# Intro to Podman (Red Hat Enterprise Linux 7.6 Beta)



[Red Hat Enterprise Linux (RHEL) 7.6 Beta](https://www.redhat.com/en/blog/red-hat-enterprise-linux-76-beta-now-available) was released a few days ago and one of the first new features I noticed is Podman. Podman complements [Buildah](https://www.projectatomic.io/blog/2017/06/introducing-buildah/) and [Skopeo](https://www.projectatomic.io/blog/2016/07/working-with-containers-image-made-easy/) by offering an experience similar to the Docker command line: allowing users to run standalone (non-orchestrated) containers. And Podman doesn’t require a daemon to run containers and pods, so we can easily say goodbye to big fat daemons.

Podman implements almost all the Docker CLI commands (apart from the ones related to Docker Swarm, of course). For container orchestration, I suggest you take a look at Kubernetes and [Red Hat OpenShift](http://openshift.com/).

Podman consists of just a single command to run on the command line. There are no daemons in the background doing stuff, and this means that Podman can be integrated into system services through systemd.

We’ll cover some real examples that show how easy it can be to transition from the Docker CLI to Podman.

## Podman installation

If you are running Red Hat Enterprise Linux 7.6 Beta, follow the steps below.  If not, you can [try Podman online with Katacoda](http://katacoda.com/courses/containers-without-docker/running-containers-with-podman).

You need to enable the extras repo:

**$ su -**

**# subscription-manager repos --enable rhel-7-server-extras-beta-rpms**

**Please note:** at the time this was written RHEL 7.6 is still in beta. Once GA occurs, please change the repository name by removing the -beta-.

Then, launch the proper installation command:

**# yum -y install podman**

This command will install Podman and also its dependencies: atomic-registries, runC, skopeo-containers, and SELinux policies.

That’s all. Now you can now play with Podman.

#### Develop using Red Hat's most valuable products

Your membership unlocks Red Hat products and technical training on enterprise cloud application development.

[**JOIN RED HAT DEVELOPER**](https://developers.redhat.com/register/?intcmp=701f2000001OMHaAAO)

[](https://developers.redhat.com/register/?intcmp=701f2000001OMHaAAO)

## Command-line examples

### Run a RHEL container

For the first example, suppose we want to just  run a RHEL container. We are on a RHEL system and we want to run a RHEL container, so it should work:

**[root@localhost ~]# docker run -it rhel sh**

**-bash: docker: command not found**

As you can see, there is no docker command on my RHEL 7.6 host. Just replace the docker command with podman:

**[root@localhost ~]# podman run -it rhel sh**

**Trying to pull registry.access.redhat.com/rhel:latest...Getting image source signatures**

**Copying blob sha256:367d845540573038025f445c654675aa63905ec8682938fb45bc00f40849c37b**

**71.46 MB / ? [------------=----------------------------------------------] 23s**

**Copying blob sha256:b82a357e4f15fda58e9728fced8558704e3a2e1d100e93ac408edb45fe3a5cb9**

**1.27 KB / ? [----=--------------------------------------------------------] 0s**

**Copying config sha256:f5ea21241da8d3bc1e92d08ca4888c2f91ed65280c66acdefbb6d2dba6cd0b29**

**6.52 KB / 6.52 KB [========================================================] 0s**

**Writing manifest to image destination**

**Storing signatures**

**sh-4.2#**

We now have our RHEL container. Let’s play with it, check its status, and then delete it and its source image:

**sh-4.2# ps ax**

**PID TTY STAT TIME COMMAND**

**1 pts/0 Ss 0:00 sh**

**10 pts/0 R+ 0:00 ps ax**

**sh-4.2# exit**

**[root@localhost ~]# podman ps -a**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**deda2991f9fd registry.access.redhat.com/rhel:latest sh 3 minutes ago Exited (0) Less than a second ago reverent\_torvalds**

**[root@localhost ~]# podman rm deda2991f9fd**

**deda2991f9fd43400566abceaa917ecbd59a2e83354c5c9021ba1830a7ab196d**

**[root@localhost ~]# podman image rm rhel**

**f5ea21241da8d3bc1e92d08ca4888c2f91ed65280c66acdefbb6d2dba6cd0b29**

As you can see, we used the same syntax as we’d use with docker. There are no differences at the moment. I didn’t check the Podman documentation and I started working immediately!

