

Introducing `piladb` to the Gopher

by Fernando Álvarez
Golang Barcelona — 21/03/2017

Summary

- ``whoami``

Summary

- ``whoami``
- Introduction to **piladb**

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo

Summary


- ``whoami``
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo
- Future plans

Summary

- ``whoami``
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo
- Future plans
- Help!

`whoami`

Fernando Álvarez

- Software Engineer from Madrid
- Infrastructure at  **BeBanjo**
- Gopher since 2013
- Open Source at oscillatingworks
- Author of **piladb**
- @fern4lvarez

piladb *[pee-lah-dee-bee].*



piladb



piladb



battery

piladb

charge



battery

piladb

energy

battery

charge



piladb

energy

battery

charge



superpowers

~~piladb~~

~~energy~~

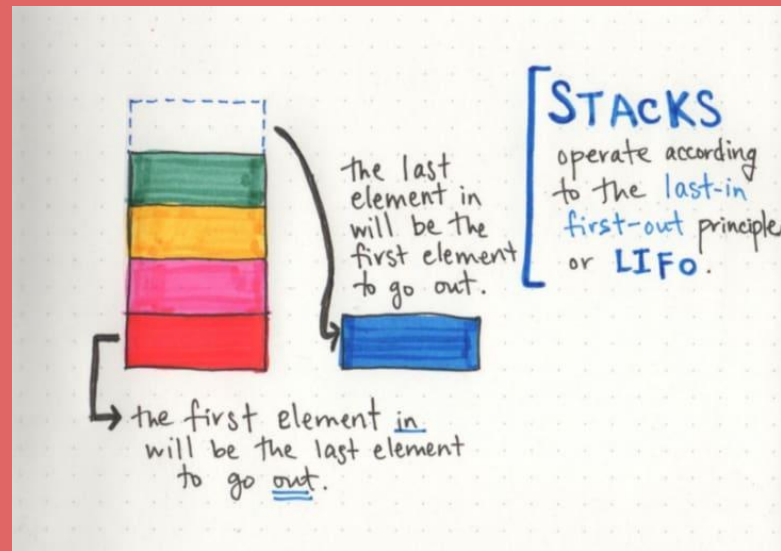
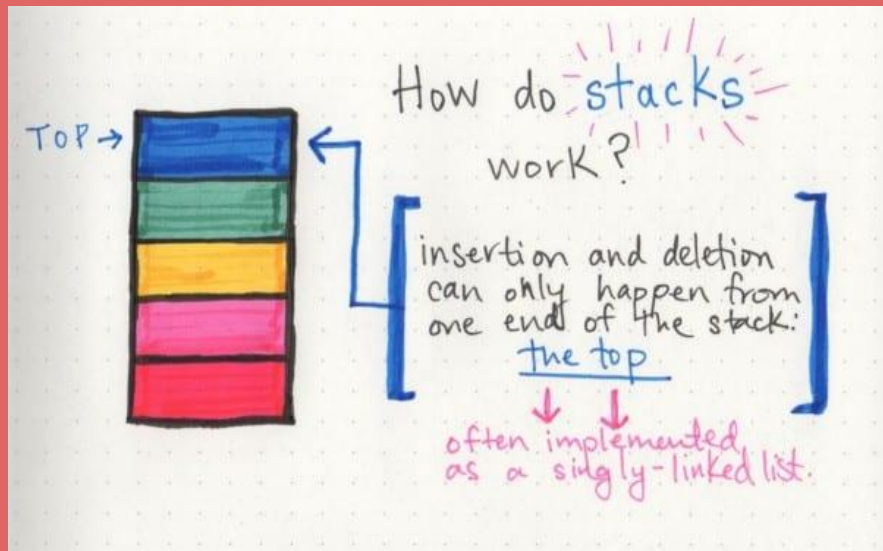
~~battery~~

