Configuração e Monitoração Docker

Processos e monitoração Docker.

- Bash script to log and restart docker container based on cpu usage
- how to get docker stats using shell script
- Docker Stats | Understand how to monitor Docker Metrics with docker stats
- Monitora containers
- Coletar estatísticas Docker com CTOP
- 15 Scripts to Automate Docker Container Management

Bash script to log and restart docker container based on cpu usage

Link: https://dev.to/tomasggarcia/bash-script-to-log-and-restart-docker-container-cpu-usage-1j2o

I wrote this post to share with you one job experience that I had to live with recently, one problem I had and my temporal solution to this problem.
I'm new in this tech world, I will be grateful if any of you have some improvement recommendations and very pleased if this post it's useful for someone else.
A few mounths ago in my company we discovered that some Docker container was having problems with CPU usage. Out of nowhere the CPU usage of that container was increasing abruptly. So while the dev team was searching for the code error, I implemented a temporary solution. I made one script to log all cpu usage every 5 seconds:

```
#!/bin/bash
logs=/var/log/process_name.log
container_name=container_name

while :
do
  # Get a variable with the cpu usage for a specific container
  var=`docker stats --no-stream --format "{{.CPUPerc}}" $container_name`
  length=${#var}

  if (( $length==0 )); then
     echo "Container ${container_name} does not exist"
     echo "$(date +'%d-%m-%Y %H:%M') | Container $container_name does not exist" >> $logs
  else
    # CPU usage in number
    percent="${var[@]::-4}"

    echo "Actual cpu usage: ${percent}"

    # Save actual CPU usage in file
```

```
    echo "$(date +'%d-%m-%Y %H:%M') | ${percent}" >> $logs
  fi
  sleep 5
```

After that I created a supervisor config to run this process:

```
[program:process]
command=/opt/scripts/script.sh
autostart=true
autorestart=true
stderr_logfile=/var/log/process.err.log
stdout_logfile=/var/log/process.err.log
```

Then I wrote a script to restart the problematic container, based on the logs of the previous script:

```
#!/bin/bash
container_name=container_name
logs_evaluated_lines=5
logs=/var/log/process_name.log
max_cpu=90

while :
do
    # Lines in file
    num=$(wc -l < $logs)

    counter=0

    # For 'logs_evaluated_lines' lines in logs increase counter if cpu is greater than 100%
    for ((index=$num;index>=$num-$logs_evaluated_lines+1;index--))
    do
    value=$(sed "${index}q;d" $logs)
    percent=$(echo $value | cut -c 20-)
    #echo $percent
    if (( $percent >= max_cpu )); then
    #    echo 'mayor'
        counter=$((counter+1))
    #  else
    #    echo 'menor'
    fi
    done
```

```
    echo "$(date +'%d-%m-%Y %H:%M') | Logs up to 100%: ${counter}"
    echo "$(date +'%d-%m-%Y %H:%M') | Logs lines analyzed: ${logs_evaluated_lines}"
    if (( $counter == $logs_evaluated_lines )); then
        echo "$(date +'%d-%m-%Y %H:%M') | CPU Full usage";
        echo "$(date +'%d-%m-%Y %H:%M') | Restarting Container"
        docker restart $container_name
        echo "$(date +'%d-%m-%Y %H:%M') | Container Restarted"
        echo "$(date +'%d-%m-%Y %H:%M') | Container Restarted" >> $logs
    else
        echo "$(date +'%d-%m-%Y %H:%M') | CPU Usage OK"
    fi
    echo "$(date +'%d-%m-%Y %H:%M') |"
    sleep 5
done
```

This script evaluate 'logs_evaluated_lines' lines in log and restarts the container if the count is upper 'max_cpu' variable