### Run a MariaDB persistent container

Let’s move forward and try a more complicated test: run MariaDB 10.2 with some custom variables and try to let its “data” be persistent.

First, let’s download the MariaDB container image and inspect its details:

**[root@localhost ~]# podman pull registry.access.redhat.com/rhscl/mariadb-102-rhel7Trying to pull registry.access.redhat.com/rhscl/mariadb-102-rhel7...Getting image source signatures**

**Copying blob sha256:367d845540573038025f445c654675aa63905ec8682938fb45bc00f40849c37b**

**71.46 MB / ? [------------=----------------------------------------------] 10s**

**Copying blob sha256:b82a357e4f15fda58e9728fced8558704e3a2e1d100e93ac408edb45fe3a5cb9**

**1.27 KB / ? [----=--------------------------------------------------------] 0s**

**Copying blob sha256:ddec0f65683ad89fc27298921921b2f8cbf57f674ed9eb71eef4e23a9dd9bbfe**

**6.40 MB / ? [--------------=----------------------------------------------] 1s**

**Copying blob sha256:105cfda934d478ffbf65d74a89af55cc5de1d5bc94874c2d163c45e31a937047**

**58.25 MB / ? [-------------------------------------------=---------------] 10s**

**Copying config sha256:7ac0a23445fec91d4b458f3062e64d1ca4af4755387604f8d8cbec08926867d7**

**6.79 KB / 6.79 KB [========================================================] 0s**

**Writing manifest to image destination**

**Storing signatures**

**7ac0a23445fec91d4b458f3062e64d1ca4af4755387604f8d8cbec08926867d7**

**[root@localhost ~]# podman images**

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**registry.access.redhat.com/rhscl/mariadb-102-rhel7 latest 7ac0a23445fe 9 days ago 445MB**

**[root@localhost ~]# podman inspect 7ac0a23445fe**

**...**

Then we can set up a folder that will handle MariaDB’s data once we start our container:

**[root@localhost ~]# mkdir mysql-data**

**[root@localhost ~]# chown 27:27 mysql-data**

**Please note:** “27” is the ID of the mysql user that will run the MariaDB’s processes in the container. For this reason, we have to allow it to read from and write to the directory.

And finally, run it:

**[root@localhost ~]# podman run -d -v /root/mysql-data:/var/lib/mysql/data:Z -e MYSQL\_USER=user -e MYSQL\_PASSWORD=pass -e MYSQL\_DATABASE=db -p 3306:3306 registry.access.redhat.com/rhscl/mariadb-102-rhel7**

**71da2bb210b36aaab28a2dc81b8e77da4e1024d1f2d025c0a7b97b075dec1425**

**[root@localhost ~]# podman ps**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**71da2bb210b3 registry.access.redhat.com/rhscl/mariadb-102-rhel7:latest container-entrypoin... 3 seconds ago Up 3 seconds ago 0.0.0.0:3306->3306/udp, 0.0.0.0:3306->3306/tcp cranky\_mahavira**

As you can see, the container is up and running, but what is it doing? Let’s check:

**[root@localhost ~]# podman logs 71da2bb210b3 | head**

**=> sourcing 20-validate-variables.sh ...**

**=> sourcing 25-validate-replication-variables.sh ...**

**=> sourcing 30-base-config.sh ...**

**---> 13:12:43 Processing basic MySQL configuration files ...**

**=> sourcing 60-replication-config.sh ...**

**=> sourcing 70-s2i-config.sh ...**

**---> 13:12:43 Processing additional arbitrary MySQL configuration provided by s2i ...**

**=> sourcing 40-paas.cnf ...**

**=> sourcing 50-my-tuning.cnf ...**

**---> 13:12:43 Initializing database ...**

Ah! It’s just started and initialized its database. Let’s play with it:

**[root@localhost ~]# mysql --user=user --password=pass -h 127.0.0.1 -P 3306 -t**

**Welcome to the MariaDB monitor. Commands end with ; or \g.**

**Your MariaDB connection id is 8**

**Server version: 10.2.8-MariaDB MariaDB Server**

**Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.**

**Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.**

**MariaDB [(none)]> show databases;**

**+--------------------+**

**| Database |**

**+--------------------+**

**| db |**

**| information\_schema |**

**| test |**

**+--------------------+**

**3 rows in set (0.00 sec)**

**MariaDB [(none)]> use test;**

**Database changed**

**MariaDB [test]> show tables;**

**Empty set (0.00 sec)**

Perfect. Now we’ll create at least a table and then we’ll terminate the container:

**MariaDB [db]> CREATE TABLE mytest (username VARCHAR(20), date DATETIME);**

**Query OK, 0 rows affected (0.02 sec)**

**MariaDB [db]> show tables;**

**+--------------+**

**| Tables\_in\_db |**

**+--------------+**

**| mytest |**

**+--------------+**

**1 row in set (0.00 sec)**

**MariaDB [db]> Bye**

**[root@localhost ~]# podman kill 71da2bb210b3**

**71da2bb210b36aaab28a2dc81b8e77da4e1024d1f2d025c0a7b97b075dec1425**

Inspecting the content of the folder, we can see that data is still there, but let’s start a new container for checking the data persistence:

**[root@localhost ~]# ls -la mysql-data/**

**total 41024**

**drwxr-xr-x. 6 27 27 4096 Aug 24 09:12 .**

**dr-xr-x---. 4 root root 219 Aug 24 09:28 ..**

**-rw-rw----. 1 27 27 2 Aug 24 09:12 71da2bb210b3.pid**

**-rw-rw----. 1 27 27 16384 Aug 24 09:12 aria\_log.00000001**

**-rw-rw----. 1 27 27 52 Aug 24 09:12 aria\_log\_control**

**drwx------. 2 27 27 56 Aug 24 09:27 db**

**-rw-rw----. 1 27 27 2799 Aug 24 09:12 ib\_buffer\_pool**

**-rw-rw----. 1 27 27 12582912 Aug 24 09:27 ibdata1**

**-rw-rw----. 1 27 27 8388608 Aug 24 09:27 ib\_logfile0**

**-rw-rw----. 1 27 27 8388608 Aug 24 09:12 ib\_logfile1**

**-rw-rw----. 1 27 27 12582912 Aug 24 09:12 ibtmp1**

**-rw-rw----. 1 27 27 0 Aug 24 09:12 multi-master.info**

**drwx------. 2 27 27 4096 Aug 24 09:12 mysql**

**-rw-r--r--. 1 27 27 14 Aug 24 09:12 mysql\_upgrade\_info**

**drwx------. 2 27 27 20 Aug 24 09:12 performance\_schema**

**-rw-rw----. 1 27 27 24576 Aug 24 09:12 tc.log**

**drwx------. 2 27 27 6 Aug 24 09:12 test**

**[root@localhost ~]# podman run -d -v /root/mysql-data:/var/lib/mysql/data:Z -e MYSQL\_USER=user -e MYSQL\_PASSWORD=pass -e MYSQL\_DATABASE=db -p 3306:3306 registry.access.redhat.com/rhscl/mariadb-102-rhel7**

**0364513f6b6ae1b86ea3752ec732bad757770ca14ec1f879e7487f3f4293004d**

**[root@localhost ~]# podman ps**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**0364513f6b6a registry.access.redhat.com/rhscl/mariadb-102-rhel7:latest container-entrypoin... 3 seconds ago Up 2 seconds ago 0.0.0.0:3306->3306/udp, 0.0.0.0:3306->3306/tcp heuristic\_northcutt**

