AgentZero Documentation

Release 0.4.1

Gabriel Falcão

Apr 27, 2018

Contents

AgentZero lets you create, connect, bind, and modify zeromq sockets in runtime with ease.

It works great with gevent, making it possible to create network applications with simple code that performs complex operations.

CHAPTER 1

Features:

- Create labeled sockets, every ZMQ socket in AgentZero has a name.
- seamlessly poll across connected/bound sockets
- seamlessly subscribe to events, which return the boxed type: `Event`
- easily publish events
- bind sockets to random ports automatically
- bind to hostnames, with automatic DNS resolution
- ability to wait until a socket has received data
- builtin python log handler that publishes logs in a ZMQ PUB socket

CHAPTER 2

Table of Contents:

2.1 Internals Reference

2.1.1 SocketManager

class agentzero.**SocketManager**(*zmq*, *context*, *serialization_backend=None*, *polling_timeout=10000*, *timeout=10*)

High-level abstraction for zeromq's non-blocking api.

This component provides utility methods to create, retrieve, connect and bind sockets by name.

It can wait for a socket to become available in either receiving data, sending data or both at the same time.

Parameters

- **zmq** – a reference to the zmq module (either from `import zmq` or `import zmq.green as zmq`)
- **context** – the context where the sockets will be created
- **serialization_backend** – an instance of a valid `agentzero.serializers.BaseSerializer`. This is completely optional safe for the cases where you utilize the methods `send_safe` and `recv_safe` when communicating to other nodes.

Note: An extra useful feature that comes with using a `SocketManager` is that you can use a SocketManager to create an application that dynamically connects to new nodes based on scaling instructions coming from other nodes

Warning: Always use the same context per each thread. If you are using gevent, please using a single instance for your whole main process, across all greenlets that you manage.

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>> from agentzero.serializers import JSON
>>>
>>> context = zmq.Context()
>>>
>>> sockets = SocketManager(zmq, context, serialization_backend=JSON())
>>> sockets.ensure_and_connect(
...       "requester",
...       zmq.REQ,
...       'tcp://192.168.2.42:5051',
...       zmq.POLLIN | zmq.POLLOUT
... )
<zmq.green.core._Socket at ...>
```

create

SocketManager.**create**(*name*, *socket_type*)

Creates a named socket by type. Can raise a SocketAlreadyExists.

Returns the socket itself

Parameters

- **name** – the socket name
- **socket_type** – a valid socket type (i.e: zmq.REQ, zmq.PUB, zmq.PAIR, ...)

get_by_name

SocketManager.**get_by_name**(*name*)

Returns an existing socket by name. It can raise a SocketNotFound exception.

Returns the socket

Parameters **name** – the socket name

get_or_create

SocketManager.**get_or_create**(*name*, *socket_type*, *polling_mechanism*)

ensure that a socket exists and is registered with a given polling_mechanism (POLLIN, POLLOUT or both)

returns the socket itself.

Parameters

- **name** – the socket name
- **socket_type** – a valid socket type (i.e: zmq.REQ, zmq.PUB, zmq.PAIR, ...)
- **polling_mechanism** – one of (zmq.POLLIN, zmq.POLLOUT, zmq.POLLIN | zmq.POLLOUT)

register_socket

SocketManager.**register_socket**(*socket*, *polling_mechanism*)
registers a socket with a given polling_mechanism (POLLIN, POLLOUT or both)

returns the socket itself.

Parameters

- **socket** – the socket instance
- **polling_mechanism** – one of (zmq.POLLIN, zmq.POLLOUT, zmq.POLLIN | zmq.POLLOUT)

bind

SocketManager.**bind**(*socket_name*, *address*, *polling_mechanism*)
binds a socket to an address and automatically registers it with the given polling mechanism.

returns the socket itself.

Parameters

- **socket_name** – the socket name
- **address** – a valid zeromq address (i.e: inproc://whatevs)
- **polling_mechanism** – zmq.POLLIN, zmq.POLLOUT or zmq.POLLIN | zmq.POLLOUT

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.create('pipe-in', zmq.PULL)
>>> sockets.bind('pipe-in', 'tcp://*:6000', zmq.POLLIN)
```

ensure_and_bind

SocketManager.**ensure_and_bind**(*socket_name*, *socket_type*, *address*, *polling_mechanism*)
Ensure that a socket exists, that is *binded* to the given address and that is registered with the given polling mechanism.