how to get docker stats using shell script

Link: https://iqcode.com/code/typescript/how-to-get-docker-stats-using-shell-script

```
#!/bin/bash

# This script is used to complete the output of the docker stats command.
# The docker stats command does not compute the total amount of resources (RAM or CPU)

# Get the total amount of RAM, assumes there are at least 1024*1024 KiB, therefore > 1 GiB
HOST_MEM_TOTAL=$(grep MemTotal /proc/meminfo | awk '{print $2/1024/1024}')

# Get the output of the docker stat command. Will be displayed at the end
# Without modifying the special variable IFS the ouput of the docker stats command won't have
# the new lines thus resulting in a failure when using awk to process each line
IFS=;
DOCKER_STATS_CMD=`docker stats --no-stream --format "table
{{.MemPerc}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.Name}}"`

SUM_RAM=`echo $DOCKER_STATS_CMD | tail -n +2 | sed "s/%//g" | awk '{s+=$1} END {print s}'`
SUM_CPU=`echo $DOCKER_STATS_CMD | tail -n +2 | sed "s/%//g" | awk '{s+=$2} END {print s}'`
SUM_RAM_QUANTITY=`LC_NUMERIC=C printf %.2f $(echo "$SUM_RAM*$HOST_MEM_TOTAL*0.01" | bc)`

# Output the result
echo $DOCKER_STATS_CMD
echo -e "${SUM_RAM}%\t\t\t${SUM_CPU}%\t\t${SUM_RAM_QUANTITY}GiB / ${HOST_MEM_TOTAL}GiB\tTOTAL"
```

Docker Stats | Understand how to monitor Docker Metrics with docker stats

Link: https://signoz.io/blog/docker-stats/
Docker containers are transient (lasting for a very short time), spawning quickly and in high numbers, which causes metrics bursts. This makes monitoring a challenge due to Docker's scaling and redeployment features. Docker stats is a built-in feature of Docker containers. The `docker stats` command returns a live data stream of your running containers.

Docker is a containerization platform that lets you separate your applications from your infrastructure to deliver software quickly. To monitor Docker, it is crucial to gather performance-related metrics from various system elements, such as containers, hosts, and databases.

Methods of monitoring Docker Metrics

There are several ways in which Docker metrics can be monitored. These include;

- Docker stats command
- Pseudo-files in sysfs
- REST API exposed by the Docker daemon

Note that these are in-built features of Docker.

> Observability for your containerized application
> Observability is critical for modern cloud-native applications. It helps engineering teams have more confidence in their production environment. Troubleshooting performance issues is easier with a robust observability framework in place. SigNoz, an open-source observability tool, can help make

> your containerized applications observable.

To get started with SigNoz, please visit the documentation.

In this article, we will deep dive into the `docker stats` command that can be used to monitor Docker metrics right from the terminal.

What is the `docker stats` command?

The `docker stats` command is a built-in feature of Docker that displays resource consumption statistics for the container in real-time. By default, it shows CPU and memory utilization for all containers. Stats here refers to “statistics”. You can restrict the statistics by entering the container names or IDs you're interested in monitoring. The `docker stats` command output includes CPU stats, memory metrics, block I/O, and network IO metrics for all active containers.

A Practical Approach

Running the `docker stats` command produces an output that looks like the code snippet below. This command shows the stats for `all` the running Docker containers.

```
$ docker stats

CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM%                NET I/O             BLOCK I/O           PIDS
56b3f523b0sd        nginx-container     0.35%               2.534MiB / 16.455GiB
0.37%               568B / 0B           134kb / 0B          3
049996113bc8        ubuntu              0.14%               1.437MiB / 16.455GiB
0.10%               3.56kb / 0B         5.12MB / 0B         1
a3f78cb32a8e        hello-world         0.00%               1.228MiB / 16.455GiB
0.06%               65.45kb / 0B        550kb / 0B          0
```

Understanding the `docker stats` command output

The `docker stats` command returns a live snapshot of resource usage by Docker containers. Let’s break down all the stats given by the command.

CPU% stats

CPU is expressed as a percentage (%) of the overall host capacity. One can optimize the resource usage of Docker hosts by being aware of how much CPU the hosts and containers consume. One active/busy container shouldn't slow down other containers by consuming all of the CPU resources. Containers can be optimized based on the amount of CPU they are using.

MEM USAGE / LIMIT Stats

`MEM USAGE` lists the available memory. It gives a quick overview of the container's memory usage and allocation, providing information about the container's memory statistics, including usage and memory limit. Except when it is defined for a specific container, the memory usage limit corresponds to the host machine's memory limit.

MEM % Stat

`MEM %` shows the memory percentage that the container is using from its host machine.

Network(NET) I/O Stats

`NET I/O` shows the volume of information the container's network interface has transmitted(TX) and received (RX). It represents network traffic.

BLOCK I/O Stats

`BLOCK I/O` helps to identify containers that are writing data and shows the total number of bytes read and written to the container file system. Block I/O stats can give you an idea about issues with data persistence.

PIDS

PIDS is a count of the processes that the container has created or the number of kernel process IDs running inside the corresponding container.

