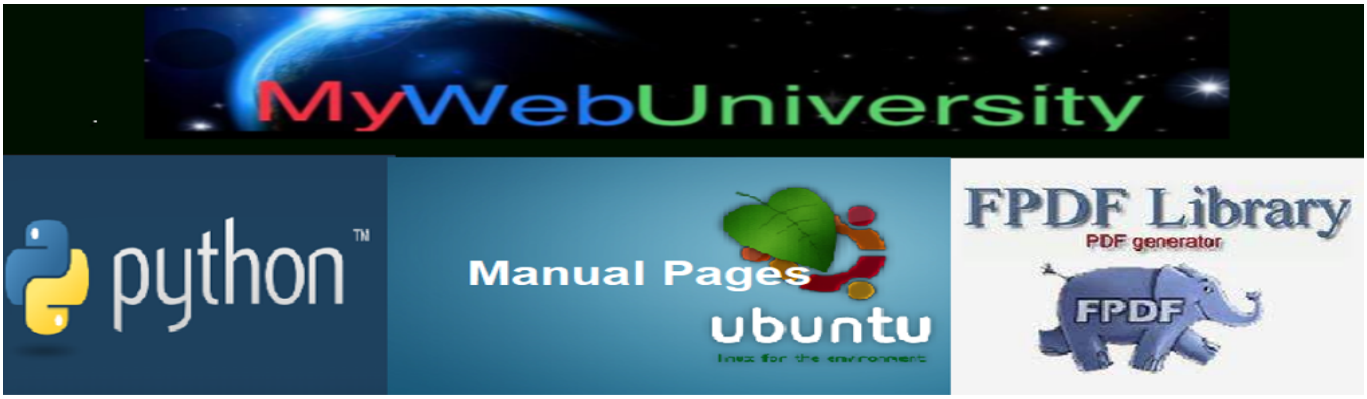

Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

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

$ man podman-build.1

podman-build(1)() podman-build(1)()

NAME

podman-build - Build a container image using a Containerfile

SYNOPSIS

podman build [options] [context]

podman image build [options] [context]

DESCRIPTION

podman build Builds an image using instructions from one or more Containerfiles or Docker? files and a specified build context directory. A Containerfile uses the same syntax as a Dockerfile internally. For this document, a file referred to as a Containerfile can be a file named either 'Containerfile' or 'Dockerfile'.

The build context directory can be specified as the http(s) URL of an archive, git reposi? tory or Containerfile.

If no context directory is specified, then Podman will assume the current working direc? tory as the build context, which should contain the Containerfile.

Containerfiles ending with a ".in" suffix will be preprocessed via CPP(1). This can be useful to decompose Containerfiles into several reusable parts that can be used via CPP's #include directive. Notice, a Containerfile.in file can still be used by other tools when manually preprocessing them via cpp -E.

When the URL is an archive, the contents of the URL is downloaded to a temporary location and extracted before execution.

When the URL is an Containerfile, the Containerfile is downloaded to a temporary location.

When a Git repository is set as the URL, the repository is cloned locally and then set as

the context.

NOTE: podman build uses code sourced from the buildah project to build container images. This buildah code creates buildah containers for the RUN options in container storage. In certain situations, when the podman build crashes or users kill the podman build process, these external containers can be left in container storage. Use the podman ps --all --storage command to see these containers. External containers can be removed with the podman rm --storage command.

podman buildx build command is an alias of podman build. Not all buildx build features are available in Podman. The buildx build option is provided for scripting compatibility.

OPTIONS

--add-host=host

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 mul? tiple times.

--annotation=annotation

Add an image annotation (e.g. annotation=value) to the image metadata. Can be used multi? ple times.

Note: this information is not present in Docker image formats, so it is discarded when writing images in Docker formats.

--arch=arch

Set the architecture of the image to be built, and that of the base image to be pulled, if the build uses one, to the provided value instead of using the architecture of the build host. (Examples: arm, arm64, 386, amd64, ppc64le, s390x)

--authfile=path

Path of the authentication file. Default is ${XDG_RUNTIME_DIR}/containers/auth.json, which is set using podman login. If the authorization state is not found there, $HOME/.docker/config.json is checked, which is set using docker login.

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

--build-arg=arg=value

Specifies a build argument and its value, which will be interpolated in instructions read from the Containerfiles in the same way that environment variables are, but which will not be added to environment variable list in the resulting image's configuration.

