NSQ & Python Worker

Felinx Lee

May 18, 2013

NSQ

BILLIONS AND BILLIONS
OF MESSAGES SERVED

NSQ is a realtime message processing system designed to operate at bitly's scale, handling billions of messages per day.

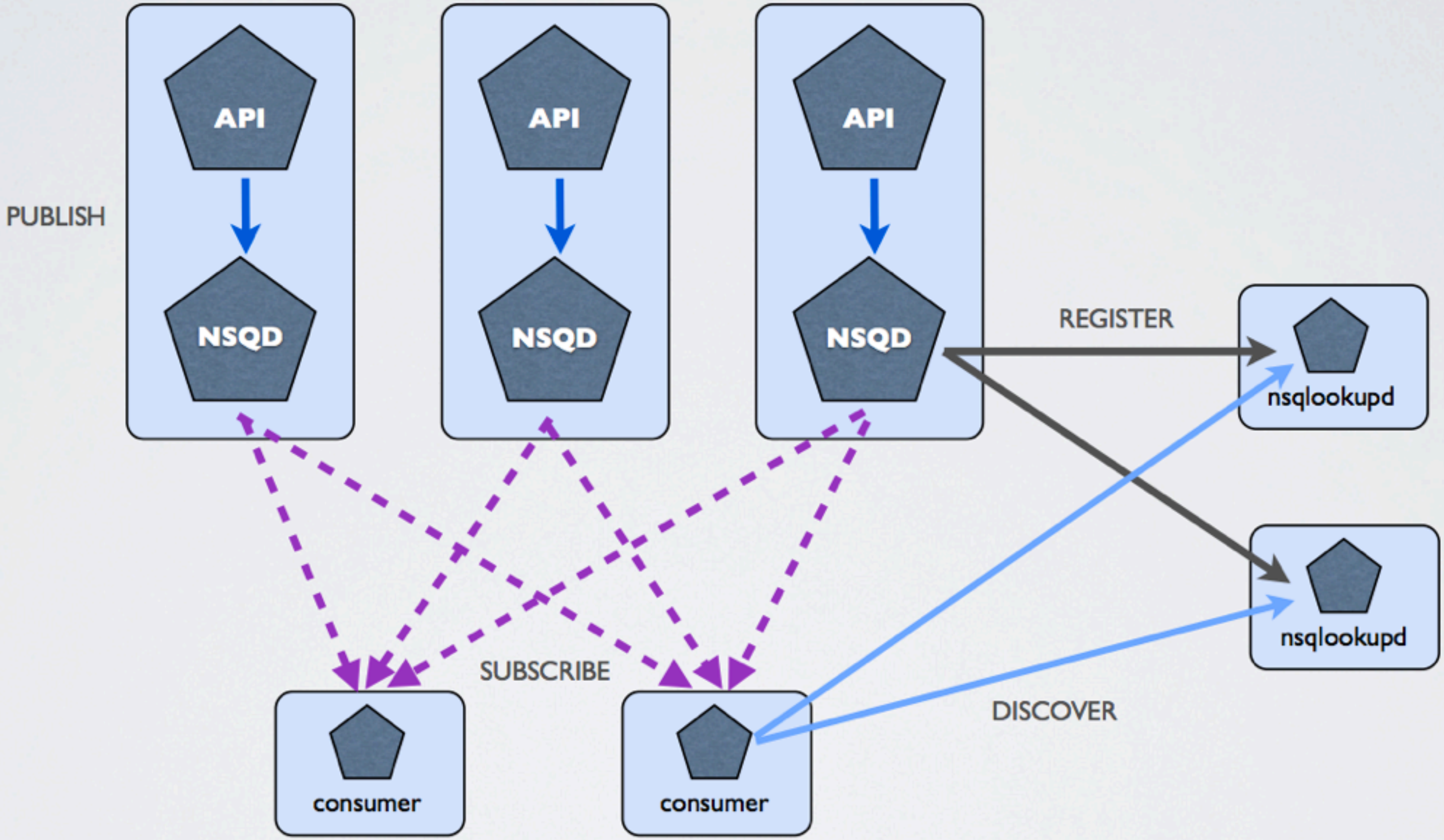
Keywords

- realtime
- distributed
- decentralized
- fault tolerance
- message delivery guarantee
- developed in Go Language
- developed by bitly

