Rocky Enterprise Linux 9.2 Manual Pages on command 'docker-run.1'

$ man docker-run.1

podman-run(1)() podman-run(1)()

NAME

podman-run - Run a command in a new container

SYNOPSIS

podman run [options] image [command [arg ...]]

podman container run [options] image [command [arg ...]]

DESCRIPTION

Run a process in a new container. podman run starts a process with its own file system,
its own networking, and its own isolated process tree. The image which starts the process
may define defaults related to the process that will be run in the container, the network?
ing to expose, and more, but podman run gives final control to the operator or administra?
tor who starts the container from the image. For that reason podman run has more options
than any other Podman command.

If the image is not already loaded then podman run will pull the image, and all image de?
pendencies, from the repository in the same way running podman pull image , before it
starts the container from that image.

Several files will be automatically created within the container. These include
/etc/hosts, /etc/hostname, and /etc/resolv.conf to manage networking. These will be based
on the host's version of the files, though they can be customized with options (for exam?
ple, --dns will override the host's DNS servers in the created resolv.conf). Additionally,
a container environment file is created in each container to indicate to programs they are
running in a container. This file is located at /run/.containerenv. When using the --priv?
ileged flag the .containerenv contains name/value pairs indicating the container engine

version, whether the engine is running in rootless mode, the container name and id, as well as the image name and id that the container is based on.

When running from a user defined network namespace, the /etc/netns/NSNAME/resolv.conf will be used if it exists, otherwise /etc/resolv.conf will be used.

Default settings are defined in containers.conf. Most settings for remote connections use the servers containers.conf, except when documented in man pages.

IMAGE

The image is specified using transport:path format. If no transport is specified, the docker (container registry) transport will be used by default. For remote Podman, docker is the only allowed transport.

dir:path

An existing local directory path storing the manifest, layer tarballs and signatures as individual files. This is a non-standardized format, primarily useful for debugging or noninvasive container inspection.

```
$ podman save --format docker-dir fedora -o /tmp/fedora
$ podman run dir:/tmp/fedora echo hello
```

docker://docker-reference (Default)

An image reference stored in a remote container image registry. Example: "quay.io/pod?man/stable:latest". The reference can include a path to a specific registry; if it does not, the registries listed in registries.conf will be queried to find a matching image.

By default, credentials from podman login (stored at $XDG_RUNTIME_DIR/containers/auth.json by default) will be used to authenticate; otherwise it falls back to using credentials in $HOME/.docker/config.json.

```
$ podman run registry.fedoraproject.org/fedora:latest echo hello
```

docker-archive:path[:docker-reference] An image stored in the docker save formatted file.

docker-reference is only used when creating such a file, and it must not contain a digest.

```
$ podman save --format docker-archive fedora -o /tmp/fedora
$ podman run docker-archive:/tmp/fedora echo hello
```

docker-daemon:docker-reference

An image in docker-reference format stored in the docker daemon internal storage. The docker-reference can also be an image ID (docker-daemon:algo:digest).

```
$ sudo docker pull fedora
$ sudo podman run docker-daemon:docker.io/library/fedora echo hello
```

oci-archive:path:tag

An image in a directory compliant with the "Open Container Image Layout Specification" at the specified path and specified with a tag.

```
$ podman save --format oci-archive fedora -o /tmp/fedora

$ podman run oci-archive:/tmp/fedora echo hello
```

OPTIONS

--add-host=host:ip

Add a line to container's /etc/hosts for custom host-to-IP mapping. This option can be set multiple times.

--annotation=key=value

Add an annotation to the container. This option can be set multiple times.

--arch=ARCH

Override the architecture, defaults to hosts, of the image to be pulled. For example, arm.

--attach, -a=stdin|stdout|stderr

Attach to STDIN, STDOUT or STDERR.

In foreground mode (the default when -d is not specified), podman run can start the process in the container and attach the console to the process's standard input, output, and error. It can even pretend to be a TTY (this is what most commandline executables ex? pect) and pass along signals. The -a option can be set for each of stdin, stdout, and stderr.

--authfile[=path]

Path to the authentication file. Default is ${XDG_RUNTIME_DIR}/containers/auth.json.

Note: You can also override the default path of the authentication file by setting the REGISTRY_AUTH_FILE environment variable.

--blkio-weight=weight

Block IO relative weight. The weight is a value between 10 and 1000.

--blkio-weight-device=device:weight

Block IO relative device weight.

--cap-add=capability

Add Linux capabilities.

--cap-drop=capability

Drop Linux capabilities.

--cgroupns=mode

Set the cgroup namespace mode for the container.

? host: use the host's cgroup namespace inside the container.

? container:id: join the namespace of the specified container.

? private: create a new cgroup namespace.

? ns:path: join the namespace at the specified path.

If the host uses cgroups v1, the default is set to host. On cgroups v2, the default is private.

--cgroups=enabled|disabled|no-conmon|split

Determines whether the container will create CGroups.

Default is enabled.

The enabled option will create a new cgroup under the cgroup-parent. The disabled option will force the container to not create CGroups, and thus conflicts with CGroup options (--cgroupns and --cgroup-parent). The no-conmon option disables a new CGroup only for the conmon process. The split option splits the current CGroup in two sub-cgroups: one for conmon and one for the container payload. It is not possible to set --cgroup-parent with split.

--cgroup-parent=path

Path to cgroups under which the cgroup for the container will be created. If the path is not absolute, the path is considered to be relative to the cgroups path of the init process. Cgroups will be created if they do not already exist.

--cgroup-conf=KEY=VALUE

When running on cgroup v2, specify the cgroup file to write to and its value. For example --cgroup-conf=memory.high=1073741824 sets the memory.high limit to 1GB.

--cidfile=file

Write the container ID to file.

--conmon-pidfile=file

Write the pid of the conmon process to a file. As conmon runs in a separate process than Podman, this is necessary when using systemd to restart Podman containers. (This option is not available with the remote Podman client)

--cpu-period=limit

Set the CPU period for the Completely Fair Scheduler (CFS), which is a duration in microseconds. Once the container's CPU quota is used up, it will not be scheduled to run until the current period ends. Defaults to 100000 microseconds.

On some systems, changing the CPU limits may not be allowed for non-root users. For more details, see https://github.com/containers/podman/blob/master/troubleshooting.md#26-run?ning-containers-with-cpu-limits-fails-with-a-permissions-error

--cpu-quota=limit

Limit the CPU Completely Fair Scheduler (CFS) quota.

Limit the container's CPU usage. By default, containers run with the full CPU resource. The limit is a number in microseconds. If you provide a number, the container will be al?lowed to use that much CPU time until the CPU period ends (controllable via --cpu-period).

On some systems, changing the CPU limits may not be allowed for non-root users. For more details, see https://github.com/containers/podman/blob/master/troubleshooting.md#26-run?ning-containers-with-cpu-limits-fails-with-a-permissions-error

--cpu-rt-period=microseconds

Limit the CPU real-time period in microseconds.

Limit the container's Real Time CPU usage. This flag tell the kernel to restrict the con?tainer's Real Time CPU usage to the period you specify.

This flag is not supported on cgroups V2 systems.

--cpu-rt-runtime=microseconds

Limit the CPU real-time runtime in microseconds.

Limit the containers Real Time CPU usage. This flag tells the kernel to limit the amount of time in a given CPU period Real Time tasks may consume. Ex: Period of 1,000,000us and Runtime of 950,000us means that this container could consume 95% of available CPU and leave the remaining 5% to normal priority tasks.

The sum of all runtimes across containers cannot exceed the amount allotted to the parent cgroup.

This flag is not supported on cgroups V2 systems.

--cpu-shares=shares

CPU shares (relative weight).

By default, all containers get the same proportion of CPU cycles. This proportion can be modified by changing the container's CPU share weighting relative to the combined weight of all the running containers. Default weight is 1024.

The proportion will only apply when CPU-intensive processes are running. When tasks in one container are idle, other containers can use the left-over CPU time. The actual amount of CPU time will vary depending on the number of containers running on the system.

For example, consider three containers, one has a cpu-share of 1024 and two others have a cpu-share setting of 512. When processes in all three containers attempt to use 100% of CPU, the first container would receive 50% of the total CPU time. If you add a fourth container with a cpu-share of 1024, the first container only gets 33% of the CPU. The remaining containers receive 16.5%, 16.5% and 33% of the CPU.

