()
Return True if the queue is full, False otherwise (not reliable!).

get(*block=True*, *timeout=None*)
Remove and return an item from the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until an item is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Empty exception if no item was available within that time. Otherwise ('block' is false), return an item if one is immediately available, else raise the Empty exception ('timeout' is ignored in that case).

get_nowait()
Remove and return an item from the queue without blocking.

Only get an item if one is immediately available. Otherwise raise the Empty exception.

join()
Blocks until all items in the Queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls task_done() to indicate the item was retrieved and all work on it is complete.

When the count of unfinished tasks drops to zero, join() unblocks.

put(*item*, *block=True*, *timeout=None*)
Put an item into the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until a free slot is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Full exception if no free slot was available within that time. Otherwise ('block' is false), put an item on the queue if a free slot is immediately available, else raise the Full exception ('timeout' is ignored in that case).

put_nowait(*item*)
Put an item into the queue without blocking.

Only enqueue the item if a free slot is immediately available. Otherwise raise the Full exception.

qsize()
Return the approximate size of the queue (not reliable!).

task_done()
Indicate that a formerly enqueued task is complete.

Used by Queue consumer threads. For each get() used to fetch a task, a subsequent call to task_done() tells the queue that the processing on the task is complete.

If a join() is currently blocking, it will resume when all items have been processed (meaning that a task_done() call was received for every item that had been put() into the queue).

Raises a ValueError if called more times than there were items placed in the queue.

ThreadingHandler.**QueueEmptyError**
alias of Empty

class pykafka.handlers.**RequestHandler**(*handler*, *connection*)
Bases: object

Uses a Handler instance to dispatch requests.

class Task(*request*, *future*)
Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

static __new__(*_cls*, *request*, *future*)
Create new instance of Task(request, future)

__repr__()
Return a nicely formatted representation string

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x7b4ba0>*, *len=<built-in function len>*)
Make a new Task object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new Task object replacing specified fields with new values

future
Alias for field number 1

request
Alias for field number 0

RequestHandler.**__init__**(*handler*, *connection*)

RequestHandler.**__weakref__**
list of weak references to the object (if defined)

RequestHandler.**_start_thread**()
Run the request processor

RequestHandler.**request**(*request*, *has_response=True*)
Construct a new request

Parameters **has_response** – Whether this request will return a response

Returns *pykafka.handlers.ResponseFuture*

RequestHandler.**start**()
Start the request processor.

RequestHandler.**stop**()
Stop the request processor.

pykafka.partition

Author: Keith Bourgoin, Emmett Butler

class pykafka.partition.**Partition**(*topic*, *id_*, *leader*, *replicas*, *isr*)
A Partition is an abstraction over the kafka concept of a partition. A kafka partition is a logical division of the logs for a topic. Its messages are totally ordered.

__init__(*topic*, *id_*, *leader*, *replicas*, *isr*)
Instantiate a new Partition

Parameters

- **topic** (*pykafka.topic.Topic*) – The topic to which this Partition belongs
- **id** (*int*) – The identifier for this partition
- **leader** (*pykafka.broker.Broker*) – The broker that is currently acting as the leader for this partition.
- **replicas** (Iterable of *pykafka.broker.Broker*) – A list of brokers containing this partition's replicas
- **isr** (*pykafka.broker.Broker*) – The current set of in-sync replicas for this partition

earliest_available_offset()
Get the earliest offset for this partition.

fetch_offset_limit(*offsets_before*, *max_offsets=1*)

Use the Offset API to find a limit of valid offsets for this partition.

Parameters

- **offsets_before** (*int*) – Return an offset from before this timestamp (in milliseconds)
- **max_offsets** (*int*) – The maximum number of offsets to return

id
The identifying int for this partition, unique within its topic

isr
The current list of in-sync replicas for this partition

latest_available_offset()
Get the latest offset for this partition.

leader
The broker currently acting as leader for this partition

replicas
The list of brokers currently holding replicas of this partition

topic
The topic to which this partition belongs

update(*brokers*, *metadata*)
Update this partition with fresh metadata.