~~charge~~

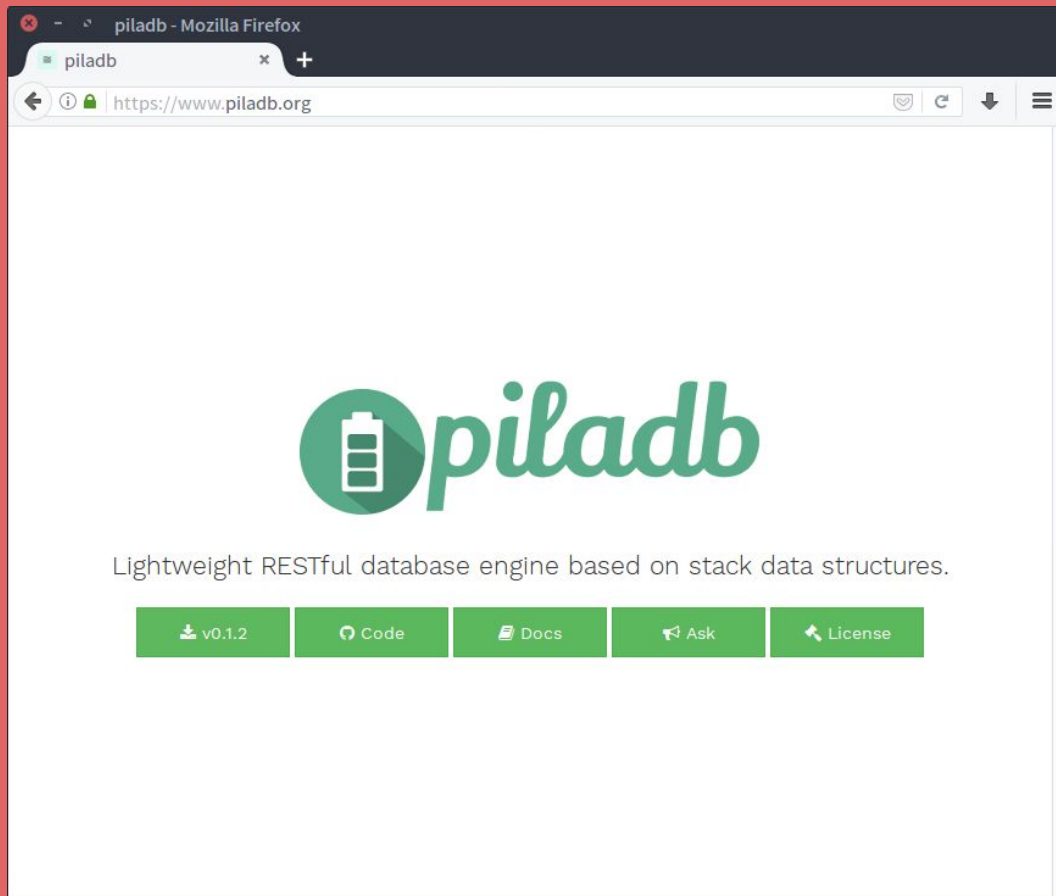


~~superpowers~~

stack data structure



<https://dev.to/vaidehijoshi/stacks-and-overflows>



database engine written in Go

lightweight and fast*

* no benchmarks yet, but feels fast!


```
piladb|master ⇒ find . -name '*.go' | grep -v vendor/ | grep -v "_test.go" | xargs wc -l
 17 ./pkg/stack/stacker.go
 83 ./pkg/stack/stack.go
 35 ./pkg/version/version.go
  9 ./pkg/date/date.go
 31 ./pkg/uuid/uuid.go
107 ./pila/pila.go
165 ./pila/stack.go
145 ./pila/database.go
 65 ./pila/stack_status.go
  9 ./main.go
405 ./pilad/conn.go
 62 ./pilad/router.go
 25 ./pilad/logo.go
148 ./pilad/config.go
 68 ./pilad/utils.go
 30 ./pilad/main.go
 61 ./pilad/status.go
 65 ./config/value.go
 45 ./config/config.go
 59 ./config/vars/vars.go
1634 total
```

Latest release

v0.1.2

78929cb

Verified

Version 0.1.2

Edit

 fern4lvarez released this 13 days ago · 4 commits to master since this release







Everyday is like a release on Sunday 🎵

What's new?

- Fix decoding bug when pushing a malformed payload
- New shell client: <https://github.com/oscillatingworks/piladb-sh>
- pilad: Add Go Version to Status
- pilad: Add /_ping endpoint
- godoc: Extend packages documentation
- vendor: Update dependencies
- dev: Add pila.sh utilities to Docker image
- pilad: Show links of interest in / endpoint.

More about this release here: <https://blog.oscillating.works/piladb-0-1-2-released-announcing-piladb-sh-and-more-89d742ea55bf#.9pra33e41>

Downloads

 piladb0.1.2.darwin-amd64.tar.gz	2.25 MB
 piladb0.1.2.darwin-amd64.zip	2.25 MB
 piladb0.1.2.linux-amd64.tar.gz	2.27 MB
 piladb0.1.2.linux-amd64.zip	2.27 MB
 Source code (zip)	
 Source code (tar.gz)	

RESTful communication

PUSH, POP, PEEK, SIZE, FLUSH

JSON compatible elements

strings, numbers, objects, arrays, booleans, *null* ^[1]

[1] https://www.w3schools.com/js/js_json_datatypes.asp

no configuration files

no configuration files

environment variables

CLI parameters

inject via REST API

100% test coverage ^[1]

[1] <https://codecov.io/gh/fern4lvarez/piladb>

in-memory store

context and use cases

context

- Project started in October 2015
- More a Proof of Concept, less a solution to a problem
- Solo, side project
- First release in December 2016 (0.1.0)
- Main influences: Redis, Elasticsearch, SQLite, BoltDB
- Not persistent, not distributed, *yet*

use cases

- Caching system
 - Invalidation using dates
 - All read and write ops are $O(1)$

use cases

- Caching system
 - Invalidation using dates
 - All read and write ops are $O(1)$
- Key-Value store with version history
 - Key: name of a Stack, Value: elements of the Stack

use cases

- Caching system
 - Invalidation using dates
 - All read and write ops are $O(1)$
- Key-Value store with version history
 - Key: name of a Stack, Value: elements of the Stack
- Undo/Redo mechanism

use cases

- Caching system
 - Invalidation using dates
 - All read and write ops are $O(1)$
- Key-Value store with version history
 - Key: name of a Stack, Value: elements of the Stack
- Undo/Redo mechanism
- Message processing, Pub/Sub, etc.

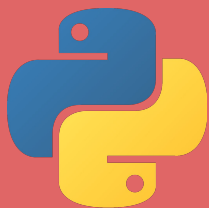
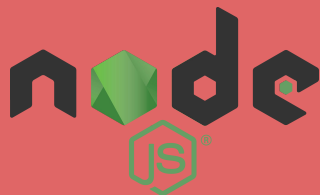
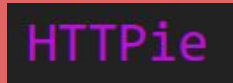
functionality and main components


```

~ =>
~ => pilad
2017/03/18 11:05:49
2017/03/18 11:05:49      d8b 888      888 888
2017/03/18 11:05:49      Y8P 888      888 888
2017/03/18 11:05:49      888      888 888
2017/03/18 11:05:49 888888b. 888 888 8888b. .d88888 888888b.
2017/03/18 11:05:49 888 "88b 888 888 "88b d88" 888 888 "88b
2017/03/18 11:05:49 888 888 888 888 .d888888 888 888 888 888
2017/03/18 11:05:49 888 d88P 888 888 888 888 Y88b 888 888 d88P
2017/03/18 11:05:49 88888P" 888 888 "Y888888 "Y88888 88888P"
2017/03/18 11:05:49 888
2017/03/18 11:05:49 888
2017/03/18 11:05:49 888
2017/03/18 11:05:49
2017/03/18 11:05:49 Version:      0.1.2
2017/03/18 11:05:49 Go Version:   go1.8
2017/03/18 11:05:49 Host:        linux_amd64
2017/03/18 11:05:49 Port:        1205
2017/03/18 11:05:49 PID:         11106
2017/03/18 11:05:49

```

with your favourite HTTP client...



