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
- 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 only 5 : READ1, READ2, WRITE, MAIN, SUBMIT
  - each worker has access to the C++ classes needed to access the KV database
- queues per worker
  - each worker with its own queue
  - sequential processing of the requests
    - 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 GET

- the webserver, as well as the kv itself, will be embedded in the same exe

Process of a request
- the webserver main thread gets a request
- the main thread checks if the exe is in master mode

  - if the server is in slave state
    - the request is redirected to the actual master of the domain

  - if the server is in master state
    - the main thread parses the request and obtain the cmd

    - if the command starts with W
      - passes the request to the Write queue (write thread)
      - activates the event in this worker

```
                if the command starts with S
                        passes the request to the Submit queue  (submit thread)
                        activates the event in this worker

                if the command starts with M
                        passes the request to the Main queue (main thread)
                        activates the event in this worker

                if the command starts with R
                        does random of 1 and total number of read workers, usually 2
                        get the random number
                        passes the request to the associated queue (read01, read02) (read
                        thread)
                        activates the event in this worker

                if the command starts with [XXXXXX] (the XX indicates the queue number
                of the read thread)
                        passes the cmd to the XX queue of the read thread
                        activates the event in this worker

        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 and number comparison
  - low resource needed per request
    - usually less than 1 mseg per request (except request with huge 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 ?