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')

def on_send_success(record_metadata):
    print(record_metadata.topic)
    print(record_metadata.partition)
    print(record_metadata.offset)

def on_send_error(excp):
    log.error('I am an errback', exc_info=excp)
    # handle exception

# produce asynchronously with callbacks
```

```
producer.send('my-topic', b'raw_bytes').add_callback(on_send_success).add_errback(on_send_error)

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

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