CREATE DATABASE:

```
PUT /databases?name=MYDATABASE
```

```
{  
  "id": "22ffa4116b38da7988ebe505f9e129ba",  
  "name": "MYDATABASE",  
  "number_of_stacks": 0  
}
```

CREATE STACK:

PUT /databases/MYDATABASE/stacks?name=MYSTACK

```
{  
  "id": "0c39814bad28d8b2ec5b4d697701c125",  
  "name": "MYSTACK",  
  "peek": null,  
  "size": 0,  
  "created_at": "2017-02-15T03:35:22.446530542+01:00",  
  "updated_at": "2017-02-15T03:35:22.446530542+01:00",  
  "read_at": "2017-02-15T03:35:22.446530542+01:00"  
}
```

PUSH:

```
POST /databases/MYDATABASE/stacks/MYSTACK {"element":"this is an element"}
```

```
{  
  "id": "0c39814bad28d8b2ec5b4d697701c125",  
  "name": "MYSTACK",  
  "peek": "this is an element",  
  "size": 1,  
  "created_at": "2017-02-15T03:35:22.446530542+01:00",  
  "updated_at": "2017-02-15T03:45:30.264123933+01:00",  
  "read_at": "2017-02-15T03:46:18.060685036+01:00"  
}
```

PUSH x 3:

```
POST /databases/MYDATABASE/stacks/MYSTACK {"element":"this is the 4th element"}
```

```
{  
  "id": "0c39814bad28d8b2ec5b4d697701c125",  
  "name": "MYSTACK",  
  "peek": "this is the 4th element",  
  "size": 4,  
  "created_at": "2017-02-15T03:35:22.446530542+01:00",  
  "updated_at": "2017-02-15T03:48:47.699231061+01:00",  
  "read_at": "2017-02-15T03:48:50.109298271+01:00"  
}
```

POP: DELETE /databases/MYDATABASE/stacks/MYSTACK

```
{  
  "id": "0c39814bad28d8b2ec5b4d697701c125",  
  "name": "MYSTACK",  
  "peek": "this is the 3rd element",  
  "size": 3,  
  "created_at": "2017-02-15T03:35:22.446530542+01:00",  
  "updated_at": "2017-02-15T03:50:02.882637946+01:00",  
  "read_at": "2017-02-15T03:50:34.287258581+01:00"  
}
```


PEEK: GET /databases/MYDATABASE/stacks/MYSTACK?peek

```
{  
  "element": "this is the 3rd element"  
}
```

FLUSH: DELETE /databases/MYDATABASE/stacks/MYSTACK?flush

```
{  
  "id": "0c39814bad28d8b2ec5b4d697701c125",  
  "name": "MYSTACK",  
  "peek": null,  
  "size": 0,  
  "created_at": "2017-02-15T03:35:22.446530542+01:00",  
  "updated_at": "2017-02-15T03:52:27.494852286+01:00",  
  "read_at": "2017-02-15T03:52:27.494852286+01:00"  
}
```

stack

- LIFO: Last in, First out
- Stores JSON-compatible elements
- Unique name and identifier
- Created by user

database

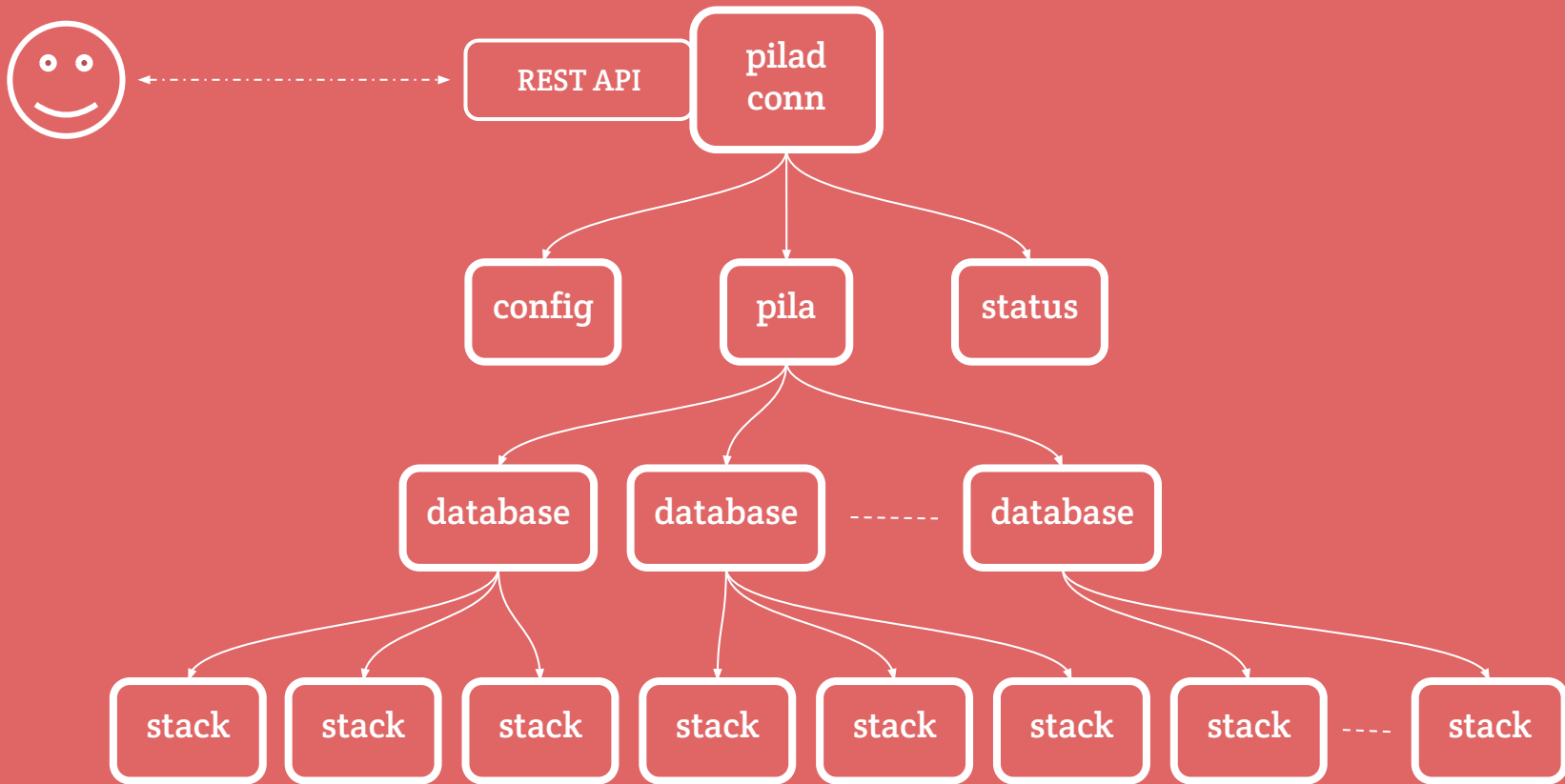
- Contains Stacks
- Unique name and identifier
- Created by user

