Rocky Enterprise Linux 9.2 Manual Pages on command 'podman-pod-clone.1'

$ man podman-pod-clone.1

podman-pod-clone(1) General Commands Manual podman-pod-clone(1)

NAME

podman-pod-clone - Creates a copy of an existing pod

SYNOPSIS

podman pod clone [options] pod name

DESCRIPTION

podman pod clone creates a copy of a pod, recreating the identical con?

fig for the pod and for all of its containers. Users can modify the

pods new name and select pod details within the infra container

OPTIONS

--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.

--cgroup-parent=path

Path to cgroups under which the cgroup for the pod will be created. If

the path is not absolute, the path is considered to be relative to the

cgroups path of the init process. Cgroups will be created if they do not already exist.

--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 contain? ers. Default weight is 1024.

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

For example, consider three containers, one has a cpu-share of 1024 and two others have a cpu-share setting of 512. When processes in all three containers attempt to use 100% of CPU, the first container would re? ceive 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 con? tainer C0 is started with --cpu-shares=512 running one process, and an? other 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

Set a number of CPUs for the pod that overrides the original pods CPU limits. If none are specified, the original pod's Nano CPUs are used.

--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.

If none are specified, the original pod's CPUset is used.

--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 will only use 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.

--destroy

Remove the original pod that we are cloning once used to mimic the configuration.

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

Add a host device to the pod. 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 will be resolved first.

The pod will only store the major and minor numbers of the host device.

Podman may load kernel modules required for using the specified device.

The devices that Podman will load 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. Be? cause 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: the pod implements devices by storing the initial configuration passed by the user and recreating the device on each container added to the pod.

--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/contain? ers/podman/blob/main/troubleshooting.md#26-running-containers-with-re? source-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/contain?

ers/podman/blob/main/troubleshooting.md#26-running-containers-with-re?
source-limits-fails-with-a-permissions-error

This option is not supported on cgroups V1 rootless systems.

--gidmap=pod_gid:host_gid:amount

GID map for the user namespace. Using this flag will run all containers in the pod with user namespace enabled. It conflicts with the --userns and --subgidname flags.

--help, -h

Print usage statement.

--hostname=name

Set a hostname to the pod.

--infra-command=command

The command that will be run to start the infra container. Default: "/pause".

--infra-conmon-pidfile=file

Write the pid of the infra container's conmon process to a file. As conmon runs in a separate process than Podman, this is necessary when using systemd to manage Podman containers and pods.

--infra-name=name

The name that will be used for the pod's infra container.

--label, -l=key=value

Add metadata to a pod.

--label-file=file

Read in a line-delimited file of labels.

--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 would be very large, that's millions of trillions).

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 should always be larger than that of

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

Set number to -1 to enable unlimited swap.

This option is not supported on cgroups V1 rootless systems.

--name, -n

Set a custom name for the cloned pod. The default if not specified is of the syntax: -clone

--pid=pid

Set the PID mode for the pod. The default is to create a private PID namespace for the pod. Requires the PID namespace to be shared via --share.

host: use the host?s PID namespace for the pod

ns: join the specified PID namespace

private: create a new namespace for the pod (default)

--security-opt=option

Security Options

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

? apparmor=alternate-profile : Set the apparmor confinement pro? file for the pod

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

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

? label=type:TYPE: Set the label process type for the pod pro? cesses

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

? label=filetype:TYPE: Set the label file type for the pod files

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

Note: Labeling can be disabled for all pods/containers by setting la?

bel=false in the containers.conf (/etc/containers/containers.conf or $HOME/.config/containers/containers.conf) file.

? mask=/path/1:/path/2: The paths to mask separated by a colon. A masked path cannot be accessed inside the containers within the pod.

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

? seccomp=unconfined: Turn off seccomp confinement for the pod.

? seccomp=profile.json: JSON file to be used as a seccomp fil? ter. 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 will unmask all the paths that are masked or made read-only by default. The default masked paths are /proc/acpi, /proc/kcore, /proc/keys, /proc/latency_stats, /proc/sched_debug, /proc/scsi, /proc/timer_list, /proc/timer_stats, /sys/firmware, and /sys/fs/selinux. The default paths that are read-only are /proc/asound, /proc/bus, /proc/fs, /proc/irq, /proc/sys, /proc/sysrq-trigger, /sys/fs/cgroup.

Note: Labeling can be disabled for all containers by setting la? bel=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 pod. This option conflicts with --ipc=host.

--start

When set to true, this flag starts the newly created pod after the clone process has completed. All containers within the pod are started.

--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 for all containers in the pod.

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 the ipc namespace is not shared within the pod, the above sysctls are not allowed.

For the network namespace, only sysctls beginning with net.* are al? lowed.

Note: if the network namespace is not shared within the pod, the above sysctls are not allowed.

--uidmap=container_uid:from_uid:amount

Run all containers in the pod in a new user namespace using the sup? plied 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.

--userns=mode

Set the user namespace mode for all the containers in a pod. It de?

faults to the PODMAN_USERNS environment variable. An empty value ("")

means user namespaces are disabled.

Rootless user --userns=Key mappings:

??

?Key ? Host User ? Container User ?

??

?"" ? $UID ? 0 (Default User ac? ?

? ? ? count mapped to ?

? ? ? root user in con? ?

? ? ? tainer.) ?

??

?keep-id ? $UID ? $UID (Map user ac? ?