On a multi-core system, the shares of CPU time are distributed over all CPU cores. Even if a container is limited to less than 100% of CPU time, it can use 100% of each individual CPU core.

For example, consider a system with more than three cores. If you start one container {C0} with --cpu-shares=512 running one process, and another container {C1} with --cpu-shares=1024 running two processes, this can result in the following division of CPU shares:

```
??????????????????????????????????????????
?PID ? container ? CPU ? CPU share    ?
??????????????????????????????????????????
?100 ? {C0}      ? 0   ? 100% of CPU0 ?
??????????????????????????????????????????
?101 ? {C1}      ? 1   ? 100% of CPU1 ?
??????????????????????????????????????????
?102 ? {C1}      ? 2   ? 100% of CPU2 ?
??????????????????????????????????????????
```

--cpus=number

Number of CPUs. The default is 0.0 which means no limit. This is shorthand for --cpu-period and --cpu-quota, so you may only set either

--cpus or --cpu-period and --cpu-quota.

On some systems, changing the CPU limits may not be allowed for non-root users. For more details, see https://github.com/containers/podman/blob/master/troubleshooting.md#26-running-containers-with-cpu-limits-fails-with-a-permissions-error

--cpuset-cpus=number

CPUs in which to allow execution. Can be specified as a comma-separated list (e.g. 0,1), as a range (e.g. 0-3), or any combination thereof (e.g. 0-3,7,11-15).

--cpuset-mems=nodes

Memory nodes (MEMs) in which to allow execution. Only effective on NUMA systems.

For example, if you have four memory nodes (0-3) on your system, use --cpuset-mems=0,1 to only use memory from the first two memory nodes.

--detach, -d=true|false

Detached mode: run the container in the background and print the new container ID. The de? fault is false.

At any time you can run podman ps in the other shell to view a list of the running con? tainers. You can reattach to a detached container with podman attach.

When attached in the tty mode, you can detach from the container (and leave it running) using a configurable key sequence. The default sequence is ctrl-p,ctrl-q. Configure the keys sequence using the --detach-keys option, or specifying it in the containers.conf file: see containers.conf(5) for more information.

--detach-keys=sequence

Specify the key sequence for detaching a container. Format is a single character [a-Z] or one or more ctrl-<value> characters where <value> is one of: a-z, @, ^, [, , or _. Speci? fying "" will set the sequence to the default value of ctrl-p,ctrl-q.

This option can also be set in containers.conf(5) file.

--device=host-device[:container-device][:permissions]

Add a host device to the container. Optional permissions parameter can be used to specify device permissions, it is combination of r for read, w for write, and m for mknod(2).

Example: --device=/dev/sdc:/dev/xvdc:rwm.

Note: if _hostdevice is a symbolic link then it will be resolved first. The container will only store the major and minor numbers of the host device.

Note: if the user only has access rights via a group, accessing the device from inside a rootless container will fail. Use the --group-add keep-groups flag to pass the user's sup? plementary group access into the container.

Podman may load kernel modules required for using the specified device. The devices that Podman will load modules when necessary are: /dev/fuse.

--device-cgroup-rule=rule

Add a rule to the cgroup allowed devices list

--device-read-bps=path:rate

Limit read rate (in bytes per second) from a device (e.g. --device-read-bps=/dev/sda:1mb).

--device-read-iops=path:rate

Limit read rate (in IO operations per second) from a device (e.g. --device-read-iops=/dev/sda:1000).

--device-write-bps=path:rate

Limit write rate (in bytes per second) to a device (e.g. --device-write-bps=/dev/sda:1mb).

--device-write-iops=path:rate

Limit write rate (in IO operations per second) to a device (e.g. --device-write-iops=/dev/sda:1000).

--disable-content-trust

This is a Docker specific option to disable image verification to a Docker registry and is not supported by Podman. This flag is a NOOP and provided solely for scripting compatibil?ity.

--dns=ipaddr

Set custom DNS servers. Invalid if using --dns with --network that is set to none or con?tainer:id.

This option can be used to override the DNS configuration passed to the container. Typi?cally this is necessary when the host DNS configuration is invalid for the container (e.g., 127.0.0.1). When this is the case the --dns flags is necessary for every run.

The special value none can be specified to disable creation of /etc/resolv.conf in the container by Podman. The /etc/resolv.conf file in the image will be used without changes.

--dns-opt=option

Set custom DNS options. Invalid if using --dns-opt with --network that is set to none or container:id.

--dns-search=domain

Set custom DNS search domains. Invalid if using --dns-search and --network that is set to none or container:id. Use --dns-search=. if you don't wish to set the search domain.

--entrypoint="command" | '["command", arg1 , ...]'

Overwrite the default ENTRYPOINT of the image.

This option allows you to overwrite the default entrypoint of the image.

The ENTRYPOINT of an image is similar to a COMMAND because it specifies what executable to run when the container starts, but it is (purposely) more difficult to override. The EN?TRYPOINT gives a container its default nature or behavior, so that when you set an ENTRY?POINT you can run the container as if it were that binary, complete with default options, and you can pass in more options via the COMMAND. But, sometimes an operator may want to run something else inside the container, so you can override the default ENTRYPOINT at runtime by using a --entrypoint and a string to specify the new ENTRYPOINT.

You need to specify multi option commands in the form of a json string.

--env, -e=env

Set environment variables.

This option allows arbitrary environment variables that are available for the process to be launched inside of the container. If an environment variable is specified without a value, Podman will check the host environment for a value and set the variable only if it is set on the host. If an environment variable ending in * is specified, Podman will search the host environment for variables starting with the prefix and will add those variables to the container. If an environment variable with a trailing ***** is specified, then a value must be supplied.

See Environment ?#environment? note below for precedence and examples.

--env-host=true|false

Use host environment inside of the container. See Environment note below for precedence. (This option is not available with the remote Podman client)

--env-file=file

Read in a line delimited file of environment variables. See Environment note below for precedence.

--expose=port

Expose a port, or a range of ports (e.g. --expose=3300-3310) to set up port redirection on the host system.

--gidmap=container_gid:host_gid:amount

Run the container in a new user namespace using the supplied mapping. This option con?flicts with the --userns and --subgidname flags. This option can be passed several times to map different ranges. If calling podman run as an unprivileged user, the user needs to have the right to use the mapping. See subuid(5). The example maps gids 0-1999 in the container to the gids 30000-31999 on the host: --gidmap=0:30000:2000.

Important note: The new user namespace mapping based on --gidmap is based on the initial mapping made in the /etc/subgid file. Assuming there is a /etc/subgid mapping group?name:100000:65536, then groupname is initially mapped to a namespace starting with gid 100000 for 65536 ids. From here the --gidmap mapping to the new namespace starts from 0 again, but is based on the initial mapping. Meaning groupname is initially mapped to gid 100000 which is referenced as 0 in the following --gidmap mapping. In terms of the example above: The group groupname is mapped to group 100000 of the initial namespace then the 30000st id of this namespace (which is gid 130000 in this namespace) is mapped to con?tainer namespace group id 0. (groupname -> 100000 / 30000 -> 0)

--group-add=group|keep-groups

Add additional groups to assign to primary user running within the container process.

? keep-groups is a special flag that tells Podman to keep the supplementary group access.

Allows container to use the user's supplementary group access. If file systems or devices are only accessible by the rootless user's group, this flag tells the OCI runtime to pass the group access into the container. Currently only available with the crun OCI runtime. Note: keep-groups is exclusive, you cannot add any other groups with this flag. (Not available for remote commands)

--health-cmd="command" | '["command", arg1 , ...]'

Set or alter a healthcheck command for a container. The command is a command to be exe?cuted inside your container that determines your container health. The command is required for other healthcheck options to be applied. A value of none disables existing healthchecks.

Multiple options can be passed in the form of a JSON array; otherwise, the command will be interpreted as an argument to /bin/sh -c.

--health-interval=interval

Set an interval for the healthchecks. An interval of disable results in no automatic timer setup. The default is 30s.

--health-retries=retries

The number of retries allowed before a healthcheck is considered to be unhealthy. The de?fault value is 3.

--health-start-period=period

The initialization time needed for a container to bootstrap. The value can be expressed in time format like 2m3s. The default value is 0s.