This method is a handy replacement for calling .get_or_create(), .bind() and then .engage().

returns the socket itself.

Parameters

- **socket_name** – the socket name
- **socket_type** – a valid socket type (i.e: zmq.REQ, zmq.PUB, zmq.PAIR, ...)
- **address** – a valid zeromq address (i.e: inproc://whatevs)
- **polling_mechanism** – zmq.POLLIN, zmq.POLLOUT or zmq.POLLIN | zmq.POLLOUT

bind_to_random_port

SocketManager.**bind_to_random_port**(*socket_name*, *polling_mechanism*, *local_address=u'tcp://0.0.0.0'*)

binds the socket to a random port

returns a 2-item tuple with the socket instance and the address string

Parameters

- **socket_name** – the socket name
- **polling_mechanism** – zmq.POLLIN, zmq.POLLOUT or zmq.POLLIN | zmq.POLLOUT

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.create('api-server', zmq.REP)
>>> _, address = sockets.bind_to_random_port(
...     'api-server',
...     zmq.POLLIN | zmq.POLLOUT,
...     local_address='tcp://192.168.10.24
... )
>>> address
'tcp://192.168.10.24:61432'
```

connect

SocketManager.**connect**(*socket_name*, *address*, *polling_mechanism*)

connects a socket to an address and automatically registers it with the given polling mechanism.

returns the socket itself.

Parameters

- **socket_name** – the socket name
- **address** – a valid zeromq address (i.e: inproc://whatevs)
- **polling_mechanism** – zmq.POLLIN, zmq.POLLOUT or zmq.POLLIN | zmq.POLLOUT

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_connect(
...   socket_name='logs',
...   zmq.PUB,
...   'tcp://192.168.10.24:6000',
...   zmq.POLLOUT
... )
>>> sockets.publish_safe('logs', 'output', 'some data')
```

ensure_and_connect

SocketManager.**ensure_and_connect**(*socket_name*, *socket_type*, *address*, *polling_mechanism*)

Ensure that a socket exists, that is *connected* to the given address and that is registered with the given polling mechanism.

This method is a handy replacement for calling `.get_or_create()`, `.connect()` and then `.engage()`.

returns the socket itself.

Parameters

- **socket_name** – the socket name
- **socket_type** – a valid socket type (i.e: `zmq.REQ`, `zmq.PUB`, `zmq.PAIR`, ...)
- **address** – a valid zeromq address (i.e: inproc://whatevs)
- **polling_mechanism** – `zmq.POLLIN`, `zmq.POLLOUT` or `zmq.POLLIN | zmq.POLLOUT`

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_connect(
...   socket_name='logs',
...   zmq.REQ,
...   'tcp://192.168.10.24:7000',
...   zmq.POLLIN | zmq.POLLOUT
... )
```

engage

SocketManager.**engage**(*timeout=None*)

polls all registered sockets with the given timeout in miliseconds

returns a dictionary with the sockets that are ready to be used in their respective state (`zmq.POLLIN` *or* `zmq.POLLOUT`)

Parameters **timeout** – how long should it poll until a socket becomes available. defaults to `agentzero.core.DEFAULT_POLLING_TIMEOUT`

send_safe

SocketManager.**send_safe**(*name*, *data*, **args*, ***kw*)

serializes the data with the configured `serialization_backend`, waits for the socket to become available, then sends it over through the provided socket name.

returns `True` if the message was sent, or `False` if the socket never became available.

Note: you can safely use this function without waiting for a socket to become ready, as it already does it for you.

---

raises SocketNotFound when the socket name is wrong.

Parameters

- **name** – the name of the socket where data should be sent through
- **data** – the data to be serialized then sent
- ***args** – args to be passed to wait_until_ready
- ****kw** – kwargs to be passed to wait_until_ready

recv_safe

SocketManager.**recv_safe**(*name*, **args*, ***kw*)

waits for the socket to become available then receives data through it and deserializes the result using the configured `serialization_backend` before returning.

Note: you can safely use this function without waiting for a socket to become ready, as it already does it for you.

---

raises SocketNotFound when the socket name is wrong.

returns the deserialized data, or `None` if the socket never became available

Parameters

- **name** – the name of the socket where data will pad through
- ***args** – args to be passed to wait_until_ready
- ****kw** – kwargs to be passed to wait_until_ready

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('pipe-in', zmq.PULL, 'tcp://*:6000', zmq.POLLIN)
>>> sockets.recv_safe('pipe-in')
{
    "pipeline": "video-download",
    "instructions": {
      "url": "https://www.youtube.com/watch?v=FPZ6mVsv4EI"
    }
}
```

