Operations Guide for Linux

October 25, 2021

Contents

This document provides an operations guide for Linux based OS. The guide can be followed for installations, where the software have been deployed in a Docker based setup, as described in the installation guides. As there are small differences in the syntax of commands on different Linux distributions, the administrator may have to adjust commands accordingly for the specific Linux distribution.

You should read the Deployment and Installation documentation beforehand, in order to understand the components and their roles. Administrative knowdledge of Linux and Docker administration is assumed.

Health Check of a System

Login to the backend server

```
ssh user@<server>
sudo su
```

```
cd deploy
```

Check Docker Host

The administrator must check and monitor the Docker Host performance and ressources. These commands are available under all flavors of Linux and can be useful to monitor and find the actual causes of performance problem.

Memory and CPU

Linux `vmstat` command is used to display statistics of virtual memory, kernerl threads, disks, system processes, I/O blocks, interrupts, CPU activity and much more. By default `vmstat` command is not available under Linux systems you need to install a package called `sysstat` that includes the `vmstat` program. Example `vmstat` usage:

```
# vmstat
procs -----------memory---------- ---swap-- -----io---- -system--
    ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy
      id wa st
 3  0      0 529780   2088 393428    0    0     0     1    3   23  0  0
      100  0  0
```

Also the `free` command could be used to look into the memory usage and avaliability on the Docker Host. The `free` command provides information about unused and used memory and swap space

```
# free
               total        used        free      shared  buff/cache
   available
Mem:         1014992       91936      418376       63876      504680
      685536
Swap:             0           0           0
```

Consult the man pages for options and usage of `vmstat`.

Disk Usage

Disk usage and free space, can be observed using the `df` command:

```
# df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  3.8G   22G  16% /
devtmpfs        473M     0  473M   0% /dev
tmpfs           496M     0  496M   0% /dev/shm
tmpfs           496M   63M  434M  13% /run
tmpfs           496M     0  496M   0% /sys/fs/cgroup
tmpfs           100M     0  100M   0% /run/user/0
```

Consult the man pages for options and usage of `df`.

Running Processes

`top` and `htop` commands are performance monitoring programs used by many system administrators to monitor Linux performance. The commands are used to dipslay all the running and active real-time processes in ordered list and updates it regularly. It display CPU usage, Memory usage, Swap Memory, Cache Size, Buffer Size, Process PID, User, Commands and much more. It also shows high memory and cpu utilization of a running processess. The top command is much userful for system administrator to monitor and take correct action when required. (`htop` is a third party tool and isn't included in Linux systems, you need to install it using the package manager)

```
# top
top - 11:23:31 up 21 days, 21:03,  1 user,  load average: 0.13, 0.08,
    0.06
Tasks:  89 total,   2 running,  87 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  0.7 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,
     0.0 st
KiB Mem :  1014992 total,   421240 free,    93980 used,   499772 buff/
    cache
KiB Swap:        0 total,        0 free,        0 used.   684172 avail
    Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+
      COMMAND
 6472 root      20   0  158748   5280   3976 S  0.7  0.5   0:00.02 sshd
 6471 root      20   0  161896   2216   1560 R  0.3  0.2   0:00.04 top
 6473 sshd      20   0  117204   2828   1712 S  0.3  0.3   0:00.01 sshd
    1 root      20   0   46092   6532   4128 S  0.0  0.6   0:37.97
      systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.16
      kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:27.02
      ksoftirqd/0
```

```
15      5 root       0 -20       0      0      0 S   0.0   0.0   0:00.00
           kworker/0:0H
16      7 root      rt   0       0      0      0 S   0.0   0.0   0:00.00
           migration/0
17  ...
```

Open Files

`lsof` command is used to display a list of all the open files and the processes. The open files included are disk files, network sockets, pipes, devices and processes. One of the main reason for using this command is when a disk cannot be unmounted and displays the error that files are being used or opened. With this commmand you can easily identify which files are in use. Another use is if the Docker Host is running out af filehandles.