--cache-from

Images to utilize as potential cache sources. Podman does not currently support caching so this is a NOOP. (This option is not available with the remote Podman client)

--cap-add=CAP_xxx

When executing RUN instructions, run the command specified in the instruction with the specified capability added to its capability set. Certain capabilities are granted by de? fault; this option can be used to add more.

--cap-drop=CAP_xxx

When executing RUN instructions, run the command specified in the instruction with the specified capability removed from its capability set. The CAP_AUDIT_WRITE, CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID, CAP_KILL, CAP_MKNOD, CAP_NET_BIND_SERVICE, CAP_SETFCAP, CAP_SETGID, CAP_SETPCAP, CAP_SETUID, and CAP_SYS_CHROOT capabilities are granted by default; this option can be used to remove them.

If a capability is specified to both the --cap-add and --cap-drop options, it will be dropped, regardless of the order in which the options were given.

--cert-dir=path

Use certificates at path (*.crt, *.cert, *.key) to connect to the registry. (Default: /etc/containers/certs.d) Please refer to containers-certs.d(5) for details. (This option is not available with the remote Podman client)

--cgroup-parent=path

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

--compress

This option is added to be aligned with other containers CLIs. Podman doesn't communicate with a daemon or a remote server. Thus, compressing the data before sending it is irrele? vant to Podman. (This option is not available with the remote Podman client)

--cni-config-dir=directory

Location of CNI configuration files which will dictate which plugins will be used to con? figure network interfaces and routing for containers created for handling RUN instruc? tions, if those containers will be run in their own network namespaces, and networking is not disabled.

--cni-plugin-path=directory[:directory[:directory[...]]]

List of directories in which the CNI plugins which will be used for configuring network namespaces can be found.

--cpu-period=limit

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

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

--cpu-quota=limit

Limit the CPU Completely Fair Scheduler (CFS) quota.

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

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

--cpu-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 weighting of all other running containers.

To modify the proportion from the default of 1024, use the --cpu-shares flag to set the weighting to 2 or higher.

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

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

On a multi-core system, the shares of CPU time are distributed over all CPU cores. Even if

a container is limited to less than 100% of CPU time, it can use 100% of each individual CPU core.

For example, consider a system with more than three cores. If you start one container {C0} with -c=512 running one process, and another container {C1} with -c=1024 running two pro? cesses, 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

--cpuset-cpus=num

CPUs in which to allow execution (0-3, 0,1)

--cpuset-mems=nodes

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

If you have four memory nodes on your system (0-3), use --cpuset-mems=0,1 then processes in your container will only use memory from the first two memory nodes.

--creds=creds

The [username[:password]] to use to authenticate with the registry if required. If one or both values are not supplied, a command line prompt will appear and the value can be en? tered. The password is entered without echo.

--decryption-key=key[:passphrase]

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

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

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

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

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

Note: if the user only has access rights via a group, accessing the device from inside a rootless container will fail. The crun(1) runtime offers a workaround for this by adding the option

--annotation run.oci.keep_original_groups=1.

--disable-compression, -D

Don't compress filesystem layers when building the image unless it is required by the lo? cation where the image is being written. This is the default setting, because image lay? ers are compressed automatically when they are pushed to registries, and images being written to local storage would only need to be decompressed again to be stored. Compres? sion can be forced in all cases by specifying --disable-compression=false.

--disable-content-trust

This is a Docker specific option to disable image verification to a Docker registry and is not supported by Podman. This flag is a NOOP and provided solely for scripting compati? bility. (This option is not available with the remote Podman client)

--dns=dns

Set custom DNS servers to be used during the build.

This option can be used to override the DNS configuration passed to the container. Typi? cally this is necessary when the host DNS configuration is invalid for the container (e.g., 127.0.0.1). When this is the case the --dns 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 will be used without changes.

--dns-option=option

Set custom DNS options to be used during the build.

--dns-search=domain

Set custom DNS search domains to be used during the build.

--file, -f=Containerfile

Specifies a Containerfile which contains instructions for building the image, either a lo? cal file or an http or https URL. If more than one Containerfile is specified, FROM in? structions will only be accepted from the first specified file.

If a build context is not specified, and at least one Containerfile is a local file, the directory in which it resides will be used as the build context.

If you specify -f -, the Containerfile contents will be read from stdin.