? ? ? count to same UID ?

? ? ? within container.) ?

??

?auto ? $UID ? nil (Host User UID ?

? ? ? is not mapped into ?

? ? ? container.) ?

??

?nomap ? $UID ? nil (Host User UID ?

? ? ? is not mapped into ?

? ? ? container.) ?

??

Valid mode values are:

? auto[:OPTIONS,...]: automatically create a namespace. It is

possible to specify these options to auto:

? 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 speci?
fied, auto will estimate a size for the user namespace.

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

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

? keep-id: creates a user namespace where the current rootless user's UID:GID are mapped to the same values in the container. This option is not allowed for containers created by the root user.

? 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.

--uts=mode

Set the UTS namespace mode for the pod. The following values are sup?
ported:

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

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

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

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

Create a bind mount. If -v /HOST-DIR:/CONTAINER-DIR is specified, Pod?
man bind mounts /HOST-DIR from the host into /CONTAINER-DIR in the Pod?
man container. Similarly, -v SOURCE-VOLUME:/CONTAINER-DIR will mount the named volume from the host into the container. If no such named volume exists, Podman will create one. If no source is given, the vol?
ume will be created as an anonymously named volume with a randomly gen?
erated name, and will be removed when the pod is removed via the --rm flag or the podman rm --volumes command.

(Note when using the remote client, including Mac and Windows (exclud?
ing WSL2) machines, the volumes will be mounted from the remote server, not necessarily the client machine.)

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

? rw|ro

? z|Z

? [O]

? [U]

? [no]copy

? [no]dev

? [no]exec

? [no]suid

? [r]bind

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

? idmap[=options]

The CONTAINER-DIR must be an absolute path such as /src/docs. The vol?
ume will be 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 rela?
tive; relative paths are resolved relative to the directory Podman is
run in. If the source does not exist, Podman will return an error.
Users must pre-create the source files or directories.

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

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

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 pod 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 pod, 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, it the volume has thou? sands of inodes, this process will take a long time, delaying the start of the pod.

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

Labeling Volume Mounts

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

To change a label in the pod context, add either of two suffixes :z or :Z to the volume mount. These suffixes tell Podman to relabel file ob? jects on the shared volumes. The z option tells Podman that two or more pods 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 pod can use a private volume. Relabeling walks the file system under the volume and changes the label on each file, it the volume has thousands of inodes, this process will take a long time, delaying the start of the pod. 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 -R 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 separa? tion. The option --security-opt label=disable disables SELinux separa? tion for the pod. For example if a user wanted to volume mount their entire home directory into a pod, they need to disable SELinux separa?

tion.

$ podman pod clone --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 tem? porary storage using the overlay file system. The pod processes can modify content within the mountpoint which is stored in the container storage in a separate directory. In overlay terms, the source directory will be the lower, and the container storage directory will be the up? per. Modifications to the mount point are destroyed when the pod fin? ishes 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 will not remove it on lifecycle completion. Example :O,up? perdir=/some/upper,workdir=/some/work

Subsequent executions of the container will see the original source di? rectory content, any changes from previous pod executions no longer ex? ist.

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 read? able by the pod infra container label. Usually containers can read/exe? cute 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 pod or infra container to work.

- The source directory mounted into the pod with an overlay mount should not be modified, it can cause unexpected failures. It is recom? mended to not modify the directory until the container finishes run? ning.

Mounts propagation

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

To control mount propagation property of 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 vol? umes, the source mount point has to be either shared or slave. [1] ?#Footnote1?

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

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 dif? ferent containers. The copy option is ignored on bind mounts and has no effect.

Mounting the volume with the nosuid options means that SUID applica? tions on the volume will not be able to change their privilege. By de? fault volumes are mounted with nosuid.

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

Mounting the volume with the nodev option means that no devices on the volume will be able to be used by processes within the pod. 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] ?#Footnote1?

To change propagation properties of a mount point, use mount(8) command. For example, if one wants to bind mount source directory /foo, one can do mount --bind /foo /foo and mount --make-private --make-shared /foo. This will convert /foo into a shared mount point. 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 pod will fail.

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 #.

--volumes-from=CONTAINER[:OPTIONS]

Mount volumes from the specified container(s). Used to share volumes between containers and pods. 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 pod. 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 pod. 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 pod 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 pod, then the volume hides that data on the target.

EXAMPLES

podman pod clone pod-name

6b2c73ff8a1982828c9ae2092954bcd59836a131960f7e05221af9df5939c584

podman pod clone --name=cloned-pod

d0cf1f782e2ed67e8c0050ff92df865a039186237a4df24d7acba5b1fa8cc6e7

6b2c73ff8a1982828c9ae2092954bcd59836a131960f7e05221af9df5939c584

podman pod clone --destroy --cpus=5

d0cf1f782e2ed67e8c0050ff92df865a039186237a4df24d7acba5b1fa8cc6e7

6b2c73ff8a1982828c9ae2092954bcd59836a131960f7e05221af9df5939c584

podman pod clone 2d4d4fca7219b4437e0d74fcdc272c4f031426a6eacd207372691207079551de new_name

5a9b7851013d326aa4ac4565726765901b3ecc01fcbc0f237bc7fd95588a24f9

SEE ALSO

HISTORY

May 2022, Originally written by Charlie Doern cdoern@redhat.com ?mailto:cdoern@redhat.com?

FOOTNOTES

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