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 networking to expose, and more, but **podman run** gives final control to the operator or administrator 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 dependencies, 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 example, **--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* (or */var/run/.containerenv* for FreeBSD containers). When using the --privileged 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. Note: */run/.containerenv* will not be created when a volume is mounted on /run.

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 is used by default. For remote Podman, including Mac and Windows (excluding WSL2) machines, 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/podman/stable:latest”. The reference can include a path to a specific registry; if it does not, the registries listed in registries.conf are queried to find a matching image. By default, credentials from podman login (stored at $XDG_RUNTIME_DIR/containers/auth.json by default) are 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 custom host-to-IP mapping (host:ip)

Add a line to /etc/hosts. The format is hostname:ip. The **--add-host** option can be set multiple times. Conflicts with the **--no-hosts** option.

--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. Unless overridden, subsequent lookups of the same image in the local storage matches this architecture, regardless of the host.

--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 command-line executables expect) and pass along signals. The **-a** option can be set for each of **stdin**, **stdout**, and **stderr**.

--authfile=*path*

Path of the authentication file. Default is ${XDG_RUNTIME_DIR}/containers/auth.json on Linux, and $HOME/.config/containers/auth.json on Windows/macOS. The file is created by **podman login**. If the authorization state is not found there, $HOME/.docker/config.json is checked, which is set using **docker login**.

Note: There is also the option to override the default path of the authentication file by setting the REGISTRY_AUTH_FILE environment variable. This can be done with **export REGISTRY_AUTH_FILE=*path***.

--blkio-weight=*weight*

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

This option is not supported on cgroups V1 rootless systems.

--blkio-weight-device=*device:weight*

Block IO relative device weight.

--cap-add=*capability*

Add Linux capabilities.

--cap-drop=*capability*

Drop Linux capabilities.

--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.

--cgroup-parent=*path*

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

--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=*how*

Determines whether the container creates CGroups.

Default is **enabled**.

The **enabled** option creates a new cgroup under the cgroup-parent. The **disabled** option forces 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**.

--chrootdirs=*path*

Path to a directory inside the container that is treated as a chroot directory. Any Podman managed file (e.g., /etc/resolv.conf, /etc/hosts, etc/hostname) that is mounted into the root directory is mounted into that location as well. Multiple directories are separated with a comma.

--cidfile=*file*

Write the container ID to *file*. The file is removed along with the container, except when used with podman --remote run on detached containers.

--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, including Mac and Windows (excluding WSL2) machines)

--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 resource limits may not be allowed for non-root users. For more details, see https://github.com/containers/podman/blob/main/troubleshooting.md#26-running-containers-with-resource-limits-fails-with-a-permissions-error

This option is not supported on cgroups V1 rootless systems.

--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 a number is provided, the container is allowed to use that much CPU time until the CPU period ends (controllable via **--cpu-period**).

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

This option is not supported on cgroups V1 rootless systems.

--cpu-rt-period=*microseconds*

Limit the CPU real-time period in microseconds.

Limit the container's Real Time CPU usage. This option tells the kernel to restrict the container's Real Time CPU usage to the period specified.

This option is only supported on cgroups V1 rootful systems.

--cpu-rt-runtime=*microseconds*

Limit the CPU real-time runtime in microseconds.

Limit the containers Real Time CPU usage. This option 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 can 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 option is only supported on cgroups V1 rootful systems.

--cpu-shares, -c=*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 only applies 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 varies 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 receives 50% of the total CPU time. If a fourth container is added 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 the container *C0* is started 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

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

This option is not supported on cgroups V1 rootless systems.

--cpus=*number*

Number of CPUs. The default is *0.0* which means no limit. This is shorthand for **--cpu-period** and **--cpu-quota**, therefore the option cannot be specified with **--cpu-period** or **--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/main/troubleshooting.md#26-running-containers-with-resource-limits-fails-with-a-permissions-error

This option is not supported on cgroups V1 rootless systems.

--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**).

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

This option is not supported on cgroups V1 rootless systems.

--cpuset-mems=*nodes*

Memory nodes (MEMs) in which to allow execution (0-3, 0,1). Only effective on NUMA systems.

If there are four memory nodes on the system (0-3), use **--cpuset-mems=0,1** then processes in the container only uses memory from the first two memory nodes.

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

This option is not supported on cgroups V1 rootless systems.

--decryption-key=*key[:passphrase]*

The [key[:passphrase]] to be used for decryption of images. Key can point to keys and/or certificates. Decryption is tried with all keys. If the key is protected by a passphrase, it is required to be passed in the argument and omitted otherwise.

--detach, -d

Detached mode: run the container in the background and print the new container ID. The default is *false*.

At any time run **podman ps** in the other shell to view a list of the running containers. Reattach to a detached container with **podman attach** command.

When attached via tty mode, detach from the container (and leave it running) using a configurable key sequence. The default sequence is ctrl-p,ctrl-q. Specify the key sequence using the **--detach-keys** option, or configure 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 _. Specifying "" disables this feature. The default is *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 by combining **r** for read, **w** for write, and **m** for **mknod**(2).

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

Note: if *host-device* is a symbolic link then it is resolved first. The container only stores the major and minor numbers of the host device.

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

In rootless mode, the new device is bind mounted in the container from the host rather than Podman creating it within the container space. Because the bind mount retains its SELinux label on SELinux systems, the container can get permission denied when accessing the mounted device. Modify SELinux settings to allow containers to use all device labels via the following command:

$ sudo setsebool -P container_use_devices=true

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

--device-cgroup-rule=*"type major:minor mode"*

Add a rule to the cgroup allowed devices list. The rule is expected to be in the format specified in the Linux kernel documentation admin-guide/cgroup-v1/devices:

- *type*: a (all), c (char), or b (block);
- *major* and *minor*: either a number, or * for all;
- *mode*: a composition of r (read), w (write), and m (mknod(2)).

--device-read-bps=*path:rate*

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

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

This option is not supported on cgroups V1 rootless systems.

--device-read-iops=*path:rate*

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

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

This option is not supported on cgroups V1 rootless systems.

--device-write-bps=*path:rate*

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

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

This option is not supported on cgroups V1 rootless systems.

--device-write-iops=*path:rate*

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

On some systems, changing the resource limits may not be allowed for non-root users. For more details, see

https://github.com/containers/podman/blob/main/troubleshooting.md#26-running-containers-with-resource-limits-fails-with-a-permissions-error

This option is not supported on cgroups V1 rootless systems.

--disable-content-trust

This is a Docker-specific option to disable image verification to a container registry and is not supported by Podman. This option is a NOOP and provided solely for scripting compatibility.

--dns=*ipaddr*

Set custom DNS servers.

This option can be used to override the DNS configuration passed to the container. Typically 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** flag 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 is used without changes.

This option cannot be combined with **--network** that is set to **none** or **container:***id*.

--dns-option=*option*

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

--dns-search=*domain*

Set custom DNS search domains. Invalid if using **--dns-search** with **--network** that is set to **none** or **container:***id*. Use **--dns-search=.** to remove the search domain.

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

Override the default ENTRYPOINT from 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 ENTRYPOINT gives a container its default nature or behavior. When the ENTRYPOINT is set, the container runs as if it were that binary, complete with default options. More options can be passed in via the COMMAND. But, if a user wants to run something else inside the container, the **--entrypoint** option allows a new ENTRYPOINT to be specified.