pila

- Main entity of the engine
- Contains Databases
- Created by pilad on startup
- github.com/fern4lvarez/piladb/pila

pilad

- Daemon that starts piladb engine
- Implements REST API + configuration
- Creates a single instance of the Pila entity



internals in Go

implementation of a Stack

implementation is decoupled from the type

```
15 // Stack represents a stack entity in piladb.
16 type Stack struct {
17     // ID is a unique identifier of the Stack
18     ID fmt.Stringer
19
20     // Name of the Stack
21     Name string
22
23     // Database associated to the Stack
24     Database *Database
25
26     // CreatedAt represents the date when the Stack was created
27     CreatedAt time.Time
28
29     // UpdatedAt represents the date when the Stack was updated for the last time.
30     // This date must be updated when a Stack is created, and when receives a PUSH,
31     // POP, or FLUSH operation.
32     // Note that unlike CreatedAt, UpdatedAt is not triggered automatically
33     // when one of these events happens, but it needs to be set by hand.
34     UpdatedAt time.Time
35
36     // ReadAt represents the date when the Stack was read for the last time.
37     // This date must be updated when a Stack is created, accessed, and when it
38     // receives a PUSH, POP, or FLUSH operation.
39     // Note that unlike CreatedAt, ReadAt is not triggered automatically
40     // when one of these events happens, but it needs to be set by hand.
41     ReadAt time.Time
42
43     // base represents the Stack data structure
44     base stack.Stacker
45 }
```

./pila/stack.go


interface that contains all Stack operations

```
1 package stack
2
3 // Stacker represents an interface that contains all the
4 // required methods to implement a Stack that can be
5 // used in piladb.
6 type Stacker interface {
7     // Push an element into a Stack
8     Push(element interface{})
9     // Pop the topmost element of a stack
10    Pop() (interface{}, bool)
11    // Size returns the size of the Stack
12    Size() int
13    // Peek returns the topmost element of the Stack
14    Peek() interface{}
15    // Flush flushes a Stack
16    Flush()
17 }
```

`./pkg/stack/stacker.go`

use another Stack implementation by changing one parameter

```
47 // NewStack creates a new Stack given a name and a creation date,  
48 // without an association to any Database. It uses the default  
49 // ./pkg/stack implementation as a base Stack.  
50 func NewStack(name string, t time.Time) *Stack {  
51     return NewStackWithBase(name, t, stack.NewStack())  
52 }  
53  
54 // NewStackWithBase creates a new Stack given a name, a creation date,  
55 // and a stack.Stacker base implementation, without an association to any Database.  
56 func NewStackWithBase(name string, t time.Time, base stack.Stacker) *Stack {  
57     s := &Stack{}  
58     s.Name = name  
59     s.SetID()  
60     s.CreatedAt = t  
61     s.base = base  
62     return s  
63 }
```



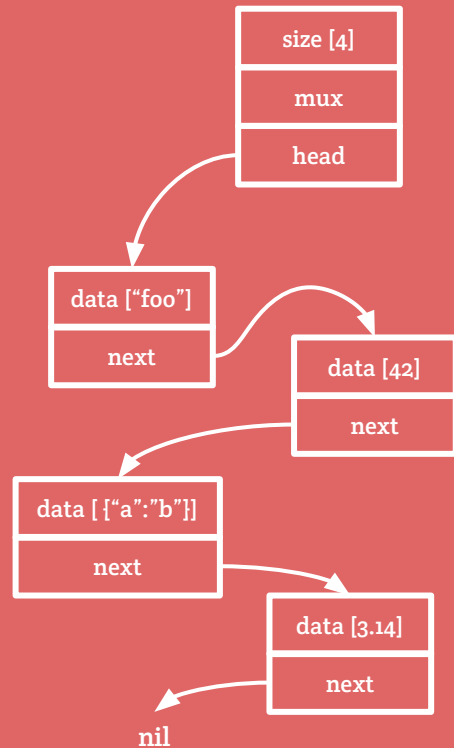
./pila/stack.go

current implementation uses linked lists

implementation using linked lists

```
7 // Stack implements the Stacker interface, and represents the stack
8 // data structure as a linked list, containing a pointer
9 // to the first Frame as a head and the size of the stack.
10 // It also contain a mutex to lock and unlock
11 // the access to the stack at I/O operations.
12 type Stack struct {
13     head *frame
14     size int
15     mux  sync.Mutex
16 }
17
18 // frame represents an element of the stack. It contains
19 // data and the link to the next Frame as a pointer.
20 type frame struct {
21     data interface{}
22     next *frame
23 }
```

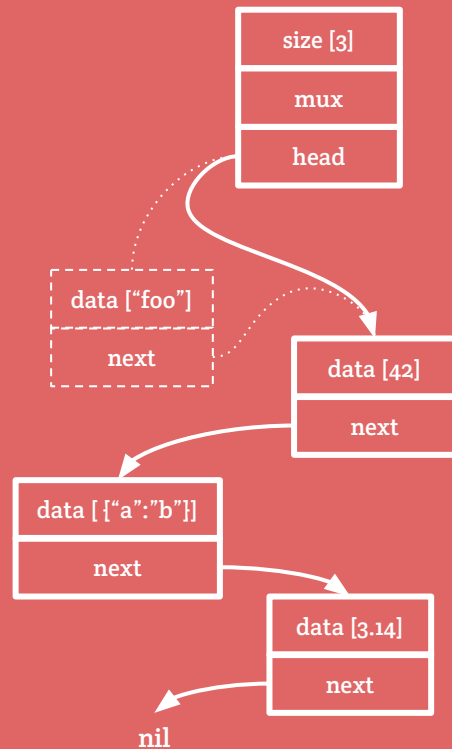
./pkg/stack/stack.go



implementation using linked lists

```
46 // Pop removes and returns the element on top of the stack,  
47 // updating its head to the next Frame. If the stack was empty,  
48 // it returns false.  
49 func (s *Stack) Pop() (interface{}, bool) {  
50     s.mux.Lock()  
51     defer s.mux.Unlock()  
52  
53     if s.head == nil {  
54         return nil, false  
55     }  
56  
57     element := s.head.data  
58     s.head = s.head.next  
59     s.size--  
60     return element, true  
61 }
```

