Server-Monitor

A report on package submitted by

Manoj Kumar M

Roll no. 18PT21

18XT44 - OPERATING SYSTEMS

April 2020

M.Sc. THEORETICAL COMPUTER SCIENCE

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

PSG COLLEGE OF TECHNOLOGY

COIMBATORE – 641 004.

ABSTRACT

This report aims to cover the basics of process management and memory management and explains different commands used in monitoring the process and memory-related information.

Server Monitor

1.1 Introduction

The operating system is a program that links the user and the computer system. This operating system must be capable of managing resources.

To find whether these resources are utilized properly by the operating system, we need a tool. This report explains the tool used to monitor the resources and show all the hardware and network-related information.

1.2 Description

To start with everyone has used a System Monitor. This tool helps us to visualize the system-related information in a more precise way. We can see the Hardware related information like the processor used, number of cores, cache size, memory size, etc.., we can also see the operating system-related information like the kernel version, swap memory size, kernel version, etc.., also network and process-related information. This information helps us to know whether the system is functioning properly, the performance of the system, and also to understand the concepts of operating systems in a precise way. We can find system monitors easily for local machines and local servers but it's tricky to find system monitors for cloud servers. Most of the time we would just connect it through a secure shell and get the information through commands. But this tool would help us to easily monitor them through a website hosted by the server in a particular port. The advantage of this over a secure shell is immense. A secure shell runs in the kernel-mode most of the time, so it is not as easily accessible as the terminals in local systems. Second one, SSH is costlier. It provides a hefty amount of stuff. Instead we need only a few things. The last thing is we cannot monitor the server easily all the time and only computer professionals with good knowledge of Linux commands can monitor them. This tool overcomes all of them. First it provides a nice GUI from which you can monitor the server from any device

easily (Accessibility). It runs in a private port mostly in the user space(More secure). It provides only the information we want(Less costly).

1.3 Informations Monitored

Basic Info

- System Time - Hardware time
- Duration the system is running
- Number of users logged in

Memory Info

- Main memory usage
- Swap memory usage

Task Info

- Running Process
- Sleeping Process
- Zombie Process

Cpu Info

- Time in User Space
- Time in Kernel Space
- Idle time

Hardware Info

- RAM Information
- CPU architecture
- CPU modes
- Number of CPUs
- Threads per core

- Sockets
- Vendor ID
- CPU model
- CPU speed
- Cache info
- Network Information

Operating System Info

- Kernel Name
- Kernel-Version
- Operating System Name
- Host Name

Running Process Info

- PID
- State
- User who created the process
- Amt of memory used
- Command
- Duration

Network Info

- Number of Client hits per second
- Active Connections
- Routing Table
- Network connections