recv_event_safe

SocketManager.**recv_event_safe**(*name*, *topic=False*, **args*, ***kw*)

waits for the socket to become available then receives multipart data assuming that it's a pub/sub event, thus it parses the topic and the serialized data, then it deserializes the result using the configured `serialization_backend` before returning.

Note: you can safely use this function without waiting for a socket to become ready, as it already does it for you.

---

raises SocketNotFound when the socket name is wrong.

returns the deserialized data, or `None` if the socket never became available

Parameters

- **name** – the name of the socket where data will pad through
- ***args** – args to be passed to wait_until_ready
- ****kw** – kwargs to be passed to wait_until_ready

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('events', zmq.SUB, 'tcp://*:6000', zmq.POLLIN)
>>>
>>> # subscribe only to topics beginning with "logs"
>>> sockets.set_topic('events', 'logs')
>>> event = sockets.recv_event_safe('events')
>>> event.topic, event.data
'logs:2016-06-20', {'stdout': 'hello world'}
```

subscribe

`SocketManager.`**`subscribe`**(*name*, *topic=None*, *keep_polling=None*, **args*, ***kw*)

waits for the socket to become available then receives data through it and deserializes the result using the configured `serialization_backend` before returning.

Note: you can safely use this function without waiting for a socket to become ready, as it already does it for you.

---

raises SocketNotFound when the socket name is wrong.

returns an :py:class‘~agentzero.core.Event‘, or `None` if the socket never became available

Parameters

- **name** – the name of the socket where data will pad through
- ***args** – args to be passed to wait_until_ready
- ****kw** – kwargs to be passed to wait_until_ready

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('logs', zmq.SUB, 'tcp://*:6000', zmq.POLLIN)
>>> for topic, data in sockets.subscribe('logs', 'output'):
...     print topic, '==>', data
...
output:0 ==> some data
output:1 ==> more data
...
```

set_socket_option

SocketManager.**set_socket_option**(*name*, *option*, *value*)
calls zmq.setsockopt on the given socket.

Parameters

- **name** – the name of the socket where data will pad through
- **option** – the option from the zmq module
- **value** –

Here are some examples of options:

- zmq.HWM: Set high water mark
- zmq.AFFINITY: Set I/O thread affinity
- zmq.IDENTITY: Set socket identity
- zmq.SUBSCRIBE: Establish message filter
- zmq.UNSUBSCRIBE: Remove message filter
- zmq.SNDBUF: Set kernel transmit buffer size
- zmq.RCVBUF: Set kernel receive buffer size
- zmq.LINGER: Set linger period for socket shutdown
- zmq.BACKLOG: Set maximum length of the queue of outstanding connections
- for the full list go to http://api.zeromq.org/4-0:zmq-setsockopt

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.create('pipe-in', zmq.PULL)
>>>
>>> # block after 10 messages are queued
>>> sockets.set_socket_option('pipe-in', zmq.HWM, 10)
```

set_topic

SocketManager.**set_topic**(*name*, *topic*)
shortcut to :py:meth:SocketManager.set_socket_option(zmq.TOPIC, topic)

Parameters

- **name** – the name of the socket where data will pad through
- **topic** – the option from the zmq module

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('events', zmq.SUB, 'tcp://*:6000', zmq.POLLIN)
```

```
>>>
>>> # subscribe only to topics beginning with "logs"
>>> sockets.set_topic('events', 'logs')
>>> event = sockets.recv_event_safe('events')
>>> event.topic, event.data
'logs:2016-06-20', {'stdout': 'hello world'}
```

publish_safe

`SocketManager.`**`publish_safe`**(*name*, *topic*, *data*)

serializes the data with the configured `serialization_backend`, waits for the socket to become available, then sends it to the given topic through `socket.send_multipart`.

returns `True` if the message was sent, or `False` if the socket never became available.