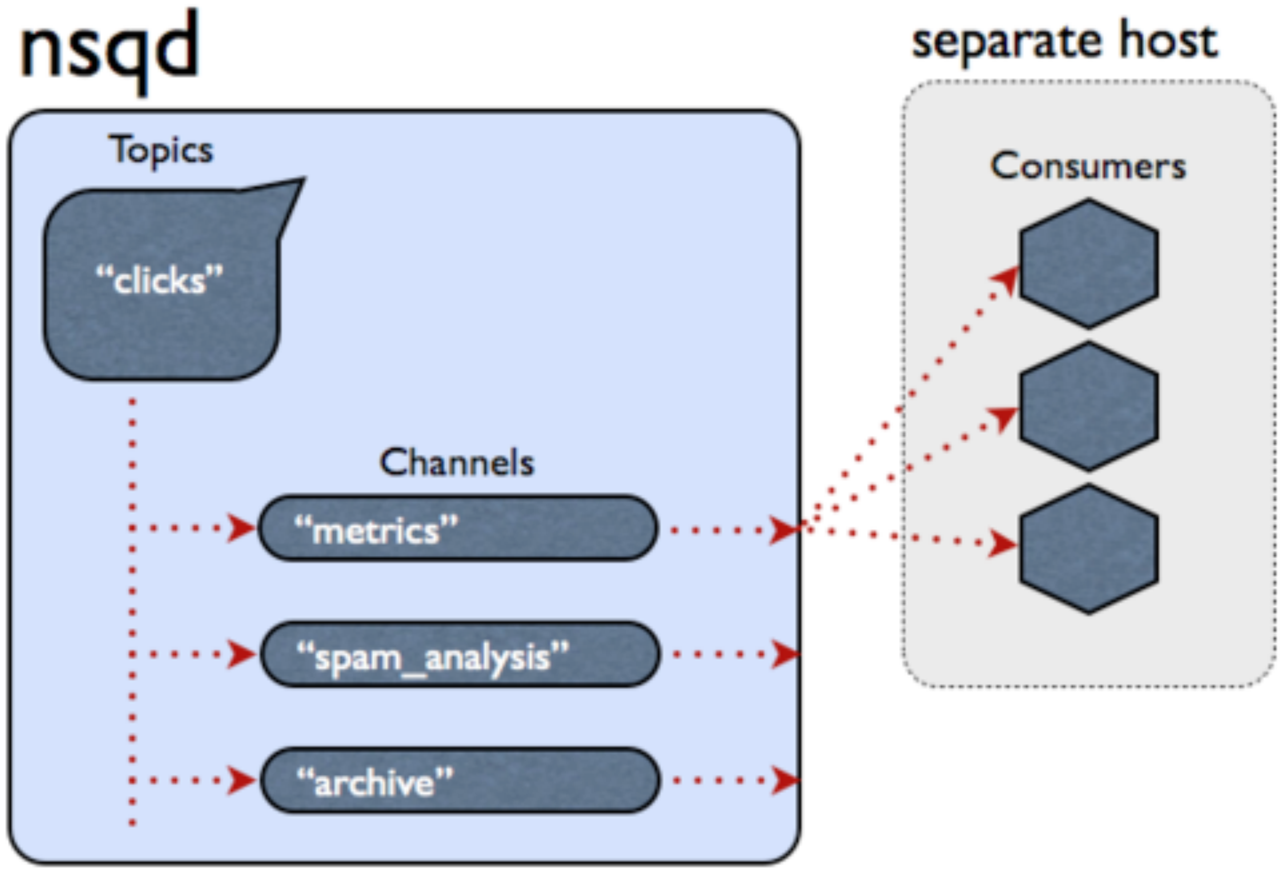
Features

- no SPOF, designed for distributed environments
- messages are guaranteed to be delivered at least once
- low-latency push based message delivery (performance)
- combination load-balanced and multicast style message routing
- configurable high-water mark after which messages are transparently kept on disk
- few dependencies, easy to deploy, and sane, bounded, default configuration
- runtime discovery service for consumers to find producers (nsqlookupd)
- HTTP interface for stats, administrative actions, and producers (no client libraries needed!)
- memcached-like TCP protocol for producers/consumers