Parameters

- **brokers** (List of *pykafka.broker.Broker*) – Brokers on which partitions exist
- **metadata** (pykafka.protocol.PartitionMetadata) – Metadata for the partition

pykafka.partitioners

Author: Keith Bourgoin, Emmett Butler

pykafka.partitioners.**random_partitioner**(*partitions*, *key*)
Returns a random partition out of all of the available partitions.

class pykafka.partitioners.**BasePartitioner**
Bases: object

Base class for custom class-based partitioners.

A partitioner is used by the *pykafka.producer.Producer* to decide which partition to which to produce messages.

__weakref__
list of weak references to the object (if defined)

class pykafka.partitioners.**HashingPartitioner**(*hash_func=<built-in function hash>*)
Bases: *pykafka.partitioners.BasePartitioner*

Returns a (relatively) consistent partition out of all available partitions based on the key.

Messages that are published with the same keys are not guaranteed to end up on the same broker if the number of brokers changes (due to the addition or removal of a broker, planned or unplanned) or if the number of topics per partition changes. This is also unreliable when not all brokers are aware of a topic, since the number of available partitions will be in flux until all brokers have accepted a write to that topic and have declared how many partitions that they are actually serving.

__call__(*partitions*, *key*)

Parameters

- **partitions** (sequence of pykafka.base.BasePartition) – The partitions from which to choose
- **key** (Any hashable type if using the default hash() implementation, any valid value for your custom hash function) – Key used for routing

Returns A partition

Return type pykafka.base.BasePartition

__init__ (*hash_func=<built-in function hash>*)

Parameters **hash_func** (*function*) – hash function (defaults to `hash()`), should return an *int*. If hash randomization (Python 2.7) is enabled, a custom hashing function should be defined that is consistent between interpreter restarts.

pykafka.producer

class `pykafka.producer.`**Producer**(*cluster*, *topic*, *partitioner=<function random_partitioner>*, *compression=0*, *max_retries=3*, *retry_backoff_ms=100*, *required_acks=1*, *ack_timeout_ms=10000*, *batch_size=200*)

This class implements the synchronous producer logic found in the JVM driver.

__init__(*cluster*, *topic*, *partitioner=<function random_partitioner>*, *compression=0*, *max_retries=3*, *retry_backoff_ms=100*, *required_acks=1*, *ack_timeout_ms=10000*, *batch_size=200*)

Instantiate a new Producer.

Parameters

- **cluster** (*pykafka.cluster.Cluster*) – The cluster to which to connect
- **topic** (*pykafka.topic.Topic*) – The topic to which to produce messages
- **partitioner** (*pykafka.partitioners.BasePartitioner*) – The partitioner to use during message production
- **compression** (*pykafka.common.CompressionType*) – The type of compression to use.
- **max_retries** (*int*) – How many times to attempt to produce messages before raising an error.
- **retry_backoff_ms** (*int*) – The amount of time (in milliseconds) to back off during produce request retries.
- **required_acks** (*int*) – How many other brokers must have committed the data to their log and acknowledged this to the leader before a request is considered complete?
- **ack_timeout_ms** (*int*) – Amount of time (in milliseconds) to wait for acknowledgment of a produce request.
- **batch_size** (*int*) – Size (in bytes) of batches to send to brokers.

_partition_messages(*messages*)

Assign messages to partitions using the partitioner.

Parameters **messages** – Iterable of messages to publish.

Returns Generator of ((key, value), partition_id)

_produce(*message_partition_tups*, *attempt*)

Publish a set of messages to relevant brokers.

Parameters **message_partition_tups** (*tuples of ((key, value), partition_id)*) – Messages with partitions assigned.

_send_request(*broker*, *req*, *attempt*)

Send the produce request to the broker and handle the response.

Parameters

- **broker** (*pykafka.broker.Broker*) – The broker to which to send the request
- **req** (*pykafka.protocol.ProduceRequest*) – The produce request to send

- **attempt** (*int*) – The current attempt count. Used for retry logic

produce(*messages*)
Produce a set of messages.