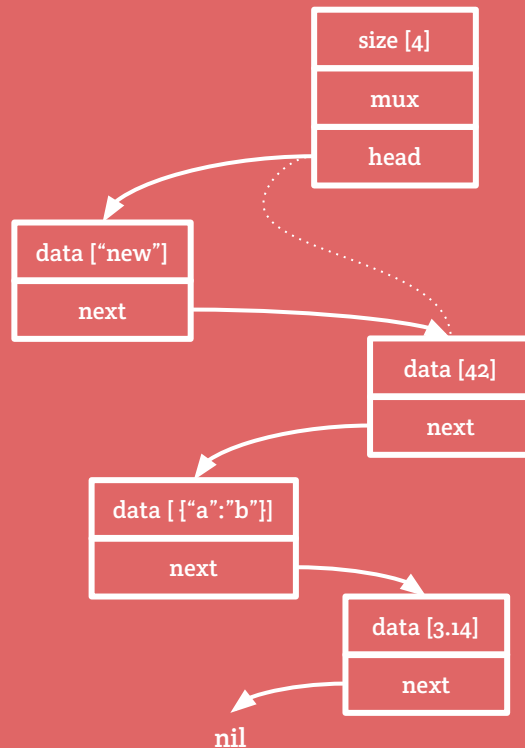
./pkg/stack/stack.go



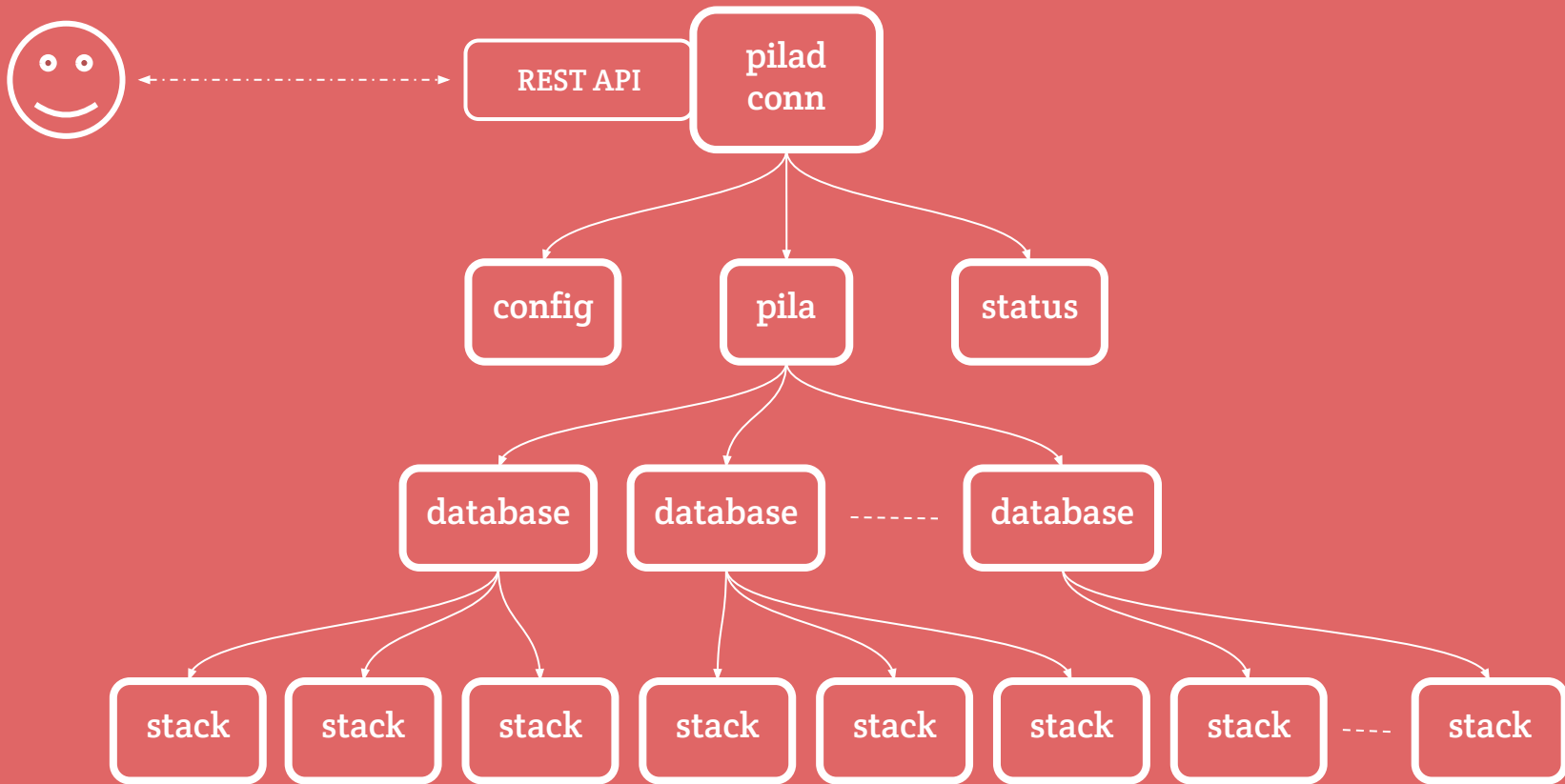
implementation using linked lists

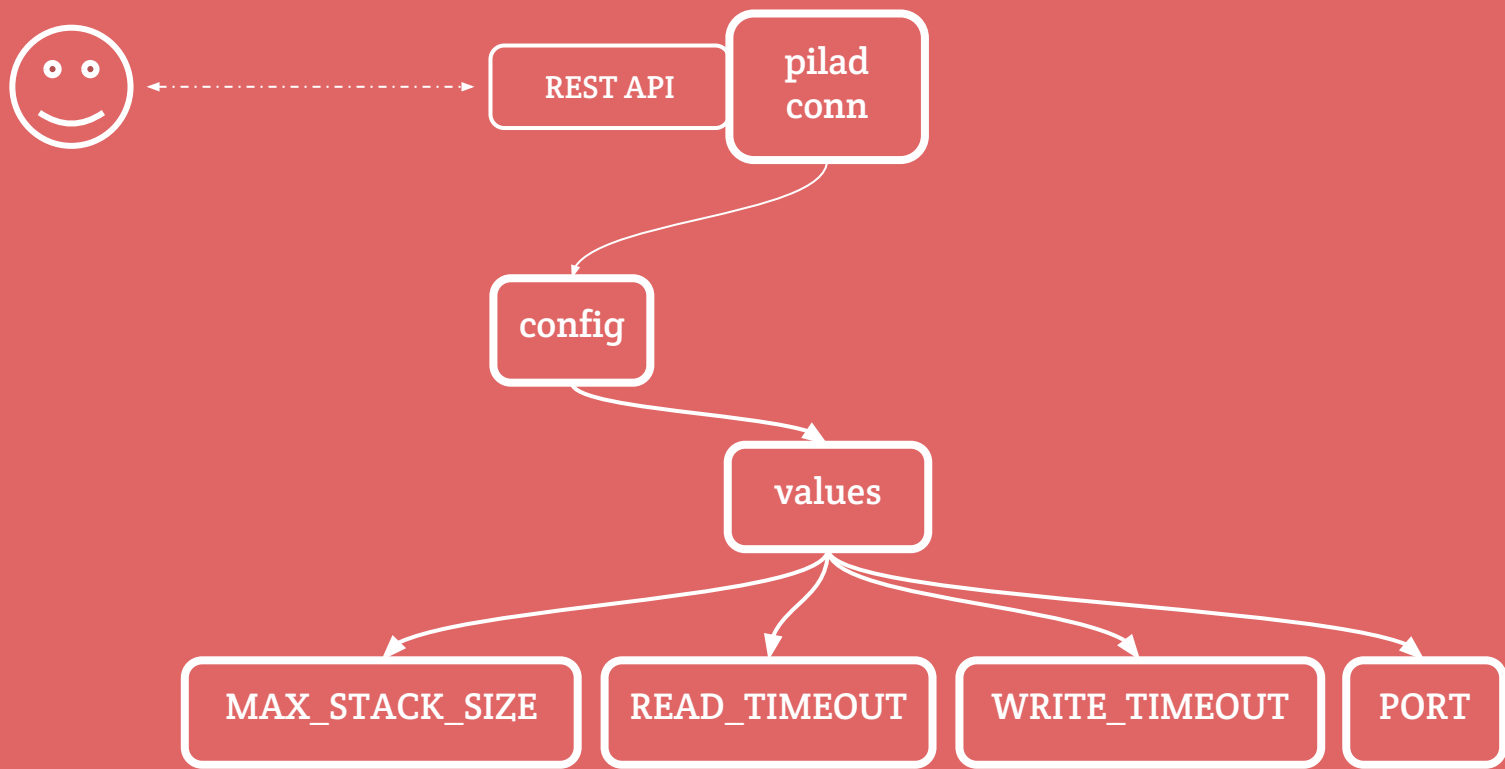
```
31 // Push adds a new element on top of the stack, creating
32 // a new head holding this data and updating its head to
33 // the previous stack's head.
34 func (s *Stack) Push(element interface{}) {
35     s.mux.Lock()
36     defer s.mux.Unlock()
37
38     head := &frame{
39         data: element,
40         next: s.head,
41     }
42     s.head = head
43     s.size++
44 }
```

./pkg/stack/stack.go



implementation of Config





config is like a Pila with a single Database

```
15 // Config represents a Database containing all
16 // configuration values that will be
17 // updated and consumed by piladb.
18 type Config struct {
19     Values *pila.Database
20 }
```

```
35 // Get gets a config value from a key.
36 func (c *Config) Get(key string) interface{} {
37     s, ok := c.Values.Stacks[uuid.New(CONFIG+key)]
38     if !ok {
39         return nil
40     }
41     return s.Peek()
42 }
43
44 // Set sets a config value having a key and the value.
45 func (c *Config) Set(key string, value interface{}) {
46     s, ok := c.Values.Stacks[uuid.New(CONFIG+key)]
47     if !ok {
48         sID := c.Values.CreateStack(key, time.Now().UTC())
49         s, _ = c.Values.Stacks[sID]
50     }
51     s.Push(value)
52 }
53 }
```