--health-timeout=timeout

The maximum time allowed to complete the healthcheck before an interval is considered failed. Like start-period, the value can be expressed in a time format such as 1m22s. The default value is 30s.

--help

Print usage statement

--hostname=name, -h

Container host name

Sets the container host name that is available inside the container. Can only be used with a private UTS namespace --uts=private (default). If --pod is specified and the pod shares the UTS namespace (default) the pod's hostname will be used.

--http-proxy=true|false

By default proxy environment variables are passed into the container if set for the Podman process. This can be disabled by setting the value to false. The environment variables passed in include http_proxy, https_proxy, ftp_proxy, no_proxy, and also the upper case versions of those. This option is only needed when the host system must use a proxy but the container should not use any proxy. Proxy environment variables specified for the con? tainer in any other way will override the values that would have been passed through from the host. (Other ways to specify the proxy for the container include passing the values with the --env flag, or hard coding the proxy environment at container build time.) (This option is not available with the remote Podman client)

Defaults to true.

--image-volume, builtin-volume=bind|tmpfs|ignore

Tells Podman how to handle the builtin image volumes. Default is bind.

? bind: An anonymous named volume will be created and mounted into the container.

? tmpfs: The volume is mounted onto the container as a tmpfs, which allows the users to create content that disappears when the container is stopped.

? ignore: All volumes are just ignored and no action is taken.

--init

Run an init inside the container that forwards signals and reaps processes.

--init-path=path

Path to the container-init binary.

--interactive, -i=true|false

When set to true, keep stdin open even if not attached. The default is false.

--ip6=ip

Not implemented.

--ip=ip

Specify a static IP address for the container, for example 10.88.64.128. This option can

only be used if the container is joined to only a single network - i.e., --network=_net?
work-name_ is used at most once and if the container is not joining another container's
network namespace via --network=container:_id_. The address must be within the CNI net?
work's IP address pool (default 10.88.0.0/16).

--ipc=mode

Set the IPC namespace mode for a container. The default is to create a private IPC name?
space.

? container:id: reuses another container shared memory, semaphores and message
queues

? host: use the host shared memory,semaphores and message queues inside the con?
tainer. Note: the host mode gives the container full access to local shared mem?
ory and is therefore considered insecure.

? ns:path: path to an IPC namespace to join.

--kernel-memory=number[unit]

Kernel memory limit. A unit can be b (bytes), k (kilobytes), m (megabytes), or g (giga?
bytes).

Constrains the kernel memory available to a container. If a limit of 0 is specified (not
using --kernel-memory), the container's kernel memory is not limited. If you specify a
limit, it may be rounded up to a multiple of the operating system's page size and the
value can be very large, millions of trillions.

This flag is not supported on cgroups V2 systems.

--label, -l=key=value

Add metadata to a container.

--label-file=file

Read in a line-delimited file of labels.

--link-local-ip=ip

Not implemented.

--log-driver="driver"

Logging driver for the container. Currently available options are k8s-file, journald, and
none, with json-file aliased to k8s-file for scripting compatibility.

--log-opt=name=value

Logging driver specific options.

Set custom logging configuration. The following *name*s are supported:

path: specify a path to the log file

(e.g. --log-opt path=/var/log/container/mycontainer.json);

max-size: specify a max size of the log file

(e.g. --log-opt max-size=10mb);

tag: specify a custom log tag for the container

(e.g. --log-opt tag="{{.ImageName}}".

This option is currently supported only by the journald log driver.

--mac-address=address

Container MAC address (e.g. 92:d0:c6:0a:29:33).

Remember that the MAC address in an Ethernet network must be unique. The IPv6 link-local address will be based on the device's MAC address according to RFC4862.

--memory, -m=number[unit]

Memory limit. A unit can be b (bytes), k (kilobytes), m (megabytes), or g (gigabytes).

Allows you to constrain the memory available to a container. If the host supports swap memory, then the -m memory setting can be larger than physical RAM. If a limit of 0 is specified (not using -m), the container's memory is not limited. The actual limit may be rounded up to a multiple of the operating system's page size (the value would be very large, that's millions of trillions).

--memory-reservation=number[unit]

Memory soft limit. A unit can be b (bytes), k (kilobytes), m (megabytes), or g (giga? bytes).

After setting memory reservation, when the system detects memory contention or low memory, containers are forced to restrict their consumption to their reservation. So you should always set the value below --memory, otherwise the hard limit will take precedence. By de? fault, memory reservation will be the same as memory limit.

--memory-swap=number[unit]

A limit value equal to memory plus swap. A unit can be b (bytes), k (kilobytes), m (megabytes), or g (gigabytes).

Must be used with the -m (--memory) flag. The argument value should always be larger than that of

-m (--memory) By default, it is set to double the value of --memory.

Set number to -1 to enable unlimited swap.

--memory-swappiness=number

Tune a container's memory swappiness behavior. Accepts an integer between 0 and 100.

This flag is not supported on cgroups V2 systems.

--mount=type=TYPE,TYPE-SPECIFIC-OPTION[,...]

Attach a filesystem mount to the container

Current supported mount TYPEs are bind, volume, image, tmpfs and devpts. [1] ?#Footnote1?

e.g.

type=bind,source=/path/on/host,destination=/path/in/container

type=bind,src=/path/on/host,dst=/path/in/container,relabel=shared

type=volume,source=vol1,destination=/path/in/container,ro=true

type=tmpfs,tmpfs-size=512M,destination=/path/in/container

type=image,source=fedora,destination=/fedora-image,rw=true

type=devpts,destination=/dev/pts

Common Options:

? src, source: mount source spec for bind and volume. Mandatory for bind.

? dst, destination, target: mount destination spec.

Options specific to volume:

? ro, readonly: true or false (default).

Options specific to image:

? rw, readwrite: true or false (default).

Options specific to bind:

? ro, readonly: true or false (default).

? bind-propagation: shared, slave, private, unbindable, rshared, rslave, runbindable, or rprivate(default). See also mount(2).

. bind-nonrecursive: do not setup a recursive bind mount. By default it is recursive.

. relabel: shared, private.

Options specific to tmpfs:

? ro, readonly: true or false (default).

? tmpfs-size: Size of the tmpfs mount in bytes. Unlimited by default in Linux.

? tmpfs-mode: File mode of the tmpfs in octal. (e.g. 700 or 0700.) Defaults to 1777 in Linux.

? tmpcopyup: Enable copyup from the image directory at the same location to the tmpfs. Used by default.

? notmpcopyup: Disable copying files from the image to the tmpfs.

--name=name

Assign a name to the container.

The operator can identify a container in three ways:

? UUID long identifier (?f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778?);

? UUID short identifier (?f78375b1c487?);

? Name (?jonah?).

Podman generates a UUID for each container, and if a name is not assigned to the container with --name then it will generate a random string name. The name is useful any place you need to identify a container. This works for both background and foreground containers.

--network=mode, --net

Set the network mode for the container. Invalid if using --dns, --dns-opt, or --dns-search with --network that is set to none or container:id. If used together with --pod, the container will not join the pods network namespace.

Valid mode values are:

? bridge: Create a network stack on the default bridge. This is the default for rootfull containers.

? none: Create a network namespace for the container but do not configure network interfaces for it, thus the container has no network connectivity.

? container:id: Reuse another container's network stack.

? host: Do not create a network namespace, the container will use the host's network. Note: The host mode gives the container full access to local system services such as D-bus and is therefore considered insecure.

? network: Connect to a user-defined network, multiple networks should be comma-separated.

? ns:path: Path to a network namespace to join.

? private: Create a new namespace for the container. This will use the bridge mode for rootfull containers and slirp4netns for rootless ones.

? slirp4netns[:OPTIONS,...]: use slirp4netns(1) to create a user network stack. This is the default for rootless containers. It is possible to specify these additional options:

? allow_host_loopback=true|false: Allow the slirp4netns to reach the host loopback IP (10.0.2.2, which is added to /etc/hosts as host.containers.internal for your convenience). Default is false.

? mtu=MTU: Specify the MTU to use for this network. (Default is 65520).

? cidr=CIDR: Specify ip range to use for this network. (Default is 10.0.2.0/24).