Parameters **messages** (*Iterable of str or (str, str) tuples*) – The messages to produce

pykafka.protocol

class pykafka.protocol.**MetadataRequest**(*topics=None*)
Bases: *pykafka.protocol.Request*

Metadata Request

MetadataRequest => [TopicName] TopicName => string

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*topics=None*)
Create a new MetadatRequest

Parameters **topics** – Topics to query. Leave empty for all available topics.

__len__()
Length of the serialized message, in bytes

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**MetadataResponse**(*buff*)
Bases: *pykafka.protocol.Response*

Response from MetadataRequest

MetadataResponse => [Broker][TopicMetadata] Broker => NodeId Host Port NodeId => int32 Host => string Port => int32 TopicMetadata => TopicErrorCode TopicName [PartitionMetadata] TopicErrorCode => int16 PartitionMetadata => PartitionErrorCode PartitionId Leader Replicas Isr PartitionErrorCode => int16 PartitionId => int32 Leader => int32 Replicas => [int32] Isr => [int32]

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**ProduceRequest**(*compression_type=0*, *required_acks=1*, *timeout=10000*)
Bases: *pykafka.protocol.Request*

Produce Request

ProduceRequest => RequiredAcks Timeout [TopicName [Partition MessageSetSize MessageSet]] RequiredAcks => int16 Timeout => int32 Partition => int32 MessageSetSize => int32

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*compression_type=0*, *required_acks=1*, *timeout=10000*)
Create a new ProduceRequest

required_acks determines how many acknowledgement the server waits for before returning. This is useful for ensuring the replication factor of published messages. The behavior is:

-1: Block until all servers acknowledge 0: No waiting – server doesn't even respond to the Produce request 1: Wait for this server to write to the local log and then return 2+: Wait for N servers to acknowledge

Parameters

- **partition_requests** – Iterable of kafka.pykafka.protocol.PartitionProduceRequest for this request
- **compression_type** – Compression to use for messages
- **required_acks** – see docstring
- **timeout** – timeout (in ms) to wait for the required acks

__len__()
Length of the serialized message, in bytes

add_message(*message*, *topic_name*, *partition_id*)
Add a list of kafka.common.Message to the waiting request

Parameters

- **messages** – an iterable of kafka.common.Message to add
- **topic_name** – the name of the topic to publish to
- **partition_id** – the partition to publish to

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

message_count()
Get the number of messages across all MessageSets in the request.

messages
Iterable of all messages in the Request

class pykafka.protocol.**ProduceResponse**(*buff*)
Bases: *pykafka.protocol.Response*

Produce Response. Checks to make sure everything went okay.

ProduceResponse => [TopicName [Partition ErrorCode Offset]] TopicName => string Partition => int32 ErrorCode => int16 Offset => int64

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**OffsetRequest**(*partition_requests*)
Bases: *pykafka.protocol.Request*

An offset request

OffsetRequest => ReplicaId [TopicName [Partition Time MaxNumberOfOffsets]] ReplicaId => int32 TopicName => string Partition => int32 Time => int64 MaxNumberOfOffsets => int32

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*partition_requests*)
Create a new offset request

__len__()
Length of the serialized message, in bytes

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**OffsetResponse**(*buff*)
Bases: *pykafka.protocol.Response*

An offset response

OffsetResponse => [TopicName [PartitionOffsets]] PartitionOffsets => Partition ErrorCode [Offset] Partition => int32 ErrorCode => int16 Offset => int64

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**OffsetCommitRequest**(*consumer_group*, *consumer_group_generation_id*, *consumer_id*, *partition_requests=[]*)
Bases: *pykafka.protocol.Request*

An offset commit request

OffsetCommitRequest => ConsumerGroupId ConsumerGroupGenerationId ConsumerId [TopicName [Partition Offset T
ConsumerGroupId => string ConsumerGroupGenerationId => int32 ConsumerId => string TopicName => string Partition => int32 Offset => int64 TimeStamp => int64 Metadata => string

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*consumer_group*, *consumer_group_generation_id*, *consumer_id*, *partition_requests=[]*)
Create a new offset commit request

Parameters **partition_requests** – Iterable of kafka.pykafka.protocol.PartitionOffsetCommitRequest for this request