Note: you can safely use this function without waiting for a socket to become ready, as it already does it for you.

---

raises SocketNotFound when the socket name is wrong.

Parameters

- **`name`** – the name of the socket where data should be sent through
- **`topic`** – the name of the topic
- **`data`** – the data to be serialized then sent

ready

`SocketManager.`**`ready`**(*name*, *polling_mechanism*, *timeout=None*)

Polls all sockets and checks if the socket with the given name is ready for either `zmq.POLLIN` or `zmq.POLLOUT`.

returns the socket if available, or `None`

Parameters

- **`socket_name`** – the socket name
- **`polling_mechanism`** – either `zmq.POLLIN` or `zmq.POLLOUT`
- **`timeout`** – the polling timeout in miliseconds that will be passed to `zmq.Poller().poll()` (optional, defaults to `core.DEFAULT_POLLING_TIMEOUT`)

wait_until_ready

`SocketManager.`**`wait_until_ready`**(*name*, *polling_mechanism*, *timeout=None*, *polling_timeout=None*)

Briefly waits until the socket is ready to be used, yields to other greenlets until the socket becomes available.

returns the socket if available within the given timeout, or `None`

Parameters

- **`socket_name`** – the socket name

- **polling_mechanism** – either `zmq.POLLIN` or `zmq.POLLOUT`
- **timeout** – the timeout in seconds (accepts float) in which it should wait for the socket to become available (optional, defaults to `core.DEFAULT_TIMEOUT_IN_SECONDS`)
- **polling_timeout** – the polling timeout in miliseconds that will be passed to `zmq.Poller().poll()`. (optional, defaults to `core.DEFAULT_POLLING_TIMEOUT`)

ready

`SocketManager.`**ready**(*name*, *polling_mechanism*, *timeout=None*)

Polls all sockets and checks if the socket with the given name is ready for either `zmq.POLLIN` or `zmq.POLLOUT`.

returns the socket if available, or `None`

Parameters

- **socket_name** – the socket name
- **polling_mechanism** – either `zmq.POLLIN` or `zmq.POLLOUT`
- **timeout** – the polling timeout in miliseconds that will be passed to `zmq.Poller().poll()` (optional, defaults to `core.DEFAULT_POLLING_TIMEOUT`)

get_log_handler

`SocketManager.`**get_log_handler**(*socket_name*, *topic_name=u'logs'*)

returns an instance of :py:class:ZMQPubHandler attached to a previously-created socket.

Parameters

- **socket_name** – the name of the socket, previously created with :py:meth:SocketManager.create
- **topic_name** – the name of the topic in which the logs will be PUBlished

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('logs', zmq.PUB, 'tcp://*:6000', zmq.POLLOUT)
>>> app_logger = sockets.get_logger('logs', logger_name='myapp'))
>>> app_logger.info("Server is up!")
>>> try:
        url = sockets.recv_safe('download_queue')
...     requests.get(url)
... except:
...     app_logger.exception('failed to download url: %s', url)
```

get_logger

`SocketManager.`**get_logger**(*socket_name*, *topic_name=u'logs'*, *logger_name=None*)

returns an instance of :py:class:logging.Logger that contains a :py:class:ZMQPubHandler attached to.

Parameters

- **socket_name** – the name of the socket, previously created with :py:meth:SocketManager.create
- **topic_name** – (optional) the name of the topic in which the logs will be PUBlished, defaults to **"logs"**
- **logger_name** – (optional) defaults to the given socket name

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.ensure_and_bind('logs', zmq.PUB, 'tcp://*:6000', zmq.POLLOUT)
>>> app_logger = sockets.get_logger('logs', logger_name='myapp'))
>>> app_logger.info("Server is up!")
>>> try:
        url = sockets.recv_safe('download_queue')
...     requests.get(url)
... except:
...     app_logger.exception('failed to download url: %s', url)
```