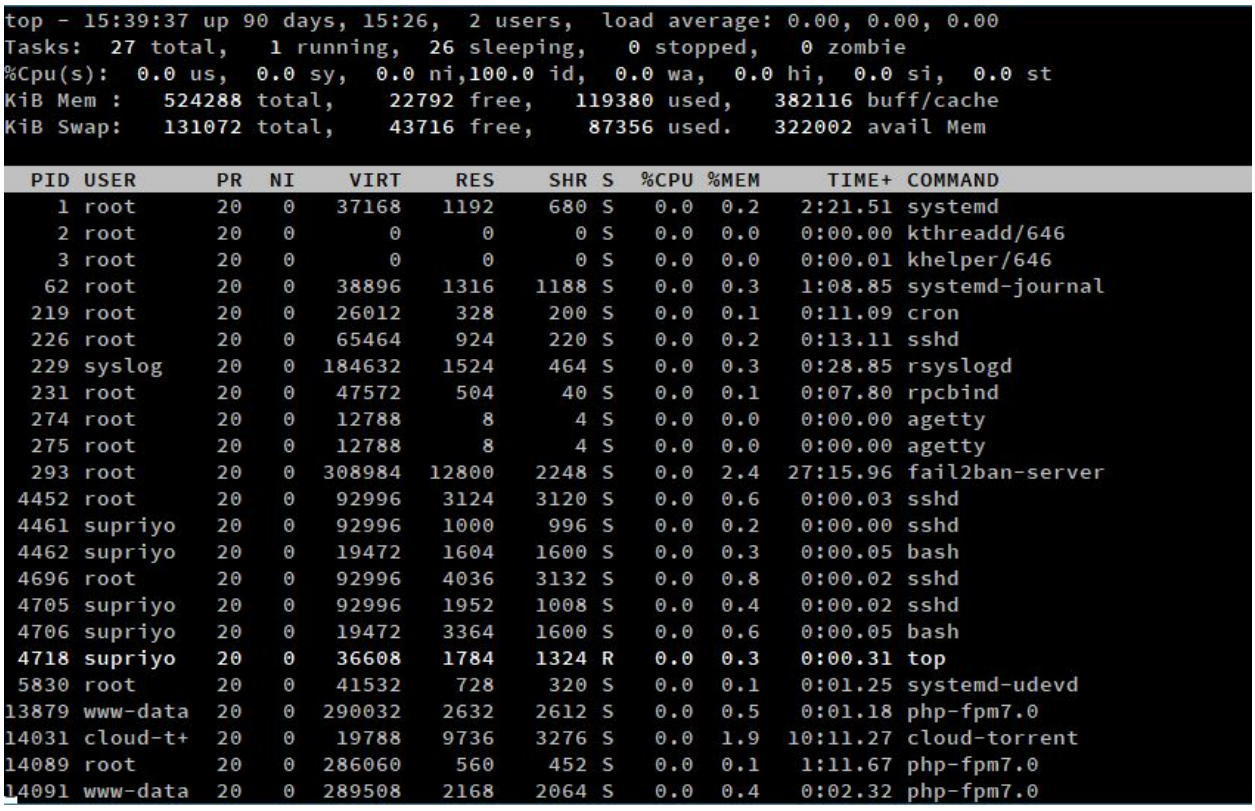
1.3 System Calls Used

To monitor the system we need to interact with the kernel with the help of system calls. The following lines explain the commands used and the concepts of Operating Systems.

1.3.1 Commands and their Description

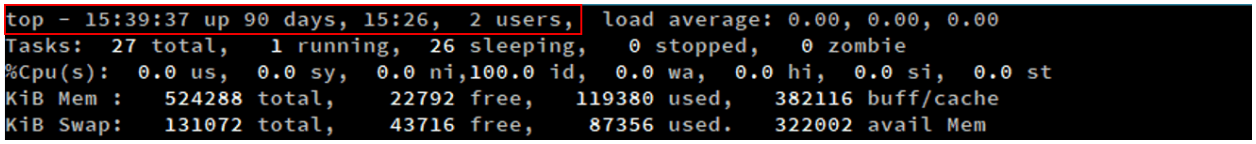
Top

This starts up an interactive command-line application, similar to one in the screenshot below. The upper half of the output contains statistics on processes and resource usage, while the lower half contains a list of the currently running processes. You can use the arrow keys and Page Up/Down keys to browse through the list. If you want to quit, simply press “q”.



As we have previously seen, the top’s output is divided into two different sections. In this part of the article, we’re going to focus on the elements in half of the output. This region is also called the “summary area”.

System time, uptime and user sessions

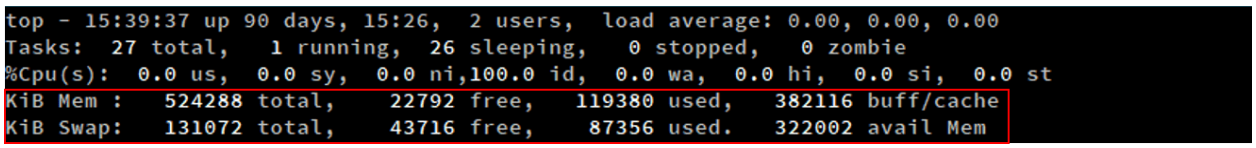


At the very top left of the screen (as marked in the screenshot above), top displays the current time. This is followed by the system uptime, which tells us the time for which the system has been running. For instance, in our example, the current time is “15:39:37”, and the system has been running for 90 days, 15 hours and 26 minutes.

Next comes the number of active user sessions. In this example, there are two active user sessions. These sessions may be either made on a TTY (physically on the system, either through the command line or a desktop environment) or a PTY (such as a terminal emulator window or over SSH). In fact, if you log in to a Linux system through a desktop environment, and then start a terminal emulator, you will find there will be two active sessions.

