distconfig Documentation

Release 0.1.0

Delivery Hero Holding GmbH

April 13, 2015

Contents

Library to manage distributed configuration using either ZooKeeper or Etcd or Consul.

Contents:

CHAPTER 1

Installation

To use **ZooKeeper** as backend you should install `distconfig` using

```
$ pip install distconfig[zookeeper]
```

with **etcd**:

```
$ pip install distconfig[etcd]
```

with **consul**:

```
$ pip install distconfig[consul]
```

To use distconfig with gevent you can do

```
$ pip install distconfig[zookeeper, gevent]
```

CHAPTER 2

Usage

Example using zookeeper as a backend

```
import kazoo

from distconfig import Proxy

client = kazoo.Client()

proxy = Proxy.configure(
    'distconfig.backends.zookeeper.ZooKeeperBackend',
    client=client,
)

# config is a read only mapping-like object.
config = proxy.get_config('/distconfig/service_name/config')

print config['key']

# Getting nested values works by supplying dotted key, this of course
# mean that key cannot have a dot in them.
print config['key.inner']

# You can assert key value type by using typed get function e.g.
# get_int, get_float, get_unicode, get_bytes ... .
print config.get_int('key.inner.int_key')

# Getting a inner config.
print config.get_config('key.inner.dict_key')
```

CHAPTER 3

API Docs

Contents:

3.1 Backend Proxy

class distconfig.api.**Proxy**(*backend*)
Proxy class for differents backend.

backend
Readonly property for backend.

classmethod configure(*backend_qualname*, ***backend_options*)
Configure backend.

Parameters

- **backend_qualname** – Backend class qualifed name (dotted-name) e.g. distconfig.backends.zookeeper.ZooKeeperBackend.
- **logger** – logging.Logger instance to use for logging.
- **parser** – Callable that accept the raw config data saved in the backend and should return the config data parsed, default: ujson.loads.
- **backend_options** – Keyword arguments to pass to backend class.

get_config(*path*, *config_cls=<class 'distconfig.config.Config'>*)
Get configuration from path.

Parameters

- **path** – Location of the configuratin in the backend.
- **config_cls** – configuration class to return, default: distconfig.config.Config.

Returns config_cls instance.

3.2 Configuration Management

class distconfig.config.**Config**(*data*)
Read only mapping-like for holding configuration.

Note: At the opposite of `ConfigParser.RawConfigParser` in stdlib, methods like `get_<type>` in this class do **not** coerse, instead the use case is type assertion, i.e. When a user call `Config.get_int` on a key where the value is a string e.g. '2', the call will fail with a `TypeError`.

get(*path*, *default=None*, *type_=None*)
Same as `dict.get(k[,d])` with an extra argument `type_` that assert type matching if given.

`path` argument can be in the form 'key1/key2/key3' as a shortcut for doing `config['key1']['key2']['key3']`. In case a key have a / in it, we can escape it like so 'key1/key2/key3' this will be translated to `config['key1/key2']['key3']`

Parameters

- **path** – path expression e.g. 'key1/key2/key3'
- **default** – default value to return when key is missing, in case it's supplied `default` with `type_`, `default` must be of of type `type_`, default: None.
- **type** – Python type to validate returned value, default: None.

Returns the value of the requested path as a string.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and `default` is set to `distconfig.config.UNDEFINED`.

get_boolean(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as boolean.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type boolean, default: UNDEFINED.

Returns the value of the requested path as a Boolean.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

get_bytes(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as bytes.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type bytes, default: UNDEFINED.

Returns the value of the requested path as a Unicode.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

get_config(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as `Config`.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type dict, default: UNDEFINED.

Returns the value of the requested path as a `Config` instance.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

get_float(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as float.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type float, default: UNDEFINED.

Returns the value of the requested path as a Float.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

get_int(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as integer.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type integer, default: UNDEFINED.

Returns the value of the requested path as a Integer.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

get_unicode(*path*, *default=<object object at 0x7f30266183c0>*)
Get path value as unicode.

Parameters

- **path** – path expression e.g. 'key1.key2.key3'
- **default** – default value to return when key is missing, if supplied `default` must be of type unicode, default: UNDEFINED.

Returns the value of the requested path as a Unicode.

Raises

- **TypeError** – if path value is not of type `type_`.
- **KeyError** – if path doesn't exist and no default was given.

3.3 Backends

3.3.1 Base class

class distconfig.backends.base.**BaseBackend**(*parser=<built-in function loads>*, *logger=<logging.Logger object at 0x7f3026232950>*)

Base abstract backend class.

Backend implementation should inherit and implement `get_raw` method.

Parameters

- **parser** – Callable that accept a string and parse it, default: `ujson.loads`.
- **logger** – `logging.Logger` ` instance.

add_listener(*callback*)

Add callback to be called when data change in the backend.

If the same callback is added more than once, then it will be notified more than once. That is, no check is made to ensure uniqueness.

Parameters **callback** – Callable that accept one argument the new data.

get(*path*)

Get parsed path value in backend.

Path key in the backend.

Returns path value parsed.

get_raw(*path*)

Get path value from backend as it is.

Path key in the backend.

Returns path value as saved in backend or None if not found in backend.

remove_listener(*callback*)

Remove previously added callback.

If callback had been added more than once, then only the first occurrence will be removed.

Parameters **callback** – Callable as with `:meth: add_listener`.

Raises ValueError In case callback was not previously registered.

3.3.2 Existing backends

The current backends exist out of the box:

class distconfig.backends.consul.**ConsulBackend**(*client*, *execution_context=<distconfig.backends.execution_context.ThreadingExe object at 0x7f3025cbf090>*, ***kwargs*)

Consul backend implementation.

Parameters

- **client** – Instance of `consul.Consul`.
- **execution_context** – Instance of `distconfig.backends.execution_context.ExecutionContext`

3.3.3 Execution Contexts

class distconfig.backends.execution_context.**GeventExecutionContext**
Execution context that run background function as a Greenlet.

gevent monkey patching must be done by user.

run(*func*, **args*, ***kwargs*)
Run given function in a Greenlet.

class distconfig.backends.execution_context.**ThreadingExecutionContext**
Execution context that run background function as a OS Thread.

run(*func*, **args*, ***kwargs*)
Run given function in a daemon OS thread.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

d

A

B

C

D

G

P

R

T