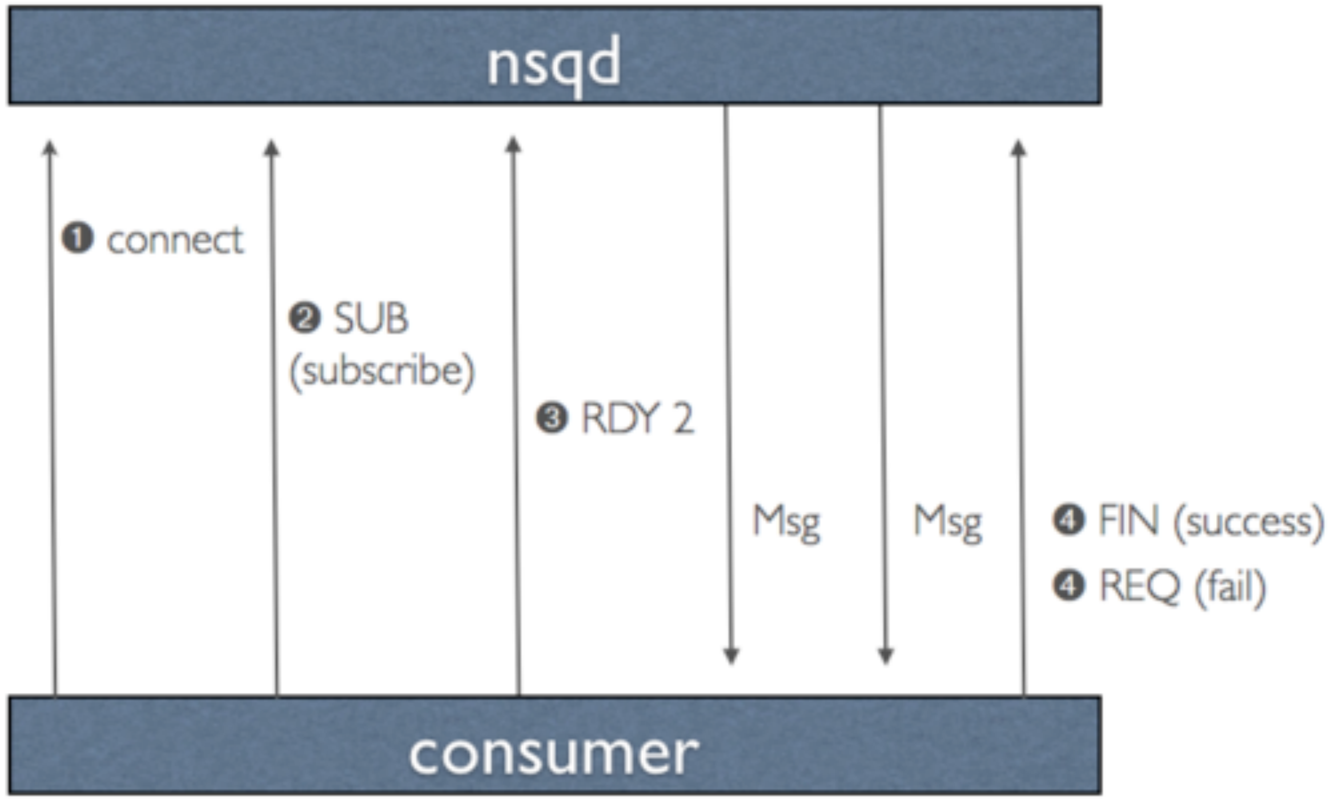
NSQ ARCHITECTURE



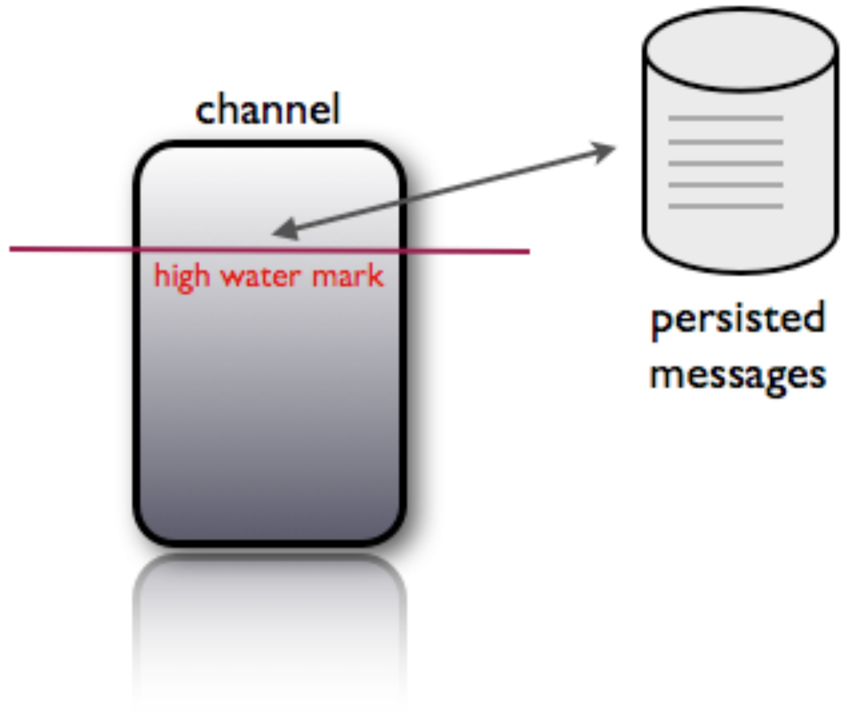
Topics&Channels



Efficiency(Push)