If you want to get more details about the active user sessions, use the who command.

Memory usage



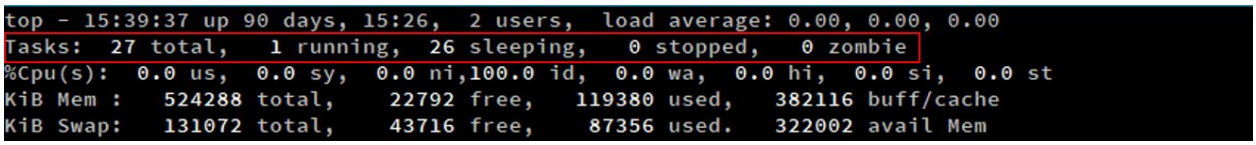
The “memory” section shows information regarding the memory usage of the system. The lines marked “Mem” and “Swap” show information about RAM and swap space respectively. Simply put, a swap space is a part of the hard disk that is used like RAM. When the RAM usage gets nearly full, infrequently used regions of the RAM are written into the swap space, ready to be retrieved later

when needed. However, because accessing disks are slow, relying too much on swapping can harm system performance.

As you would naturally expect, the “total”, “free” and “used” values have their usual meanings. The “avail mem” value is the amount of memory that can be allocated to processes without causing more swapping.

The Linux kernel also tries to reduce disk access times in various ways. It maintains a “disk cache” in RAM, where frequently used regions of the disk are stored. In addition, disk writes are stored to a “disk buffer”, and the kernel eventually writes them out to the disk. The total memory consumed by them is the “buff/cache” value. It might sound like a bad thing, but it really isn’t — memory used by the cache will be allocated to processes if needed.

Tasks



The “Tasks” section shows statistics regarding the processes running on your system. The “total” value is simply the total number of processes. For example, in the above screenshot, there are 27 processes running. To understand the rest of the values, we need a little bit of background on how the Linux kernel handles processes.

Processes perform a mix of I/O-bound work (such as reading disks) and CPU-bound work (such as performing arithmetic operations). The CPU is idle when a process performs I/O, so OSes switch to executing other processes during this time. In addition, the OS allows a given process to execute for a very small amount of time, and then it switches over to another process. This is how

OSes appear as if they were “multitasking”. Doing all this requires us to keep track of the “state” of a process. In Linux, a process may be in of these states:

- Runnable (R): A process in this state is either executing on the CPU, or it is present on the run queue, ready to be executed.
- Interruptible sleep (S): Processes in this state are waiting for an event to complete.
- Uninterruptible sleep (D): In this case, a process is waiting for an I/O operation to complete.
- Stopped (T): These processes have been stopped by a job control signal (such as by pressing Ctrl+Z) or because they are being traced.
- Zombie (Z): The kernel maintains various data structures in memory to keep track of processes. A process may create a number of child processes, and they may exit while the parent is still around. However, these data structures must be kept around until the parent obtains the status of the child processes. Such terminated processes whose data structures are still around are called zombies.

Processes in the D and S states are shown in “sleeping”, and those in the T state are shown in “stopped”. The number of zombies are shown as the “zombie” value.

CPU usage

The CPU usage section shows the percentage of CPU time spent on various tasks. The us value is the time the CPU spends executing processes in

userspace. Similarly, the sy value is the time spent on running kernel-space processes.

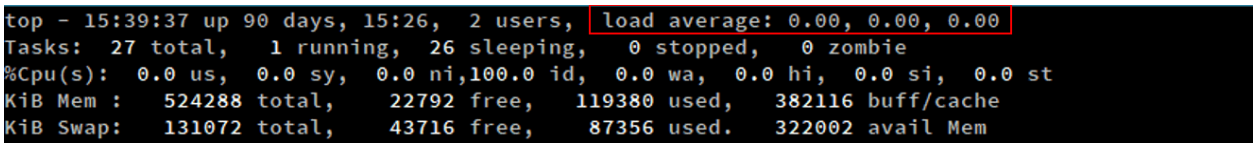
Linux uses a “nice” value to determine the priority of a process. A process with a high “nice” value is “nicer” to other processes, and gets a low priority. Similarly, processes with a lower “nice” gets higher priority. As we shall see later, the default “nice” value can be changed. The time spent on executing processes with a manually set “nice” appear as the ni value.

