
Operations Guide for Linux

all

Sirenia

September 23, 2020



Contents

1	Content	2
2	Health Check of a System	3
2.1	Check Docker Host	3
2.2	Memory and CPU	3
2.3	Disk Usage	4
2.4	Running Processes	4
2.5	Open Files	5
2.6	Check Containers	6
3	Operational Monitoring	7
3.1	Levels of Monitoring	8
3.2	Monitor Docker Host	8
3.3	Monitoring Containers	8
3.4	Monitoring Application Endpoints	9
4	Issue hunting on Postgres Database	9
5	Issue hunting on Kwanza	10
6	Issue hunting on Cuesta	11
7	Backup and Restore	11
7.1	Backup Kwanza Database	11
7.2	Restore Kwanza Database	12

1 Content

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 knowledge of Linux and Docker administration is assumed.

2 Health Check of a System

Login to the backend server

```
1 ssh user@<server>
2 sudo su
3 cd deploy
```

2.1 Check Docker Host

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

2.2 Memory and CPU

Linux `vmstat` command is used to display statistics of virtual memory, kernel 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:

```
1 # vmstat
2 procs -----memory----- ---swap-- -----io----- -system--
   -----cpu-----
3  r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sy
   id wa st
4  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 availability on the Docker Host. The `free` command provides information about unused and used memory and swap space

```
1 # free
2              total        used        free        shared  buff/cache
3 Mem:         1014992        91936        418376        63876        504680
              685536
4 Swap:          0           0           0
```

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

2.3 Disk Usage

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

```
1 # df -h
2 Filesystem      Size  Used Avail Use% Mounted on
3 /dev/vda1        25G   3.8G   22G  16% /
4 devtmpfs         473M    0   473M   0% /dev
5 tmpfs            496M    0   496M   0% /dev/shm
6 tmpfs            496M   63M   434M  13% /run
7 tmpfs            496M    0   496M   0% /sys/fs/cgroup
8 tmpfs            100M    0   100M   0% /run/user/0
```

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

2.4 Running Processes

`top` and `htop` commands are performance monitoring programs used by many system administrators to monitor Linux performance. The commands are used to display 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 processes. The `top` command is much useful 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)

```
1 # top
2 top - 11:23:31 up 21 days, 21:03,  1 user,  load average: 0.13, 0.08,
   0.06
3 Tasks:  89 total,   2 running,  87 sleeping,   0 stopped,   0 zombie
4 %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
5 KiB Mem : 1014992 total,   421240 free,    93980 used,   499772 buff/
   cache
6 KiB Swap:      0 total,      0 free,      0 used.   684172 avail
   Mem
7
8  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+
   COMMAND
9  6472 root      20   0  158748   5280   3976 S   0.7   0.5    0:00.02 sshd
10  6471 root      20   0  161896   2216   1560 R   0.3   0.2    0:00.04 top
11  6473 sshd      20   0  117204   2828   1712 S   0.3   0.3    0:00.01 sshd
```

12	1	root	20	0	46092	6532	4128	S	0.0	0.6	0:37.97
		systemd									
13	2	root	20	0	0	0	0	S	0.0	0.0	0:00.16
		kthreadd									
14	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	...										

2.5 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 command you can easily identify which files are in use. Another use is if the Docker Host is running out of filehandles.

To display all open files:

1	#	lsof									
2	COMMAND	PID	TID	USER	FD	TYPE		DEVICE	SIZE/		
	OFF		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
2     372 tuned
3     234 gssproxy
4     159 polkitd
5     146 sshd
6     146 gmain
7     130 master
8     106 JS
9      88 dbus-daem
10     72 systemd
11     68 auditd

```

2.6 Check Containers

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

```

1 # docker-compose ps
2      Name                                Command                                State
3      -----
4  deploy_cuesta_1    /bin/sh -c /bin/sh -c "if ...    Up
5  deploy_kwanza_1    kwanza serve                        Up
6  deploy_postgres_1  docker-entrypoint.sh postgres    Up

```

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.

```

1 # docker-compose ps
2      Name                                Command                                State
3      -----
4  deploy_cuesta_1    /bin/sh -c /bin/sh -c "if ...    Up
5  deploy_kwanza_1    kwanza serve                        Up
6  deploy_postgres_1  docker-entrypoint.sh postgres    Up

```

```

3 -----
4 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
5 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
6 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.