**[root@localhost ~]# mysql --user=user --password=pass -h 127.0.0.1 -P 3306 -t**

**Welcome to the MariaDB monitor. Commands end with ; or \g.**

**Your MariaDB connection id is 8**

**Server version: 10.2.8-MariaDB MariaDB Server**

**Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.**

**Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.**

**MariaDB [(none)]> use db;**

**Reading table information for completion of table and column names**

**You can turn off this feature to get a quicker startup with -A**

**Database changed**

**MariaDB [db]> show tables;**

**+--------------+**

**| Tables\_in\_db |**

**+--------------+**

**| mytest |**

**+--------------+**

**1 row in set (0.00 sec)**

**MariaDB [db]> Bye**

**[root@localhost ~]# podman kill 0364513f6b6a**

**0364513f6b6ae1b86ea3752ec732bad757770ca14ec1f879e7487f3f4293004d**

Great! MariaDB’s data is still there and the new container read it and showed it once as requested.

### Manage containers as system services through systemd and Podman

Finally, we’ll create a simple systemd resource for handling the previously created MariaDB container.

First, we need to create a systemd resource file for handling the brand new container service:

**[root@localhost ~]# cat /etc/systemd/system/mariadb-podman.service**

**[Unit]**

**Description=Custom MariaDB Podman Container**

**After=network.target**

**[Service]**

**Type=simple**

**TimeoutStartSec=5m**

**ExecStartPre=-/usr/bin/podman rm "mariadbpodman"**

**ExecStart=/usr/bin/podman run --name mariadbpodman -v /root/mysql-data:/var/lib/mysql/data:Z -e MYSQL\_USER=user -e MYSQL\_PASSWORD=pass -e MYSQL\_DATABASE=db -p 3306:3306 registry.access.redhat.com/rhscl/mariadb-102-rhel7**

**ExecReload=-/usr/bin/podman stop "mariadbpodman"**

**ExecReload=-/usr/bin/podman rm "mariadbpodman"**

**ExecStop=-/usr/bin/podman stop "mariadbpodman"**

**Restart=always**

**RestartSec=30**

**[Install]**

Then we can reload the systemd catalog and start the service:

**[root@localhost ~]# systemctl daemon-reload**

**[root@localhost ~]# systemctl start mariadb-podman**

**[root@localhost ~]# systemctl status mariadb-podman**

**mariadb-podman.service - Custom MariaDB Podman Container**

**Loaded: loaded (/etc/systemd/system/mariadb-podman.service; static; vendor preset: disabled)**

**Active: active (running) since Fri 2018-08-24 10:14:36 EDT; 3s ago**

**Process: 19147 ExecStartPre=/usr/bin/podman rm mariadbpodman (code=exited, status=0/SUCCESS)**

**Main PID: 19172 (podman)**

**CGroup: /system.slice/mariadb-podman.service**

**└─19172 /usr/bin/podman run --name mariadbpodman -v /root/mysql-data:/var/lib/mysql/data:Z -e MYSQL\_USER=user -e MYSQL\_PASSWORD=pass -e MYSQL\_DA...**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140578968823552 [Note] InnoDB: Buffer pool(s) load completed at 180824 14:14:39**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Note] Plugin 'FEEDBACK' is disabled.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Note] Server socket created on IP: '::'.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Warning] 'user' entry 'root@71da2bb210b3' ignored in --sk...ve mode.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Warning] 'user' entry '@71da2bb210b3' ignored in --skip-n...ve mode.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Warning] 'proxies\_priv' entry '@% root@71da2bb210b3' igno...ve mode.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Note] Reading of all Master\_info entries succeded**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Note] Added new Master\_info '' to hash table**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: 2018-08-24 14:14:39 140579889719488 [Note] /opt/rh/rh-mariadb102/root/usr/libexec/mysqld: read...ections.**

**Aug 24 10:14:39 localhost.localdomain podman[19172]: Version: '10.2.8-MariaDB' socket: '/var/lib/mysql/mysql.sock' port: 3306 MariaDB Server**

**Hint: Some lines were ellipsized, use -l to show in full.**

**[root@localhost ~]# systemctl stop mariadb-podman**

**[root@localhost ~]#**

Awesome! We just set up a custom system service based on a container managed through Podman.