This is followed by id, which is the time the CPU remains idle. Most operating systems put the CPU on a power-saving mode when it is idle. Next comes the wa value, which is the time the CPU spends waiting for I/O to complete.

Interrupts are signals to the processor about an event that requires immediate attention. Hardware interrupts are typically used by peripherals to tell the system about events, such as a keypress on a keyboard. On the other hand, software interrupts are generated due to specific instructions executed on the processor. In either case, the OS handles them, and the time spent on handling hardware and software interrupts are given by hi and si respectively.

In a virtualized environment, a part of the CPU resources are given to each virtual machine (VM). The OS detects when it has work to do, but it cannot perform them because the CPU is busy on some other VM. The amount of time lost in this way is the “steal” time, shown as st.

Load average



The load average section represents the average “load” over one, five and fifteen minutes. “Load” is a measure of the amount of computational work a system performs. On Linux, the load is the number of processes in the R and D states at any given moment. The “load average” value gives you a relative measure of how long you must wait for things to get done.

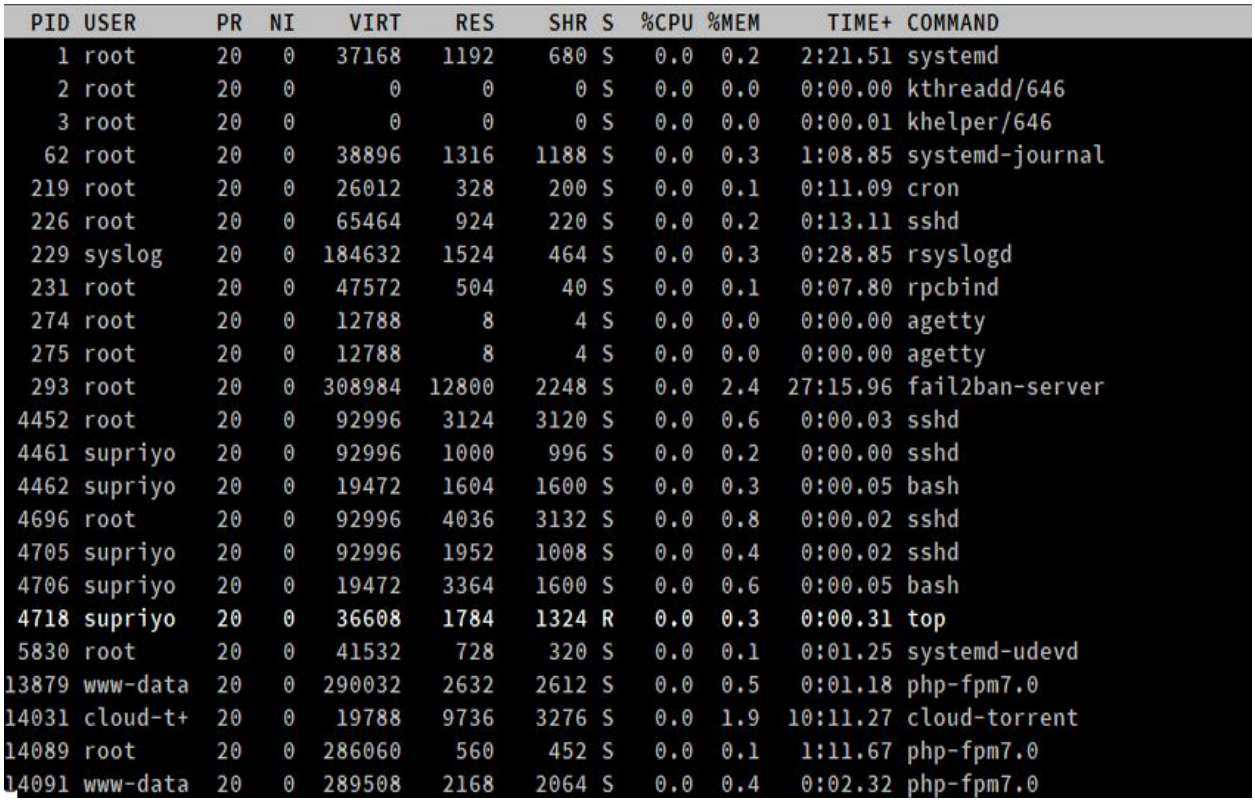
Let us consider a few examples to understand this concept. On a single-core system, a load average of 0.4 means the system is doing only 40% of the work it can do. A load average of 1 means that the system is exactly at capacity — the system will be overloaded by adding even a little bit of additional work. A system with a load average of 2.12 means that it is overloaded by 112% more work than it can handle.