? enable_ipv6=true|false: Enable IPv6. Default is false. (Required for out?
bound_addr6).

? outbound_addr=INTERFACE: Specify the outbound interface slirp should bind to
(ipv4 traffic only).

? outbound_addr=IPv4: Specify the outbound ipv4 address slirp should bind to.

? outbound_addr6=INTERFACE: Specify the outbound interface slirp should bind to
(ipv6 traffic only).

? outbound_addr6=IPv6: Specify the outbound ipv6 address slirp should bind to.

? port_handler=rootlesskit: Use rootlesskit for port forwarding. Default. Note:
Rootlesskit changes the source IP address of incoming packets to a IP address
in the container network namespace, usually 10.0.2.100. If your application re?
quires the real source IP address, e.g. web server logs, use the slirp4netns
port handler. The rootlesskit port handler is also used for rootless containers
when connected to user-defined networks.

? port_handler=slirp4netns: Use the slirp4netns port forwarding, it is slower
than rootlesskit but preserves the correct source IP address. This port handler
cannot be used for user-defined networks.

--network-alias=alias

Add network-scoped alias for the container. NOTE: A container will only have access to aliases on the first network that it joins. This is a limitation that will be removed in a later release.

--no-healthcheck=true|false

Disable any defined healthchecks for container.

--no-hosts=true|false

Do not create /etc/hosts for the container.

By default, Podman will manage /etc/hosts, adding the container's own IP address and any hosts from --add-host.

--no-hosts disables this, and the image's /etc/hosts will be preserved unmodified.

This option conflicts with --add-host.

--oom-kill-disable=true|false

Whether to disable OOM Killer for the container or not.

--oom-score-adj=num

Tune the host's OOM preferences for containers (accepts values from -1000 to 1000).

--os=OS

Override the OS, defaults to hosts, of the image to be pulled. For example, windows.

--personality=persona

Personality sets the execution domain via Linux personality(2).

--pid=mode

Set the PID namespace mode for the container. The default is to create a private PID namespace for the container.

? container:id: join another container's PID namespace;

? host: use the host's PID namespace for the container. Note the host mode gives the container full access to local PID and is therefore considered insecure;

? private: create a new namespace for the container (default)

? ns:path: join the specified PID namespace.

--pids-limit=limit

Tune the container's pids limit. Set to -1 to have unlimited pids for the container. The default is 4096 on systems that support "pids" cgroup controller.

--platform=OS/ARCH

Specify the platform for selecting the image. (Conflicts with --arch and --os) The --platform option can be used to override the current architecture and operating system.

--pod=name

Run container in an existing pod. If you want Podman to make the pod for you, prefix the pod name with new:. To make a pod with more granular options, use the podman pod create command before creating a container. If a container is run with a pod, and the pod has an infra-container, the infra-container will be started before the container is.

--pod-id-file=path

Run container in an existing pod and read the pod's ID from the specified file. If a con? tainer is run within a pod, and the pod has an infra-container, the infra-container will be started before the container is.

--preserve-fds=N

Pass down to the process N additional file descriptors (in addition to 0, 1, 2). The to? tal FDs will be 3+N. (This option is not available with the remote Podman client)

--privileged=true|false

Give extended privileges to this container. The default is false.

By default, Podman containers are unprivileged (=false) and cannot, for example, modify parts of the operating system. This is because by default a container is only allowed lim?ited access to devices. A "privileged" container is given the same access to devices as the user launching the container.

A privileged container turns off the security features that isolate the container from the host. Dropped Capabilities, limited devices, read-only mount points, Apparmor/SELinux sep?aration, and Seccomp filters are all disabled.

Rootless containers cannot have more privileges than the account that launched them.

--publish, -p=ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | con?tainerPort

Publish a container's port, or range of ports, to the host.

Both hostPort and containerPort can be specified as a range of ports.

When specifying ranges for both, the number of container ports in the range must match the number of host ports in the range.

If host IP is set to 0.0.0.0 or not set at all, the port will be bound on all IPs on the host.

Host port does not have to be specified (e.g. podman run -p 127.0.0.1::80). If it is not, the container port will be randomly assigned a port on the host.

Use podman port to see the actual mapping: podman port $CONTAINER $CONTAINERPORT.

Note: if a container will be run within a pod, it is not necessary to publish the port for the containers in the pod. The port must only be published by the pod itself. Pod network stacks act like the network stack on the host - you have a variety of containers in the pod, and programs in the container, all sharing a single interface and IP address, and as?sociated ports. If one container binds to a port, no other container can use that port within the pod while it is in use. Containers in the pod can also communicate over local?host by having one container bind to localhost in the pod, and another connect to that port.

--publish-all, -P=true|false

Publish all exposed ports to random ports on the host interfaces. The default is false.

When set to true, publish all exposed ports to the host interfaces. The default is false.

If the operator uses -P (or -p) then Podman will make the exposed port accessible on the host and the ports will be available to any client that can reach the host.

When using this option, Podman will bind any exposed port to a random port on the host within an ephemeral port range defined by /proc/sys/net/ipv4/ip_local_port_range. To find the mapping between the host ports and the exposed ports, use podman port.

--pull=always|missing|never

Pull image before running. The default is missing.

? missing: attempt to pull the latest image from the registries listed in reg?istries.conf if a local image does not exist.Raise an error if the image is not in any listed registry and is not present locally.

? always: Pull the image from the first registry it is found in as listed in reg?istries.conf. Raise an error if not found in the registries, even if the image is present locally.

? never: do not pull the image from the registry, use only the local version. Raise an error if the image is not present locally.

--quiet, -q

Suppress output information when pulling images

--read-only=true|false

Mount the container's root filesystem as read only.

By default a container will have its root filesystem writable allowing processes to write files anywhere. By specifying the --read-only flag, the container will have its root filesystem mounted as read only prohibiting any writes.

--read-only-tmpfs=true|false

If container is running in --read-only mode, then mount a read-write tmpfs on /run, /tmp, and /var/tmp. The default is true.

--replace=true|false

If another container with the same name already exists, replace and remove it. The default is false.

--requires=container

Specify one or more requirements. A requirement is a dependency container that will be started before this container. Containers can be specified by name or ID, with multiple containers being separated by commas.

--restart=policy

Restart policy to follow when containers exit. Restart policy will not take effect if a container is stopped via the podman kill or podman stop commands.

Valid policy values are:

? no   : Do not restart containers on exit

? on-failure[:max_retries] : Restart containers when they exit with a non-zero exit code, retrying indefinitely or until the optional max_retries count is hit

? always   : Restart containers when they exit, regardless of sta? tus, retrying indefinitely

? unless-stopped   : Identical to always

Please note that restart will not restart containers after a system reboot. If this func? tionality is required in your environment, you can invoke Podman from a systemd.unit(5) file, or create an init script for whichever init system is in use. To generate systemd unit files, please see podman generate systemd.

--rm=true|false

Automatically remove the container when it exits. The default is false.

--rmi=true|false

After exit of the container, remove the image unless another container is using it. The default is false.

--rootfs

If specified, the first argument refers to an exploded container on the file system.

This is useful to run a container without requiring any image management, the rootfs of the container is assumed to be managed externally.

Note: On SELinux systems, the rootfs needs the correct label, which is by default uncon? fined_u:object_r:container_file_t.

--sdnotify=container|conmon|ignore

Determines how to use the NOTIFY_SOCKET, as passed with systemd and Type=notify.

Default is container, which means allow the OCI runtime to proxy the socket into the con? tainer to receive ready notification. Podman will set the MAINPID to conmon's pid. The conmon option sets MAINPID to conmon's pid, and sends READY when the container has started. The socket is never passed to the runtime or the container. The ignore option removes NOTIFY_SOCKET from the environment for itself and child processes, for the case where some other process above Podman uses NOTIFY_SOCKET and Podman should not use it.

--seccomp-policy=policy

Specify the policy to select the seccomp profile. If set to image, Podman will look for a "io.containers.seccomp.profile" label in the container-image config and use its value as a seccomp profile. Otherwise, Podman will follow the default policy by applying the default profile unless specified otherwise via --security-opt seccomp as described below.

Note that this feature is experimental and may change in the future.

--secret=secret[,opt=opt ...]

Give the container access to a secret. Can be specified multiple times.