More on the usage of `docker stats`

Getting stats of a particular container

To get the stats of a particular container, provide the container Id and run the command `docker stats <containerID>`

```
$ docker stats 56b3f523b0sd

CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM%                NET I/O             BLOCK I/O           PIDS
56b3f523b0sd        nginx-container     0.35%               2.534MiB / 16.455GiB
0.37%               568B / 0B           134kb / 0B          3
```

You can also get the stats of multiple containers by name and id if you run

`docker stats <containerName> <containerId>`

```
$ docker stats ubuntu 56b3f523b0sd

CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM%                NET I/O             BLOCK I/O           PIDS
049996113bc8        ubuntu              0.14%               1.437MiB / 16.455GiB
0.10%               3.56kb / 0B         5.12MB / 0B         1
56b3f523b0sd        nginx-container     0.35%               2.534MiB / 16.455GiB
0.37%               568B / 0B           134kb / 0B          3
```

Display options that Docker Provides

These display options allow you to specify how you want the output to be shown.

Docker stats offers the following options for display:

1. `--all` which shows all containers, whether stopped or running.
2. `--format` which uses the Go Template syntax to print images out.
3. `--no-stream` which disables streaming stats and only pulls the first result
4. `--no-trunc` which instructs Docker not to truncate (shorten) output.

The syntax for this is shown below:

```
$ docker stats [OPTIONS] [CONTAINER...]
```

Using `docker stats --format`

Let's take a look at the `--format` option.

Docker format is used to modify the output format of commands that have the `--format` option. If a command supports this option, it can be used to change the output format of the command to suit our needs since the default command does not display all the fields connected to that object.

By using the Go Template syntax, the formatting option `--format` presents container output in an easy-to-read way.

For example,

```
$ docker stats --format "{{.Container}}: {{.CPUPerc}}"


049996113bc8: 0.14%
56b3f523b0sd: 0.35%
```

This prints out all images with the Container and CPUPerc (CPU Percentage) elements, separated by a colon (:) and it uses a template without headers.

To display all container information in a table format, including name, CPU percentage, and memory consumption, use the following syntax:

```
$ docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"

CONTAINER ID        CPU %               PRIV WORKING SET
56b3f523b0sd        0.35%               2.534MiB / 16.455GiB
049996113bc8        0.14%               1.437MiB / 16.455GiB
a3f78cb32a8e        0.00%               1.228MiB / 16.455GiB
```

Here is the list of applicable placeholders to use with the Go template syntax:

Placeholder	Description
.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)

Final Thoughts

In this article, we discussed ways to monitor resource usage metrics in Docker focused on the `docker stats` command. Other ways of using The Docker stats, Pseudo-files in sysfs, and REST API exposed by the Docker daemon are native ways of monitoring resource utilization metrics.

Docker container monitoring is critical for running containerized applications. For a robust monitoring and observability setup, you need to use a tool that visualizes the metrics important for container monitoring and also lets you set alerts on critical metrics. SigNoz is an open-source observability tool that can help you do that.

It uses OpenTelemetry to collect metrics from your containers for monitoring. OpenTelemetry is becoming the world standard for instrumentation of cloud-native applications, and it is backed by CNCF foundation, the same foundation under which Kubernetes graduated.

If you want to set up a robust observability framework for your containerized application, you can use SigNoz. You can create unified views to monitor your Docker containers effectively.

It is easy to get started with SigNoz. It can be installed on macOS or Linux computers in just three steps by using a simple installation script.

The install script automatically installs Docker Engine on Linux. However, you must manually install Docker Engine on macOS before running the install script.

```
git clone -b main https://github.com/SigNoz/signoz.git
cd signoz/deploy/
./install.sh
```

Monitora containers

Link: https://gist.github.com/haukurk/a6e0751a8b8746265f8b2c55d9476230

```
#!/bin/bash
# Author: Haukur Kristinsson / Erik Kristensen
# Email: haukur@hauxi.is / erik@erikkristensen.com
# License: MIT
# Nagios Usage: check_nrpe!check_docker_container!_container_id_
# Usage: ./check_docker_container.sh _container_id_
#
# The script checks if a container is running.
# OK - running
# WARNING - container is ghosted
# CRITICAL - container is stopped
# UNKNOWN - does not exist

CONTAINER=$1

RUNNING=$(docker inspect --format="{{ .State.Running }}" $CONTAINER 2> /dev/null)

if [ $? -eq 1 ]; then
echo "UNKNOWN - $CONTAINER does not exist."
exit 3
fi

if [ "$RUNNING" == "false" ]; then
echo "CRITICAL - $CONTAINER is not running."
exit 2
fi

STARTED=$(docker inspect --format="{{ .State.StartedAt }}" $CONTAINER)
NAME=$(docker inspect --format="{{ .Name }}" $CONTAINER)
```

```
NETWORKMODE=$(docker inspect --format="{{ .HostConfig.NetworkMode }}" $CONTAINER)
NETWORK=$(docker inspect --format="{{ .NetworkSettings.Networks."$NETWORKMODE".IPAddress }}" $CONTAINER)

echo "OK - $CONTAINER is running. IP: $NETWORK, StartedAt: $STARTED, Named: $NAME"
```