On a multi-core system, you should first divide the load average with the number of CPU cores to get a similar measure.

In addition, “load average” isn’t actually the typical “average” most of us know. It is an “exponential moving average“, which means a small part of the previous load averages are factored into the current value. If you’re interested, this article covers all the technical details.

Understanding top’s interface: the task area

The summary area is comparatively simpler, and it contains a list of processes. In this section, we will learn about the different columns shown in the top’s default output.



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0   37168   1192    680 S   0.0   0.2   2:21.51 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd/646
    3 root      20   0       0      0      0 S   0.0   0.0   0:00.01 khelper/646
   62 root      20   0   38896   1316   1188 S   0.0   0.3   1:08.85 systemd-journal
  219 root      20   0   26012    328    200 S   0.0   0.1   0:11.09 cron
  226 root      20   0   65464    924    220 S   0.0   0.2   0:13.11 sshd
  229 syslog    20   0  184632   1524    464 S   0.0   0.3   0:28.85 rsyslogd
  231 root      20   0   47572    504     40 S   0.0   0.1   0:07.80 rpcbind
  274 root      20   0   12788      8      4 S   0.0   0.0   0:00.00 agetty
  275 root      20   0   12788      8      4 S   0.0   0.0   0:00.00 agetty
  293 root      20   0  308984  12800   2248 S   0.0   2.4  27:15.96 fail2ban-server
 4452 root      20   0   92996   3124   3120 S   0.0   0.6   0:00.03 sshd
 4461 supriyo   20   0   92996   1000    996 S   0.0   0.2   0:00.00 sshd
 4462 supriyo   20   0   19472   1604   1600 S   0.0   0.3   0:00.05 bash
 4696 root      20   0   92996   4036   3132 S   0.0   0.8   0:00.02 sshd
 4705 supriyo   20   0   92996   1952   1008 S   0.0   0.4   0:00.02 sshd
 4706 supriyo   20   0   19472   3364   1600 S   0.0   0.6   0:00.05 bash
 4718 supriyo   20   0   36608   1784   1324 R   0.0   0.3   0:00.31 top
 5830 root      20   0   41532    728    320 S   0.0   0.1   0:01.25 systemd-udevd
13879 www-data  20   0  290032   2632   2612 S   0.0   0.5   0:01.18 php-fpm7.0
14031 cloud-t+  20   0   19788   9736   3276 S   0.0   1.9  10:11.27 cloud-torrent
14089 root      20   0  286060    560    452 S   0.0   0.1   1:11.67 php-fpm7.0
14091 www-data  20   0  289508   2168   2064 S   0.0   0.4   0:02.32 php-fpm7.0
```

- PID

This is the process ID, a unique positive integer that identifies a process.

- USER

This is the “effective” username (which maps to a user ID) of the user who started the process. Linux assigns a real user ID and an effective user ID to processes; the latter allows a process to act on behalf of another user. (For example, a non-root user can elevate to root in order to install a package.)

- PR and NI

The “NI” field shows the “nice” value of a process. The “PR” field shows the scheduling priority of the process from the perspective of the kernel. The nice value affects the priority of a process.

- VIRT, RES, SHR and %MEM

These three fields are related to memory consumption of the processes. “VIRT” is the total amount of memory consumed by a process. This includes the program’s code, the data stored by the process in memory, as well as any regions of memory that have been swapped to the disk. “RES” is the memory consumed by the process in RAM, and “%MEM” expresses this value as a percentage of the total RAM available. Finally, “SHR” is the amount of memory shared with other processes.

- S

As we have seen before, a process may be in various states. This field shows the process state in the single-letter form.

- TIME+

This is the total CPU time used by the process since it started, precise to the hundredths of a second.

- COMMAND

The COMMAND column shows the name of the processes.

Uname

It prints all the system information in the following order:

- Kernel name,
- Network node hostname
- Kernel release date
- Kernel version
- Machine hardware name
- Hardware platform
- Operating System

lscpu

lscpu gathers CPU architecture information from sysfs and /proc/cpuinfo. The command output can be optimized for parsing or for easy readability by humans. The information includes, for example, the number of CPUs, threads, cores, sockets, and Non-Uniform Memory Access (NUMA) nodes. There is also information about the CPU caches and cache sharing, family, model, bogoMIPS, byte order, and stepping.

Options that result in an output table have a list argument. Use this argument to customize the command output. Specify a comma-separated list of column labels to limit the output table to only the specified columns, arranged in the specified order. See COLUMNS for a list of valid column labels. The column labels are not case sensitive. Not all columns are supported on all architectures. If an unsupported column is specified, lscpu prints the column but does not provide any data for it.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Model name:            Intel(R) Core(TM) i3-4130 CPU @ 3.4
0GHz
Stepping:              3
CPU MHz:               3392.467
CPU max MHz:           3400.0000
CPU min MHz:           800.0000
BogoMIPS:              6784.93
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8
apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm co
nstant_tsc arch_perfmon pebs bts rep_good nopl xtopology n
onstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor d
s_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_
1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16
c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti tp
r_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bm
i1 avx2 smep bmi2 erms invpcid xsaveopt dtherm arat pln pt
s
```