`./config/config.go`

set or modify a config var on runtime:

POST /_config/READ_TIMEOUT {"element":15}

```
104         if r.Method == "POST" {
105             if r.Body == nil {
106                 log.Println(r.Method, r.URL, http.StatusBadRequest,
107                     "no element provided")
108                 w.WriteHeader(http.StatusBadRequest)
109                 return
110             }
111             err := element.Decode(r.Body)
112             if err != nil {
113                 log.Println(r.Method, r.URL, http.StatusBadRequest,
114                     "error on decoding element:", err)
115                 w.WriteHeader(http.StatusBadRequest)
116                 return
117             }
118
119             c.Config.Set(vars["key"], element.Value)
120         }
```

./pilad/config.go

export a func per config value

```
25 // WriteTimeout returns the value of WRITE_TIMEOUT.
26 // Type: time.Duration, Default: 45
27 func (c *Config) WriteTimeout() time.Duration {
28     writeTimeout := c.Get(vars.WriteTimeout)
29     t := intValue(writeTimeout, vars.WriteTimeoutDefault)
30     return time.Duration(t)
31 }
32
33 // Port returns the value of PORT.
34 // Type: int, Default: 1205
35 func (c *Config) Port() int {
36     port := c.Get(vars.Port)
37     t := intValue(port, vars.PortDefault)
38
39     if t < 1025 || t > 65536 {
40         return vars.PortDefault
41     }
42     return t
43 }
```