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 checks the host environment for a value and set the variable only if it is set on the host. As a special case, if an environment variable ending in ***** is specified without a value, Podman searches the host environment for variables starting with the prefix and adds those variables to the container.

See **Environment** note below for precedence and examples.

--env-file=*file*

Read in a line-delimited file of environment variables.

See **Environment** note below for precedence and examples.

--env-host

Use host environment inside of the container. See **Environment** note below for precedence. (This option is not available with the remote Podman client, including Mac and Windows (excluding WSL2) machines)

--env-merge=*env*

Preprocess default environment variables for the containers. For example if image contains environment variable hello=world user can preprocess it using --env-merge hello=${hello}-some so new value is hello=world-some.

Please note that if the environment variable hello is not present in the image, then it'll be replaced by an empty string and so using --env-merge hello=${hello}-some would result in the new value of hello=-some, notice the leading - delimiter.

--expose=*port[/protocol]*

Expose a port or a range of ports (e.g. **--expose=3300-3310**). The protocol can be tcp, udp or sctp and if not given tcp is assumed. This option matches the EXPOSE instruction for image builds and has no effect on the actual networking rules unless **-P/--publish-all** is used to forward to all exposed ports from random host ports. To forward specific ports from the host into the container use the **-p/--publish** option instead.

--gidmap=*[flags]container_uid:from_uid[:amount]*

Run the container in a new user namespace using the supplied GID mapping. This option conflicts with the **--userns** and **--subgidname** options. This option provides a way to map host GIDs to container GIDs in the same way as **--uidmap** maps host UIDs to container UIDs. For details see **--uidmap**.

Note: the **--gidmap** option cannot be called in conjunction with the **--pod** option as a gidmap cannot be set on the container level when in a pod.

--gpus=*ENTRY*

GPU devices to add to the container ('all' to pass all GPUs) Currently only Nvidia devices are supported.

--group-add=*group* | *keep-groups*

Assign additional groups to the 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, other groups cannot be specified with this flag. (Not available for remote commands, including Mac and Windows (excluding WSL2) machines)

--group-entry=*ENTRY*

Customize the entry that is written to the /etc/group file within the container when --user is used.

The variables $GROUPNAME, $GID, and $USERLIST are automatically replaced with their value at runtime if present.

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

Set or alter a healthcheck command for a container. The command is a command to be executed inside the container that determines the 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 is 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-on-failure=*action*

Action to take once the container transitions to an unhealthy state. The default is **none**.

- **none**: Take no action.

- **kill**: Kill the container.
- **restart**: Restart the container. Do not combine the restart action with the --restart flag. When running inside of a systemd unit, consider using the kill or stop action instead to make use of systemd's restart policy.
- **stop**: Stop the container.

--health-retries=*retries*

The number of retries allowed before a healthcheck is considered to be unhealthy. The default 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**.

Note: The health check command is executed as soon as a container is started, if the health check is successful the container's health state will be updated to healthy. However, if the health check fails, the health state will stay as starting until either the health check is successful or until the --health-start-period time is over. If the health check command fails after the --health-start-period time is over, the health state will be updated to unhealthy. The health check command is executed periodically based on the value of --health-interval.

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

Set a startup healthcheck command for a container. This command is executed inside the container and is used to gate the regular healthcheck. When the startup command succeeds, the regular healthcheck begins and the startup healthcheck ceases. Optionally, if the command fails for a set number of attempts, the container is restarted. A startup healthcheck can be used to ensure that containers with an extended startup period are not marked as unhealthy until they are fully started. Startup healthchecks can only be used when a regular healthcheck (from the container's image or the **--health-cmd** option) is also set.

--health-startup-interval=*interval*

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

--health-startup-retries=*retries*

The number of attempts allowed before the startup healthcheck restarts the container. If set to **0**, the container is never restarted. The default is **0**.

--health-startup-success=*retries*

The number of successful runs required before the startup healthcheck succeeds and the regular healthcheck begins. A value of **0** means that any success begins the regular healthcheck. The default is **0**.

--health-startup-timeout=*timeout*

The maximum time a startup healthcheck command has to complete before it is marked as failed. The value can be expressed in a time format like **2m3s**. The default value is **30s**.

--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, -h=*name*

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 is used.

--hostuser=*name*

Add a user account to /etc/passwd from the host to the container. The Username or UID must exist on the host system.

--http-proxy

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 does not use any proxy. Proxy environment variables specified for the container in any other way overrides the values that 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.) When used with the remote client it uses the proxy environment variables that are set on the server process.

Defaults to **true**.

--image-volume=bind | *tmpfs* | *ignore*

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

- **bind**: An anonymous named volume is 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. The container-init binary is mounted at /run/podman-init. Mounting over /run breaks container execution.

--init-path=*path*

Path to the container-init binary.

--interactive, -i

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

--ip=*ipv4*

Specify a static IPv4 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=network-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 network's IP address pool (default **10.88.0.0/16**).

To specify multiple static IP addresses per container, set multiple networks using the **--network** option with a static IP address specified for each using the ip mode for that option.

--ip6=*ipv6*

Specify a static IPv6 address for the container, for example **fd46:db93:aa76:ac37::10**. This option can only be used if the container is joined to only a single network - i.e., **--network=network-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 network's IPv6 address pool.

To specify multiple static IPv6 addresses per container, set multiple networks using the **--network** option with a static IPv6 address specified for each using the ip6 mode for that option.

--ipc=*ipc*

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

- “”: Use Podman’s default, defined in containers.conf.
- **container:***id*: reuses another container’s shared memory, semaphores, and message queues
- **host**: use the host’s shared memory, semaphores, and message queues inside the container. Note: the host mode gives the container full access to local shared memory and is therefore considered insecure.
- **none**: private IPC namespace, with /dev/shm not mounted.
- **ns:***path*: path to an IPC namespace to join.
- **private**: private IPC namespace.
- **shareable**: private IPC namespace with a possibility to share it with other containers.

--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**, **none**, **passthrough** and **passthrough-tty**, with **json-file** aliased to **k8s-file** for scripting compatibility. (Default **journald**).

The podman info command below displays the default log-driver for the system.

$ podman info --format '{{ .Host.LogDriver }}'

journald

The **passthrough** driver passes down the standard streams (stdin, stdout, stderr) to the container. It is not allowed with the remote Podman client, including Mac and Windows (excluding WSL2) machines, and on a tty, since it is vulnerable to attacks via TIOCSTI.

The **passthrough-tty** driver is the same as **passthrough** except that it also allows it to be used on a TTY if the user really wants it.

--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}}"**. It supports the same keys as **podman inspect --format**. This option is currently supported only by the **journald** log driver.

--mac-address=*address*

Container network interface MAC address (e.g. 92:d0:c6:0a:29:33) This option can only be used if the container is joined to only a single network - i.e., **--network=*network-name*** is used at most once - and if the container is not joining another container's network namespace via **--network=container:*id***.

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

To specify multiple static MAC addresses per container, set multiple networks using the **--network** option with a static MAC address specified for each using the mac mode for that option.