A secret is a blob of sensitive data which a container needs at runtime but should not be stored in the image or in source control, such as usernames and passwords, TLS certificates and keys, SSH keys or other important generic strings or binary content (up to 500 kb in size).

When secrets are specified as type mount, the secrets are copied and mounted into the container when a container is created. When secrets are specified as type env, the secret will be set as an environment variable within the container. Secrets are written in the container at the time of container creation, and modifying the secret using podman secret commands after the container is created will not affect the secret inside the container.

Secrets and its storage are managed using the podman secret command.

Secret Options

? type=mount|env : How the secret will be exposed to the container. Default mount.

? target=target : Target of secret. Defaults to secret name.

? uid=0 : UID of secret. Defaults to 0. Mount secret type only.

? gid=0 : GID of secret. Defaults to 0. Mount secret type only.

? mode=0 : Mode of secret. Defaults to 0444. Mount secret type only.

--security-opt=option

Security Options

? apparmor=unconfined : Turn off apparmor confinement for the container

? apparmor=your-profile : Set the apparmor confinement profile for the container

? label=user:USER: Set the label user for the container processes

? label=role:ROLE: Set the label role for the container processes

? label=type:TYPE: Set the label process type for the container processes

? label=level:LEVEL: Set the label level for the container processes

? label=filetype:TYPE_: Set the label file type for the container files

? label=disable: Turn off label separation for the container

Note: Labeling can be disabled for all containers by setting label=false in the containers.conf (/etc/containers/containers.conf or $HOME/.config/containers/containers.conf) file.

? mask=/path/1:/path/2: The paths to mask separated by a colon. A masked path can?
not be accessed inside the container.

? no-new-privileges: Disable container processes from gaining additional privileges

? seccomp=unconfined: Turn off seccomp confinement for the container

? seccomp=profile.json: Allowed syscall list seccomp JSON file to be used as a sec?
comp filter

? proc-opts=OPTIONS : Comma-separated list of options to use for the /proc mount.
More details for the possible mount options are specified in the proc(5) man
page.

? unmask=ALL or /path/1:/path/2, or shell expanded paths (/proc/*): Paths to unmask
separated by a colon. If set to ALL, it will unmask all the paths that are masked
or made read only by default. The default masked paths are /proc/acpi,
/proc/kcore, /proc/keys, /proc/latency_stats, /proc/sched_debug, /proc/scsi,
/proc/timer_list, /proc/timer_stats, /sys/firmware, and /sys/fs/selinux.. The
default paths that are read only are /proc/asound, /proc/bus, /proc/fs,
/proc/irq, /proc/sys, /proc/sysrq-trigger, /sys/fs/cgroup.

Note: Labeling can be disabled for all containers by setting label=false in the contain?
ers.conf(5) file.

--shm-size=number[unit]

Size of /dev/shm. A unit can be b (bytes), k (kilobytes), m (megabytes), or g (gigabytes).
If you omit the unit, the system uses bytes. If you omit the size entirely, the default is
64m. When size is 0, there is no limit on the amount of memory used for IPC by the con?
tainer.

--sig-proxy=true|false

Sets whether the signals sent to the podman run command are proxied to the container
process. SIGCHLD, SIGSTOP, and SIGKILL are not proxied. The default is true.

--stop-signal=signal

Signal to stop a container. Default is SIGTERM.

--stop-timeout=seconds

Timeout to stop a container. Default is 10. Remote connections use local containers.conf
for defaults

--subgidname=name

Run the container in a new user namespace using the map with name in the /etc/subgid file. If calling podman run as an unprivileged user, the user needs to have the right to use the mapping. See subgid(5). This flag conflicts with --userns and --gidmap.

--subuidname=name

Run the container in a new user namespace using the map with name in the /etc/subuid file. If calling podman run as an unprivileged user, the user needs to have the right to use the mapping. See subuid(5). This flag conflicts with --userns and --uidmap.

--sysctl=name=value

Configure namespaced kernel parameters at runtime.

For the IPC namespace, the following sysctls are allowed:

? kernel.msgmax

? kernel.msgmnb

? kernel.msgmni

? kernel.sem

? kernel.shmall

? kernel.shmmax

? kernel.shmmni

? kernel.shm_rmid_forced

? Sysctls beginning with fs.mqueue.*

Note: if you use the --ipc=host option, the above sysctls will not be allowed.

For the network namespace, the following sysctls are allowed:

? Sysctls beginning with net.*

Note: if you use the --network=host option, these sysctls will not be allowed.

--systemd=true|false|always

Run container in systemd mode. The default is true.

The value always enforces the systemd mode is enforced without looking at the executable name. Otherwise, if set to true and the command you are running inside the container is systemd, /usr/sbin/init, /sbin/init or /usr/local/sbin/init.

If the command you are running inside of the container is systemd Podman will setup tmpfs mount points in the following directories:

? /run

? /run/lock

? /tmp

? /sys/fs/cgroup/systemd

? /var/lib/journal

It will also set the default stop signal to SIGRTMIN+3.

This allows systemd to run in a confined container without any modifications.

Note that on SELinux systems, systemd attempts to write to the cgroup file system. Con?tainers writing to the cgroup file system are denied by default. The container_man?age_cgroup boolean must be enabled for this to be allowed on an SELinux separated system.

```
setsebool -P container_manage_cgroup true
```

--timeout=seconds

Maximum time a container is allowed to run before conmon sends it the kill signal. By de?fault containers will run until they exit or are stopped by podman stop.

--tls-verify=true|false

Require HTTPS and verify certificates when contacting registries (default: true). If ex?plicitly set to true, then TLS verification will be used. If set to false, then TLS veri?fication will not be used. If not specified, TLS verification will be used unless the tar?get registry is listed as an insecure registry in registries.conf.

--tmpfs=fs

Create a tmpfs mount.

Mount a temporary filesystem (tmpfs) mount into a container, for example:

```
$ podman run -d --tmpfs /tmp:rw,size=787448k,mode=1777 my_image
```

This command mounts a tmpfs at /tmp within the container. The supported mount options are the same as the Linux default mount flags. If you do not specify any options, the systems uses the following options: rw,noexec,nosuid,nodev.

--tty, -t=true|false

Allocate a pseudo-TTY. The default is false.

When set to true, Podman will allocate a pseudo-tty and attach to the standard input of the container. This can be used, for example, to run a throwaway interactive shell. The default is false.

NOTE: The --tty flag prevents redirection of standard output. It combines STDOUT and STDERR, it can insert control characters, and it can hang pipes. This option should only be used when run interactively in a terminal. When feeding input to Podman, use -i only, not -it.

```
echo "asdf" | podman run --rm -i someimage /bin/cat
```

--tz=timezone

Set timezone in container. This flag takes area-based timezones, GMT time, as well as local, which sets the timezone in the container to match the host machine. See /usr/share/zoneinfo/ for valid timezones. Remote connections use local containers.conf for defaults

--umask=umask

Set the umask inside the container. Defaults to 0022. Remote connections use local containers.conf for defaults

--uidmap=container_uid:from_uid:amount

Run the container in a new user namespace using the supplied mapping. This option conflicts with the --userns and --subuidname options. This option provides a way to map host UIDs to container UIDs. It can be passed several times to map different ranges.

The _fromuid value is based upon the user running the command, either rootfull or rootless users. * rootfull user: container_uid:host_uid:amount * rootless user: container_uid:intermediate_uid:amount

When podman run is called by a privileged user, the option --uidmap works as a direct mapping between host UIDs and container UIDs.

host UID -> container UID

The amount specifies the number of consecutive UIDs that will be mapped. If for example amount is 4 the mapping would look like:

| host UID | container UID | | - | - | | _fromuid | _containeruid | | _fromuid + 1 | _containeruid + 1 | | _fromuid + 2 | _containeruid + 2 | | _fromuid + 3 | _containeruid + 3 |

When podman run is called by an unprivileged user (i.e. running rootless), the value _fromuid is interpreted as an "intermediate UID". In the rootless case, host UIDs are not mapped directly to container UIDs. Instead the mapping happens over two mapping steps:

host UID -> intermediate UID -> container UID

The --uidmap option only influences the second mapping step.

The first mapping step is derived by Podman from the contents of the file /etc/subuid and the UID of the user calling Podman.