To display all open files:

```
1  # lsof
2  COMMAND     PID  TID    USER   FD      TYPE             DEVICE  SIZE/
       OFF       NODE NAME
3  systemd       1         root  cwd       DIR              253,1
       224         64 /
4  systemd       1         root  rtd       DIR              253,1
       224         64 /
5  systemd       1         root  txt       REG              253,1
       1620384     299920 /usr/lib/systemd/systemd
6  systemd       1         root  mem       REG              253,1
       20112     109252 /usr/lib64/libuuid.so.1.3.0
7  systemd       1         root  mem       REG              253,1
       265624     109256 /usr/lib64/libblkid.so.1.1.0
8  systemd       1         root  mem       REG              253,1
       90248     332379 /usr/lib64/libz.so.1.2.7
9  systemd       1         root  mem       REG              253,1
       157424     109251 /usr/lib64/liblzma.so.5.2.2
10 systemd       1         root  mem       REG              253,1
       23968     109278 /usr/lib64/libcap-ng.so.0.0.0
11 systemd       1         root  mem       REG              253,1
       19896     109037 /usr/lib64/libattr.so.1.1.0
12 ...
```

Or to find top 10 processes using file handles:

```
1  # lsof | awk '{print $1}' | sort | uniq -c | sort -r | head -10
```

```
    372 tuned
    234 gssproxy
    159 polkitd
    146 sshd
    146 gmain
    130 master
    106 JS
     88 dbus-daem
     72 systemd
     68 auditd
```

Check Containers

Ensure that the backend services are running on the server as expected

```
# docker-compose ps
      Name                     Command               State
                                                Ports
-------------------------------------------------------------------------------

deploy_cuesta_1     /bin/sh -c /bin/sh -c "if  ...   Up
    0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
deploy_kwanza_1     kwanza serve                     Up
    0.0.0.0:6060->6060/tcp, 0.0.0.0:8000->8000/tcp, 0.0.0.0:8001->8001/
    tcp
deploy_postgres_1   docker-entrypoint.sh postgres    Up
    0.0.0.0:5444->5432/tcp
```

It is important that all three containers are in `Up` state. If one is not running, then it is a problem, as in this example, where the Postgres database container have stopped for some reason.

```
# docker-compose ps
      Name                     Command                State
                                                  Ports
-------------------------------------------------------------------------------

deploy_cuesta_1     /bin/sh -c /bin/sh -c "if  ...   Up
    0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
deploy_kwanza_1     kwanza serve                     Up
    0.0.0.0:6060->6060/tcp, 0.0.0.0:8000->8000/tcp, 0.0.0.0:8001->8001/
    tcp
deploy_postgres_1   docker-entrypoint.sh postgres    Exit 137
```

If a container hosting a service have stopped, try to start it again to resolve the issue. Here we start the stopped Postgres database container.

```
# docker-compose up -d postgres
Starting deploy_postgres_1 ... done
# docker-compose ps
      Name                     Command               State
                                                Ports
-------------------------------------------------------------------------------

deploy_cuesta_1     /bin/sh -c /bin/sh -c "if  ...   Up
    0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
deploy_kwanza_1     kwanza serve                     Up
    0.0.0.0:6060->6060/tcp, 0.0.0.0:8000->8000/tcp, 0.0.0.0:8001->8001/
    tcp
deploy_postgres_1   docker-entrypoint.sh postgres    Up
    0.0.0.0:5444->5432/tcp
```

If the container is unable to start, the issue must be located and resolved in order to restore correct operations.

Operational Monitoring

In order to realise a reliable operation of the backend services, and thus of the complete system, monitoring is the first step towards this goal. Docker containers, are normally brought up and down on demand. They are ephemeral as they are lightweight and can be started up with little system overhead so they could be discarded when not actively in use.

Dockerization ensures the applications to be designed to work as distributed systems with each functional element is run in one more containers. That enabled a container based system to be scaled easily and the available compute resources could be allocated much more efficiently.

The benefits of monitroring are mainly:

- Monitoring helps to identify issues proactively that would help to avoid system outages.
- The monitoring time-series data provide insights to fine-tune applications for better performance and robustness.
- Changes could be rolled out safely as issues will be caught early on and be resolved quickly.
- Environmental changes and the impact these gets monitored indirectly.
- Availability of application services can be determined immediately.

Levels of Monitoring

In order to monitor a container based application environment systematically, the monitoring should be implemented at various levels of the infrastructure and application.

Monitor Docker Host

Docker containers are run on bare-metal or virtual machines. Monitoring of these machines for their availability and performance is important. This falls into the traditional infrastructure monitoring.

Typically, CPU, memory and storage usages are tracked and alerted based on the thresholds setup for those metrics. Implementing those are relatively easy as any monitoring tool would support it as part of core features.

Monitoring Containers

The Docker containers are run on a set of hosts and a specific Docker instance could be running on any one of those hosts. You should monitor the running container instances. Tracking information on the up and running containers would be handy in monitoring the complete system avaliablility and proformance.

As with bare-metal and virtual machines, CPU, memory and storage metrics can be monitored for Docker containers as well. Container specific metrics related to CPU throttling, a situation when CPU cycles are allocated based on priorities set when there would be competition for available CPU, can also be tracked.