**Deploying SQL Server 2019 container on RHEL 8 with podman**

Having a fresh install of RHEL8 on my lab environment, I was curious to take a look at new containerization stuff from Red Hat in the context of SQL Server 2019. Good chances are the future version of SQL Server should be available and supported on with the latest version of Red Hat but for now this blog post is purely experimental. This time I wanted to share with you some thoughts about the new Podman command.

[](https://blog.dbi-services.com/wp-insides/uploads/sites/2/2019/07/159-0-RHEL8-podman-SQL.jpg)

First of all, we should be aware that since RHEL8 Red Hat decided to replace docker with CRI-O/podman in order to provide a “daemonless” container world and especially for Kubernetes. By 2016, Kubernetes project introduced the [Container Runtime Interface](https://github.com/kubernetes/kubernetes/blob/242a97307b34076d5d8f5bbeb154fa4d97c9ef1d/docs/devel/container-runtime-interface.md) (CRI).  Basically, with CRI, Kubernetes can be container runtime-agnostic. [CRI-O](https://cri-o.io/) that is an open source project initiated by Red Hat the same year that gives the ability to run containers directly from Kubernetes without any unnecessary code or tooling as long as the container remains OCI-compliant. Because Docker is not implemented anymore (and officially not supported) by Red Hat since RHEL8, we need a client tool for working with containers and this is where [Podman](https://developers.redhat.com/blog/2018/08/29/intro-to-podman/) steps in. To cut the story short, Podman implements almost all the Docker CLI commands and more.

So, let’s have an overview of Podman commands through the installation of a SQL Server 2019 based container. It is worth noting that Podman is not intended to be used in the context of a “standalone” container environnement and should be used with an container orchestrator like K8s or an orchestration platform like OpenShift.  That said,  let’s first create a host directory to persist the SQL Server database files.

|  |  |
| --- | --- |
| 1  2 | $ sudo mkdir -p  /var/mssql/data  $ sudo chmod 755 -R /var/mssql/data |

Then let’s download the SQL Server 2019 RHEL image. We will use the following Podman command:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $ sudo podman pull mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1  Trying to pull mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1...Getting image source signatures  Copying blob 079e961eee89: 70.54 MiB / 70.54 MiB [========================] 1m3s  Copying blob 1b493d38a6d3: 1.20 KiB / 1.20 KiB [==========================] 1m3s  Copying blob 89e62e5b4261: 333.24 MiB / 333.24 MiB [======================] 1m3s  Copying blob d39017c722a8: 174.82 MiB / 174.82 MiB [======================] 1m3s  Copying config dbba412361d7: 4.98 KiB / 4.98 KiB [==========================] 0s  Writing manifest to image destination  Storing signatures  dbba412361d7ca4fa426387e1d6fc3ec85e37d630bfe70e6599b5116d392394d |