__len__()
Length of the serialized message, in bytes

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**FetchRequest**(*partition_requests=[]*, *timeout=1000*, *min_bytes=1024*)
Bases: *pykafka.protocol.Request*

A Fetch request sent to Kafka

FetchRequest => ReplicaId MaxWaitTime MinBytes [TopicName [Partition FetchOffset MaxBytes]] ReplicaId => int32 MaxWaitTime => int32 MinBytes => int32 TopicName => string Partition => int32 FetchOffset => int64 MaxBytes => int32

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*partition_requests=[]*, *timeout=1000*, *min_bytes=1024*)
Create a new fetch request

Kafka 0.8 uses long polling for fetch requests, which is different from 0.7x. Instead of polling and waiting, we can now set a timeout to wait and a minimum number of bytes to be collected before it returns. This way we can block effectively and also ensure good network throughput by having fewer, large transfers instead of many small ones every time a byte is written to the log.

Parameters

- **partition_requests** – Iterable of `kafka.pykafka..protocol.PartitionFetchRequest` for this request
- **timeout** – Max time to wait (in ms) for a response from the server
- **min_bytes** – Minimum bytes to collect before returning

__len__()
Length of the serialized message, in bytes

add_request(*partition_request*)
Add a topic/partition/offset to the requests

Parameters

- **topic_name** – The topic to fetch from
- **partition_id** – The partition to fetch from
- **offset** – The offset to start reading data from
- **max_bytes** – The maximum number of bytes to return in the response

get_bytes()
Serialize the message

Returns Serialized message

Return type `bytearray`

class `pykafka.protocol.`**FetchResponse**(*buff*)
Bases: *pykafka.protocol.Response*

Unpack a fetch response from the server

FetchResponse => [TopicName [Partition ErrorCode HighwaterMarkOffset MessageSetSize MessageSet]] TopicName => string Partition => int32 ErrorCode => int16 HighwaterMarkOffset => int64 MessageSetSize => int32

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (`bytearray`) – Serialized message

_unpack_message_set(*buff*, *partition_id=-1*)
MessageSets can be nested. Get just the Messages out of it.

class pykafka.protocol.**PartitionFetchRequest**
Bases: *pykafka.protocol.PartitionFetchRequest*

Fetch request for a specific topic/partition

Variables

- **topic_name** – Name of the topic to fetch from
- **partition_id** – Id of the partition to fetch from
- **offset** – Offset at which to start reading
- **max_bytes** – Max bytes to read from this partition (default: 300kb)

class pykafka.protocol.**OffsetCommitResponse**(*buff*)
Bases: *pykafka.protocol.Response*

An offset commit response

OffsetCommitResponse => [TopicName [Partition ErrorCode]]] TopicName => string Partition => int32 ErrorCode => int16

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**OffsetFetchRequest**(*consumer_group*, *partition_requests=[]*)
Bases: *pykafka.protocol.Request*

An offset fetch request

OffsetFetchRequest => ConsumerGroup [TopicName [Partition]] ConsumerGroup => string TopicName => string Partition => int32

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*consumer_group*, *partition_requests=[]*)
Create a new offset fetch request

Parameters **partition_requests** – Iterable of kafka.pykafka.protocol.PartitionOffsetFetchRequest for this request

__len__()
Length of the serialized message, in bytes

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**OffsetFetchResponse**(*buff*)
Bases: *pykafka.protocol.Response*

An offset fetch response

OffsetFetchResponse => [TopicName [Partition Offset Metadata ErrorCode]] TopicName => string Partition => int32 Offset => int64 Metadata => string ErrorCode => int16

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**PartitionOffsetRequest**
Bases: *pykafka.protocol.PartitionOffsetRequest*

Offset request for a specific topic/partition

Variables

- **topic_name** – Name of the topic to look up
- **partition_id** – Id of the partition to look up
- **offsets_before** – Retrieve offset information for messages before this timestamp (ms). -1 will retrieve the latest offsets and -2 will retrieve the earliest available offset. If -2,only 1 offset is returned
- **max_offsets** – How many offsets to return

class pykafka.protocol.**ConsumerMetadataRequest**(*consumer_group*)
Bases: *pykafka.protocol.Request*