--memory, -m=*number[unit]*

Memory limit. A *unit* can be **b** (bytes), **k** (kibibytes), **m** (mebibytes), or **g** (gibibytes).

Allows the memory available to a container to be constrained. 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 is very large, that's millions of trillions).

This option is not supported on cgroups V1 rootless systems.

--memory-reservation=*number[unit]*

Memory soft limit. A *unit* can be **b** (bytes), **k** (kibibytes), **m** (mebibytes), or **g** (gibibytes).

After setting memory reservation, when the system detects memory contention or low memory, containers are forced to restrict their consumption to their reservation. So always set the value below **--memory**, otherwise the hard limit takes precedence. By default, memory reservation is the same as memory limit.

This option is not supported on cgroups V1 rootless systems.

--memory-swap=*number[unit]*

A limit value equal to memory plus swap. A *unit* can be **b** (bytes), **k** (kibibytes), **m** (mebibytes), or **g** (gibibytes).

Must be used with the **-m** (**--memory**) flag. The argument value must 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.

This option is not supported on cgroups V1 rootless systems.

--memory-swappiness=*number*

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

This flag is only supported on cgroups V1 rootful systems.

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

Attach a filesystem mount to the container

Current supported mount TYPEs are **bind**, **devpts**, **glob**, **image**, **ramfs**, **tmpfs** and **volume**.

Options common to all mount types:

- *src*, *source*: mount source spec for **bind**, **glob**, and **volume**. Mandatory for **bind** and **glob**.
- *dst*, *destination*, *target*: mount destination spec.

When source globs are specified without the destination directory, the files and directories are mounted with their complete path within the container. When the destination is specified, the files and directories matching the glob on the base file name on the destination directory are mounted. The option type=glob,src=/foo*,destination=/tmp/bar tells container engines to mount host files matching /foo* to the /tmp/bar/ directory in the container.

Options specific to type=**volume**:

- *ro*, *readonly*: *true* or *false* (default if unspecified: *false*).
- *U*, *chown*: *true* or *false* (default if unspecified: *false*). Recursively change the owner and group of the source volume based on the UID and GID of the container.

- *idmap*: If specified, create an idmapped mount to the target user namespace in the container. The idmap option supports a custom mapping that can be different than the user namespace used by the container. The mapping can be specified after the idmap option like: idmap=uids=0-1-10#10-11-10;gids=0-100-10. For each triplet, the first value is the start of the backing file system IDs that are mapped to the second value on the host. The length of this mapping is given in the third value. Multiple ranges are separated with #. If the specified mapping is prepended with a '@' then the mapping is considered relative to the container user namespace. The host ID for the mapping is changed to account for the relative position of the container user in the container user namespace.

Options specific to type=**image**:

- *rw*, *readwrite*: *true* or *false* (default if unspecified: *false*).
- *subpath*: Mount only a specific path within the image, instead of the whole image.

Options specific to **bind** and **glob**:

- *ro*, *readonly*: *true* or *false* (default if unspecified: *false*).
- *bind-propagation*: *shared*, *slave*, *private*, *unbindable*, *rshared*, *rslave*, *runbindable*, or **rprivate** (default).[1] See also mount(2).
- *bind-nonrecursive*: do not set up a recursive bind mount. By default it is recursive.
- *relabel*: *shared*, *private*.
- *idmap*: *true* or *false* (default if unspecified: *false*). If true, create an idmapped mount to the target user namespace in the container.
- *U*, *chown*: *true* or *false* (default if unspecified: *false*). Recursively change the owner and group of the source volume based on the UID and GID of the container.
- *no-dereference*: do not dereference symlinks but copy the link source into the mount destination.

Options specific to type=**tmpfs** and **ramfs**:

- *ro*, *readonly*: *true* or *false* (default if unspecified: *false*).
- *tmpfs-size*: Size of the tmpfs/ramfs mount, in bytes. Unlimited by default in Linux.
- *tmpfs-mode*: Octal file mode of the tmpfs/ramfs (e.g. 700 or 0700.).
- *tmpcopyup*: Enable copyup from the image directory at the same location to the tmpfs/ramfs. Used by default.

- *notmpcopyup*: Disable copying files from the image to the tmpfs/ramfs.
- *U*, *chown*: *true* or *false* (default if unspecified: *false*). Recursively change the owner and group of the source volume based on the UID and GID of the container.

Options specific to type=**devpts**:

- *uid*: numeric UID of the file owner (default: 0).
- *gid*: numeric GID of the file owner (default: 0).
- *mode*: octal permission mask for the file (default: 600).
- *max*: maximum number of PTYs (default: 1048576).

Examples:

- type=bind,source=/path/on/host,destination=/path/in/container
- type=bind,src=/path/on/host,dst=/path/in/container,relabel=shared
- type=bind,src=/path/on/host,dst=/path/in/container,relabel=shared,U=true
- type=devpts,destination=/dev/pts
- type=glob,src=/usr/lib/libfoo*,destination=/usr/lib,ro=true
- type=image,source=fedora,destination=/fedora-image,rw=true
- type=ramfs,tmpfs-size=512M,destination=/path/in/container
- type=tmpfs,tmpfs-size=512M,destination=/path/in/container
- type=tmpfs,destination=/path/in/container,noswap
- type=volume,source=vol1,destination=/path/in/container,ro=true

--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 generates a random string name. The name can be useful

as a more human-friendly way to identify containers. This works for both background and foreground containers.

--network=*mode*, --net

Set the network mode for the container.

Valid *mode* values are:

- **bridge[:OPTIONS,...]**: Create a network stack on the default bridge. This is the default for rootful containers. It is possible to specify these additional options:
    - **alias=***name*: Add network-scoped alias for the container.
    - **ip=***IPv4*: Specify a static IPv4 address for this container.
    - **ip6=***IPv6*: Specify a static IPv6 address for this container.
    - **mac=***MAC*: Specify a static MAC address for this container.
    - **interface_name=***name*: Specify a name for the created network interface inside the container.

For example, to set a static ipv4 address and a static mac address, use --network bridge:ip=10.88.0.10,mac=44:33:22:11:00:99.

- <*network name or ID*>**[:OPTIONS,...]**: Connect to a user-defined network; this is the network name or ID from a network created by **podman network create**. Using the network name implies the bridge network mode. It is possible to specify the same options described under the bridge mode above. Use the **--network** option multiple times to specify additional networks.
  For backwards compatibility it is also possible to specify comma-separated networks on the first **--network** argument, however this prevents you from using the options described under the bridge section above.
- **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 uses 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.
- **ns:***path*: Path to a network namespace to join.
- **private**: Create a new namespace for the container. This uses the **bridge** mode for rootful containers and **slirp4netns** for rootless ones.