close

SocketManager.**close**(*socket_name*)

closes a socket if it exists

Parameters

- **socket_name** – the socket name
- **address** – a valid zeromq address (i.e: inproc://whatevs)
- **polling_mechanism** – zmq.POLLIN, zmq.POLLOUT or zmq.POLLIN | zmq.POLLOUT

Example:

```
>>> import zmq
>>> from agentzero.core import SocketManager
>>>
>>> sockets = SocketManager()
>>> sockets.create('logs', zmq.SUB)
>>> sockets.bind('logs', 'tcp://*:6000', zmq.POLLIN)
>>> sockets.close('logs')
```

2.1.2 Utility Functions

get_free_tcp_port

agentzero.util.**get_free_tcp_port**()

returns a TCP port that can be used for listen in the host.

get_default_bind_address

agentzero.util.**get_default_bind_address**()

get_public_ip_address

agentzero.util.**get_public_ip_address**(*hostname=None*)

extract_hostname_from_tcp_address

agentzero.util.**extract_hostname_from_tcp_address**(*address*)

resolve_hostname

agentzero.util.**resolve_hostname**(*hostname*)

fix_zeromq_tcp_address

agentzero.util.**fix_zeromq_tcp_address**(*address*)

get_public_zmq_address

agentzero.util.**get_public_zmq_address**()

seconds_since

agentzero.util.**seconds_since**(*timestamp*)

datetime_from_seconds

agentzero.util.**datetime_from_seconds**(*timestamp*)

serialized_exception

agentzero.util.**serialized_exception**(*e*)

2.1.3 Exceptions

AgentZeroSocketError

exception agentzero.errors.**AgentZeroSocketError**
Base exception class for errors originated in SocketManager

SocketAlreadyExists

exception agentzero.errors.**SocketAlreadyExists**(*manager*, *socket_name*)
raised by SocketManager when trying to create a named socket that already exists

```
>>> from agentzero.core import zmq
>>> from agentzero.core import SocketManager
>>> sockets = SocketManager()
>>> sockets.create('foo', zmq.REP)
>>> sockets.create('foo', zmq.REP)
Traceback (most recent call last):
    ...
SocketAlreadyExists: SocketManager(sockets=['foo']) already has a socket named
↪'foo'.
```

SocketNotFound

exception agentzero.errors.**SocketNotFound**(*manager*, *socket_name*)
raised by SocketManager when trying to retrieve an unexisting socket

```
>>> from agentzero.core import zmq
>>> from agentzero.core import SocketManager
>>> sockets = SocketManager()
>>> sockets.get_by_name('some-name', zmq.PUB)
Traceback (most recent call last):
    ...
SocketNotFound: SocketManager(sockets=['']) has no sockets named 'some-name'.
```

SocketBindError

exception agentzero.errors.**SocketBindError**
raised by SocketManager when a **:py:method:‘~agentzero.core.SocketManager.bind‘** operation fails.

