Nginx Config Builder

Release

May 04, 2018

Contents

A python library for constructing nginx configuration files.

Contents:

CHAPTER 1

Builder API

The Builder API defines a pluggable builder framework for manipulating nginx configs from within python.

1.1 Building a config

Every config built using the builder pattern starts off with creating a :class:NginxConfigBuilder:

```
from nginx.config.builder import NginxConfigBuilder

nginx = NginxConfigBuilder()
```

By default, this comes loaded with a bunch of helpful tools to easily create routes and servers in nginx:

```
with nginx.add_server() as server:
    server.add_route('/foo').end()
    with server.add_route('/bar') as bar:
         bar.add_route('/baz')
```

This generates a simple config that looks like this:

```
error_log logs/nginx.error.log;
worker_processes auto;
daemon on;
http {
    include ../conf/mime.types;
    server {
        server_name _;
        location /foo {
        }
        location /bar {
            location /baz {
            }
        }
    }
```

```
}
events {
    worker_connections 1024;
}
```

1.2 Plugins

A plugin is a class that inherits from `nginx.config.builder.baseplugins.Plugin` that provides additional methods which can be chained off of the *NginxConfigBuilder* object. These plugins provide convenience methods that manipulate the underlying nginx configuration that gets built by the *NginxConfigBuilder*.

A simple plugin only needs to define what methods it's going to export:

```
class NoopPlugin(Plugin):
    name = 'noop'

    @property
    def exported_methods(self):
         return {'noop': self.noop}

    def noop(self):
         pass
```

This NoopPlugin provides a simple function that can be called off of a *NginxConfigBuilder* that does nothing successfully. More complex plugins can be found in `nginx.config.builder.plugins`

To use this NoopPlugin, we need to create a config builder and then register the plugin with it:

```
nginx = NginxConfigBuilder()
nginx.noop()  # AttributeError :(
nginx.register_plugin(NoopPlugin())
nginx.noop()  # it works!
```

A more complex plugin would actually do something, like a plugin that adds an expiry directive to a route:

```
class ExpiryPlugin(Plugin):
    name = 'expiry'
    @property
```

class nginx.config.builder.**NginxConfigBuilder**(*worker_processes='auto'*, *worker_connections=512*, *error_log='logs/error.log'*, *daemon='off'*)

Helper that builds a working nginx configuration

Exposes a plugin-based architecture for generating nginx configurations.

register_plugin(*plugin*)

Registers a new nginx builder plugin.

Plugins must inherit from nginx.builder.baseplugins.Plugin and not expose methods that conflict with already loaded plugins

Parameters **nginx.builder.baseplugins.Plugin** (*plugin*) – nginx plugin to add to builder

top

Returns the logical top of the config hierarchy.

This is a convenience method for any plugins that need to quickly access the top of the config tree.

:returns `nginx.config.Block`: Top of the config block

CHAPTER 2

Block (low-level) API

The Block API provides objects to programatically generate nginx configurations.

Example:

```
>>> from nginx.config.api import Config, Section, Location
>>> events = Section('events', worker_connections='1024')
>>> http = Section('http', include='../conf/mime.types')
>>> http.sections.add(
...     Section(
...         'server',
...         Location(
...             '/foo',
...             proxy_pass='upstream',
...         ),
...         server_name='_',
...     )
... )
>>> nginx = Config(
...     events,
...     http,
...     worker_processes='auto',
...     daemon='on',
...     error_log='var/error.log',
... )
>>> print(nginx)

error_log var/error.log;
worker_processes auto;
daemon on;
http {
    include ../conf/mime.types;
    server {
        server_name _;
        location /foo {
            proxy_pass upstream;
```

```
        }
    }
}
events {
    worker_connections 1024;
}
```

class nginx.config.api.**EmptyBlock**(**sections*, ***options*)

An unnamed block of options and/or sections.

Empty blocks are useful for representing groups of options.