--force-rm=true|false

Always remove intermediate containers after a build, even if the build fails (default true).

--format

Control the format for the built image's manifest and configuration data. Recognized for? mats include oci (OCI image-spec v1.0, the default) and docker (version 2, using schema format 2 for the manifest).

Note: You can also override the default format by setting the BUILDAH_FORMAT environment variable. export BUILDAH_FORMAT=docker

--from

Overrides the first FROM instruction within the Containerfile. If there are multiple FROM instructions in a Containerfile, only the first is changed.

-h, --help

Print usage statement

--http-proxy

Pass through HTTP Proxy environment variables.

--iidfile=ImageIDfile

Write the built image's ID to the file. When --platform is specified more than once, at? tempting to use this option will trigger an error.

--ignorefile

Path to an alternative .dockerignore file.

--ipc=how

Sets the configuration for IPC namespaces when handling RUN instructions. The configured value can be "" (the empty string) or "container" to indicate that a new IPC namespace should be created, or it can be "host" to indicate that the IPC namespace in which podman itself is being run should be reused, or it can be the path to an IPC namespace which is already in use by another process.

--isolation=type

Controls what type of isolation is used for running processes as part of RUN instructions. Recognized types include oci (OCI-compatible runtime, the default), rootless (OCI-compati? ble runtime invoked using a modified configuration and its --rootless flag enabled, with --no-new-keyring --no-pivot added to its create invocation, with network and UTS name? spaces disabled, and IPC, PID, and user namespaces enabled; the default for unprivileged users), and chroot (an internal wrapper that leans more toward chroot(1) than container technology).

Note: You can also override the default isolation type by setting the BUILDAH_ISOLATION environment variable. export BUILDAH_ISOLATION=oci

--jobs=number

Run up to N concurrent stages in parallel. If the number of jobs is greater than 1, stdin will be read from /dev/null. If 0 is specified, then there is no limit in the number of jobs that run in parallel.

--label=label

Add an image label (e.g. label=value) to the image metadata. Can be used multiple times.

Users can set a special LABEL io.containers.capabilities=CAP1,CAP2,CAP3 in a Containerfile that specified the list of Linux capabilities required for the container to run properly.

This label specified in a container image tells Podman to run the container with just these capabilities. Podman launches the container with just the specified capabilities, as long as this list of capabilities is a subset of the default list.

If the specified capabilities are not in the default set, Podman will print an error mes? sage and will run the container with the default capabilities.

--layers

Cache intermediate images during the build process (Default is true).

Note: You can also override the default value of layers by setting the BUILDAH_LAYERS en? vironment variable. export BUILDAH_LAYERS=true

--logfile=filename

Log output which would be sent to standard output and standard error to the specified file instead of to standard output and standard error.

--manifest manifest

Name of the manifest list to which the image will be added. Creates the manifest list if it does not exist. This option is useful for building multi architecture images.

--memory, -m=LIMIT

Memory limit (format: <number>[<unit>], where unit = b (bytes), k (kilobytes), m (megabytes), or g (gigabytes))

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

--memory-swap=LIMIT

A limit value equal to memory plus swap. Must be used with the -m (--memory) flag. The

swap LIMIT should always be larger than -m (--memory) value. By default, the swap LIMIT will be set to double the value of --memory.

The format of LIMIT is <number>[<unit>]. Unit can be b (bytes), k (kilobytes), m (megabytes), or g (gigabytes). If you don't specify a unit, b is used. Set LIMIT to -1 to enable unlimited swap.

--network=mode, --net

Sets the configuration for network namespaces when handling RUN instructions.

Valid mode values are:

- ? none: no networking.
- ? host: use the Podman host network stack. 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 (default).

--no-cache

Do not use existing cached images for the container build. Build from the start with a new set of cached layers.

--os=string

Set the OS of the image to be built, and that of the base image to be pulled, if the build uses one, instead of using the current operating system of the build host.

--pid=pid

Sets the configuration for PID namespaces when handling RUN instructions. The configured value can be "" (the empty string) or "container" to indicate that a new PID namespace should be created, or it can be "host" to indicate that the PID namespace in which podman itself is being run should be reused, or it can be the path to a PID namespace which is already in use by another process.

--platform="OS/ARCH[/VARIANT][,...]"

Set the OS/ARCH of the built image (and its base image, if your build uses one) to the provided value instead of using the current operating system and architecture of the host (for example linux/arm). If --platform is set, then the values of the --arch, --os, and --variant options will be overridden.