First mapping step:

| host UID | intermediate UID | | - | - | | UID for the user starting Podman | 0

| | 1st subordinate UID for the user starting Podman | 1 | | 2nd subordi?nate UID for the user starting Podman | 2 | | 3rd subordinate UID for the user starting Podman | 3 | | nth subordinate UID for the user starting Pod?man | n |

To be able to use intermediate UIDs greater than zero, the user needs to have subordinate UIDs configured in /etc/subuid. See subuid(5).

The second mapping step is configured with --uidmap.

If for example amount is 5 the second mapping step would look like:

| intermediate UID | container UID | | - | - | | _fromuid | _containeruid | | _fromuid + 1 | _containeruid + 1 | | _fromuid + 2 | _containeruid + 2 | | _fromuid + 3 | _containeruid + 3 | | _fromuid + 4 | _containeruid + 4 |

Even if a user does not have any subordinate UIDs in /etc/subuid, --uidmap could still be used to map the normal UID of the user to a container UID by running podman run --uidmap $container_uid:0:1 --user $container_uid ....

--ulimit=option

Ulimit options. You can use host to copy the current configuration from the host.

--user, -u=[user | user:group | uid | uid:gid | user:gid | uid:group]

Sets the username or UID used and optionally the groupname or GID for the specified com?mand.

Without this argument, the command will run as the user specified in the container image. Unless overridden by a USER command in the Containerfile or by a value passed to this op?tion, this user generally defaults to root.

When a user namespace is not in use, the UID and GID used within the container and on the host will match. When user namespaces are in use, however, the UID and GID in the con?tainer may correspond to another UID and GID on the host. In rootless containers, for ex?ample, a user namespace is always used, and root in the container will by default corre?spond to the UID and GID of the user invoking Podman.

--userns=mode

Set the user namespace mode for the container. It defaults to the PODMAN_USERNS environ?ment variable. An empty value ("") means user namespaces are disabled unless an explicit mapping is set with the --uidmap and --gidmap options.

Valid mode values are:

auto[:OPTIONS,...]: automatically create a unique user namespace.

The --userns=auto flag, requires that the user name containers and a range of subordinate user ids that the Podman container is allowed to use be specified in the /etc/subuid and /etc/subgid files.

Example: containers:2147483647:2147483648.

Podman allocates unique ranges of UIDs and GIDs from the containers subpordinate user ids. The size of the ranges is based on the number of UIDs required in the image. The number of UIDs and GIDs can be overridden with the size option. The auto options currently does not work in rootless mode

Valid auto options:

? gidmapping=_CONTAINER_GID:HOSTGID:SIZE: to force a GID mapping to be present in the user namespace.

? size=SIZE: to specify an explicit size for the automatic user namespace. e.g. --userns=auto:size=8192. If size is not specified, auto will estimate a size for the user namespace.

? uidmapping=_CONTAINER_UID:HOSTUID:SIZE: to force a UID mapping to be present in the user namespace.

container:id: join the user namespace of the specified container.

host: run in the user namespace of the caller. The processes running in the container will have the same privileges on the host as any other process launched by the calling user (default).

keep-id: creates a user namespace where the current rootless user's UID:GID are mapped to the same values in the container. This option is ignored for containers created by the root user.

ns:namespace: run the container in the given existing user namespace.

private: create a new namespace for the container.

This option is incompatible with --gidmap, --uidmap, --subuidname and --subgidname.

--uts=mode

Set the UTS namespace mode for the container. The following values are supported:

? host: use the host's UTS namespace inside the container.

? private: create a new namespace for the container (default).

? ns:[path]: run the container in the given existing UTS namespace.

? container:[container]: join the UTS namespace of the specified container.

--variant=VARIANT

Use VARIANT instead of the default architecture variant of the container image. Some images can use multiple variants of the arm architectures, such as arm/v5 and arm/v7.

--volume, -v[=[[SOURCE-VOLUME|HOST-DIR:]CONTAINER-DIR[:OPTIONS]]]

Create a bind mount. If you specify /HOST-DIR:/CONTAINER-DIR, Podman bind mounts host-dir in the host to CONTAINER-DIR in the Podman container. Similarly, SOURCE-VOLUME:/CONTAINER-DIR will mount the volume in the host to the container. If no such named volume exists, Podman will create one. (Note when using the remote client, the volumes will be mounted from the remote server, not necessarily the client machine.)

The options is a comma-separated list and can be: [1] ?#Footnote1?

? rw|ro

? z|Z

? [r]shared|[r]slave|[r]private[r]unbindable

? [r]bind

? [no]exec

? [no]dev

? [no]suid

? [O]

? [U]

The CONTAINER-DIR must be an absolute path such as /src/docs. The volume will be mounted into the container at this directory.

Volumes may specify a source as well, as either a directory on the host or the name of a named volume. If no source is given, the volume will be created as an anonymously named volume with a randomly generated name, and will be removed when the container is removed via the --rm flag or podman rm --volumes.

If a volume source is specified, it must be a path on the host or the name of a named volume. Host paths are allowed to be absolute or relative; relative paths are resolved relative to the directory Podman is run in. If the source does not exist, Podman will return an error. Users must pre-create the source files or directories.

Any source that does not begin with a . or / will be treated as the name of a named volume. If a volume with that name does not exist, it will be created. Volumes created with names are not anonymous, and they are not removed by the --rm option and the podman rm --volumes command.

You can specify multiple -v options to mount one or more volumes into a container.

Write Protected Volume Mounts

You can add :ro or :rw option to mount a volume in read-only or read-write mode, respec? tively. By default, the volumes are mounted read-write.

Chowning Volume Mounts

By default, Podman does not change the owner and group of source volume directories mounted into containers. If a container is created in a new user namespace, the UID and GID in the container may correspond to another UID and GID on the host.

The :U suffix tells Podman to use the correct host UID and GID based on the UID and GID within the container, to change recursively the owner and group of the source volume.

Warning use with caution since this will modify the host filesystem.

Labeling Volume Mounts

Labeling systems like SELinux require that proper labels are placed on volume content mounted into a container. Without a label, the security system might prevent the processes running inside the container from using the content. By default, Podman does not change the labels set by the OS.

To change a label in the container context, you can add either of two suffixes :z or :Z to the volume mount. These suffixes tell Podman to relabel file objects on the shared vol? umes. The z option tells Podman that two containers share the volume content. As a result, Podman labels the content with a shared content label. Shared volume labels allow all con? tainers to read/write content. The Z option tells Podman to label the content with a pri? vate unshared label.

Note: Do not relabel system files and directories. Relabeling system content might cause other confined services on your machine to fail. For these types of containers we recom? mend that disable SELinux separation. The option --security-opt label=disable disables SELinux separation for the container. For example if a user wanted to volume mount their entire home directory into a container, they need to disable SELinux separation.

```
       $ podman run --security-opt label=disable -v $HOME:/home/user fedora touch /home/user/file
```

Overlay Volume Mounts

The :O flag tells Podman to mount the directory from the host as a temporary storage using the overlay file system. The container processes can modify content within the mountpoint which is stored in the container storage in a separate directory. In overlay terms, the source directory will be the lower, and the container storage directory will be the upper.

Modifications to the mount point are destroyed when the container finishes executing, sim?
ilar to a tmpfs mount point being unmounted.

Subsequent executions of the container will see the original source directory content, any changes from previous container executions no longer exist.

One use case of the overlay mount is sharing the package cache from the host into the con?
tainer to allow speeding up builds.

Note:

- The `O` flag conflicts with other options listed above.

Content mounted into the container is labeled with the private label.

On SELinux systems, labels in the source directory must be readable by the con?
tainer label. Usually containers can read/execute container_share_t and can read/write container_file_t. If you cannot change the labels on a source volume, SELinux container separation must be disabled for the container to work.

- The source directory mounted into the container with an overlay mount should not be modified, it can cause unexpected failures. It is recommended that you do not modify the directory until the container finishes running.

Only the current container can use a private volume.

Mounts propagation

By default bind mounted volumes are private. That means any mounts done inside container will not be visible on host and vice versa. One can change this behavior by specifying a volume mount propagation property. Making a volume shared mounts done under that volume inside container will be visible on host and vice versa. Making a volume slave enables only one way mount propagation and that is mounts done on host under that volume will be visible inside container but not the other way around. [1] ?#Footnote1?