./config/value.go

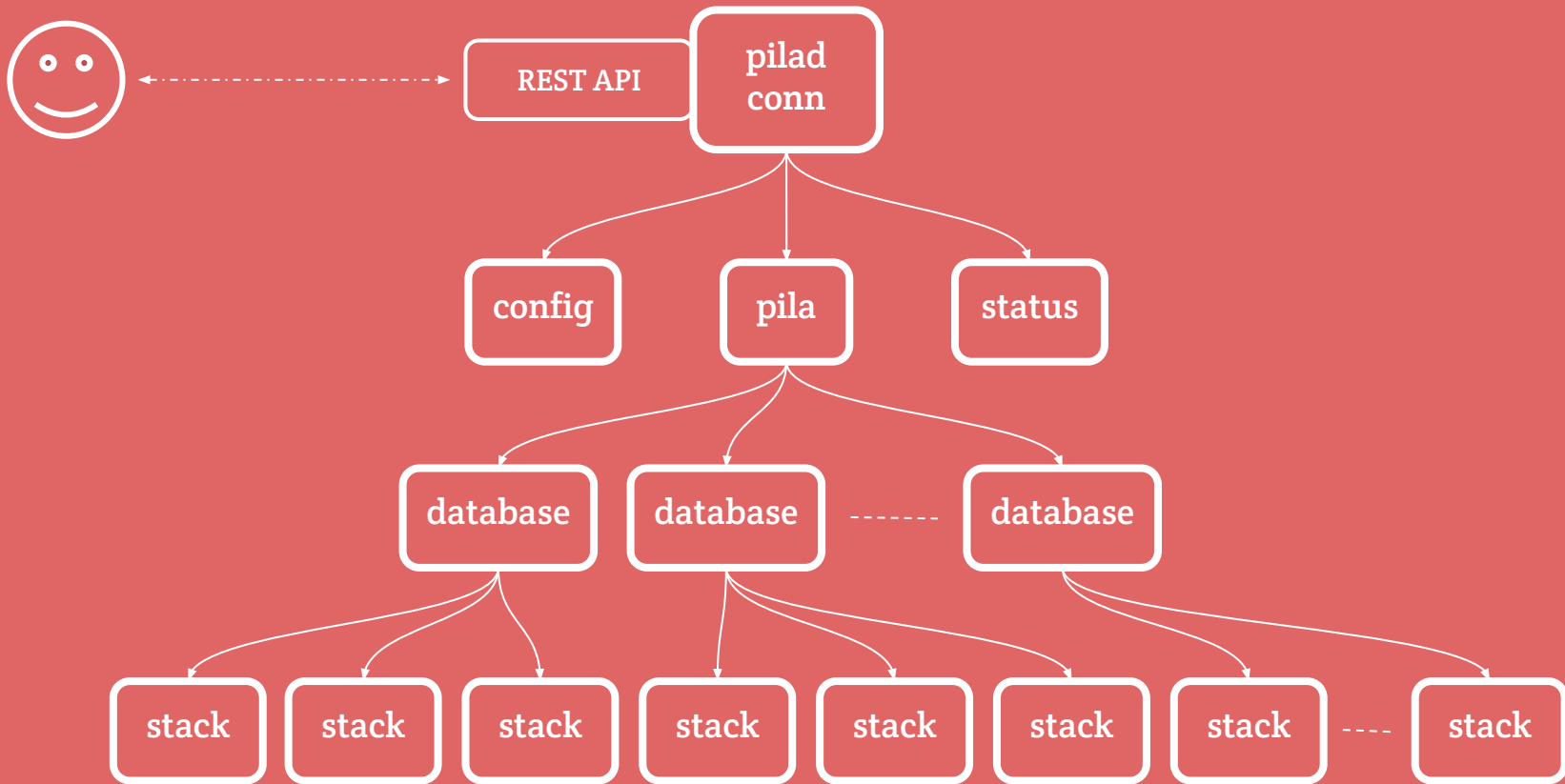
read config values on runtime: PUSH

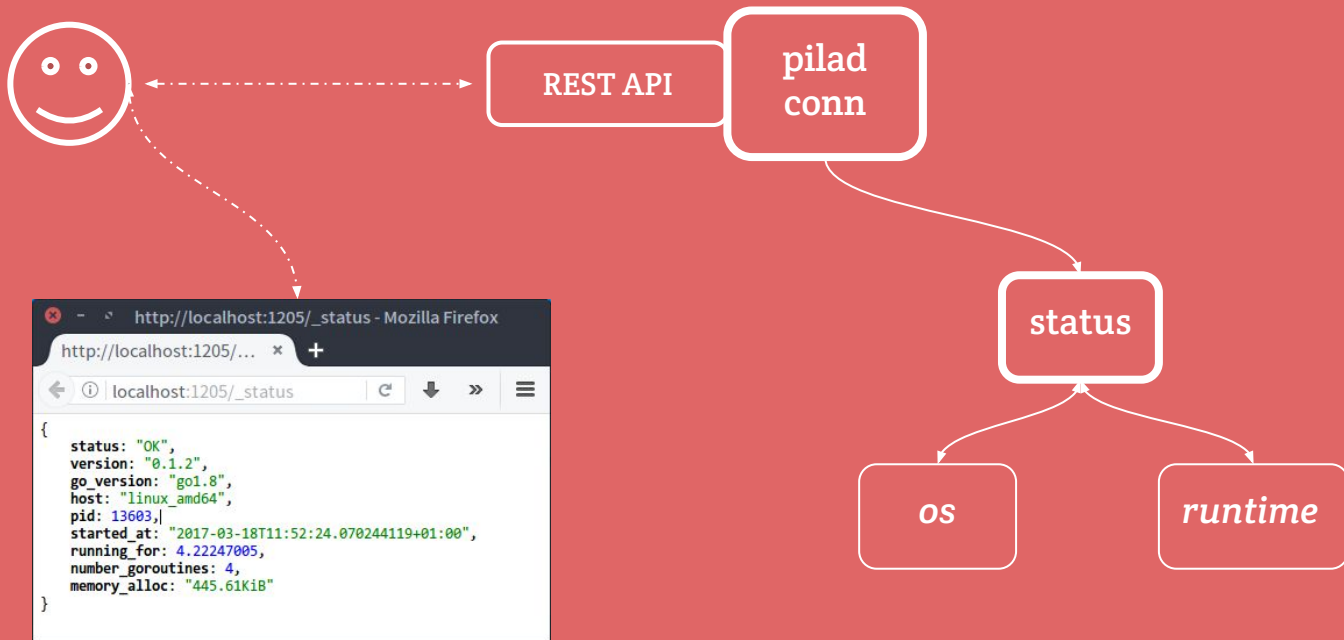
```
246         case r.Method == "POST":
247             c.checkMaxStackSize(c.pushStackHandler)(w, r, stack)
248         return
```

```
136 // checkMaxStackSize checks config value for MaxStackSize and execute the
137 // wrapped handler if check is validated.
138 func (c *Conn) checkMaxStackSize(handler stackHandlerFunc) stackHandlerFunc {
139     return func(w http.ResponseWriter, r *http.Request, stack *pila.Stack) {
140         if s := c.Config.MaxStackSize(); stack.Size() >= s && s != -1 {
141             log.Println(r.Method, r.URL, http.StatusNotAcceptable, vars.MaxStackSize, "value reached")
142             w.WriteHeader(http.StatusNotAcceptable)
143             return
144         }
145         handler(w, r, stack)
146     }
147 }
148 }
```