A consumer metadata request

ConsumerMetadataRequest => ConsumerGroup ConsumerGroup => string

API_KEY
API_KEY for this request, from the Kafka docs

__init__(*consumer_group*)
Create a new consumer metadata request

__len__()
Length of the serialized message, in bytes

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**ConsumerMetadataResponse**(*buff*)
Bases: *pykafka.protocol.Response*

A consumer metadata response

ConsumerMetadataResponse => ErrorCode CoordinatorId CoordinatorHost CoordinatorPort
ErrorCode => int16 CoordinatorId => int32 CoordinatorHost => string CoordinatorPort => int32

__init__(*buff*)
Deserialize into a new Response

Parameters **buff** (bytearray) – Serialized message

class pykafka.protocol.**PartitionOffsetCommitRequest**
Bases: *pykafka.protocol.PartitionOffsetCommitRequest*

Offset commit request for a specific topic/partition

Variables

- **topic_name** – Name of the topic to look up
- **partition_id** – Id of the partition to look up
- **offset** –
- **timestamp** –

- **metadata** – arbitrary metadata that should be committed with this offset commit

class pykafka.protocol.**PartitionOffsetFetchRequest**
Bases: *pykafka.protocol.PartitionOffsetFetchRequest*

Offset fetch request for a specific topic/partition

Variables

- **topic_name** – Name of the topic to look up
- **partition_id** – Id of the partition to look up

class pykafka.protocol.**Request**
Bases: pykafka.utils.Serializable

Base class for all Requests. Handles writing header information

API_KEY()
API key for this request, from the Kafka docs

_write_header(*buff*, *api_version=0*, *correlation_id=0*)
Write the header for an outgoing message.

Parameters

- **buff** (*buffer*) – The buffer into which to write the header
- **api_version** (*int*) – The "kafka api version id", used for feature flagging
- **correlation_id** (*int*) – This is a user-supplied integer. It will be passed back in the response by the server, unmodified. It is useful for matching request and response between the client and server.

get_bytes()
Serialize the message

Returns Serialized message

Return type bytearray

class pykafka.protocol.**Response**
Bases: object

Base class for Response objects.

__weakref__
list of weak references to the object (if defined)

raise_error(*err_code*, *response*)
Raise an error based on the Kafka error code

Parameters

- **err_code** – The error code from Kafka
- **response** – The unpacked raw data from the response

class pykafka.protocol.**Message**(*value*, *partition_key=None*, *compression_type=0*, *offset=-1*, *partition_id=-1*)
Bases: *pykafka.common.Message*, pykafka.utils.Serializable

Representation of a Kafka Message

NOTE: Compression is handled in the protocol because of the way Kafka embeds compressed MessageSets within Messages

Message => Crc MagicByte Attributes Key Value Crc => int32 MagicByte => int8 Attributes => int8 Key => bytes Value => bytes

pykafka.protocol.Message also contains *partition* and *partition_id* fields. Both of these have meaningless default values. When *pykafka.protocol.Message* is used by the producer. When used in a *pykafka.protocol.FetchRequest*, *partition_id* is set to the id of the partition from which the message was sent on receipt of the message. In the *pykafka.simpleconsumer.SimpleConsumer*, *partition* is set to the *pykafka.partition.Partition* instance from which the message was sent.

pack_into(*buff*, *offset*)
Serialize and write to `buff` starting at offset `offset`.

Intentionally follows the pattern of `struct.pack_into`

Parameters

- **buff** – The buffer to write into
- **offset** – The offset to start the write at

class pykafka.protocol.**MessageSet**(*compression_type=0*, *messages=None*)
Bases: `pykafka.utils.Serializable`

Representation of a set of messages in Kafka

This isn't useful outside of direct communications with Kafka, so we keep it hidden away here.

N.B.: MessageSets are not preceded by an int32 like other array elements in the protocol.

