KafkaConsumer

class **kafka.KafkaConsumer**(**topics, **configs*) [source]

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: 305000.
- **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 backoff/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. Once the maximum is reached, reconnection attempts will continue periodically with this fixed rate. 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]
- **max_poll_records** (*int*) – The maximum number of records returned in a single call to `poll()`. Default: 500
- **max_poll_interval_ms** (*int*) – The maximum delay between

invocations of `poll()` when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If `poll()` is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member. Default 300000

- **session_timeout_ms** (*int*) – The timeout used to detect failures when using Kafka's group management facilities. The consumer sends periodic heartbeats to indicate its liveness to the broker. If no heartbeats are received by the broker before the expiration of this session timeout, then the broker will remove this consumer from the group and initiate a rebalance. Note that the value must be in the allowable range as configured in the broker configuration by group.min.session.timeout.ms and group.max.session.timeout.ms. Default: 10000
- **heartbeat_interval_ms** (*int*) – The expected time in milliseconds between heartbeats to the consumer coordinator when using Kafka's group management facilities. 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
- **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')].
- **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 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.
- **ssl_ciphers** (*str*) – optionally set the available ciphers for ssl connections. It should be a string in the OpenSSL cipher list format. If no cipher can be selected (because compile-time options or other configuration forbids use of all the specified ciphers), an ssl.SSLError will be raised. See ssl.SSLContext.set_ciphers
- **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.

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 None.
- **connections_max_idle_ms** – Close idle connections after the number of milliseconds specified by this config. The broker closes idle connections after connections.max.idle.ms, so this avoids hitting unexpected socket disconnected errors on the client. Default: 540000
- **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*) – Authentication mechanism when security_protocol is configured for SASL_PLAINTEXT or SASL_SSL. Valid values are: PLAIN, GSSAPI, OAUTHBEARER, SCRAM-SHA-256, SCRAM-SHA-512.
- **sasl_plain_username** (*str*) – username for sasl PLAIN and SCRAM authentication. Required if sasl_mechanism is PLAIN or one of

the SCRAM mechanisms.

- **sasl_plain_password** (*str*) – password for sasl PLAIN and SCRAM authentication. Required if sasl_mechanism is PLAIN or one of the SCRAM mechanisms.
- **sasl_kerberos_service_name** (*str*) – Service name to include in GSSAPI sasl mechanism handshake. Default: 'kafka'
- **sasl_kerberos_domain_name** (*str*) – kerberos domain name to use in GSSAPI sasl mechanism handshake. Default: one of bootstrap servers
- **sasl_oauth_token_provider** (*AbstractTokenProvider*) – OAuthBearer token provider instance. (See kafka.oauth.abstract). Default: None

Note

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

`assign`(*partitions*) [source]

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`() [source]

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*) [source]

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.

bootstrap_connected() [source]

Return True if the bootstrap is connected.

close(*autocommit=True*) [source]

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*) [source]

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*) [source]

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*, *metadata=False*) [source]

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.
- **metadata** (*bool*, *optional*) – If True, return OffsetAndMetadata struct instead of offset int. Default: False.

Returns:

The last committed offset (int or OffsetAndMetadata), or None if there was no prior commit.

`end_offsets`(*partitions*) [source]

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*) [source]

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*) [source]

Get metrics on consumer performance.

This is ported from the Java Consumer, for details see: https://kafka.apache.org/documentation/#consumer_monitoring

Warning

This is an unstable interface. It may change in future releases without warning.

`offsets_for_times`(*timestamps*) [source]

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*) [source]

This method first checks the local metadata cache for information about the topic. If the topic is not found (either because the topic does not exist, the user is not authorized to view the topic, or the metadata cache is not populated), then it will issue a metadata update call to the cluster.

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

Returns: Partition ids

Return type: set

`pause`(**partitions*) [source]

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`() [source]

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

Returns: {partition (TopicPartition), ...}

Return type: set

`poll`(*timeout_ms=0*, *max_records=None*, *update_offsets=True*) [source]

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*) [source]

Get the offset of the next record that will be fetched

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

Returns: Offset

Return type: int

`resume`(**partitions*) [source]

Resume fetching from the specified (paused) partitions.

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

`seek`(*partition*, *offset*) [source]

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*) [source]

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*) [source]

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*) [source]

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()` [source]

Get the current topic subscription.

Returns: {topic, ...}

Return type: set

`topics()` [source]

Get all topics the user is authorized to view. This will always issue a remote call to the cluster to fetch the latest information.

Returns: topics

Return type: set

`unsubscribe()` [source]

Unsubscribe from all topics and clear all assigned partitions.