./pilad/config.go

implementation of Status





OS

- `os.Getpid()`: piladb process ID

runtime

- `runtime.GOOS`: Host operating system
- `runtime.GOARCH`: Host architecture
- `runtime.NumGoroutine()`: Number of existing goroutines
- `runtime.MemStats`: Statistics about memory allocator



Dave Cheney

@davecheney

Following

#golang top tip: operating systems misreport
your apps' mem usage. Use
runtime.ReadMemStats to get the right number.
[golang.org/pkg/runtime/#M...](https://golang.org/pkg/runtime/#MemStats)

RETWEETS

31

LIKES

80



7:32 AM - 16 Mar 2017



<https://twitter.com/davecheney/status/842262394139172864>

```

115 // A MemStats records statistics about the memory allocator.
116 type MemStats struct {
117     // General statistics.
118     Alloc      uint64 // bytes allocated and not yet freed
119     TotalAlloc uint64 // bytes allocated (even if freed)
120     Sys        uint64 // bytes obtained from system (sum of XxxSys below)
121     Lookups    uint64 // number of pointer lookups
122     Mallocs    uint64 // number of mallocs
123     Frees      uint64 // number of frees
124
125     // Main allocation heap statistics.
126     HeapAlloc  uint64 // bytes allocated and not yet freed (same as Alloc above)
127     HeapSys    uint64 // bytes obtained from system
128     HeapIdle   uint64 // bytes in idle spans
129     HeapInuse  uint64 // bytes in non-idle span
130     HeapReleased uint64 // bytes released to the OS
131     HeapObjects uint64 // total number of allocated objects
132
133     // Low-level fixed-size structure allocator statistics.
134     // Inuse is bytes used now.
135     // Sys is bytes obtained from system.
136     StackInuse uint64 // bytes used by stack allocator
137     StackSys    uint64
138     MSpanInuse  uint64 // mspan structures
139     MSpanSys    uint64
140     MCacheInuse uint64 // mcache structures
141     MCacheSys   uint64
142     BuckHashSys uint64 // profiling bucket hash table
143     GCSys       uint64 // GC metadata
144     OtherSys    uint64 // other system allocations
145
146     // Garbage collector statistics.
147     NextGC      uint64 // next collection will happen when HeapAlloc ≥ this amount
148     LastGC      uint64 // end time of last collection (nanoseconds since 1970)
149     PauseTotalNs uint64
150     PauseNs      [256]uint64 // circular buffer of recent GC pause durations, most recent at [(NumGC+255)%256]
151     PauseEnd     [256]uint64 // circular buffer of recent GC pause end times
152     NumGC        uint32
153     GCPUFraction float64 // fraction of CPU time used by GC
154     EnableGC     bool
155     DebugGC      bool
156
157     // Per-size allocation statistics.
158     // 61 is NumSizeClasses in the C code.
159     BySize [61]struct {
160         Size      uint32
161         Mallocs   uint64
162         Frees     uint64
163     }
164 }

```

[go/src/runtime/memstats.go](https://golang.org/src/runtime/memstats.go)

pull MemStats on each request

```
45 // statusHandler writes the piladb status into the response.
46 func (c *Conn) statusHandler(w http.ResponseWriter, r *http.Request) {
47     c.Status.Update(time.Now().UTC(), MemStats())
48
49     w.Header().Set("Content-Type", "application/json")
50     log.Println(r.Method, r.URL, http.StatusOK)
51     w.Write(c.Status.ToJSON())
52 }
```

./pilad/conn.go

```
40 // Update updates the Status given a current time and memory
41 // stats.
42 func (s *Status) Update(now time.Time, mem *runtime.MemStats) {
43     s.RunningFor = now.Sub(s.StartedAt).Seconds()
44     s.NumberGoroutines = runtime.NumGoroutine()
45     s.MemoryAlloc = MemOutput(mem.Alloc)
46 }
```

./pilad/status.go

```
36 // MemStats fetches the memory statistics provided
37 // by the Go stdlib.
38 func MemStats() *runtime.MemStats {
39     var mem runtime.MemStats
40     runtime.ReadMemStats(&mem)
41     return &mem
42 }
```

./pilad/utils.go

demo

future plans

O.1.X

- More docs!
 - Internals
 - Configuration
 - Go package
- Go client library

0.2.x

- New Stack operations
 - ROTATE, BASE, BLOCK, CONCAT
 - Spoiler! Stacks will be Queues too
- Option to allow pushing to Stack when it is full
 - Bottommost element will be removed
- More clients libraries?
 - Ruby, JS, Python, PHP, Java...

0.3.x

- Version control
 - Using activity logs
- Restore from any point in time
 - Persistence!
- Authentication
- New logo
 - Bye battery!

Future

- Replication
- pilaql (query language)
- TLS
- JSON API
- ... and more!

help!

oscillatingworks / [piladb-go](#)

Unwatch ▾

2

★ Star

0

🔗 Fork

0

<> Code

🔔 Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📈 Pulse

📊 Graphs

⚙️ Settings

Official piladb client library in Go <https://www.piladb.org>

Edit

[piladb](#) [database](#) [client](#) [go](#) [golang](#) [http](#) [stack](#) [Manage topics](#)

📁 2 commits

🌿 1 branch

📦 0 releases

👤 1 contributor

📄 MIT

Branch: master ▾

New pull request

Create new file


Upload files

Find file

Clone or download ▾

 **fern4lvarez** committed on **GitHub** Update README.md

Latest commit 55d867a 2 days ago

 [.gitignore](#) Initial commit 2 days ago

 [LICENSE](#) Initial commit 2 days ago

 [README.md](#) Update README.md 2 days ago

📖 README.md

piladb-go

Official piladb client library in Go

all it takes to get started

```
go get -u github.com/fern4lvarez/piladb/...  
cd $GOPATH/src/github.com/fern4lvarez/piladb  
make test
```

<https://www.piladb.org>

<https://github.com/fern4lvarez/piladb>

<https://docs.piladb.org>

<https://www.reddit.com/r/piladb/>

<https://www.oscillating.works>

@oscillatingw

thank you!
questions?