Note that if you’re already comfortable with the Docker commands, the shift to Podman will be easy thanks to the similarity between the both tools. To get information of the new fresh image, we will use the following Podman command:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | $ sudo podman images  REPOSITORY                            TAG           IMAGE ID       CREATED       SIZE  mcr.microsoft.com/mssql/rhel/server   2019-CTP3.1   dbba412361d7   3 weeks ago   1.79 GB  $ sudo podman inspect dbba  …  "GraphDriver": {              "Name": "overlay",              "Data": {                  "LowerDir": "/var/lib/containers/storage/overlay/b2769e971a1bdb62f1c0fd9dcc0e9fe727dca83f52812abd34173b49ae55e37d/diff:/var/lib/containers/storage/overlay/4b0cbf0d9d0ff230916734a790f47ab2adba69db44a79c8eac4c814ff4183c6d/diff:/var/lib/containers/storage/overlay/9197342671da8b555f200e47df101da5b7e38f6d9573b10bd3295ca9e5c0ae28/diff",                  "MergedDir": "/var/lib/containers/storage/overlay/b372c0d6ff718d2d182af4639870dc6e4247f684d81a8b2dc2649f8517b9fc53/merged",                  "UpperDir": "/var/lib/containers/storage/overlay/b372c0d6ff718d2d182af4639870dc6e4247f684d81a8b2dc2649f8517b9fc53/diff",                  "WorkDir": "/var/lib/containers/storage/overlay/b372c0d6ff718d2d182af4639870dc6e4247f684d81a8b2dc2649f8517b9fc53/work"              }          },  … |

As show above, Podman uses the CRI-O back-end store directory with the /var/lib/containers path, instead of using the Docker default storage location (/var/lib/docker).

Go ahead and let’s take a look at the Podman info command:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | $ podman info  …  OCIRuntime:      package: runc-1.0.0-54.rc5.dev.git2abd837.module+el8+2769+577ad176.x86\_64      path: /usr/bin/runc      version: 'runc version spec: 1.0.0'  …  store:    ConfigFile: /home/clustadmin/.config/containers/storage.conf    ContainerStore:      number: 0    GraphDriverName: overlay |

The same kind of information is provided by the Docker info command including the runtime and the graph driver name that is overlay in my case. Generally speaking, creating and getting information of a container with Podman is pretty similar to what we may use with the usual Docker commands. Here  for instance the command to spin up a SQL Server container based on the RHEL image:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $ sudo podman run -d -e 'ACCEPT\_EULA=Y' -e \  > 'MSSQL\_SA\_PASSWORD=Password1'  \  > --name 'sqltest' \  > -p 1460:1433 \  > -v /var/mssql/data:/var/opt/mssql/data:Z \  > mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1  4f5128d36e44b1f55d23e38cbf8819041f84592008d0ebb2b24ff59065314aa4  $ sudo podman ps  CONTAINER ID  IMAGE                                            COMMAND               CREATED        STATUS            PORTS                   NAMES  4f5128d36e44  mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1  /opt/mssql/bin/sq...  4 seconds ago  Up 3 seconds ago  0.0.0.0:1460->1433/tcp  sqltest |

Here comes the interesting part. Looking at the pstree output we may notice that there is not dependencies with any (docker) daemon with CRI-O implementation. Usually with the Docker implementation we retrieve the containerd daemon and the related shim for the process within the tree.

|  |  |
| --- | --- |
| 1  2  3  4  5 | $ pstree  systemd─┬─NetworkManager───2\*[{NetworkManager}]          ├─…          ├─conmon─┬─sqlservr─┬─sqlservr───138\*[{sqlservr}]          │        │          └─{sqlservr} |

By using the runc command below, we may notice the MSSQL container (identified by the ID here) is actually running through CRI-O and runc runtime.

|  |  |
| --- | --- |
| 1  2 | $ sudo runc list -q  4f5128d36e44b1f55d23e38cbf8819041f84592008d0ebb2b24ff59065314aa4 |

Let’s have a look at the existing namespace. The 9449 PID corresponds to the SQL Server process running in isolation mode through Linux namespaces.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | $ sudo lsns  …  4026532116 net         2  9449 root   /opt/mssql/bin/sqlservr  4026532187 mnt         2  9449 root   /opt/mssql/bin/sqlservr  4026532188 uts         2  9449 root   /opt/mssql/bin/sqlservr  4026532189 ipc         2  9449 root   /opt/mssql/bin/sqlservr  4026532190 pid         2  9449 root   /opt/mssql/bin/sqlservr    $ ps aux | grep sqlservr  root       9449  0.1  0.6 152072 25336 ?        Ssl  05:08   0:00 /opt/mssql/bin/sqlservr  root       9465  5.9 18.9 9012096 724648 ?      Sl   05:08   0:20 /opt/mssql/bin/sqlservr  clustad+   9712  0.0  0.0  12112  1064 pts/0    S+   05:14   0:00 grep --color=auto sqlservr |