To control mount propagation property of volume one can use [r]shared, [r]slave, [r]pri?
vate or [r]unbindable propagation flag. Propagation property can be specified only for bind mounted volumes and not for internal volumes or named volumes. For mount propagation to work source mount point (mount point where source dir is mounted on) has to have right propagation properties. For shared volumes, source mount point has to be shared. And for slave volumes, source mount has to be either shared or slave. [1] ?#Footnote1?

If you want to recursively mount a volume and all of its submounts into a container, then you can use the rbind option. By default the bind option is used, and submounts of the source directory will not be mounted into the container.

Mounting the volume with the nosuid options means that SUID applications on the volume will not be able to change their privilege. By default volumes are mounted with nosuid.

Mounting the volume with the noexec option means that no executables on the volume will be able to executed within the container.

Mounting the volume with the nodev option means that no devices on the volume will be able to be used by processes within the container. By default volumes are mounted with nodev.

If the host-dir is a mount point, then dev, suid, and exec options are ignored by the ker? nel.

Use df $hostdir to figure out the source mount, and then use findmnt -o TARGET,PROPAGATION source-mount-dir to figure out propagation properties of source mount. If findmnt(1) util? ity is not available, then one can look at mount entry for source mount point in /proc/self/mountinfo. Look at the "optional fields" and see if any propagation properties are specified. In there, shared:N means the mount is shared, master:N means mount is slave, and if nothing is there, the mount is private. [1] ?#Footnote1?

To change propagation properties of a mount point, use mount(8) command. For example, if one wants to bind mount source directory /foo, one can do mount --bind /foo /foo and mount --make-private --make-shared /foo. This will convert /foo into a shared mount point. Al? ternatively, one can directly change propagation properties of source mount. Say, if / is source mount for /foo, then use mount --make-shared / to convert / into a shared mount.

Note: if the user only has access rights via a group, accessing the volume from inside a rootless container will fail. Use the --group-add keep-groups flag to pass the user's sup? plementary group access into the container.

--volumes-from[=CONTAINER[:OPTIONS]]

Mount volumes from the specified container(s). Used to share volumes between containers. The options is a comma-separated list with the following available elements:

? rw|ro

? z

Mounts already mounted volumes from a source container onto another container. You must supply the source's container-id or container-name. To share a volume, use the --volumes-from option when running the target container. You can share volumes even if the source container is not running.

By default, Podman mounts the volumes in the same mode (read-write or read-only) as it is mounted in the source container. You can change this by adding a ro or rw option.

Labeling systems like SELinux require that proper labels are placed on volume content mounted into a container. Without a label, the security system might prevent the processes running inside the container from using the content. By default, Podman does not change the labels set by the OS.

To change a label in the container context, you can add z to the volume mount. This suf?fix tells Podman to relabel file objects on the shared volumes. The z option tells Podman that two containers share the volume content. As a result, Podman labels the content with a shared content label. Shared volume labels allow all containers to read/write content.

If the location of the volume from the source container overlaps with data residing on a target container, then the volume hides that data on the target.

--workdir, -w=dir

Working directory inside the container.

The default working directory for running binaries within a container is the root direc?tory (/). The image developer can set a different default with the WORKDIR instruction. The operator can override the working directory by using the -w option.

--pidfile=path

When the pidfile location is specified, the container process' PID will be written to the pidfile. (This option is not available with the remote Podman client) If the pidfile op?tion is not specified, the container process' PID will be written to /run/containers/stor?age/${storage-driver}-containers/$CID/userdata/pidfile.

After the container is started, the location for the pidfile can be discovered with the following podman inspect command:

```
$ podman inspect --format '{{ .PidFile }}' $CID
/run/containers/storage/${storage-driver}-containers/$CID/userdata/pidfile
```

Exit Status

The exit code from podman run gives information about why the container failed to run or why it exited. When podman run exits with a non-zero code, the exit codes follow the ch?root(1) standard, see below:

125 The error is with Podman itself

```
$ podman run --foo busybox; echo $?
Error: unknown flag: --foo
125
```

126 The contained command cannot be invoked

```
$ podman run busybox /etc; echo $?
Error: container_linux.go:346: starting container process caused "exec: \"/etc\": permission denied": OCI runtime error
126
```

127 The contained command cannot be found

```
$ podman run busybox foo; echo $?
Error: container_linux.go:346: starting container process caused "exec: \"foo\": executable file not found in $PATH": OCI runtime error
127
```

Exit code contained command exit code

```
$ podman run busybox /bin/sh -c 'exit 3'; echo $?
3
```

EXAMPLES

Running container in read-only mode

During container image development, containers often need to write to the image content. Installing packages into /usr, for example. In production, applications seldom need to write to the image. Container applications write to volumes if they need to write to file systems at all. Applications can be made more secure by running them in read-only mode us? ing the --read-only switch. This protects the containers image from modification. Read only containers may still need to write temporary data. The best way to handle this is to mount tmpfs directories on /run and /tmp.

```
$ podman run --read-only -i -t fedora /bin/bash

$ podman run --read-only --read-only-tmpfs=false --tmpfs /run -i -t fedora /bin/bash
```

Exposing log messages from the container to the host's log

If you want messages that are logged in your container to show up in the host's sys? log/journal then you should bind mount the /dev/log directory as follows.

```
$ podman run -v /dev/log:/dev/log -i -t fedora /bin/bash
```

From inside the container you can test this by sending a message to the log.

```
(bash)# logger "Hello from my container"
```

Then exit and check the journal.

```
(bash)# exit

$ journalctl -b | grep Hello
```

This should list the message sent to logger.

Attaching to one or more from STDIN, STDOUT, STDERR

If you do not specify -a, Podman will attach everything (stdin, stdout, stderr). You can specify to which of the three standard streams (stdin, stdout, stderr) you'd like to con?nect instead, as in:

```
$ podman run -a stdin -a stdout -i -t fedora /bin/bash
```

Sharing IPC between containers

Using shm_server.c available here: https://www.cs.cf.ac.uk/Dave/C/node27.html

Testing --ipc=host mode:

Host shows a shared memory segment with 7 pids attached, happens to be from httpd:

```
$ sudo ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01128e25 0          root       600        1000       7
```

Now run a regular container, and it correctly does NOT see the shared memory segment from the host:

```
$ podman run -it shm ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
```

Run a container with the new --ipc=host option, and it now sees the shared memory segment from the host httpd:

```
$ podman run -it --ipc=host shm ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01128e25 0          root       600        1000       7
```

Testing --ipc=container:id mode:

Start a container with a program to create a shared memory segment:

```
$ podman run -it shm bash
$ sudo shm/shm_server &
$ sudo ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000162e 0          root       666        27         1
```

Create a 2nd container correctly shows no shared memory segment from 1st container:

```
$ podman run shm ipcs -m

------ Shared Memory Segments --------

key        shmid      owner      perms      bytes      nattch     status
```

Create a 3rd container using the --ipc=container:id option, now it shows the shared memory segment from the first:

```
$ podman run -it --ipc=container:ed735b2264ac shm ipcs -m

$ sudo ipcs -m

------ Shared Memory Segments --------

key        shmid      owner      perms      bytes      nattch     status

0x0000162e 0          root       666        27         1
```

Mapping Ports for External Usage

The exposed port of an application can be mapped to a host port using the -p flag. For ex?ample, an httpd port 80 can be mapped to the host port 8080 using the following:

```
$ podman run -p 8080:80 -d -i -t fedora/httpd
```

Mounting External Volumes

To mount a host directory as a container volume, specify the absolute path to the direc?tory and the absolute path for the container directory separated by a colon. If the source is a named volume maintained by Podman, it is recommended to use its name rather than the path to the volume. Otherwise the volume will be considered as an orphan and wiped if you execute podman volume prune:

```
$ podman run -v /var/db:/data1 -i -t fedora bash

$ podman run -v data:/data2 -i -t fedora bash

$ podman run -v /var/cache/dnf:/var/cache/dnf:O -ti fedora dnf -y update

  $ podman run -d -e MYSQL_ROOT_PASSWORD=root --user mysql --userns=keep-id -v ~/data:/var/lib/mysql:z,U
mariadb
```

Using --mount flags to mount a host directory as a container folder, specify the absolute path to the directory or the volume name, and the absolute path within the container di?rectory:

```
$ podman run --mount type=bind,src=/var/db,target=/data1 busybox sh

$ podman run --mount type=bind,src=volume-name,target=/data1 busybox sh
```

When using SELinux, be aware that the host has no knowledge of container SELinux policy. Therefore, in the above example, if SELinux policy is enforced, the /var/db directory is not writable to the container. A "Permission Denied" message will occur and an avc: mes?

sage in the host's syslog.