Tracking of these system performance metrics would help to determine whether resources on bare-metal and virtual machines, the container hosting infra, need to be upgraded. It would also provide insights to finetune the resources allocated to a Docker image so its future container instances will be started up with adequate runtime resources.

The native Docker command `docker stats` returns some of these metrics, but a surveliance and metric collection system is needed to capture these statistics system wide, for getting notified on potential issues and resolving those proactively.

Monitoring Application Endpoints

A container-based environment would be running a large, highly distributed application with each service running on one or more containers. The application checks could be done both at the container level and system-wide level. REST API endpoints on both Kwanza and Cuesta are available to

perform such checks that could easily be plugged into any modern monitoring system to check the availability of related services.

Kwanza exposes a number of expvars for monitoring the health and performance of the application:

- `Streams` reports the total number of streams (update-channels) currently registered
- `BufferedStreams` reports the number of streams that uses an optimized buffered delivery strategy
- `StreamPanics` the number of panics seen when streaming updates/notifications on streams
- `Notifications` the total number of updates/notifications sent across all streams
- `NotificationsPrSec` current rate of notifications per second
- `Pings` the total number of pings sent, each stream is pinged at a configurable interval
- `PingsPrSec` the current rate of pings per second
- `ChangeNotifications` the total number of notifications that are actual changes
- `ChangeNotificationsPrSec` the current rate of change notifications per second
- `Requests` the total number of requests made across all services/endpoints
- `RequestsPrSec` the current rate of requests per second
- `AuthenticationRequests` the total number of authentication requests
- `AuthenticationSuccesses` the total number of successful auth requests
- `AuthenticationFailures` the total number of failed auth requests
- `AuthenticationRequestsPrSec` the current rate of auth requests per second
- `NotificationTimeouts` the total number of notifications that timed out
- `NotificationPanics` the total number of panics seen when sending notifications
- `NotificationsInFlight` the current number of notifications that are in transit on streams
- `PerSubscriberStreams` a map of a count of streams keyed by the subscriber id
- `ActiveSubscriptions` a count of subscriptions which are currently receiving notifications
- `Subscriptions` a total number for all subscriptions
- `GrpcInterceptorPanics` a count of intercepted panics in the gRPC communications layer
- `SavedTransmissions` a count of avoided transmissions due to the caching/diff layer

Most monitoring solutions support pulling metrics from expvars - a few commandline tools can be used for quick and dirty monitoring, e.g. expvarmon or jplot.

Issue hunting on Postgres Database

In order to resolve issues on the Postgres Database container, the administrator may follo these steps. (For resolving issues in Postgres itsefl, follow best-pratice for Postgres operations and maintenance.)

First try to start Postgres Database container in attached mode. This will give direct feedback on the process in the console. A normal startup will look similar to this (variations may occure based on the

specific version)

```
# docker-compose up postgres
Starting deploy_postgres_1 ... done
Attaching to deploy_postgres_1
postgres_1  | 2019-09-10 12:49:52.785 UTC [1] LOG:  listening on IPv4
    address "0.0.0.0", port 5432
postgres_1  | 2019-09-10 12:49:52.786 UTC [1] LOG:  listening on IPv6
    address "::", port 5432
postgres_1  | 2019-09-10 12:49:52.788 UTC [1] LOG:  listening on Unix
    socket "/var/run/postgresql/.s.PGSQL.5432"
postgres_1  | 2019-09-10 12:49:52.800 UTC [20] LOG:  database system
    was interrupted; last known up at 2019-09-10 12:43:24 UTC
postgres_1  | 2019-09-10 12:49:52.816 UTC [20] LOG:  database system
    was not properly shut down; automatic recovery in progress
postgres_1  | 2019-09-10 12:49:52.819 UTC [20] LOG:  redo starts at
    0/17FE768
postgres_1  | 2019-09-10 12:49:52.819 UTC [20] LOG:  invalid record
    length at 0/17FE7A0: wanted 24, got 0
postgres_1  | 2019-09-10 12:49:52.819 UTC [20] LOG:  redo done at 0/17
    FE768
postgres_1  | 2019-09-10 12:49:52.828 UTC [1] LOG:  database system is
    ready to accept connections
```

Look for hints as to why Postgres Database container will not start as expected. This may include issues such as disk full, low memory, to many open files, corrupt file system, etc. Follow best-pratice for Linux operation and maintenance if you encounter one of these.

Once the issue have been resolved, try to start the container again in attached mode. It startup succedes, then stop the container and start it again in detached mode `-d` on the `docker-compose up` command.