Coletar estatísticas Docker com CTOP

Link: https://github.com/bcicen/ctop

git clone https://github.com/bcicen/ctop.git

ctop

release homebrew macports scoop

Top-like interface for container metrics

`ctop` provides a concise and condensed overview of real-time metrics for multiple containers:

ctop

as well as a single container view for inspecting a specific container.

`ctop` comes with built-in support for Docker and runC; connectors for other container and cluster systems are planned for future releases.

Install

Fetch the latest release for your platform:

Debian/Ubuntu

Maintained by a third party

```
sudo apt-get install ca-certificates curl gnupg lsb-release
curl -fsSL https://azlux.fr/repo.gpg.key | sudo gpg --dearmor -o /usr/share/keyrings/azlux-archi
echo \
```

```
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/azlux-archive-keyring.gp
  $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/azlux.list >/dev/null
sudo apt-get update
sudo apt-get install docker-ctop
```

Arch

```
sudo pacman -S ctop
```

`ctop` *is also available for Arch in the AUR*

Linux (Generic)

```
sudo wget https://github.com/bcicen/ctop/releases/download/v0.7.7/ctop-0.7.7-linux-amd64 -O /usr
sudo chmod +x /usr/local/bin/ctop
```

OS X

```
brew install ctop
```

or

```
sudo port install ctop
```

or

```
sudo curl -Lo /usr/local/bin/ctop https://github.com/bcicen/ctop/releases/download/v0.7.7/ctop-0
sudo chmod +x /usr/local/bin/ctop
```

Windows

`ctop` is available in scoop:

```
scoop install ctop
```

Docker

```
docker run --rm -ti \
  --name=ctop \
  --volume /var/run/docker.sock:/var/run/docker.sock:ro \
  quay.io/vektorlab/ctop:latest
```

Building

Build steps can be found here.

Usage

`ctop` requires no arguments and uses Docker host variables by default. See connectors for further configuration options.

Config file

While running, use `S` to save the current filters, sort field, and other options to a default config path (`~/.config/ctop/config` on XDG systems, else `~/.ctop`).

Config file values will be loaded and applied the next time `ctop` is started.

Options

Option	Description
`-a`	show active containers only
`-f <string>`	set an initial filter string
`-h`	display help dialog
`-i`	invert default colors
`-r`	reverse container sort order
`-s`	select initial container sort field
`-v`	output version information and exit

Keybindings

Key	Action
<ENTER>	Open container menu

Key	Action
a	Toggle display of all (running and non-running) containers
f	Filter displayed containers (`esc` to clear when open)
H	Toggle ctop header
h	Open help dialog
s	Select container sort field
r	Reverse container sort order
o	Open single view
l	View container logs (`t` to toggle timestamp when open)
e	Exec Shell
c	Configure columns
S	Save current configuration to file
q	Quit ctop

Alternatives

See Awesome Docker list for similar tools to work with Docker.

15 Scripts to Automate Docker Container Management

Link: https://blog.devops.dev/15-scripts-to-automate-docker-container-management-4bab4c3faf73

Each example comes with functioning code and detailed explanations.

1. Automatically Start All Containers

Sometimes after a system reboot or maintenance, you may want to start all stopped containers at once.

```
#!/bin/bash
# Start all stopped containers
docker start $(docker ps -aq)
```

- ‘docker ps -aq’ lists all container IDs (stopped and running).
- ‘docker start’ starts the containers by passing the IDs as arguments.

2. Stop All Running Containers

Quickly stop all currently running containers.

```
#!/bin/bash
# Stop all running containers
docker stop $(docker ps -q)
```

- ‘docker ps -q’ lists IDs of only running containers.
- ‘docker stop’ stops these containers.