MessageSet => [Offset MessageSize Message] Offset => int64 MessageSize => int32

Variables

- ***messages*** – The list of messages currently in the MessageSet
- **compression_type** – compression to use for the messages

__init__(*compression_type=0*, *messages=None*)
Create a new MessageSet

Parameters

- **compression_type** – Compression to use on the messages
- **messages** – An initial list of messages for the set

__len__()
Length of the serialized message, in bytes

We don't put the MessageSetSize in front of the serialization because that's *technically* not part of the MessageSet. Most requests/responses using MessageSets need that size, though, so be careful when using this.

_get_compressed()
Get a compressed representation of all current messages.

Returns a Message object with correct headers set and compressed data in the value field.

classmethod decode(*buff*, *partition_id=-1*)
Decode a serialized MessageSet.

pack_into(*buff*, *offset*)
Serialize and write to `buff` starting at offset `offset`.

Intentionally follows the pattern of `struct.pack_into`

Parameters

- **buff** – The buffer to write into
- **offset** – The offset to start the write at

pykafka.simpleconsumer

class pykafka.simpleconsumer.**SimpleConsumer**(*topic*, *cluster*, *consumer_group=None*, *partitions=None*, *fetch_message_max_bytes=1048576*, *num_consumer_fetchers=1*, *auto_commit_enable=False*, *auto_commit_interval_ms=60000*, *queued_max_messages=2000*, *fetch_min_bytes=1*, *fetch_wait_max_ms=100*, *offsets_channel_backoff_ms=1000*, *offsets_commit_max_retries=5*, *auto_offset_reset=-1*, *consumer_timeout_ms=-1*, *auto_start=True*, *reset_offset_on_start=False*)

A non-balancing consumer for Kafka

__del__()

Stop consumption and workers when object is deleted

__init__(*topic*, *cluster*, *consumer_group=None*, *partitions=None*, *fetch_message_max_bytes=1048576*, *num_consumer_fetchers=1*, *auto_commit_enable=False*, *auto_commit_interval_ms=60000*, *queued_max_messages=2000*, *fetch_min_bytes=1*, *fetch_wait_max_ms=100*, *offsets_channel_backoff_ms=1000*, *offsets_commit_max_retries=5*, *auto_offset_reset=-1*, *consumer_timeout_ms=-1*, *auto_start=True*, *reset_offset_on_start=False*)

Create a SimpleConsumer.

Settings and default values are taken from the Scala consumer implementation. Consumer group is included because it's necessary for offset management, but doesn't imply that this is a balancing consumer. Use a BalancedConsumer for that.

Parameters

- **topic** (*pykafka.topic.Topic*) – The topic this consumer should consume
- **cluster** (*pykafka.cluster.Cluster*) – The cluster to which this consumer should connect
- **consumer_group** (*str*) – The name of the consumer group this consumer should use for offset committing and fetching.
- **partitions** (Iterable of *pykafka.partition.Partition*) – Existing partitions to which to connect
- **fetch_message_max_bytes** (*int*) – The number of bytes of messages to attempt to fetch
- **num_consumer_fetchers** (*int*) – The number of workers used to make FetchRequests
- **auto_commit_enable** (*bool*) – If true, periodically commit to kafka the offset of messages already fetched by this consumer. This also requires that *consumer_group* is not *None*.

- **auto_commit_interval_ms** (*int*) – The frequency (in milliseconds) at which the consumer offsets are committed to kafka. This setting is ignored if *auto_commit_enable* is *False*.
- **queued_max_messages** (*int*) – Maximum number of messages buffered for consumption
- **fetch_min_bytes** (*int*) – The minimum amount of data (in bytes) the server should return for a fetch request. If insufficient data is available the request will block until sufficient data is available.
- **fetch_wait_max_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 *fetch_min_bytes*.
- **offsets_channel_backoff_ms** (*int*) – Backoff time (in milliseconds) to retry offset commits/fetches
- **offsets_commit_max_retries** (*int*) – Retry the offset commit up to this many times on failure.
- **auto_offset_reset** (*pykafka.common.OffsetType*) – What to do if an offset is out of range. This setting indicates how to reset the consumer's internal offset counter when an *OffsetOutOfRangeError* is encountered.
- **consumer_timeout_ms** (*int*) – Amount of time (in milliseconds) the consumer may spend without messages available for consumption before returning None.
- **auto_start** (*bool*) – Whether the consumer should begin communicating with kafka after __init__ is complete. If false, communication can be started with *start()*.
- **reset_offset_on_start** (*bool*) – Whether the consumer should reset its internal offset counter to *self._auto_offset_reset* and commit that offset immediately upon starting up