We can double check that the process belongs to the SQL Server container by using the nsenter command:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | sudo nsenter -t 17182 --mount --uts --ipc --net --pid sh  sh-4.2# ps aux  USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  root          1  0.0  0.7 152076 28044 ?        Ssl  Jul23   0:00 /opt/mssql/bin/sqlservr  root          9  2.2 19.7 9034224 754820 ?      Sl   Jul23   0:28 /opt/mssql/bin/sqlservr  root        319  0.0  0.0  13908  3400 ?        S    00:01   0:00 sh  root        326  0.0  0.1  53832  3900 ?        R+   00:02   0:00 ps aux |

Well, we used different Podman commands to spin up a container that meets the OCI specification like Docker. For a sake of curiosity, let’s build a custom image from a Dockerfile. In fact, this is a custom image we developed for customers to meet our best practices requirements.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | $ ls -l  total 40  drwxrwxr-x. 2 clustadmin clustadmin   70 Jul 24 02:06 BestPractices  drwxrwxr-x. 2 clustadmin clustadmin   80 Jul 24 02:06 DMK  -rw-rw-r--. 1 clustadmin clustadmin  614 Jul 24 02:06 docker-compose.yml  -rw-rw-r--. 1 clustadmin clustadmin 2509 Jul 24 02:06 Dockerfile  -rw-rw-r--. 1 clustadmin clustadmin 3723 Jul 24 02:06 entrypoint.sh  -rw-rw-r--. 1 clustadmin clustadmin 1364 Jul 24 02:06 example.docker-swarm-compose.yml  -rw-rw-r--. 1 clustadmin clustadmin  504 Jul 24 02:06 healthcheck.sh  -rw-rw-r--. 1 clustadmin clustadmin   86 Jul 24 02:06 mssql.conf  -rw-rw-r--. 1 clustadmin clustadmin 4497 Jul 24 02:06 postconfig.sh  -rw-rw-r--. 1 clustadmin clustadmin 2528 Jul 24 02:06 Readme.md  drwxrwxr-x. 2 clustadmin clustadmin   92 Jul 24 02:06 scripts |

To build an image from a Dockerfile the corresponding Podman command is as follow:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | $ sudo podman build -t dbi\_mssql\_linux:2019-CTP3.1 .  …  --> 5db120fba51f3adc7482ec7a9fed5cc4194f13e97b855d9439a1386096797c39  STEP 65: FROM 5db120fba51f3adc7482ec7a9fed5cc4194f13e97b855d9439a1386096797c39  STEP 66: EXPOSE ${MSSQL\_TCP\_PORT}  --> 8b5e8234af47adb26f80d64abe46715637bd48290b4a6d7711ddf55c393cd5a8  STEP 67: FROM 8b5e8234af47adb26f80d64abe46715637bd48290b4a6d7711ddf55c393cd5a8  STEP 68: ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]  --> 11045806b8af7cf2f67e5a279692e6c9e25212105bcd104ed17b235cdaea97fe  STEP 69: FROM 11045806b8af7cf2f67e5a279692e6c9e25212105bcd104ed17b235cdaea97fe  STEP 70: CMD ["tail -f /dev/null"]  --> bcb8c26d503010eb3e5d72da4b8065aa76aff5d35fac4d7958324ac3d97d5489  STEP 71: FROM bcb8c26d503010eb3e5d72da4b8065aa76aff5d35fac4d7958324ac3d97d5489  STEP 72: HEALTHCHECK --interval=15s CMD [ "/usr/local/bin/healthcheck.sh" ]  --> e7eedf0576f73c95b19adf51c49459b00449da497cf7ae417e597dd39a9e4c8f  STEP 73: COMMIT dbi\_mssql\_linux:2019-CTP3.1 |