The --platform flag can be specified more than once, or given a comma-separated list of values as its argument. When more than one platform is specified, the --manifest option

should be used instead of the --tag option.

OS/ARCH pairs are those used by the Go Programming Language. In several cases the ARCH value for a platform differs from one produced by other tools such as the arch command. Valid OS and architecture name combinations are listed as values for $GOOS and $GOARCH at https://golang.org/doc/install/source#environment, and can also be found by running go tool dist list.

While podman build is happy to use base images and build images for any platform that ex?ists, RUN instructions will not be able to succeed without the help of emulation provided by packages like qemu-user-static.

--pull

When the option is specified or set to "true", pull the image. Raise an error if the im?age could not be pulled, even if the image is present locally.

If the option is disabled (with --pull=false) or not specified, pull the image from the registry only if the image is not present locally. Raise an error if the image is not found in the registries and is not present locally.

--pull-always

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

--pull-never

Do not pull the image from the registry, use only the local version. Raise an error if the image is not present locally.

--quiet, -q

Suppress output messages which indicate which instruction is being processed, and of progress when pulling images from a registry, and when writing the output image.

--rm=true|false

Remove intermediate containers after a successful build (default true).

--runtime=path

The path to an alternate OCI-compatible runtime, which will be used to run commands speci?fied by the RUN instruction.

Note: You can also override the default runtime by setting the BUILDAH_RUNTIME environment variable. export BUILDAH_RUNTIME=/usr/local/bin/runc

--secret=id=id,src=path

Pass secret information to be used in the Containerfile for building images in a safe way

that will not end up stored in the final image, or be seen in other stages. The secret will be mounted in the container at the default location of /run/secrets/id.

To later use the secret, use the --mount flag in a RUN instruction within a Containerfile:

RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecret

--security-opt=option

Security Options

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

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

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

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

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

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

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

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

? no-new-privileges : Not supported

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

? seccomp=profile.json : White listed syscalls seccomp Json file to be used as a seccomp filter

--shm-size=size

Size of /dev/shm. The format is <number><unit>. number must be greater than 0. Unit is optional and can be b (bytes), k (kilobytes), m(megabytes), or g (gigabytes). If you omit the unit, the system uses bytes. If you omit the size entirely, the system uses 64m.

--sign-by=fingerprint

Sign the image using a GPG key with the specified FINGERPRINT. (This option is not avail? able with the remote Podman client)

--squash

Squash all of the image's new layers into a single new layer; any preexisting layers are not squashed.

--squash-all

Squash all of the new image's layers (including those inherited from a base image) into a single new layer.

--ssh=default|id[=socket>|[,]

SSH agent socket or keys to expose to the build. The socket path can be left empty to use

the value of default=$SSH_AUTH_SOCK

To later use the ssh agent, use the --mount flag in a RUN instruction within a Container?

file:

RUN --mount=type=ssh,id=id mycmd

--stdin

Pass stdin into the RUN containers. Sometime commands being RUN within a Containerfile want to request information from the user. For example apt asking for a confirmation for install. Use --stdin to be able to interact from the terminal during the build.

--tag, -t=imageName

Specifies the name which will be assigned to the resulting image if the build process com? pletes successfully. If imageName does not include a registry name, the registry name lo? calhost will be prepended to the image name.

--target=stageName

Set the target build stage to build. When building a Containerfile with multiple build stages, --target can be used to specify an intermediate build stage by name as the final stage for the resulting image. Commands after the target stage will be skipped.

--timestamp seconds

Set the create timestamp to seconds since epoch to allow for deterministic builds (de? faults to current time). By default, the created timestamp is changed and written into the image manifest with every commit, causing the image's sha256 hash to be different even if the sources are exactly the same otherwise. When --timestamp is set, the created time? stamp is always set to the time specified and therefore not changed, allowing the image's sha256 hash to remain the same. All files committed to the layers of the image will be created with the timestamp.