__iter__()
Yield an infinite stream of messages until the consumer times out

_auto_commit()
Commit offsets only if it's time to do so

_build_default_error_handlers()
Set up the error handlers to use for partition errors.

_discover_offset_manager()
Set the offset manager for this consumer.

If a consumer group is not supplied to __init__, this method does nothing

_setup_autocommit_worker()
Start the autocommitter thread

_setup_fetch_workers()
Start the fetcher threads

commit_offsets()
Commit offsets for this consumer's partitions

Uses the offset commit/fetch API

consume(*block=True*)
Get one message from the consumer.

Parameters **block** (*bool*) – Whether to block while waiting for a message

fetch()
Fetch new messages for all partitions

Create a FetchRequest for each broker and send it. Enqueue each of the returned messages in the approprate OwnedPartition.

fetch_offsets()
Fetch offsets for this consumer's topic

Uses the offset commit/fetch API

Returns List of (id, `pykafka.protocol.OffsetFetchPartitionResponse`) tuples

held_offsets
Return a map from partition id to held offset for each partition

partitions
A list of the partitions that this consumer consumes

reset_offsets(*partition_offsets=None*)
Reset offsets for the specified partitions

Issue an OffsetRequest for each partition and set the appropriate returned offset in the consumer's internal offset counter.

Parameters **partition_offsets** (Iterable of (*pykafka.partition.Partition*, int)) – (*partition*, *timestamp_or_offset*) pairs to reset where *partition* is the partition for which to reset the offset and *timestamp_or_offset* is EITHER the timestamp of the message whose offset the partition should have OR the new offset the partition should have

NOTE: If an instance of *timestamp_or_offset* is treated by kafka as an invalid offset timestamp, this function directly sets the consumer's internal offset counter for that partition to that instance of *timestamp_or_offset*. On the next fetch request, the consumer attempts to fetch messages starting from that offset. See the following link for more information on what kafka treats as a valid offset timestamp: https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol#AGuideToTheKafkaProtocol-OffsetRequest

start()
Begin communicating with Kafka, including setting up worker threads

Fetches offsets, starts an offset autocommitter worker pool, and starts a message fetcher worker pool.

stop()
Flag all running workers for deletion.

topic
The topic this consumer consumes

pykafka.topic

Author: Keith Bourgoin, Emmett Butler

class `pykafka.topic.`**Topic**(*cluster*, *topic_metadata*)
A Topic is an abstraction over the kafka concept of a topic. It contains a dictionary of partitions that comprise it.

__init__(*cluster*, *topic_metadata*)
Create the Topic from metadata.

Parameters

- **cluster** (*pykafka.cluster.Cluster*) – The Cluster to use

- **topic_metadata** (pykafka.protocol.TopicMetadata) – Metadata for all topics.

earliest_available_offsets()
Get the earliest offset for each partition of this topic.

fetch_offset_limits(*offsets_before*, *max_offsets=1*)
Get earliest or latest offset.

Use the Offset API to find a limit of valid offsets for each partition in this topic.

Parameters

- **offsets_before** (*int*) – Return an offset from before this timestamp (in milliseconds)
- **max_offsets** (*int*) – The maximum number of offsets to return

get_balanced_consumer(*consumer_group*, ***kwargs*)
Return a BalancedConsumer of this topic

Parameters **consumer_group** (*str*) – The name of the consumer group to join

get_producer(***kwargs*)
Create a *pykafka.producer.Producer* for this topic.