- **slirp4netns[:OPTIONS,...]**: use **slirp4netns**(1) to create a user network stack. It is possible to specify these additional options, they can also be set with network_cmd_options in containers.conf:
    - **allow_host_loopback=true|false**: Allow slirp4netns to reach the host loopback IP (default is 10.0.2.2 or the second IP from slirp4netns cidr subnet when changed, see the cidr option below). The 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 true. (Required for outbound_addr6).
    - **outbound_addr=***INTERFACE*: Specify the outbound interface slirp binds to (ipv4 traffic only).
    - **outbound_addr=***IPv4*: Specify the outbound ipv4 address slirp binds to.
    - **outbound_addr6=***INTERFACE*: Specify the outbound interface slirp binds to (ipv6 traffic only).
    - **outbound_addr6=***IPv6*: Specify the outbound ipv6 address slirp binds to.
    - **port_handler=rootlesskit**: Use rootlesskit for port forwarding. Default. Note: Rootlesskit changes the source IP address of incoming packets to an IP address in the container network namespace, usually 10.0.2.100. If the application requires 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.
- **pasta[:OPTIONS,...]**: use **pasta**(1) to create a user-mode networking stack. This is the default for rootless containers and only supported in rootless mode. By default, IPv4 and IPv6 addresses and routes, as well as the pod interface name, are copied from the host. If port forwarding isn't configured, ports are forwarded dynamically as services are bound on either side (init namespace or container namespace). Port forwarding preserves the original source IP address. Options described in pasta(1) can be specified as comma-separated arguments. In terms of pasta(1) options, **--config-net** is given by default, in order to configure networking when the container is started, and **--no-map-gw** is also assumed by default, to avoid direct access from container to host using the gateway address.

The latter can be overridden by passing **--map-gw** in the pasta-specific options (despite not being an actual pasta(1) option).
Also, **-t none** and **-u none** are passed if, respectively, no TCP or UDP port forwarding from host to container is configured, to disable automatic port forwarding based on bound ports. Similarly, **-T none** and **-U none** are given to disable the same functionality from container to host.
Some examples:

- **pasta:--map-gw**: Allow the container to directly reach the host using the gateway address.
- **pasta:--mtu,1500**: Specify a 1500 bytes MTU for the *tap* interface in the container.
- **pasta:--ipv4-only,-a,10.0.2.0,-n,24,-g,10.0.2.2,--dns-forward,10.0.2.3,-m,1500,--no-ndp,--no-dhcpv6,--no-dhcp**, equivalent to default slirp4netns(1) options: disable IPv6, assign 10.0.2.0/24 to the tap0 interface in the container, with gateway 10.0.2.3, enable DNS forwarder reachable at 10.0.2.3, set MTU to 1500 bytes, disable NDP, DHCPv6 and DHCP support.
- **pasta:-I,tap0,--ipv4-only,-a,10.0.2.0,-n,24,-g,10.0.2.2,--dns-forward,10.0.2.3,--no-ndp,--no-dhcpv6,--no-dhcp**, equivalent to default slirp4netns(1) options with Podman overrides: same as above, but leave the MTU to 65520 bytes
- **pasta:-t,auto,-u,auto,-T,auto,-U,auto**: enable automatic port forwarding based on observed bound ports from both host and container sides
- **pasta:-T,5201**: enable forwarding of TCP port 5201 from container to host, using the loopback interface instead of the tap interface for improved performance

Invalid if using **--dns**, **--dns-option**, or **--dns-search** with **--network** set to **none** or **container:***id*.

If used together with **--pod**, the container joins the pod's network namespace.

--network-alias=*alias*

Add a network-scoped alias for the container, setting the alias for all networks that the container joins. To set a name only for a specific network, use the alias option as described under the **--network** option. If the network has DNS enabled (podman network inspect -f {{.DNSEnabled}} <name>), these aliases can be used for name resolution on the given network. This option can be specified multiple times. NOTE:

When using CNI a container only has access to aliases on the first network that it joins. This limitation does not exist with netavark/aardvark-dns.

--no-healthcheck

Disable any defined healthchecks for container.

--no-hosts

Do not create */etc/hosts* for the container. By default, Podman manages */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* is preserved unmodified.

This option conflicts with **--add-host**.

--oom-kill-disable

Whether to disable OOM Killer for the container or not.

This flag is not supported on cgroups V2 systems.

--oom-score-adj=*num*

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

When running in rootless mode, the specified value can't be lower than the oom_score_adj for the current process. In this case, the oom-score-adj is clamped to the current process value.

--os=*OS*

Override the OS, defaults to hosts, of the image to be pulled. For example, windows. Unless overridden, subsequent lookups of the same image in the local storage matches this OS, regardless of the host.

--passwd

Allow Podman to add entries to /etc/passwd and /etc/group when used in conjunction with the --user option. This is used to override the Podman provided user setup in favor of entrypoint configurations such as libnss-extrausers.

--passwd-entry=*ENTRY*

Customize the entry that is written to the /etc/passwd file within the container when --passwd is used.

The variables $USERNAME, $UID, $GID, $NAME, $HOME are automatically replaced with their value at runtime.

--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;
- **ns:***path*: join the specified PID namespace;
- **private**: create a new namespace for the container (default).

--pidfile=*path*

When the pidfile location is specified, the container process' PID is written to the pidfile. (This option is not available with the remote Podman client, including Mac and Windows (excluding WSL2) machines) If the pidfile option is not specified, the container process' PID is written to /run/containers/storage/${storage-driver}-containers/$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

--pids-limit=*limit*

Tune the container's pids limit. Set to **-1** to have unlimited pids for the container. The default is **2048** 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. Unless overridden, subsequent lookups of the same image in the local storage matches this platform, regardless of the host.

--pod=*name*

Run container in an existing pod. Podman makes the pod automatically if the pod name is prefixed with **new:**. To make a pod with more granular options, use the **podman pod create** command before creating a container. When a container is run with a pod with an infra-container, the infra-container is started first.

--pod-id-file=*file*

Run container in an existing pod and read the pod's ID from the specified *file*. When a container is run within a pod which has an infra-container, the infra-container starts first.

--preserve-fd=*FD1[,FD2,...]*

Pass down to the process the additional file descriptors specified in the comma separated list. It can be specified multiple times. This option is only supported with the crun OCI runtime. It might be a security risk to use this option with other OCI runtimes.

(This option is not available with the remote Podman client, including Mac and Windows (excluding WSL2) machines)

--preserve-fds=*N*

Pass down to the process N additional file descriptors (in addition to 0, 1, 2). The total FDs are 3+N. (This option is not available with the remote Podman client, including Mac and Windows (excluding WSL2) machines)

--privileged

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 limited access to devices. A "privileged" container is given the same access to devices as the user launching the container, with the exception of virtual consoles (*/dev/tty\d+*) when running in systemd mode (**--systemd=always**).

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 separation, and Seccomp filters are all disabled. Due to the disabled security features, the privileged field should almost never be set as containers can easily break out of confinement.

Containers running in a user namespace (e.g., rootless containers) cannot have more privileges than the user that launched them.

--publish, -p=*[[ip:][hostPort]:]containerPort[/protocol]*

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 is bound on all IPs on the host.

By default, Podman publishes TCP ports. To publish a UDP port instead, give udp as protocol. To publish both TCP and UDP ports, set --publish twice, with tcp, and udp as protocols respectively. Rootful containers can also publish ports using the sctp protocol.

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 is randomly assigned a port on the host.

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

Note that the network drivers macvlan and ipvlan do not support port forwarding, it will have no effect on these networks.

Note: If a container runs 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 - meaning a variety of containers in the pod and programs in the container all share a single interface, IP address, and associated 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 localhost by having one container bind to localhost in the pod, and another connect to that port.

--publish-all, -P

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

When using this option, Podman binds 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=*policy*

Pull image policy. The default is **missing**.