free

free displays the total amount of free and used physical and swap memory in the system, as well as the buffers and caches used by the kernel. The information is gathered by parsing /proc/meminfo.

```
              total        used        free      shared  buff/cache   available
Mem:        3912864     2255460      315388      249068     1342016     1123600
Swap:       4060156           0     4060156
```

ip link show

This command displays the network connections available with your computer. It also displays the MTU, mac address of the device

netstat

This is one of the important tools for monitoring the network-related information.

Command: netstat -at.

This displays details of all the TCP sockets, including open and close sockets in the same format

Sample output:

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	localhost:ipp	*:*	LISTEN
tcp	0	0	*:microsoft-ds	*:*	LISTEN
tcp	0	0	*:netbios-ssn	*:*	LISTEN
tcp	0	0	ubuntu:domain	*:*	LISTEN
tcp6	0	0	ip6-localhost:ipp	[::]:*	LISTEN
tcp6	0	0	[::]:microsoft-ds	[::]:*	LISTEN
tcp6	0	0	[::]:netbios-ssn	[::]:*	LISTEN
tcp6	1	0	ip6-localhost:34871	ip6-localhost:ipp	CLOSE_WAIT

1.4 Tools and Technology Used

I have used different Tech stack for this project.

1.4.1 NEXT.js

Next.js is a library built with React. I have used it for the front-end. It is a static site rendering library that renders the page in the server and sends only the raw data to the client-side.

1.4.2 Golang

Developed by Google, is one of the fastest server-side language. It is a compiled language that can handle more number of requests at a time. Its own scheduler helps it to handle the threads in the userspace. I used it for my back end

1.4.3 Cgo modules

Since GO is a language built with C, it has direct access to the c modules.

1.5 Workflow

Server-Monitor-Backend will have the c modules integrated and running at a specific port on the cloud server. Server-Monitor-Frontend connects to the backend and requests for the command output. The Backend creates a thread and gets the command output and sends it back to the frontend via WebSockets. The communication takes place at an interval of 5 seconds continuously.

1.6 Results and Discussion

This project proves that the memory and other resources consumed via the project is much smaller than that of SSH. Also the complexity of SSH and security concerns of sharing private keys is reduced. Thus the project helps all the users, even non-programmers to easily monitor the cloud servers via any device without any security concerns.

Conclusion

With the help of this project, one can easily get the information of the cloud server remotely. This project helps to monitor the server at any time and find the performance of the system easily.

Bibliography

Books and articles referred:

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, 2009. *Operating System Concepts, 8th Edition.*

Websites

1. https://linux.die.netman/
2. https://nextjs.org/
3. https://golang.org/