For a description of all available *kwargs*, see the Producer docstring.

get_simple_consumer(*consumer_group=None*, ***kwargs*)
Return a SimpleConsumer of this topic

Parameters **consumer_group** (*str*) – The name of the consumer group to join

latest_available_offsets()
Get the latest offset for each partition of this topic.

name
The name of this topic

partitions
A dictionary containing all known partitions for this topic

update(*metadata*)
Update the Partitions with metadata about the cluster.

Parameters **metadata** (pykafka.protocol.TopicMetadata) – Metadata for all topics

pykafka.utils.compression

Author: Keith Bourgoin

pykafka.utils.compression.**encode_gzip**(*buff*)
Encode a buffer using gzip

pykafka.utils.compression.**decode_gzip**(*buff*)
Decode a buffer using gzip

pykafka.utils.compression.**encode_snappy**(*buff*, *xerial_compatible=False*, *xerial_blocksize=32768*)
Encode a buffer using snappy

If xerial_compatible is set, the buffer is encoded in a fashion compatible with the xerial snappy library.

The block size (xerial_blocksize) controls how frequently the blocking occurs. 32k is the default in the xerial library.

The format is as follows: +————————+————————+————————+————————+————————+ | Header | Block1 len | Block1 data | Blockn len | Blockn data | |————————+————————+————————+————————+————————| | 16 bytes | BE int32 | snappy bytes | BE int32 | snappy bytes | +————————+————————+————————+————————+————————+

It is important to note that *blocksize* is the amount of uncompressed data presented to snappy at each block, whereas *blocklen* is the number of bytes that will be present in the stream.

Adapted from kafka-python https://github.com/mumrah/kafka-python/pull/127/files

pykafka.utils.compression.**decode_snappy**(*buff*)
Decode a buffer using Snappy

If xerial is found to be in use, the buffer is decoded in a fashion compatible with the xerial snappy library.

Adapted from kafka-python https://github.com/mumrah/kafka-python/pull/127/files

pykafka.utils.error_handlers

Author: Emmett Butler

pykafka.utils.error_handlers.**handle_partition_responses**(*error_handlers*, *parts_by_error=None*, *success_handler=None*, *response=None*, *partitions_by_id=None*)
Call the appropriate handler for each errored partition

Parameters

- **error_handlers** (*dict {int: callable(parts)}*) – mapping of error code to handler
- **parts_by_error** (dict {int: iterable(pykafka.simpleconsumer.OwnedPartition)}) – a dict of partitions grouped by error code
- **success_handler** (*callable accepting an iterable of partition responses*) – function to call for successful partitions
- **response** (*pykafka.protocol.Response*) – a Response object containing partition responses
- **partitions_by_id** (dict {int: pykafka.simpleconsumer.OwnedPartition}) – a dict mapping partition ids to OwnedPartition instances

pykafka.utils.error_handlers.**raise_error**(*error*, *info=''*)
Raise the given error

pykafka.utils.socket

Author: Keith Bourgoin, Emmett Butler

pykafka.utils.socket.**recvall_into**(*socket*, *bytea*, *size*)
Reads *size* bytes from the socket into the provided bytearray (modifies in-place.)

This is basically a hack around the fact that *socket.recv_into* doesn't allow buffer offsets.

Return type *bytearray*

pykafka.utils.struct_helpers

Author: Keith Bourgoin, Emmett Butler

pykafka.utils.struct_helpers.**unpack_from**(*fmt*, *buff*, *offset=0*)

A customized version of *struct.unpack_from*

This is a conveinence function that makes decoding the arrays, strings, and byte arrays that we get from Kafka significantly easier. It takes the same arguments as *struct.unpack_from* but adds 3 new formats:

- Wrap a section in *[]* to indicate an array. e.g.: *[ii]*
- *S* for strings (int16 followed by byte array)
- *Y* for byte arrays (int32 followed by byte array)

Spaces are ignored in the format string, allowing more readable formats

NOTE: This may be a performance bottleneck. We're avoiding a lot of memory allocations by using the same buffer, but if we could call *struct.unpack_from* only once, that's about an order of magnitude faster. However, constructing the format string to do so would erase any gains we got from having the single call.

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

J

K

L

M

N

O

P

Q

R

S

T

U