basic info about the application
- application : exe
- running on linux
- X86 64 bit server
- docker containers
- 3 copies of the database synchronized via kafka : 1 as master, 2 as slaves

use of the webserver
- we need a webserver to act as a gateway of a cpp exe with an embedded key value
  - the webserver, as well as the kv itself, will be embedded in the same exe
- webserver acting as a classic webserver
  - no load balancing
  - no reverse proxy
  - no caching
  - only request processing

specs of the webserver
- 1 main thread
  - the main thread accepts connections and distributes them among the queue of each worker
  - no connection to PHP, ..
  - the request are distributed to workers depending the command type
- N workers
  - the workers are the threads of the webserver
  - usually READ1, READ2, WRITE, MAIN, SUBMIT, REP
  - each worker has access to the C++ classes needed to access the KV database
- queues per worker
  - each worker with its own queue and only one
  - sequential processing of the requests in the queue
    - each worker don´t start a new request until the one in process is finished
    - no callback, ..
- other specs
  - HTTP 1.1
  - UTF-8
  - IPv6
  - requests via REST
  - only 1 port : 80

process of a request
- the webserver main thread gets a request
- the main thread of the webserver parses the request and gets the cmd
  - if the command starts with W(rite)
    - if the server is in master state
      - passes the request to the Write queue (write thread)
      - activates the event in this worker
    - if the webserver is in slave mode
      - return the request with status = XX
  - if the command starts with S(ubmit)
    - passes the request to the Submit queue (submit thread)
    - activates the event in this worker
  - if the command starts with M(ain)
    - passes the request to the Main queue (main thread)
    - activates the event in this worker

if the command starts with G(et)
if the webserver is in slave mode
return the request with status = XX
if the server is in master state
does random of 1 and total number of read workers, usually 2
get the random number
passes the request to the associated read queue (read1, read2)
activates the event in this worker
if the command starts with [THXXX] (the XX indicates the queue number & thread)
//request sent to a specific queue
passes the cmd to the XX queue & thread
the cmd could be
slot commando
master slave mode command
activates the event in this worker
if the command starts with R(eporting)
passes the request to the Main queue (main thread)
activates the event in this worker
if the command starts with [K]
//used for Kafka
//kafka changes the mode of each replicated server
stores the variable of the cmd in the webserver

all these threads access the kv database
the threads process the command + data in json
the threads answer with another json
the webserver resend the answer and closes the socket

load related specs
the request are all data oriented
no files, no images, no caching, no web page processing, …
type of requests
all string & number comparison
low resource needed per request
usually less than 1 mseg per request (except requests with big swapping)
request load
max 2.000 request per second (adding all the threads)
usually few hundreds request per second
request origin
the webserver will work in the internal part of the datacenter : the request send to the kv database will be generated by other nodejs and PHP servers

webserver footprint
the webserver don´t need all the functions and security levels usually associated with a generic webserver
I would prefer a small as possible
no compression (gzip, ..)
no authentication
no SSL
no files downloading
no ..
Is convenient to take out all the NOT needed code or is better to maintain the actual footprint (no code modification) to avoid potential errors ?