
Granian

A Rust HTTP server for Python applications.

Rationale

The main reasons behind Granian design are:

- Have a single, correct HTTP implementation, supporting versions 1, 2 (and eventually 3)
- Provide a single package for several platforms
- Avoid the usual Gunicorn + uvicorn + http-tools dependency composition on unix systems
- Provide stable performance when compared to existing alternatives

Features

- Supports ASGI/3, RSGI and WSGI interface applications
- Implements HTTP/1 and HTTP/2 protocols
- Supports HTTPS
- Supports Websockets

Quickstart

You can install Granian using pip:

```
1 $ pip install granian
```

Create an ASGI application in your `main.py`:

```
1 async def app(scope, receive, send):
2     assert scope['type'] == 'http'
3
4     await send({
5         'type': 'http.response.start',
6         'status': 200,
7         'headers': [
8             [b'content-type', b'text/plain'],
9         ],
10    })
11    await send({
12        'type': 'http.response.body',
13        'body': b'Hello, world!',
14    })
```

and serve it:

```
1 $ granian --interface asgi main:app
```

You can also create an app using the RSGI specification:

```
1 async def app(scope, proto):
2     assert scope.proto == 'http'
3
4     proto.response_str(
5         status=200,
6         headers=[
7             ('content-type', 'text/plain')
8         ],
9         body="Hello, world!"
10    )
```

and serve it using:

```
1 $ granian --interface rsgi main:app
```

Options

You can check all the options provided by Granian with the `--help` command:

```
1 $ granian --help
2 Usage: granian [OPTIONS] APP
3
4   APP  Application target to serve.  [required]
5
6   Options:
7     --host TEXT                Host address to bind to [env var:
8                               GRANIAN_HOST; default: (127.0.0.1)]
9     --port INTEGER             Port to bind to. [env var:
10                                GRANIAN_PORT;
11                                default: 8000]
12     --interface [asgi|asgini|rsgi|wsgi] Application interface type [env var:
13                               GRANIAN_INTERFACE; default: (rsgi)]
14     --http [auto|1|2]          HTTP version [env var: GRANIAN_HTTP;
15                               default: (auto)]
16     --ws / --no-ws             Enable websockets handling [env var:
17                               GRANIAN_WEBSOCKETS; default: (enabled
18                               )]
19     --workers INTEGER RANGE     Number of worker processes [env var:
20                               GRANIAN_WORKERS; default: 1; x>=1]
21     --threads INTEGER RANGE     Number of threads [env var:
22                               GRANIAN_THREADS; default: 1; x>=1]
23     --blocking-threads INTEGER RANGE
```

```

23         Number of blocking threads [env var:
24         GRANIAN_BLOCKING_THREADS; default: 1;
           x>=1]
25     --threading-mode [runtime|workers]
26         Threading mode to use [env var:
27         GRANIAN_THREADING_MODE; default: (
           workers)]
28     --loop [auto|asyncio|uvloop] Event loop implementation [env var:
29         GRANIAN_LOOP; default: (auto)]
30     --opt / --no-opt Enable loop optimizations [env var:
31         GRANIAN_LOOP_OPT; default: (disabled)
           ]
32     --backlog INTEGER RANGE Maximum number of connections to hold
           in
33         backlog [env var: GRANIAN_BACKLOG;
           default:
34         1024; x>=128]
35     --http1-buffer-size INTEGER RANGE
36         Set the maximum buffer size for HTTP
           /1
37         connections [env var:
38         GRANIAN_HTTP1_BUFFER_SIZE; default:
           417792;
39         x>=8192]
40     --http1-keep-alive / --no-http1-keep-alive
41         Enables or disables HTTP/1 keep-alive
           [env
42         var: GRANIAN_HTTP1_KEEP_ALIVE;
           default:
43         (enabled)]
44     --http1-pipeline-flush / --no-http1-pipeline-flush
45         Aggregates HTTP/1 flushes to better
           support
46         pipelined responses (experimental) [
           env
47         var: GRANIAN_HTTP1_PIPELINE_FLUSH;
           default:
48         (disabled)]
49     --http2-adaptive-window / --no-http2-adaptive-window
50         Sets whether to use an adaptive flow
           control
51         for HTTP2 [env var:
52         GRANIAN_HTTP2_ADAPTIVE_WINDOW;
           default:
53         (disabled)]
54     --http2-initial-connection-window-size INTEGER
55         Sets the max connection-level flow
           control
56         for HTTP2 [env var:
           GRANIAN_HTTP2_INITIAL_C
57         ONNECTION_WINDOW_SIZE; default:

```

```

1048576]
58  --http2-initial-stream-window-size INTEGER
59      Sets the `
        SETTINGS_INITIAL_WINDOW_SIZE`
60      option for HTTP2 stream-level flow
        control
61      [env var:
62      GRANIAN_HTTP2_INITIAL_STREAM_WINDOW_SIZE
        ;
63      default: 1048576]
64  --http2-keep-alive-interval INTEGER
65      Sets an interval for HTTP2 Ping
        frames
66      should be sent to keep a connection
        alive
67      [env var:
        GRANIAN_HTTP2_KEEP_ALIVE_INTERVAL]
68  --http2-keep-alive-timeout INTEGER
69      Sets a timeout for receiving an
70      acknowledgement of the HTTP2 keep-
        alive ping
71      [env var:
        GRANIAN_HTTP2_KEEP_ALIVE_TIMEOUT;
72      default: 20]
73  --http2-max-concurrent-streams INTEGER
74      Sets the
        SETTINGS_MAX_CONCURRENT_STREAMS
75      option for HTTP2 connections [env
        var:
76      GRANIAN_HTTP2_MAX_CONCURRENT_STREAMS;
77      default: 200]
78  --http2-max-frame-size INTEGER Sets the maximum frame size to use
        for HTTP2
79      [env var:
        GRANIAN_HTTP2_MAX_FRAME_SIZE;
80      default: 16384]
81  --http2-max-headers-size INTEGER
82      Sets the max size of received header
        frames
83      [env var:
        GRANIAN_HTTP2_MAX_HEADERS_SIZE;
84      default: 16777216]
85  --http2-max-send-buffer-size INTEGER
86      Set the maximum write buffer size for
        each
87      HTTP/2 stream [env var:
88      GRANIAN_HTTP2_MAX_SEND_BUFFER_SIZE;
        default:
89      409600]
90  --log / --no-log Enable logging [env var:
91      GRANIAN_LOG_ENABLED; default: (

```

```

102         enabled))
103     --log-level [critical|error|warning|warn|info|debug]
104         Log level [env var:
105             GRANIAN_LOG_LEVEL;
106             default: (info)]
107     --log-config FILE
108         Logging configuration file (json) [
109             env var:
110             GRANIAN_LOG_CONFIG]
111     --ssl-keyfile FILE
112         SSL key file [env var:
113             GRANIAN_SSL_KEYFILE]
114     --ssl-certificate FILE
115         SSL certificate file [env var:
116             GRANIAN_SSL_CERTIFICATE]
117     --url-path-prefix TEXT
118         URL path prefix the app is mounted on
119         [env
120         var: GRANIAN_URL_PATH_PREFIX]
121     --respawn-failed-workers / --no-respawn-failed-workers
122         Enable workers respawn on unexpected
123         exit
124         [env var:
125             GRANIAN_RESPAWN_FAILED_WORKERS;
126             default: (disabled)]
127     --respawn-interval FLOAT
128         The number of seconds to sleep
129         between
130         workers respawn [env var:
131             GRANIAN_RESPAWN_INTERVAL; default:
132             3.5]
133     --reload / --no-reload
134         Enable auto reload on application's
135         files
136         changes (requires granian[reload]
137         extra)
138         [env var: GRANIAN_RELOAD; default:
139         (disabled)]
140     --process-name TEXT
141         Set a custom name for processes (
142         requires
143         granian[pname] extra) [env var:
144             GRANIAN_PROCESS_NAME]
145     --version
146         Show the version and exit.
147     --help
148         Show this message and exit.

```

Threading mode

Granian offers two different threading paradigms, due to the fact the inner Rust runtime can be multi-threaded – in opposition to what happens in Python event-loop which can only run as a single thread.

Given you specify N threads with the relevant option, in **workers** threading mode Granian will spawn N single-threaded Rust runtimes, while in **runtime** threading mode Granian will spawn a single multi-threaded runtime with N threads.

Benchmarks suggests **workers** mode to be more efficient with a small amount of processes, while **runtime** mode seems to scale more efficiently where you have a large number of CPUs. Real performance will though depend on specific application code, and thus *your mileage might vary*.

Event loop optimizations

With the `--opt` option Granian will use custom task handlers for Python coroutines and awaitables to improve Python code execution. Due to the nature of such handlers some libraries and specific application code relying on `asyncio` internals might not work.

You might test the effect such optimizations cause over your application and decide whether to enable 'em or leave 'em disabled (as per default).

Project status

Granian is currently under active development.

Granian is compatible with Python 3.8 and above versions.

License

Granian is released under the BSD License.