- **always**: Always pull the image and throw an error if the pull fails.
- **missing**: Pull the image only when the image is not in the local containers storage. Throw an error if no image is found and the pull fails.
- **never**: Never pull the image but use the one from the local containers storage. Throw an error if no image is found.
- **newer**: Pull if the image on the registry is newer than the one in the local containers storage. An image is considered to be newer when the digests are

different. Comparing the time stamps is prone to errors. Pull errors are suppressed if a local image was found.

--quiet, -q

Suppress output information when pulling images

--rdt-class=*intel-rdt-class-of-service*

Rdt-class sets the class of service (CLOS or COS) for the container to run in. Based on the Cache Allocation Technology (CAT) feature that is part of Intel's Resource Director Technology (RDT) feature set, all container processes will run within the pre-configured COS, representing a part of the cache. The COS has to be created and configured using a pseudo file system (usually mounted at /sys/fs/resctrl) that the resctrl kernel driver provides. Assigning the container to a COS requires root privileges and thus doesn't work in a rootless environment. Currently, the feature is only supported using runc as a runtime. See https://docs.kernel.org/arch/x86/resctrl.html for more details on creating a COS before a container can be assigned to it.

--read-only

Mount the container's root filesystem as read-only.

By default, container root filesystems are writable, allowing processes to write files anywhere. By specifying the **--read-only** flag, the containers root filesystem are mounted read-only prohibiting any writes.

--read-only-tmpfs

When running --read-only containers, mount a read-write tmpfs on */dev*, */dev/shm*, */run*, */tmp*, and */var/tmp*. The default is **true**.

--read-only	**--read-only-tmpfs**	**/**	**/run, /tmp, /var/tmp**
true	true	r/o	r/w
true	false	r/o	r/o
false	false	r/w	r/w
false	true	r/w	r/w

When **--read-only=true** and **--read-only-tmpfs=true** additional tmpfs are mounted on the /tmp, /run, and /var/tmp directories.

When **--read-only=true** and **--read-only-tmpfs=false** /dev and /dev/shm are marked Read/Only and no tmpfs are mounted on /tmp, /run and /var/tmp. The directories are exposed from the underlying image, meaning they are read-only by default. This makes

the container totally read-only. No writable directories exist within the container. In this mode writable directories need to be added via external volumes or mounts.

By default, when **--read-only=false**, the /dev and /dev/shm are read/write, and the /tmp, /run, and /var/tmp are read/write directories from the container image.

--replace

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 is 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 does 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
- never : Synonym for **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 status, retrying indefinitely
- unless-stopped : Identical to **always**

Podman provides a systemd unit file, podman-restart.service, which restarts containers after a system reboot.

When running containers in systemd services, use the restart functionality provided by systemd. In other words, do not use this option in a container unit, instead set the Restart= systemd directive in the [Service] section. See **podman-systemd.unit**(5) and **systemd.service**(5).

--retry=*attempts*

Number of times to retry pulling or pushing images between the registry and local storage in case of failure. Default is **3**.

--retry-delay=*duration*

Duration of delay between retry attempts when pulling or pushing images between the registry and local storage in case of failure. The default is to start at two seconds and then exponentially back off. The delay is used when this value is set, and no exponential back off occurs.

--rm

Automatically remove the container and any anonymous unnamed volume associated with the container when it exits. The default is **false**.

--rmi

After exit of the container, remove the image unless another container is using it. Implies --rm on the new container. 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.

Overlay Rootfs Mounts

The :O flag tells Podman to mount the directory from the rootfs path as storage using the overlay file system. The container processes can modify content within the mount point which is stored in the container storage in a separate directory. In overlay terms, the source directory is the lower, and the container storage directory is the upper. Modifications to the mount point are destroyed when the container finishes executing, similar to a tmpfs mount point being unmounted.

Note: On **SELinux** systems, the rootfs needs the correct label, which is by default **unconfined_u:object_r:container_file_t:s0**.

idmap

If idmap is specified, create an idmapped mount to the target user namespace in the container. The idmap option supports a custom mapping that can be different than the user namespace used by the container. The mapping can be specified after the idmap option like: idmap=uids=0-1-10#10-11-10;gids=0-100-10. For each triplet, the first value is the start of the backing file system IDs that are mapped to the second value on the host. The length of this mapping is given in the third value. Multiple ranges are separated with #.

--sdnotify=container | *conmon* | *healthy* | *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 container to receive ready notification. Podman sets 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 **healthy** option sets MAINPID to conmon's pid, and sends READY when the container has turned healthy; requires a healthcheck to be set. 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 does not use it.

--seccomp-policy=*policy*

Specify the policy to select the seccomp profile. If set to *image*, Podman looks for a "io.containers.seccomp.profile" label in the container-image config and use its value as a seccomp profile. Otherwise, Podman follows 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 is not 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 is 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 affects the secret inside the container.

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

Secret Options

- type=mount|env : How the secret is exposed to the container. mount mounts the secret into the container as a file. env exposes the secret as an environment variable. Defaults to mount.
- target=target : Target of secret. For mounted secrets, this is the path to the secret inside the container. If a fully qualified path is provided, the secret is mounted at that location. Otherwise, the secret is mounted to /run/secrets/target for linux containers or /var/run/secrets/target for freebsd containers. If the target is not

set, the secret is mounted to /run/secrets/secretname by default. For env secrets, this is the environment variable key. Defaults to secretname.

- 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.

Examples

Mount at /my/location/mysecret with UID 1:

--secret mysecret,target=/my/location/mysecret,uid=1

Mount at /run/secrets/customtarget with mode 0777:

--secret mysecret,target=customtarget,mode=0777

Create a secret environment variable called ENVSEC:

--secret mysecret,type=env,target=ENVSEC

--security-opt=*option*

Security Options

- **apparmor=unconfined** : Turn off apparmor confinement for the container
- **apparmor**=*alternate-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.

- **label=nested**: Allows SELinux modifications within the container. Containers are allowed to modify SELinux labels on files and processes, as long as SELinux policy allows. Without **nested**, containers view SELinux as disabled, even when it is enabled on the host. Containers are prevented from setting any labels.

- **mask**=*/path/1:/path/2*: The paths to mask separated by a colon. A masked path cannot 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**: JSON file to be used as a seccomp filter. Note that the io.podman.annotations.seccomp annotation is set with the specified value as shown in podman inspect.
- **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 unmasks 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**, **/sys/devices/virtual/powercap**. 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 **containers.conf**(5) file.

--shm-size=*number[unit]*

Size of */dev/shm*. A *unit* can be **b** (bytes), **k** (kibibytes), **m** (mebibytes), or **g** (gibibytes). If the unit is omitted, the system uses bytes. If the size is omitted, the default is **64m**. When *size* is **0**, there is no limit on the amount of memory used for IPC by the container. This option conflicts with **--ipc=host**.

--shm-size-systemd=*number[unit]*

Size of systemd-specific tmpfs mounts such as /run, /run/lock, /var/log/journal and /tmp. A *unit* can be **b** (bytes), **k** (kibibytes), **m** (mebibytes), or **g** (gibibytes). If the unit is omitted, the system uses bytes. If the size is omitted, the default is **64m**. When *size* is **0**, the usage is limited to 50% of the host's available memory.