--tls-verify=true|false

Require HTTPS and verify certificates when talking to container registries (defaults to true). (This option is not available with the remote Podman client)

--ulimit=type=soft-limit[:hard-limit]

Specifies resource limits to apply to processes launched when processing RUN instructions. This option can be specified multiple times. Recognized resource types include:

"core": maximum core dump size (ulimit -c)

"cpu": maximum CPU time (ulimit -t)

"data": maximum size of a process's data segment (ulimit -d)

"fsize": maximum size of new files (ulimit -f)

"locks": maximum number of file locks (ulimit -x)

"memlock": maximum amount of locked memory (ulimit -l)

"msgqueue": maximum amount of data in message queues (ulimit -q)

"nice": niceness adjustment (nice -n, ulimit -e)

"nofile": maximum number of open files (ulimit -n)

"nproc": maximum number of processes (ulimit -u)

"rss": maximum size of a process's (ulimit -m)

"rtprio": maximum real-time scheduling priority (ulimit -r)

"rttime": maximum amount of real-time execution between blocking syscalls

"sigpending": maximum number of pending signals (ulimit -i)

"stack": maximum stack size (ulimit -s)

--userns=how

Sets the configuration for user namespaces when handling RUN instructions. The configured value can be "" (the empty string) or "container" to indicate that a new user namespace should be created, it can be "host" to indicate that the user namespace in which podman itself is being run should be reused, or it can be the path to a user namespace which is already in use by another process.

--userns-uid-map=mapping

Directly specifies a UID mapping which should be used to set ownership, at the filesystem level, on the working container's contents. Commands run when handling RUN instructions will default to being run in their own user namespaces, configured using the UID and GID maps.

Entries in this map take the form of one or more triples of a starting in-container UID, a corresponding starting host-level UID, and the number of consecutive IDs which the map en? try represents.

This option overrides the remap-uids setting in the options section of /etc/contain? ers/storage.conf.

If this option is not specified, but a global --userns-uid-map setting is supplied, set? tings from the global option will be used.

If none of --userns-uid-map-user, --userns-gid-map-group, or --userns-uid-map are speci? fied, but --userns-gid-map is specified, the UID map will be set to use the same numeric values as the GID map.

--userns-gid-map=mapping

Directly specifies a GID mapping which should be used to set ownership, at the filesystem level, on the working container's contents. Commands run when handling RUN instructions will default to being run in their own user namespaces, configured using the UID and GID maps.

Entries in this map take the form of one or more triples of a starting in-container GID, a corresponding starting host-level GID, and the number of consecutive IDs which the map en? try represents.

This option overrides the remap-gids setting in the options section of /etc/contain? ers/storage.conf.

If this option is not specified, but a global --userns-gid-map setting is supplied, set? tings from the global option will be used.

If none of --userns-uid-map-user, --userns-gid-map-group, or --userns-gid-map are speci? fied, but --userns-uid-map is specified, the GID map will be set to use the same numeric values as the UID map.

--userns-uid-map-user=user

Specifies that a UID mapping which should be used to set ownership, at the filesystem level, on the working container's contents, can be found in entries in the /etc/subuid file which correspond to the specified user. Commands run when handling RUN instructions will default to being run in their own user namespaces, configured using the UID and GID maps. If --userns-gid-map-group is specified, but --userns-uid-map-user is not specified, podman will assume that the specified group name is also a suitable user name to use as the default setting for this option.

NOTE: When this option is specified by a rootless user, the specified mappings are rela? tive to the rootless user namespace in the container, rather than being relative to the host as it would be when run rootfull.

--userns-gid-map-group=group

Specifies that a GID mapping which should be used to set ownership, at the filesystem level, on the working container's contents, can be found in entries in the /etc/subgid file which correspond to the specified group. Commands run when handling RUN instructions will default to being run in their own user namespaces, configured using the UID and GID maps. If --userns-uid-map-user is specified, but --userns-gid-map-group is not specified, podman will assume that the specified user name is also a suitable group name to use as

the default setting for this option.

NOTE: When this option is specified by a rootless user, the specified mappings are rela?
tive to the rootless user namespace in the container, rather than being relative to the
host as it would be when run rootfull.

--uts=how

Sets the configuration for UTS namespaces when the handling RUN instructions. The config?
ured value can be "" (the empty string) or "container" to indicate that a new UTS name?
space should be created, or it can be "host" to indicate that the UTS namespace in which
podman itself is being run should be reused, or it can be the path to a UTS namespace
which is already in use by another process.

--variant=""

Set the architecture variant of the image to be built, and that of the base image to be
pulled, if the build uses one, to the provided value instead of using the architecture
variant of the build host.