For example, you can use them to represent options with non-unique keys:

Example:

```
>>> from nginx.config.helpers import duplicate_options
>>> dupes = duplicate_options('key', ['value', 'other_value', 'third_value'])
>>> type(dupes)
nginx.config.api.blocks.EmptyBlock
>>> print(dupes)

key third_value;
key value;
key other_value;
```

class nginx.config.api.**Block**(*name*, **sections*, ***options*)

A block represent a named section of an Nginx config, such as 'http', 'server' or 'location'

Using this object is as simple as providing a name and any sections or options, which can be other Block objects or option objects.

Example:

```
>>> from nginx.config.api import Block
>>> http = Block('http', option='value')
>>> print(http)

    http {
        option value;
    }
```

class nginx.config.api.**Location**(*location*, **args*, ***kwargs*)

A Location is just a named block with "location" prefixed

class nginx.config.api.**KeyOption**(*name*)

A KeyOption represents a directive with no value.

For example: http://nginx.org/en/docs/http/ngx_http_core_module.html#internal

class nginx.config.api.**KeyValueOption**(*name*, *value=''*)

A key/value directive. This covers most directives available for Nginx

class nginx.config.api.**KeyMultiValueOption**(*name*, *value=''*)

A key with multiple space delimited options.

For example: http://nginx.org/en/docs/http/ngx_http_log_module.html#access_log

Example:

```
>>> from nginx.config.api.options import KeyMultiValueOption
>>> a_log = KeyMultiValueOption('access_log', ['/path/to/log.gz', 'combined',
↪'gzip', 'flush=5m'])
>>> print(a_log)

access_log /path/to/log.gz combined gzip flush=5m;
```

class nginx.config.api.**Comment**(*offset=''*, *comment=''*, ***kwargs*)
A simple comment object.

nginx.config.api.**Config**
alias of *EmptyBlock*

nginx.config.api.**Section**
alias of *Block*

CHAPTER 3

Common Stanzas

This module contains functions and variables that provide a variety of commonly used nginx config boilerplate.

nginx.config.common.**_gzip_options**()
These are some decent default settings for gzip compression

nginx.config.common.**_large_buffers**()
These are some larger than default buffer settings.

Use at your own risk!

nginx.config.common.**_statsd_options_location**()
These are some good defaults to supply to Nginx when using the statsd plugin.

https://github.com/zebrafishlabs/nginx-statsd

NB! it requires you to include a "statsd_server" directive in your http section. This set of common directives should go in any Location block.

nginx.config.common.**_uwsgi_cache**()
A set of useful defaults for using nginx's response cache with uWSGI

This block of options belongs in your HTTP section.

NB! you must set "set $nocache 0;" in the Location block of your uwsgi backend.

see: http://nginx.org/en/docs/http/ngx_http_uwsgi_module.html

nginx.config.common.**_uwsgi_cache_location**()
These are some decent defaults for caching uwsgi responses

nginx.config.common.**ratelimit_options**(*qps*)
Rcreate rate limit shared memory zone, used for tracking different connections.

Parameters **qps** (*int* | *str*) – Queries per second to rate limit.

nginx.config.common.**ratelimit_options_location**(*burst_qps*)
This needs to be added to a location block in order for that location to get rate limiting

Parameters **burst_qps** (*int* | *str*) – Queries per second to allow bursting to.

CHAPTER 4

Utilities

Convienence utilities for building nginx configs

nginx.config.helpers.**dumps**(*config_list*)

Dumps a string representation of a config. Accepts a list of config objects.

Parameters **config_list** (*list*) – A list of config objects from this module

Return type str

nginx.config.helpers.**duplicate_options**(*key*, *values*)

There are many cases when building configs that you may have duplicate keys

This function will produce an EmptyBlock object with duplicate keys but unique values

Example:

```
from nginx.config.helpers import duplicate_options
from nginx.config.api import Location
loc = Location(
    '/',
    duplicate_options('uwsgi_cache_valid', (['404', '5s'], ['200', '60s'])),
)
```

Which would produce:

```
location / {
    uwsgi_cache_valid 200 60s;
    uwsgi_cache_valid 404 5s;
}
```

nginx.config.helpers.**simple_configuration**(*port=8080*)

Returns a simple nginx config.

Also serves as an example of how to build configs using this module.

Parameters **port** (*int*) – A port to populate the 'listen' paramter of the server block

Rtype str

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

Symbols

B

C

D

E

K

L

N

R

S

T