--sig-proxy

Proxy received signals to the container process. SIGCHLD, SIGURG, 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 running rootless, 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 running rootless, 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 using the **--ipc=host** option, the above sysctls are not allowed.

For the network namespace, only sysctls beginning with net.* are allowed.

Note: if using the **--network=host** option, the above sysctls are not allowed.

--systemd=*true* | *false* | *always*

Run container in systemd mode. The default is **true**.

- **true** enables systemd mode only when the command executed inside the container is *systemd*, */usr/sbin/init*, */sbin/init* or */usr/local/sbin/init*.
- **false** disables systemd mode.
- **always** enforces the systemd mode to be enabled.

Running the container in systemd mode causes the following changes:

- Podman mounts tmpfs file systems on the following directories
    - */run*
    - */run/lock*
    - */tmp*
    - */sys/fs/cgroup/systemd* (on a cgroup v1 system)
    - */var/lib/journal*
- Podman sets the default stop signal to **SIGRTMIN+3**.
- Podman sets **container_uuid** environment variable in the container to the first 32 characters of the container ID.
- Podman does not mount virtual consoles (*/dev/tty\d+*) when running with **--privileged**.
- On cgroup v2, */sys/fs/cgroup* is mounted writeable.

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. Containers writing to the cgroup file system are denied by default. The **container_manage_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 default containers run until they exit or are stopped by podman stop.

--tls-verify

Require HTTPS and verify certificates when contacting registries (default: **true**). If explicitly set to **true**, TLS verification is used. If set to **false**, TLS verification is not used. If not specified, TLS verification is used unless the target registry is listed as an insecure registry in **containers-registries.conf(5)**

--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 no options are specified, the system uses the following options: **rw,noexec,nosuid,nodev**.

--tty, -t

Allocate a pseudo-TTY. The default is **false**.

When set to **true**, Podman allocates a pseudo-tty and attach to the standard input of the container. This can be used, for example, to run a throwaway interactive shell.

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 is only 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

--uidmap=*[flags]container_uid:from_uid[:amount]*

Run the container in a new user namespace using the supplied UID 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 possible values of the optional *flags* are discussed further down on this page. The *amount* value is optional and assumed to be **1** if not given.

The *from_uid* value is based upon the user running the command, either rootful or rootless users.

- rootful user: [*flags*]*container_uid*:*host_uid*[:*amount*]
- rootless user: [*flags*]*container_uid*:*intermediate_uid*[:*amount*]

Rootful mappings

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 is mapped. If for example *amount* is **4** the mapping looks like:

host UID	**container UID**
from_uid	*container_uid*
from_uid + 1	*container_uid* + 1
from_uid + 2	*container_uid* + 2
from_uid + 3	*container_uid* + 3

Rootless mappings

When **podman run** is called by an unprivileged user (i.e. running rootless), the value *from_uid* 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 Podman user	0
1st subordinate UID	1
2nd subordinate UID	2
3rd subordinate UID	3
nth subordinate UID	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 looks like:

intermediate UID	**container UID**
from_uid	*container_uid*
from_uid + 1	*container_uid* + 1
from_uid + 2	*container_uid* + 2
from_uid + 3	*container_uid* + 3
from_uid + 4	*container_uid* + 4

When running as rootless, Podman uses all the ranges configured in the */etc/subuid* file.

The current user ID is mapped to UID=0 in the rootless user namespace. Every additional range is added sequentially afterward:

host	**rootless user namespace**	**length**
$UID	0	1
1	$FIRST_RANGE_ID	$FIRST_RANGE_LENGTH
1+$FIRST_RANGE_LENGTH	$SECOND_RANGE_ID	$SECOND_RANGE_LENGTH

Referencing a host ID from the parent namespace

As a rootless user, the given host ID in **--uidmap** or **--gidmap** is mapped from the *intermediate namespace* generated by Podman. Sometimes it is desirable to refer directly at the *host namespace*. It is possible to manually do so, by running podman unshare cat /proc/self/gid_map, finding the desired host id at the second column of the output, and getting the corresponding intermediate id from the first column.

Podman can perform all that by preceding the host id in the mapping with the @ symbol. For instance, by specifying --gidmap 100000:@2000:1, podman will look up the intermediate id corresponding to host id 2000 and it will map the found intermediate id to the container id 100000. The given host id must have been subordinated (otherwise it would not be mapped into the intermediate space in the first place).

If the length is greater than one, for instance with --gidmap 100000:@2000:2, Podman will map host ids 2000 and 2001 to 100000 and 100001, respectively, regardless of how the intermediate mapping is defined.

Extending previous mappings

Some mapping modifications may be cumbersome. For instance, a user starts with a mapping such as --gidmap="0:0:65000", that needs to be changed such as the parent id 1000 is mapped to container id 100000 instead, leaving container id 1 unassigned. The corresponding --gidmap becomes --gidmap="0:0:1" --gidmap="2:2:65534" --gidmap="100000:1:1".

This notation can be simplified using the + flag, that takes care of breaking previous mappings removing any conflicting assignment with the given mapping. The flag is given before the container id as follows: --gidmap="0:0:65000" --gidmap="+100000:1:1"

Flag	Example	Description
+	+100000:1:1	Extend the previous mapping

This notation leads to gaps in the assignment, so it may be convenient to fill those gaps afterwards: --gidmap="0:0:65000" --gidmap="+100000:1:1" --gidmap="1:65001:1"

One specific use case for this flag is in the context of rootless users. A rootless user may specify mappings with the + flag as in --gidmap="+100000:1:1". Podman will then “fill the gaps” starting from zero with all the remaining intermediate ids. This is convenient when a user wants to map a specific intermediate id to a container id, leaving the rest of subordinate ids to be mapped by Podman at will.

Passing only one of --uidmap or --gidmap

Usually, subordinated user and group ids are assigned simultaneously, and for any user the subordinated user ids match the subordinated group ids. For convenience, if only one of **--uidmap** or **--gidmap** is given, podman assumes the mapping refers to both UIDs and GIDs and applies the given mapping to both. If only one value of the two needs to be changed, the mappings should include the u or the g flags to specify that they only apply to UIDs or GIDs and should not be copied over.

flag	Example	Description
u	u20000:2000:1	The mapping only applies to UIDs
g	g10000:1000:1	The mapping only applies to GIDs

For instance given the command

podman run --gidmap "0:0:1000" --gidmap "g2000:2000:1"

Since no **--uidmap** is given, the **--gidmap** is copied to **--uidmap**, giving a command equivalent to

podman run --gidmap "0:0:1000" --gidmap "2000:2000:1" --uidmap "0:0:1000"

The --gidmap "g2000:2000:1" used the g flag and therefore it was not copied to **--uidmap**.

Rootless mapping of additional host GIDs

A rootless user may desire to map a specific host group that has already been subordinated within */etc/subgid* without specifying the rest of the mapping.

This can be done with **--gidmap "+g*container_gid*:@*host_gid*"**

Where:

- The host GID is given through the @ symbol
- The mapping of this GID is not copied over to **--usermap** thanks to the g flag.
- The rest of the container IDs will be mapped starting from 0 to n, with all the remaining subordinated GIDs, thanks to the + flag.

