Server BenchMark

Base on golang

- Pre-estimate the occupation of service resources, and rationally choose to configure the machine
- Pressure measurement in the case of concurrency

EC2 Instance Info goto

- GENERAL PURPOSE
- COMPUTE OPTIMIZED
- MEMORY OPTIMIZED
- ACCELERATED COMPUTING
- STORAGE OPTIMIZED

Bench Tool (boom)

Http interface pressure measurement

install

```
#install pip tool (package install)
 brew install pip
 #install boom
 pip install boom
```

example(bash)

```
boom http://localhost:80 -c 10 -n 100
```

comment:

-c CONCURRENCY = 10

-n REQUESTS = 100

usage

```
boom --help
usage: boom [-h] [--version] [-m {GET,POST,DELETE,PUT,HEAD,OPTIONS}]
            [--content-type CONTENT_TYPE] [-D DATA] [-c CONCURRENCY] [-a
AUTH]
            [--header HEADER] [--pre-hook PRE_HOOK] [--post-hook
POST_HOOK]
            [--json-output] [-n REQUESTS | -d DURATION]
            [url]
```

`result`

```
-------- Results --------
Successful calls        100
Total time              0.3260 s
Average                 0.0192 s
Fastest                 0.0094 s
Slowest                 0.0285 s
Amplitude               0.0191 s
RPS                     306
BSI                     Pretty good

-------- Legend --------
RPS: Request Per Second
BSI: Boom Speed Index
```

Docker Monitor

`.Container`	Container name or ID (user input)
`.Name`	Container name
`.ID`	Container ID
`.CPUPerc`	CPU percentage
`.MemUsage`	Memory usage
`.NetIO`	Network IO
`.BlockIO`	Block IO
`.MemPerc`	Memory percentage (Not available on Windows)

`.PIDs`	Number of PIDs (Not available on Windows)

```
#command
docker stats [OPTIONS] [CONTAINER...]
#usage
docker stats --format "table
{{.Name}}\t{{.MemUsage}}\t{{.CPUPerc}}\t{{.MemPerc}}\t{{.NetIO}}"
[CONTAINER...]
```

Program Test (Go)

- go tool pprof
- gops (convenient and simple)

Install

```
#go env ready
go get -u github.com/google/gops
```

Code embedding

```
import "github.com/google/gops/agent"

if err := agent.Listen(agent.Options{}); err != nil {
 fmt.Println("gops agent failed")
}
```

Usage

```
#Listing all processes running locally
gops
```

The output displays

- PID
- PPID
- Name of the program
- Go version used to build the program
- Location of the associated program

```
#To report more information about a process
gops <pid>
To display a process tree with all the running Go processes
gops tree
#To print the current memory stats
gops memstats (<pid>|<addr>)
#To enter the CPU profile
gops pprof-cpu (<pid>|<addr>)
To enter the heap profile
gops pprof-heap (<pid>|<addr>)
```

After enter pprof

```
(pprof) help
#exampel
(pprof) help list
Output annotated source for functions matching regexp
  Usage:
    list<func_regex|address> [-focus_regex]* [-ignore_regex]*
    Include functions matching func_regex, or including the address
specified.
    Include samples matching focus_regex, and exclude ignore_regex.
```

Program Function (BenchMark)

- **unit pressure measurement**

```
#Func = $(FunctionName)

func BenchmarkFunc(b *testing.B) {
 // some code
 for i := 0; i < b.N; i++ {
  Func()
 }
}
```

- **generate result to analysis**

```
#benchtime - default 1s
#memprofile - file.out (detail mem info)
#this command will generate a cpu file like filename.test
#file.out generated by test func
go test -bench AlgorithmOne -benchtime 3s -benchmem -memprofile file.out
```

- **use go pprof to analysis**

```
go tool pprof -alloc_space filename.test file.out
```

Command

pprof-cpu

- tree Outputs a text rendering of call graph
- web Visualize graph through web browser
- top   Outputs top entries in text form
- svg   Outputs a graph in SVG format

pprof-heap

- list   Output annotated source for functions matching regexp
- tree Outputs a text rendering of call graph
- web Visualize graph through web browser
- top   Outputs top entries in text form
- svg Outputs a graph in SVG format

tip:svg Can be opened directly with a browser

Let's go ,testing, select an instance based on cpu:mem