System metrics in bash scripting

Objectives:
The goal of system metrics in Bash scripting is to monitor and gather information about the performance and status of a computer system. System metrics provide valuable insights into how the system is functioning.

Case Statement:
The application will start with a user inquiry about which metric they want to check .

```
echo "Please Select which system metrics would like to check?"
echo "1) Memory"
echo "2) Disk"
echo "3) CPU"
echo "4) Processes"

case $input in
        1) check_memory;;
        2)check_disk;;
        3)processes_number;;
        4)check_cpu;;
        *) echo " please choose number from 1 to 4 only"

esac
```

The user choice will trigger one of the below functions:

Functions

1) **check_disk**

The purpose of this function is to ascertain whether disk space utilization surpasses a predefined threshold. The initial step involves declaring the threshold limit. Subsequently, a variable named disk_usage is established. This variable utilizes the df (disk file) command, configured with the -H option to provide disk storage information in a human-readable format. Employing the pipe (|) command, the output of df is then passed to an awk command. This awk command scans the first line of the df output and extracts the fifth value from that line, representing the current disk usage. This value is then compared with the predefined threshold to determine if the disk space has exceeded the specified limit. If the disk_usage

is greater than disk_threshold
we will call the log function and pass log "High disk usage: $disk_usage%"
If it is equals or less will print
log "Disk usage within normal range: $disk_usage%"

code:

```
check_disk() {
  disk_threshold=90
  disk_usage=$(df -H | awk '$NF=="/" {print int($5)}')

  if [ "$disk_usage" -gt "$disk_threshold" ]; then
    log "High disk usage: $disk_usage%"
  else
    log "Disk usage within normal range: $disk_usage%"
  fi
}
```

2) **Check_cpu**

The purpose of the check_cpu function is to evaluate the current CPU usage and ascertain whether it exceeds a predefined threshold. The initial step involves declaring the threshold limit for CPU usage. Subsequently, a variable named cpu_usage is established to store the results. The top command is utilized to retrieve information about processes and their resource usage. Through the use of the awk command, the CPU usage value is extracted from the top output. This value is then compared with the predefined threshold to determine if the CPU usage has exceeded the specified limit. If the cpu_usage

is greater than cpu_threshold we will call the log function and pass log "High CPU usage: $cpu_usage%"

If it is equals or less will print
log "CPU usage within normal range: $cpu_usage%"

Code :
```
# function to check the CPU
check_cpu() {
  cpu_threshold=90
  cpu_usage=$(top -l 1 | awk '/CPU usage:/ {print int($3)}')

  if [ "$cpu_usage" -gt "$cpu_threshold" ]; then
    log "High CPU usage: $cpu_usage%"
  else
    log "CPU usage within normal range: $cpu_usage%"
  fi
}
```

3) **Processes_number**
The goal of the processes_number function is to monitor the total number of processes running on the system and determine whether it surpasses a predefined threshold. The function begins by declaring the threshold limit for processes. A variable named process_usage is created to store the result, utilizing the top command to gather information about processes. The awk command is then employed to extract the relevant value from the top output, representing the total number of processes. The extracted value is compared with the predefined threshold to determine if there are an excessive number of processes running at the moment. If the process_usage is greater than process_threshold we will call the log function and pass log "There are too many processes running at this moment: $process_usage".
If it is equals or less will print
log "There are few processes running: $process_usage"

Code :
```
# Function to check the number of processes
processes_number() {
    process_threshold=300
    process_usage=$(top -l 1 | awk '/Processes:/ {print int($2)}')

    if [ "$process_usage" -gt "$process_threshold" ]; then
        log "There are too many processes running at this moment: $process_usage"
    else
```

```
        log "There are few processes running: $process_usage"
      fi
    }
```

4) Check_memory

The primary function of check_memory is to examine the system's memory usage and ascertain whether it exceeds a predefined threshold. The initial step involves declaring the threshold limit for memory usage. A variable named memory_usage is then created to store the results. The top command is utilized to retrieve information about memory usage, and the grep -E will retrieve the value of ‘PhysMem’ and pass to the awk command is employed to extract the second $2 relevant value from the top output. This extracted value, representing memory usage, is compared with the predefined threshold to determine if the system's memory is approaching full capacity. If the memory_usage
is greater than memory_threshold we will call the log function and pass log "Memory is almost full: $memory_usage%"

If it is equals or less will print
log "Memory usage within normal range: $memory_usage%"

Code:

```
# Function to check memory usage
check_memory(){
  memory_threshold=50
  memory_usage=$(top -l 1 | grep -E 'PhysMem:' | awk '{print int($2)}')

  if [ "$memory_usage" -gt "$memory_threshold" ]; then
    log "Memory is almost full: $memory_usage%"
  else
    log "Memory usage within normal range: $memory_usage%"
  fi
}
```

5) Log function

We create or pass the value to metrics.txt file with the current date.

```
log() {
    echo "$(date +"%Y-%m-%d %H:%M:%S") - $1" >> metrics.txt
}
```