For instance, if a user belongs to the group 2000 and that group is subordinated to that user (with usermod --add-subgids 2000-2000 $USER), the user can map the group into the container with: **--gidmap=+g100000:@2000**.

If this mapping is combined with the option, **--group-add=keep-groups**, the process in the container will belong to group 100000, and files belonging to group 2000 in the host will appear as being owned by group 100000 inside the container.

podman run --group-add=keep-groups --gidmap="+g100000:@2000" ...

No subordinate UIDs

Even if a user does not have any subordinate UIDs in */etc/subuid*, **--uidmap** can 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 ....

Pods

The **--uidmap** option cannot be called in conjunction with the **--pod** option as a uidmap cannot be set on the container level when in a pod.

--ulimit=*option*

Ulimit options. Sets the ulimits values inside of the container.

--ulimit with a soft and hard limit in the format =[:]. For example:

$ podman run --ulimit nofile=1024:1024 --rm ubi9 ulimit -n 1024

Set -1 for the soft or hard limit to set the limit to the maximum limit of the current process. In rootful mode this is often unlimited.

Use **host** to copy the current configuration from the host.

Don't use nproc with the ulimit flag as Linux uses nproc to set the maximum number of processes available to a user, not to a container.

Use the --pids-limit option to modify the cgroup control to limit the number of processes within a container.

--umask=*umask*

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

--unsetenv=*env*

Unset default environment variables for the container. Default environment variables include variables provided natively by Podman, environment variables configured by the image, and environment variables from containers.conf.

--unsetenv-all

Unset all default environment variables for the container. Default environment variables include variables provided natively by Podman, environment variables configured by the image, and environment variables from containers.conf.

--user, -u=*user[:group]*

Sets the username or UID used and, optionally, the groupname or GID for the specified command. Both *user* and *group* may be symbolic or numeric.

Without this argument, the command runs as the user specified in the container image. Unless overridden by a USER command in the Containerfile or by a value passed to this option, 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 match. When user namespaces are in use, however, the UID and GID in the container may correspond to another UID and GID on the host. In rootless containers, for example, a user namespace is always used, and root in the container by default corresponds to the UID and GID of the user invoking Podman.

--userns=*mode*

Set the user namespace mode for the container.

If --userns is not set, the default value is determined as follows.

- If --pod is set, --userns is ignored and the user namespace of the pod is used.
- If the environment variable **PODMAN_USERNS** is set its value is used.

- If userns is specified in containers.conf this value is used.
- Otherwise, --userns=host is assumed.

--userns="" (i.e., an empty string) is an alias for --userns=host.

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

Rootless user --userns=Key mappings:

Key	Host User	Container User
auto	$UID	nil (Host User UID is not mapped into container.)
host	$UID	0 (Default User account mapped to root user in container.)
keep-id	$UID	$UID (Map user account to same UID within container.)
keep-id:uid=200,gid=210	$UID	200:210 (Map user account to specified UID, GID value within container.)
nomap	$UID	nil (Host User UID is not mapped into container.)

Valid *mode* values are:

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

- rootful mode: The --userns=auto flag requires that the user name **containers** be specified in the /etc/subuid and /etc/subgid files, with an unused range of subordinate user IDs that Podman containers are allowed to allocate.
- Example: `containers:2147483647:2147483648`.
- rootless mode: The users range from the /etc/subuid and /etc/subgid files will be used. Note running a single container without using --userns=auto will use the entire range of UIDs and not allow further subdividing. See subuid(5).

Podman allocates unique ranges of UIDs and GIDs from the containers subordinate 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 option --userns=keep-id uses all the subuids and subgids of the user. The option --userns=nomap uses all the subuids and subgids of the user except the user's own ID. Using --userns=auto when starting new containers does not work as long as any containers exist that were started with --userns=keep-id or --userns=nomap.

Valid auto options:

- *gidmapping=CONTAINER_GID:HOST_GID: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 estimates a size for the user namespace.
- *uidmapping=CONTAINER_UID:HOST_UID:SIZE*: to force a UID mapping to be present in the user namespace.

The host UID and GID in *gidmapping* and *uidmapping* can optionally be prefixed with the @ symbol. In this case, podman will look up the intermediate ID corresponding to host ID and it will map the found intermediate ID to the container id. For details see **--uidmap**.

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

host or **“”** (empty string): run in the user namespace of the caller. The processes running in the container have the same privileges on the host as any other process launched by the calling user.

keep-id: creates a user namespace where the current user’s UID:GID are mapped to the same values in the container. For containers created by root, the current mapping is created into a new user namespace.

Valid keep-id options:

- *uid*=UID: override the UID inside the container that is used to map the current user to.
- *gid*=GID: override the GID inside the container that is used to map the current user to.

nomap: creates a user namespace where the current rootless user’s UID:GID are not mapped into the container. This option is not allowed for containers created by the root user.

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

--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 -v /HOST-DIR:/CONTAINER-DIR is specified, Podman bind mounts /HOST-DIR from the host into /CONTAINER-DIR in the Podman container. Similarly, -v SOURCE-VOLUME:/CONTAINER-DIR mounts the named volume from the host into the container. If no such named volume exists, Podman creates one. If no source is given, the volume is created as an anonymously named volume with a randomly generated name, and is removed when the container is removed via the --rm flag or the podman rm --volumes command.

(Note when using the remote client, including Mac and Windows (excluding WSL2) machines, the volumes are mounted from the remote server, not necessarily the client machine.)

The *OPTIONS* is a comma-separated list and can be one or more of:

- **rw**|**ro**
- **z**|**Z**
- [**O**]
- [**U**]
- [**no**]**copy**
- [**no**]**dev**
- [**no**]**exec**
- [**no**]**suid**
- [**r**]**bind**
- [**r**]**shared**|[**r**]**slave**|[**r**]**private**[**r**]**unbindable** [1]
- **idmap**[=**options**]

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

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 returns an error. Users must pre-create the source files or directories.

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

Specify multiple **-v** options to mount one or more volumes into a container.

Write Protected Volume Mounts

Add **:ro** or **:rw** option to mount a volume in read-only or read-write mode, respectively. By default, the volumes are mounted read-write. See examples.

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. Chowning walks the file system under the volume and changes the UID/GID on each file. If the volume has thousands of inodes, this process takes a long time, delaying the start of the container.

Warning use with caution since this modifies 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, add either of two suffixes **:z** or **:Z** to the volume mount. These suffixes tell Podman to relabel file objects on the shared volumes. The **z** option tells Podman that two or more 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. The **Z** option tells Podman to label the content with a private unshared label Only the current container can use a private volume. Relabeling walks the file system under the volume and changes the label on each file, if the volume has thousands of inodes, this process takes a long time, delaying the start of the container. If the volume was previously relabeled with the z option, Podman is optimized to not relabel a second time. If files are moved into the volume, then the labels can be manually change with the chcon -Rt container_file_t PATH command.

Note: Do not relabel system files and directories. Relabeling system content might cause other confined services on the machine to fail. For these types of containers we recommend disabling 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 is the lower, and the container storage directory is the upper. Modifications to the mount point are destroyed when the container finishes executing, similar to a tmpfs mount point being unmounted.