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

Create a bind mount. If you specify, -v /HOST-DIR:/CONTAINER-DIR, Podman
bind mounts /HOST-DIR in the host to /CONTAINER-DIR in the Podman
container. (This option is not available with the remote Podman client)

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

? [rw|ro]

? [z|Z|O]

? [U]

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

The CONTAINER-DIR must be an absolute path such as /src/docs. The HOST-DIR must be an ab?
solute path as well. Podman bind-mounts the HOST-DIR to the path you specify. For example,
if you supply /foo as the host path, Podman copies the contents of /foo to the container
filesystem on the host and bind mounts that into the container.

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

You can add the :ro or :rw suffix to a volume to mount it 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. When running using user namespaces, the UID and GID inside the namespace may cor?

respond 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 namespace, to change recursively the owner and group of the source volume. Warning use with caution since this will modify the host filesystem.

Labeling Volume Mounts

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

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

Note: Do not relabel system files and directories. Relabeling system content might cause other confined services on your machine to fail. For these types of containers, disabling SELinux separation is recommended. 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 the build containers, they need to disable SELinux separation.

```
$ podman build --security-opt label=disable -v $HOME:/home/user .
```

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 RUN command containers are allowed to modify contents within the mountpoint and are stored in the container storage in a separate directory. In Over? lay FS terms the source directory will be the lower, and the container storage directory will be the upper. Modifications to the mount point are destroyed when the RUN command finishes executing, similar to a tmpfs mount point.

Any subsequent execution of RUN commands sees the original source directory content, any changes from previous RUN commands no longer exists.

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

Note:

- Overlay mounts are not currently supported in rootless mode.

- The `O` flag is not allowed to be specified with the `Z` or `z` flags.

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

On SELinux systems, labels in the source directory needs to be readable by the con? tainer label. If not, SELinux container separation must be disabled for the container to work.

- Modification of the directory volume mounted into the container with an overlay mount can cause unexpected failures. It is recommended that you do not modify the direc? tory until the container finishes running.

By default bind mounted volumes are private. That means any mounts done inside containers will not be visible on the host and vice versa. This behavior can be changed by specifying a volume mount propagation property.

When the mount propagation policy is set to shared, any mounts completed inside the con? tainer on that volume will be visible to both the host and container. When the mount prop? agation policy is set to slave, one way mount propagation is enabled and any mounts com? pleted on the host for that volume will be visible only inside of the container. To con? trol the mount propagation property of volume use the :[r]shared, :[r]slave or :[r]private propagation flag. The propagation property can be specified only for bind mounted volumes and not for internal volumes or named volumes. For mount propagation to work on the source mount point (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 has to be either shared or slave. [1] ?#Footnote1?

Use df <source-dir> to determine the source mount and then use findmnt -o TARGET,PROPAGA? TION <source-mount-dir> to determine propagation properties of source mount, if findmnt utility is not available, the source mount point can be determined by looking at the mount entry in /proc/self/mountinfo. Look at optional fields and see if any propagation proper? ties are specified. shared:X means the mount is shared, master:X means the mount is slave and if nothing is there that means the mount is private. [1] ?#Footnote1?

To change propagation properties of a mount point use the mount command. For example, to bind mount the source directory /foo do mount --bind /foo /foo and mount --make-private --make-shared /foo. This will convert /foo into a shared mount point. The propagation properties of the source mount can be changed directly. For instance if / is the source mount for /foo, then use mount --make-shared / to convert / into a shared mount.

EXAMPLES

Build an image using local Containerfiles

$ podman build .

$ podman build -f Containerfile.simple .

$ cat $HOME/Dockerfile | podman build -f - .

$ podman build -f Dockerfile.simple -f Containerfile.notsosimple .

$ podman build -f Dockerfile.in $HOME

$ podman build -t imageName .

$ podman build --tls-verify=true -t imageName -f Dockerfile.simple .

$ podman build --tls-verify=false -t imageName .

$ podman build --runtime-flag log-format=json .

$ podman build --runtime-flag debug .

$ podman build --authfile /tmp/auths/myauths.json --cert-dir $HOME/auth --tls-verify=true --creds=username:password -t imageName -f Dockerfile.simple .

$ podman build --memory 40m --cpu-period 10000 --cpu-quota 50000 --ulimit nofile=1024:1028 -t imageName .

$ podman build --security-opt label=level:s0:c100,c200 --cgroup-parent /path/to/cgroup/parent -t imageName .