To work around this, at time of writing this man page, the following command needs to be run in order for the proper SELinux policy type label to be attached to the host direc?
tory:

```
$ chcon -Rt svirt_sandbox_file_t /var/db
```

Now, writing to the /data1 volume in the container will be allowed and the changes will also be reflected on the host in /var/db.

Using alternative security labeling

You can override the default labeling scheme for each container by specifying the --secu?
rity-opt flag. For example, you can specify the MCS/MLS level, a requirement for MLS sys?
tems. Specifying the level in the following command allows you to share the same content between containers.

```
podman run --security-opt label=level:s0:c100,c200 -i -t fedora bash
```

An MLS example might be:

```
$ podman run --security-opt label=level:TopSecret -i -t rhel7 bash
```

To disable the security labeling for this container versus running with the
--permissive flag, use the following command:

```
$ podman run --security-opt label=disable -i -t fedora bash
```

If you want a tighter security policy on the processes within a container, you can specify an alternate type for the container. You could run a container that is only allowed to listen on Apache ports by executing the following command:

```
$ podman run --security-opt label=type:svirt_apache_t -i -t centos bash
```

Note you would have to write policy defining a svirt_apache_t type.

To mask additional specific paths in the container, specify the paths separated by a colon using the mask option with the --security-opt flag.

```
$ podman run --security-opt mask=/foo/bar:/second/path fedora bash
```

To unmask all the paths that are masked by default, set the unmask option to ALL. Or to only unmask specific paths, specify the paths as shown above with the mask option.

```
$ podman run --security-opt unmask=ALL fedora bash
```

To unmask all the paths that start with /proc, set the unmask option to /proc/*.

```
$ podman run --security-opt unmask=/proc/* fedora bash
$ podman run --security-opt unmask=/foo/bar:/sys/firmware fedora bash
```

Setting device weight

If you want to set /dev/sda device weight to 200, you can specify the device weight by --blkio-weight-device flag. Use the following command:

```
$ podman run -it --blkio-weight-device "/dev/sda:200" ubuntu
```

Using a podman container with input from a pipe

```
$ echo "asdf" | podman run --rm -i --entrypoint /bin/cat someimage
asdf
```

Setting automatic user namespace separated containers

```
# podman run --userns=auto:size=65536 ubi8-micro cat /proc/self/uid_map
0 2147483647      65536
# podman run --userns=auto:size=65536 ubi8-micro cat /proc/self/uid_map
0 2147549183      65536
```

Setting Namespaced Kernel Parameters (Sysctls)

The --sysctl sets namespaced kernel parameters (sysctls) in the container. For example, to turn on IP forwarding in the containers network namespace, run this command:

```
$ podman run --sysctl net.ipv4.ip_forward=1 someimage
```

Note that not all sysctls are namespaced. Podman does not support changing sysctls inside of a container that also modify the host system. As the kernel evolves we expect to see more sysctls become namespaced.

See the definition of the --sysctl option above for the current list of supported sysctls.

Set UID/GID mapping in a new user namespace

Running a container in a new user namespace requires a mapping of the uids and gids from the host.

```
$ podman run --uidmap 0:30000:7000 --gidmap 0:30000:7000 fedora echo hello
```

Configuring Storage Options from the command line

Podman allows for the configuration of storage by changing the values in the /etc/con?tainer/storage.conf or by using global options. This shows how to setup and use fuse-over?layfs for a one time run of busybox using global options.

```
podman --log-level=debug --storage-driver overlay --storage-opt "overlay.mount_program=/usr/bin/fuse-overlayfs" run busybox /bin/sh
```

Configure timezone in a container

```
$ podman run --tz=local alpine date
$ podman run --tz=Asia/Shanghai alpine date
$ podman run --tz=US/Eastern alpine date
```

Adding dependency containers

The first container, container1, is not started initially, but must be running before con?tainer2 will start. The podman run command will start the container automatically before starting container2.

```
$ podman create --name container1 -t -i fedora bash

$ podman run --name container2 --requires container1 -t -i fedora bash
```

Multiple containers can be required.

```
$ podman create --name container1 -t -i fedora bash

$ podman create --name container2 -t -i fedora bash

$ podman run --name container3 --requires container1,container2 -t -i fedora bash
```

Configure keep supplemental groups for access to volume

```
$ podman run -v /var/lib/design:/var/lib/design --group-add keep-groups ubi8
```

Configure execution domain for containers using personality flag

```
$ podman run --name container1 --personaity=LINUX32 fedora bash
```

Rootless Containers

Podman runs as a non root user on most systems. This feature requires that a new enough version of shadow-utils be installed. The shadow-utils package must include the newuidmap(1) and newgidmap(1) executables.

Note: RHEL7 and Centos 7 will not have this feature until RHEL7.7 is released.

In order for users to run rootless, there must be an entry for their username in /etc/sub?uid and /etc/subgid which lists the UIDs for their user namespace.

Rootless Podman works better if the fuse-overlayfs and slirp4netns packages are installed. The fuse-overlayfs package provides a userspace overlay storage driver, otherwise users need to use the vfs storage driver, which is diskspace expensive and does not perform well. slirp4netns is required for VPN, without it containers need to be run with the --network=host flag.

ENVIRONMENT

Environment variables within containers can be set using multiple different options, in the following order of precedence (later entries override earlier entries):

? Container image: Any environment variables specified in the container image.

? --http-proxy: By default, several environment variables will be passed in from the host, such as http_proxy and no_proxy. See --http-proxy for details.

? --env-host: Host environment of the process executing Podman is added.

? --env-file: Any environment variables specified via env-files. If multiple files specified, then they override each other in order of entry.

? --env: Any environment variables specified will override previous settings.

Run containers and set the environment ending with a * and a *****:

```
$ export ENV1=a

$ podman run --env ENV* alpine printenv ENV1

a

$ podman run --env ENV*****=b alpine printenv ENV*****

b
```

CONMON

When Podman starts a container it actually executes the conmon program, which then exe?cutes the OCI Runtime. Conmon is the container monitor. It is a small program whose job is to watch the primary process of the container, and if the container dies, save the exit code. It also holds open the tty of the container, so that it can be attached to later. This is what allows Podman to run in detached mode (backgrounded), so Podman can exit but conmon continues to run. Each container has their own instance of conmon. Conmon waits for the container to exit, gathers and saves the exit code, and then launches a Podman process to complete the container cleanup, by shutting down the network and storage. For more information on conmon, please reference the conmon(8) man page.

FILES

/etc/subuid

/etc/subgid

NOTE: Use the environment variable TMPDIR to change the temporary storage location of downloaded container images. Podman defaults to use /var/tmp.

SEE ALSO

podman(1), podman-save(1), podman-ps(1), podman-attach(1), podman-pod-create(1), podman-port(1), podman-start(1), podman-kill(1), podman-stop(1), podman-generate-systemd(1) pod?man-rm(1), subgid(5), subuid(5), containers.conf(5), systemd.unit(5), setsebool(8), slirp4netns(1), fuse-overlayfs(1), proc(5), conmon(8), personality(2).

HISTORY

September 2018, updated by Kunal Kushwaha <kushwaha_kunal_v7@lab.ntt.co.jp>

October 2017, converted from Docker documentation to Podman by Dan Walsh for Podman <dwalsh@redhat.com>

November 2015, updated by Sally O'Malley <somalley@redhat.com>

June 2014, updated by Sven Dowideit <SvenDowideit@home.org.au>

April 2014, Originally compiled by William Henry <whenry@redhat.com> based on docker.com source material and internal work.

FOOTNOTES

1: The Podman project is committed to inclusivity, a core value of open source. The master and slave mount propagation terminology used here is problematic and divisive, and should be changed. However, these terms are currently used within the Linux kernel and must be used as-is at this time. When the kernel maintainers rectify this usage, Podman will fol? low suit immediately.