Issue hunting on Kwanza

In order to resolve issues on the Kwanza container, the administrator may follo these steps.

First try to start Kwanza container in attached mode. This will give direct feedback on the process in the console. A normal startup will look similar to this (variations may occure based on the specific version)

```
# docker-compose up kwanza
deploy_postgres_1 is up-to-date
```

```
Starting deploy_kwanza_1 ... done
Attaching to deploy_kwanza_1
kwanza_1    | Either cert or key already exists, aborted cert- and key-generation
kwanza_1    | ######### INTERACTIVE #########
kwanza_1    | {"level":"info","ts":1568120211.728028,"caller":"runner/runner.go:71","msg":"expvars available","port":8080}
kwanza_1    | {"level":"info","ts":1568120211.7392752,"caller":"runner/runner.go:169","msg":"Successfully initialized PostgreSQL registry"}
kwanza_1    | {"level":"info","ts":1568120211.7431538,"caller":"runner/runner.go:136","msg":"Server started","grpc_port":8001,"http_port":8000}
kwanza_1    | {"level":"info","ts":1568120211.7432065,"caller":"runner/runner.go:132","msg":"Profiling interface running on port 6060"}
```

Look for hints as to why the Kwanza container will not start as expected. This may include issues such as disk full, low memory, to many open files, corrupt file system, etc. Follow best-pratice for Linux operation and maintenance if you encounter one of these.

Once the issue have been resolved, try to start the container again in attached mode. It startup succedes, then stop the container and start it again in detached mode `-d` on the `docker-compose up` command.

Issue hunting on Cuesta

In order to resolve issues on the Cuesta container, the administrator may follo these steps.

First try to start Cuesta container in attached mode. This will give direct feedback on the process in the console. A normal startup will look similar to this (variations may occure based on the specific version)

```
# docker-compose up cuesta
deploy_postgres_1 is up-to-date
Starting deploy_kwanza_1 ... done
Starting deploy_cuesta_1 ... done
Attaching to deploy_cuesta_1
cuesta_1    | Serving with SSL
```

Look for hints as to why the Cuesta container will not start as expected. This may include issues such as disk full, low memory, to many open files, corrupt file system, etc. Follow best-pratice for Linux operation and maintenance if you encounter one of these.

Once the issue have been resolved, try to start the container again in attached mode. It startup succedes, then stop the container and start it again in detached mode `-d` on the `docker-compose up` command.

Backup and Restore

The configuration data of Sirenia Automation and Sirenia Context Management is stored in the Postgres Database hosted in the Postgres container. The actual data is stored on a Volumen mounted to the container on container start time. Backing up databases and the ability to restore, is one of the most critical tasks in secure system operation. The administrator should follow best pratice for Postgres Database backup and restore.

Backup Kwanza Database

Before backing up the databases, the administrator should consider the following points:

- Full / partial databases
- Both data and structures, or only structures
- Point In Time recovery
- Restore performance

PostgreSQL provides `pg_dump` and `pg_dumpall` tools to help you backup databases easily and effectively. To backup one database, you can use the `pg_dump` tool. The `pg_dump` dumps out the content of all database objects into a single file.

```
1 #pg_dump -U postgres -W -F t kwanza > kwanza.tar
```

- -U postgres: specifies the user to connect to PostgreSQL database server.
- -W: forces pg_dump to prompt for the password before connecting to the PostgreSQL database server.
- -F t: specifies the output file format to be tar format.
- kwanza: is the name of the database that we want to backup

Restore Kwanza Database

You can use `pg_restore` program to restore databases backed up by the `pg_dump` or `pg_dumpall` tools. With `pg_restore` program, you have various options for restoration databases, for example:

- The `pg_restore` allows you to perform parallel restores using the `-j` option to specify the number of threads for restoration. Each thread restores a separate table simultaneously, which

speeds up the process dramatically. Currently, the `pg_restore` support this option for the only custom file format.

- The `pg_restore` enables you to restore specific database objects in a backup file that contains the full database.
- The `pg_restore` can take a database backed up in the older version and restore it in the newer version.

To restore a Kwanza database backup, create a new database named kwanza:

```
1 CREATE DATABASE kwanza;
```

You can restore the kwanza database in tarfile format generated by the `pg_dump` tool using the following command:

```
1 #pg_restore --dbname=kwanza --verbose kwanza.tar
```

If you restore the database, which is the same as the one that you backed up, you can use the following command:

```
1 #pg_restore --dbname=kwanza --create --verbose kwanza.tar
```