3. Remove Stopped Containers

Free up space by cleaning up stopped containers.

```
#!/bin/bash
# Remove all stopped containers
docker rm $(docker ps -aq -f "status=exited")
```

- `docker ps -aq -f "status=exited"` filters stopped containers.
- ‘docker rm’ removes them.

4. Remove Dangling Images

Clear unused Docker images to save disk space.

```
#!/bin/bash
# Remove dangling images
docker rmi $(docker images -q -f "dangling=true")
```

- `docker images -q -f "dangling=true"` lists image IDs with no tags (dangling).
- ‘docker rmi’ removes these images.

5. Backup a Container’s Data

Export the filesystem of a running container to a tar file.

```
#!/bin/bash
# Backup a container's data
CONTAINER_ID=$1
BACKUP_FILE="${CONTAINER_ID}_backup_$(date +%F).tar"
docker export $CONTAINER_ID > $BACKUP_FILE
echo "Backup saved to $BACKUP_FILE"
```

- ‘docker export’ exports the filesystem of the container.
- Pass the container ID as an argument to the script.

6. Restore a Container from Backup

Recreate a container from a tar backup file.

```
#!/bin/bash
# Restore a container from a tar backup
BACKUP_FILE=$1
docker import $BACKUP_FILE restored_container:latest
echo "Container restored as 'restored_container:latest'"
```

- ‘docker import’ creates a new image from the tar file.
- The image can be used to start new containers.

7. Monitor Container Resource Usage

Display real-time stats for all running containers.

```
#!/bin/bash
# Monitor resource usage of all running containers
docker stats --all
```

- ‘docker stats’ shows real-time CPU, memory, and network stats.
- ‘--all’ includes stopped containers.

8. Restart a Container Automatically

Ensure critical containers restart after failure.

```
#!/bin/bash
# Restart a container with restart policy
CONTAINER_NAME=$1
docker update --restart always $CONTAINER_NAME
echo "$CONTAINER_NAME will now restart automatically on failure."
```

- ‘docker update --restart always’ configures the restart policy.
- Pass the container name as an argument.

9. Run a Container and Clean Up After Exit

Automatically remove a container after it stops.

```
#!/bin/bash
# Run a container and clean up
IMAGE_NAME=$1
docker run --rm $IMAGE_NAME
```

- ‘--rm’ removes the container when it stops.
- Useful for one-off tasks.

10. Check Logs of All Containers

Combine logs from multiple containers into one output.

```
#!/bin/bash
# Display logs of all containers
docker ps -q | xargs -I {} docker logs {}
```

- ‘docker ps -q’ lists running container IDs.
- ‘xargs’ passes these IDs to ‘docker logs’.

11. Auto-Prune Unused Resources

Schedule automated cleanup of unused Docker resources.

```
#!/bin/bash
# Prune unused resources
```

```
docker system prune -f --volumes
```

- ‘docker system prune’ removes unused containers, networks, and images.
- ‘--volumes’ also deletes unused volumes.

12. Update Running Containers

Recreate containers with the latest image version.

```
#!/bin/bash
# Update a running container
CONTAINER_NAME=$1
IMAGE_NAME=$(docker inspect --format='{{.Config.Image}}' $CONTAINER_NAME)
docker pull $IMAGE_NAME
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME
docker run -d --name $CONTAINER_NAME $IMAGE_NAME
```

- ‘docker inspect’ fetches the image name of a container.
- The script pulls the latest image and recreates the container.

13. Copy Files from a Container

Extract files or directories from a container to the host.

```
#!/bin/bash
# Copy files from a container
CONTAINER_ID=$1
SOURCE_PATH=$2
DEST_PATH=$3
docker cp $CONTAINER_ID:$SOURCE_PATH $DEST_PATH
echo "Copied $SOURCE_PATH from $CONTAINER_ID to $DEST_PATH"
```

- ‘docker cp’ copies files between the container and the host.
- Pass container ID, source path, and destination path as arguments.

14. Restart All Containers

Restart all running containers quickly.

```
#!/bin/bash
# Restart all containers
docker restart $(docker ps -q)
```

- ‘docker restart’ restarts containers by their IDs.

15. List All Exposed Ports

Check the exposed ports of running containers.

```
#!/bin/bash
# List all exposed ports
docker ps --format '{{.ID}}: {{.Ports}}'
```

- ‘docker ps --format’ customizes the output to show container IDs and ports.

Feel free to tweak, experiment and customize them to your needs.

Docker

Bash

Bash Script

Programming