```

1 # docker-compose up -d postgres
2 Starting deploy_postgres_1 ... done
3 # docker-compose ps
4      Name                                Command                                State
5 -----
6 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
7 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
8 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.

3 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 monitoring 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.

3.1 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.

3.2 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.

3.3 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 availability and performance.

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 surveillance and metric collection system is needed to capture these statistics system wide, for getting notified on potential issues and resolving those proactively.

3.4 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.

4 Issue hunting on Postgres Database

In order to resolve issues on the Postgres Database container, the administrator may follow these steps. (For resolving issues in Postgres itself, follow best-practice 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 occur based on the specific version)

```
1 # docker-compose up postgres
2 Starting deploy_postgres_1 ... done
3 Attaching to deploy_postgres_1
4 postgres_1 | 2019-09-10 12:49:52.785 UTC [1] LOG:  listening on IPv4
   address "0.0.0.0", port 5432
5 postgres_1 | 2019-09-10 12:49:52.786 UTC [1] LOG:  listening on IPv6
   address ":::", port 5432
6 postgres_1 | 2019-09-10 12:49:52.788 UTC [1] LOG:  listening on Unix
   socket "/var/run/postgresql/.s.PGSQL.5432"
7 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
8 postgres_1 | 2019-09-10 12:49:52.816 UTC [20] LOG:  database system
   was not properly shut down; automatic recovery in progress
9 postgres_1 | 2019-09-10 12:49:52.819 UTC [20] LOG:  redo starts at
   0/17FE768
10 postgres_1 | 2019-09-10 12:49:52.819 UTC [20] LOG:  invalid record
   length at 0/17FE7A0: wanted 24, got 0
11 postgres_1 | 2019-09-10 12:49:52.819 UTC [20] LOG:  redo done at 0/17
   FE768
```

```
12 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, too many open files, corrupt file system, etc. Follow best-practice for Linux operation and maintenance if you encounter one of these.

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

5 Issue hunting on Kwanza

In order to resolve issues on the Kwanza container, the administrator may follow 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 occur based on the specific version)

```
1 # docker-compose up kwanza
2 deploy_postgres_1 is up-to-date
3 Starting deploy_kwanza_1 ... done
4 Attaching to deploy_kwanza_1
5 kwanza_1      | Either cert or key already exists, aborted cert- and key-
    generation
6 kwanza_1      | ##### INTERACTIVE #####
7 kwanza_1      | {"level":"info","ts":1568120211.728028,"caller":"runner/
    runner.go:71","msg":"expvars available","port":8080}
8 kwanza_1      | {"level":"info","ts":1568120211.7392752,"caller":"runner/
    runner.go:169","msg":"Successfully initialized PostgreSQL registry"}
9 kwanza_1      | {"level":"info","ts":1568120211.7431538,"caller":"runner/
    runner.go:136","msg":"Server started","grpc_port":8001,"http_port"
    :8000}
10 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, too many open files, corrupt file system, etc. Follow best-practice for Linux operation and maintenance if you encounter one of these.

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

command.

6 Issue hunting on Cuesta

In order to resolve issues on the Cuesta container, the administrator may follow 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 occur based on the specific version)

```
1 # docker-compose up cuesta
2 deploy_postgres_1 is up-to-date
3 Starting deploy_kwanza_1 ... done
4 Starting deploy_cuesta_1 ... done
5 Attaching to deploy_cuesta_1
6 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, too many open files, corrupt file system, etc. Follow best-practice for Linux operation and maintenance if you encounter one of these.

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

7 Backup and Restore

The configuration data of Siren timer Automation and Siren timer Context Management is stored in the Postgres Database hosted in the Postgres container. The actual data is stored on a Volume 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 practice for Postgres Database backup and restore.

7.1 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

7.2 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
```