The image built is now available in the local repository:

|  |  |
| --- | --- |
| 1  2  3  4 | $ sudo podman images  REPOSITORY                            TAG           IMAGE ID       CREATED         SIZE  localhost/dbi\_mssql\_linux             2019-CTP3.1   e7eedf0576f7   2 minutes ago   1.79 GB  mcr.microsoft.com/mssql/rhel/server   2019-CTP3.1   dbba412361d7   3 weeks ago     1.79 GB |

The next step will consist in spinning up a SQL Server container based on this new image. Note that I used a custom parameter DMK=Y to drive the creation of the DMK maintenance tool in our case which including the deployment of a custom dbi\_tools database ans related objects that carry out the database maintenance.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ sudo podman run -d -e 'ACCEPT\_EULA=Y' \  > -e 'MSSQL\_SA\_PASSWORD=Password1' -e 'DMK=Y'  \  > --name 'sqltest2' \  > -p 1470:1433 \  > localhost/dbi\_mssql\_linux:2019-CTP3.1  d057e0ca41f08a948de4206e9aa07b53450c2830590f2429e50458681d230f6b |

Let’s check if the dbi\_tools has been created during the container runtime phase:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | $ sudo podman exec -ti d057 /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Password1 -Q"SELECT name from sys.databases"  name  --------------------------------------------------------------------------------------------------------------------------------  master  tempdb  model  msdb  dbi\_tools |

Finally, to make the transition with a future blog post, the Podman tool comes with extra commands (under development) that is not available with Docker CLI. The following example generates a YAML deployment file and the corresponding service from an existing container. Please note however that containers with volumes are not supported yet.

The container definition is a follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ sudo podman run -d -e 'ACCEPT\_EULA=Y' -e \  'MSSQL\_SA\_PASSWORD=Password1'  \  --name 'sqltestwithnovolumes' \  -p 1480:1433 \  mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1  7e99581eaec4c91d7c13af4525bfb3805d5b56e675fdb53d0061c231294cd442 |

And we get the corresponding YAML file generated by the Podman command:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | $ sudo podman generate kube -s 7e99  # Generation of Kubernetes YAML is still under development!  #  # Save the output of this file and use kubectl create -f to import  # it into Kubernetes.  #  # Created with podman-1.0.2-dev  apiVersion: v1  kind: Pod  metadata:    creationTimestamp: 2019-07-24T03:52:18Z    labels:      app: sqltestwithnovolumes    name: sqltestwithnovolumes  spec:    containers:    - command:      - /opt/mssql/bin/sqlservr      env:      - name: PATH        value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin      - name: TERM        value: xterm      - name: HOSTNAME      - name: container        value: oci      - name: ACCEPT\_EULA        value: "Y"      - name: MSSQL\_SA\_PASSWORD        value: Password1      image: mcr.microsoft.com/mssql/rhel/server:2019-CTP3.1      name: sqltestwithnovolumes      ports:      - containerPort: 1433        hostPort: 1480        protocol: TCP      resources: {}      securityContext:        allowPrivilegeEscalation: true        capabilities: {}        privileged: false        readOnlyRootFilesystem: false      workingDir: /  status: {}  ---  apiVersion: v1  kind: Service  metadata:    creationTimestamp: 2019-07-24T03:52:18Z    labels:      app: sqltestwithnovolumes    name: sqltestwithnovolumes  spec:    ports:    - name: "1433"      nodePort: 30309      port: 1433      protocol: TCP      targetPort: 0    selector:      app: sqltestwithnovolumes    type: NodePort  status:    loadBalancer: {} |

By default the service type NodePort has been created by the command. This latest command needs further testing for sure!