For advanced users, the **overlay** option also supports custom non-volatile **upperdir** and **workdir** for the overlay mount. Custom **upperdir** and **workdir** can be fully managed by the users themselves, and Podman does not remove it on lifecycle completion. Example **:O,upperdir=/some/upper,workdir=/some/work**

Subsequent executions of the container sees 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 container 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 container label. Usually containers can read/execute container_share_t and can read/write container_file_t. If unable to change the labels on a source volume, SELinux container separation must be disabled for the container to work.

Do not modify the source directory mounted into the container with an overlay mount, it can cause unexpected failures. Only modify the directory after the container finishes running.

Mounts propagation

By default, bind-mounted volumes are private. That means any mounts done inside the container are not visible on the host and vice versa. One can change this behavior by specifying a volume mount propagation property. When a volume is shared, mounts

done under that volume inside the container are visible on host and vice versa. Making a volume **slave**[1] enables only one-way mount propagation: mounts done on the host under that volume are visible inside the container but not the other way around.

To control mount propagation property of a volume one can use the [**r**]**shared**, [**r**]**slave**, [**r**]**private** or the [**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 the source mount point (the mount point where source dir is mounted on) has to have the right propagation properties. For shared volumes, the source mount point has to be shared. And for slave volumes, the source mount point has to be either shared or slave. [1]

To recursively mount a volume and all of its submounts into a container, use the **rbind** option. By default the bind option is used, and submounts of the source directory is not mounted into the container.

Mounting the volume with a **copy** option tells podman to copy content from the underlying destination directory onto newly created internal volumes. The **copy** only happens on the initial creation of the volume. Content is not copied up when the volume is subsequently used on different containers. The **copy** option is ignored on bind mounts and has no effect.

Mounting volumes with the **nosuid** options means that SUID executables on the volume can not be used by applications 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 can be executed within the container.

Mounting the volume with the **nodev** option means that no devices on the volume can 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 kernel.

Use **df HOST-DIR** to figure out the source mount, then use **findmnt -o TARGET,PROPAGATION *source-mount-dir*** to figure out propagation properties of source mount. If **findmnt**(1) utility is not available, then one can look at the mount entry for the 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]

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 converts /foo into a shared mount point. Alternatively, one can directly change propagation properties of

source mount. Say */* 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 fails.

Idmapped mount

If idmap is specified, create an idmapped mount to the target user namespace in the container. The idmap option supports a custom mapping that can be different than the user namespace used by the container. The mapping can be specified after the idmap option like: idmap=uids=0-1-10#10-11-10;gids=0-100-10. For each triplet, the first value is the start of the backing file system IDs that are mapped to the second value on the host. The length of this mapping is given in the third value. Multiple ranges are separated with #.

Use the **--group-add keep-groups** option to pass the user's supplementary 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. *CONTAINER* may be a name or ID. To share a volume, use the --volumes-from option when running the target container. Volumes can be shared 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. This can be changed 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, add z to the volume mount. This suffix tells Podman to relabel file objects on the shared volumes. The z option tells Podman that two entities 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 directory (**/**). The image developer can set a different default with the WORKDIR instruction. The operator can override the working directory by using the **-w** option.

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 **chroot**(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 using the **--read-only** switch. This protects the container's image from modification. By default read-only containers can write to temporary data. Podman mounts a tmpfs on */run* and */tmp* within the container.

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

If the container does not write to any file system within the container, including tmpfs, set --read-only-tmpfs=false.

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

Exposing shared libraries inside of container as read-only using a glob

$ podman run --mount type=glob,src=/usr/lib64/libnvidia*,ro=true -i -t fedora /bin/bash

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

Bind mount the */dev/log* directory to have messages that are logged in the container show up in the host's syslog/journal.

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

From inside the container 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 lists the message sent to the logger.

Attaching to one or more from STDIN, STDOUT, STDERR

Without specifying the **-a** option, Podman attaches everything (stdin, stdout, stderr). Override the default by specifying -a (stdin, stdout, stderr), 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 example, 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 directory 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 is considered an orphan and wiped by the **podman volume prune** command:

$ 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

If the container needs a writeable mounted volume by a non root user inside the container, use the **U** option. This option tells Podman to chown the source volume to match the default UID and GID used within the container.

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

Alternatively if the container needs a writable volume by a non root user inside of the container, the --userns=keep-id option allows users to specify the UID and GID of the user executing Podman to specific UIDs and GIDs within the container. Since the processes running in the container run as the user's UID, they can read/write files owned by the user.

$ podman run -d -e MARIADB_ROOT_PASSWORD=root --user mysql --userns=keep-id:uid=999,gid=999 -v ~/data:/var/lib/mysql:Z 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 directory:

$ 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 occurs, and an **avc:** message is added to 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 directory:

$ chcon -Rt svirt_sandbox_file_t /var/db

Now, writing to the */data1* volume in the container is allowed and the changes are reflected on the host in */var/db*.

Using alternative security labeling

Override the default labeling scheme for each container by specifying the **--security-opt** flag. For example, specify the MCS/MLS level, a requirement for MLS systems. Specifying the level in the following command allows the same content to be shared 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

Tighten the security policy on the processes within a container by specifying an alternate type for the container. For example, 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 that an SELinux policy defining a **svirt_apache_t** type must be written.

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 via --blkio-weight-device flag.

$ 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/container/storage.conf* or by using global options. This shows how to set up and use fuse-overlayfs 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 container2 starts. The podman run command starts 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 --personality=LINUX32 fedora bash

Run a container with external rootfs mounted as an overlay

$ podman run --name container1 --rootfs /path/to/rootfs:O bash

Handling Timezones in java applications in a container.

In order to use a timezone other than UTC when running a Java application within a container, the TZ environment variable must be set within the container. Java applications ignores the value set with the --tz option.

Example run

podman run -ti --rm -e TZ=EST mytzimage

lrwxrwxrwx. 1 root root 29 Nov 3 08:51 /etc/localtime -> ../usr/share/zoneinfo/Etc/UTC

Now with default timezone:

Fri Nov 19 18:10:55 EST 2021

Java default sees the following timezone:

2021-11-19T18:10:55.651130-05:00

Forcing UTC:

Fri Nov 19 23:10:55 UTC 2021

Run a container connected to two networks (called net1 and net2) with a static ip

$ podman run --network net1:ip=10.89.1.5 --network net2:ip=10.89.10.10 alpine ip addr

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.

In order for users to run rootless, there must be an entry for their username in */etc/subuid* 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 can be disk space expensive and less performant than other drivers.

To enable VPN on the container, slirp4netns or pasta needs to be specified; without either, 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 are 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 are specified, then they override each other in order of entry.
- **--env**: Any environment variables specified overrides previous settings.

Run containers and set the environment ending with a *****. The trailing ***** glob functionality is only active when no value is specified:

$ export ENV1=a

$ podman run --env 'ENV*' alpine env | grep ENV

ENV1=a

$ podman run --env 'ENV*=b' alpine env | grep ENV

ENV*=b

CONMON

When Podman starts a container it actually executes the conmon program, which then executes 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 about conmon, see 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)**, **podman-rm(1)**, **subgid(5)**, **subuid(5)**, **containers.conf(5)**, **systemd.unit(5)**, **setsebool(8)**, **slirp4netns(1)**, **pasta(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 needs to 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 follow suit immediately.