SocketConnectError

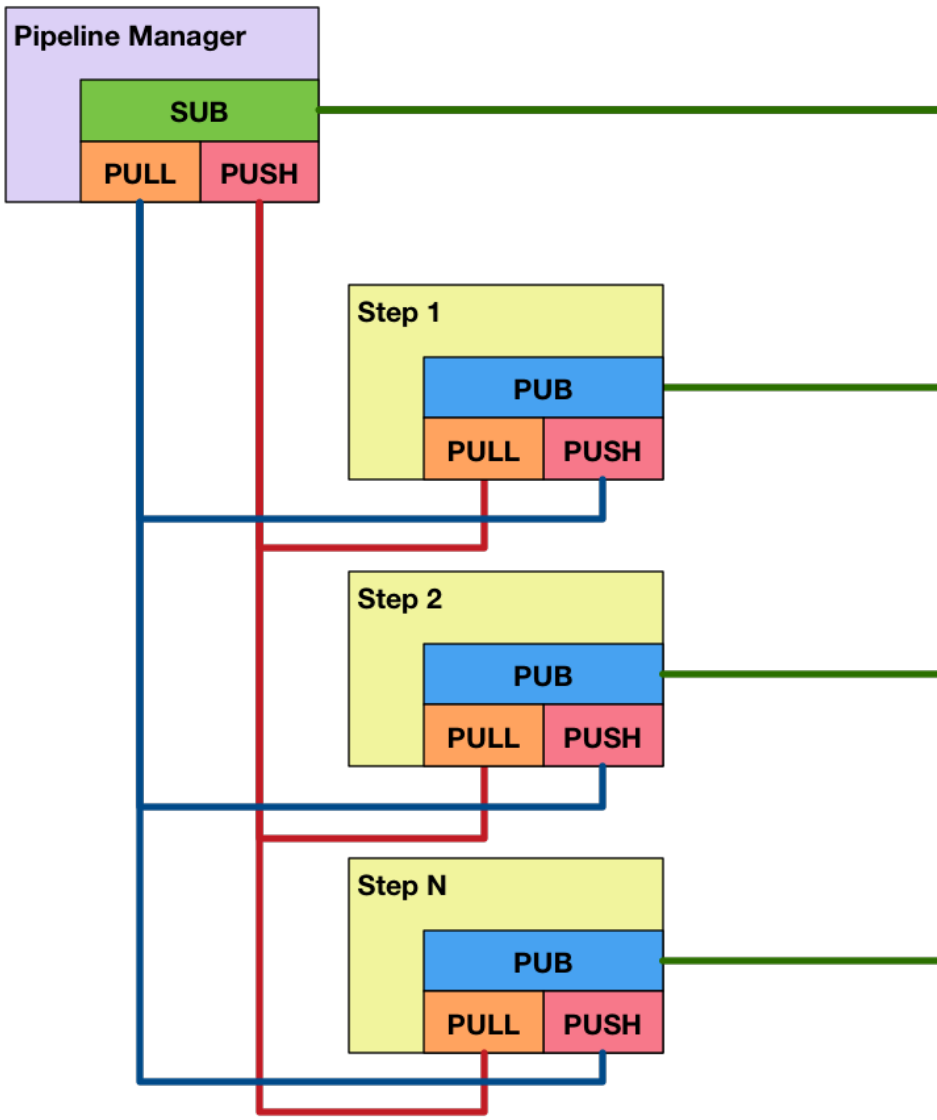
exception agentzero.errors.**SocketConnectError**
raised by SocketManager when a **:py:method:‘~agentzero.core.SocketManager.connect‘** operation fails.

2.2 Indices and tables

- genindex
- modindex
- search

2.3 Building a distributed worker pipeline

Let’s build a worker pipeline where Steps that will execute specific job types, and can be scaled individually.



Here is an overview of the socket architecture:

2.3.1 Coding the pipeline entity

The main pipeline contains the following sockets:

- a `SUB` socket where it will `bind()` at the given `bind_address` and subscribe to `Step` events
- a `REP` socket where it will respond to pipeline execution `bind()` at the given `reply_address` and reply with a job id for later status querying
- a `REP` socket where it will respond to pipeline execution `bind()` at the given `reply_address`

```
import zmq.green as zmq
from agentzero.core import SocketManager


class Pipeline(object):
    steps = []

```

```
    def __init__(self):
        self.context = zmq.Context()
        self.sockets = SocketManager(zmq, self.context)
        self.sockets.create('pipe-sub', zmq.SUB)
        self.sockets.create('pipe-in', zmq.PULL)
        self.sockets.create('pipe-out', zmq.PUSH)
        self.children = []
```

2.3.2 Coding the step entity

A `Step` contains a PUB socket where it will send the following events:

- announce its `JobType` as well as its PUSH/PULL `address` pair
- announce failed jobs, so that they can be auto-recovered later
- announce succeeded jobs
- announce exceptions and auto-schedule a later retry
- live metrics
- live logging output

Index