Bounded Memory Footprint



In Production

bitly | Life360 | HAIL O | Path | Trendrr | simplereach | EnergyHub

蝴蝶互动
HOODINN INTERACTIVE

Bootstrap

```python
def main():
    worker = load_worker()
    logging.debug("Starting worker: %s" % worker)

    r = nsq.Reader(worker.tasks,
                   lookupd_http_addresses=options.nsqlookupd_http_addresses,
                   topic=options.topic, channel=options.channel)

    # override default preprocess and validate method with worker's method
    r.validate_message = worker.validate_message
    r.preprocess_message = worker.preprocess_message

    nsq.run()


if __name__ == "__main__":
    main()
```

Loader

```
def load_worker(workers_dir=None):
    """Load worker according to topic and channel options.

     Preload all of workers to make tornado options work like a magic,
     so a worker can define custom options in worker's header.

     It only loads worker from file named as {topic}/{topic}_{channel}_worker.py,
     eg. apiview/apiview_pageview_worker.py, the other files will be ignored.
    """
    # Preload all of workers
    if workers_dir is None:
        workers_dir = os.path.dirname(os.path.abspath(__file__))

    for d in os.listdir(workers_dir):
        dir_ = os.path.join(workers_dir, d)
        if os.path.isdir(dir_):
            # ...

    # Enable tornado command line options
    options.parse_command_line()

    # Then load the worker object according to topic and channel options.
    try:
        worker, name = _load_worker(options.topic, options.channel)
        if worker:
            return worker()
        else:
            raise ImportError("%s not found!" % name)
    except ImportError, e:
        logging.error(traceback.format_exc())
        raise e
```

Worker

```python
class Worker(object):
    def preprocess_message(self, message):
        pass

    def validate_message(self, message):
        pass

    @property
    def tasks(self):
        _tasks = {}
        for func in dir(self):
            if func.endswith("_task"):
                _tasks[func] = _task(getattr(self, func))

        return _tasks

def _task(func):
    """Worker task decorator"""
    q = _nsq_processed_messages_queues.get(func.__name__, None)
    if q is None:
        q = deque(maxlen=options.nsq_max_processed_messages_queue)
        _nsq_processed_messages_queues[func.__name__] = q

    def wrapper(message):
        logging.debug("Raw message: %s, %s", message.id, message.body)
        if message.id not in q:
            try:
                r = func(message)
                if r:
                    q.append(message.id)
                return r
            except Exception, e:
                logging.error(e)
                logging.error(traceback.format_exc())

        return True
    return wrapper
```

Demo

```python
import logging
from tornado.options import options, define
from nsqworker.workers.worker import Worker

define("demoname", "demo")


class TopicChannelWorker(Worker):
    def demo_task(self, message):
        logging.info("%s: %s/%s, message: %s", options.demoname,
                     options.topic, options.channel, message.body)

        return True
```

Publish

```
class MainHandler(tornado.web.RequestHandler):
    @property
    def nsq(self):
        return self.application.nsq

    def get(self):
        topic = "log"
        msg = "Hello world"
        msg_cn = "Hello 世界"

        self.nsq.pub(topic, msg) # pub
        self.nsq.mpub(topic, [msg, msg_cn]) # mpub

        # customize callback
        callback = functools.partial(self.finish_pub, topic=topic, msg=msg)
        self.nsq.pub(topic, msg, callback=callback)

        self.write(msg)

    def finish_pub(self, conn, data, topic, msg):
        if isinstance(data, Error):
            # try to re-pub message again if pub failed
            self.nsq.pub(topic, msg)
```

Reference

- https://github.com/bitly/nsq
- https://github.com/bitly/pynsq
- https://speakerdeck.com/snakes/nsq-nyc-golang-meetup
- https://github.com/bitly/nsq/blob/master/docs/protocol.md
- https://github.com/bitly/nsq/blob/master/docs/building_client_libraries.md
- https://github.com/felinx/nsqworker