$ podman build --volume /home/test:/myvol:ro,Z -t imageName .

$ podman build -v /var/lib/yum:/var/lib/yum:O -t imageName .

$ podman build --layers -t imageName .

$ podman build --no-cache -t imageName .

$ podman build --layers --force-rm -t imageName .

$ podman build --no-cache --rm=false -t imageName .

Building a multi-architecture image using the --manifest option (requires emulation software)

$ podman build --arch arm --manifest myimage /tmp/mysrc

$ podman build --arch amd64 --manifest myimage /tmp/mysrc

$ podman build --arch s390x --manifest myimage /tmp/mysrc

$ podman build --platform linux/s390x,linux/ppc64le,linux/amd64 --manifest myimage /tmp/mysrc

$ podman build --platform linux/arm64 --platform linux/amd64 --manifest myimage /tmp/mysrc

Building an image using a URL, Git repo, or archive

The build context directory can be specified as a URL to a Containerfile, a Git reposi? tory, or URL to an archive. If the URL is a Containerfile, it is downloaded to a temporary location and used as the context. When a Git repository is set as the URL, the repository is cloned locally to a temporary location and then used as the context. Lastly, if the URL

is an archive, it is downloaded to a temporary location and extracted before being used as the context.

Building an image using a URL to a Containerfile

Podman will download the Containerfile to a temporary location and then use it as the build context.

```
$ podman build https://10.10.10.1/podman/Containerfile
```

Building an image using a Git repository

Podman will clone the specified GitHub repository to a temporary location and use it as the context. The Containerfile at the root of the repository will be used and it only works if the GitHub repository is a dedicated repository.

```
$ podman build git://github.com/scollier/purpletest
```

Building an image using a URL to an archive

Podman will fetch the archive file, decompress it, and use its contents as the build con? text. The Containerfile at the root of the archive and the rest of the archive will get used as the context of the build. If you pass -f PATH/Containerfile option as well, the system will look for that file inside the contents of the archive.

```
$ podman build -f dev/Containerfile https://10.10.10.1/podman/context.tar.gz
```

Note: supported compression formats are 'xz', 'bzip2', 'gzip' and 'identity' (no compres? sion).

Files

.dockerignore

If the file .dockerignore exists in the context directory, buildah copy reads its con? tents. Use the --ignorefile flag to override .dockerignore path location. Podman uses the content to exclude files and directories from the context directory, when executing COPY and ADD directives in the Containerfile/Dockerfile

Users can specify a series of Unix shell globals in a .dockerignore file to identify files/directories to exclude.

Podman supports a special wildcard string ** which matches any number of directories (in? cluding zero). For example, */.go will exclude all files that end with .go that are found in all directories.

Example .dockerignore file:

```
# exclude this content for image
*/*.c
```

**/output*

src

/.c Excludes files and directories whose names ends with .c in any top level subdirec?

tory. For example, the source file include/rootless.c.

**/output* Excludes files and directories starting with output from any directory.

src Excludes files named src and the directory src as well as any content in it.

Lines starting with ! (exclamation mark) can be used to make exceptions to exclusions. The

following is an example .dockerignore file that uses this mechanism:

*.doc

!Help.doc

Exclude all doc files except Help.doc from the image.

This functionality is compatible with the handling of .dockerignore files described here:

https://docs.docker.com/engine/reference/builder/#dockerignore-file

registries.conf (/etc/containers/registries.conf)

registries.conf is the configuration file which specifies which container registries

should be consulted when completing image names which do not include a registry or domain

portion.

Troubleshooting

lastlog sparse file

If you are using a useradd command within a Containerfile with a large UID/GID, it will

create a large sparse file /var/log/lastlog. This can cause the build to hang forever.

Go language does not support sparse files correctly, which can lead to some huge files be?

ing created in your container image.

If you are using useradd within your build script, you should pass the --no-log-init or -l

option to the useradd command. This option tells useradd to stop creating the lastlog

file.

SEE ALSO

podman(1), buildah(1), containers-certs.d(5), containers-registries.conf(5), crun(8),

runc(8), useradd(8), podman-ps(1), podman-rm(1)

HISTORY

Aug 2020, Additional options and .dockerignore added by Dan Walsh <dwalsh@redhat.com>

May 2018, Minor revisions added by Joe Doss <joe@solidadmin.com>

December 2017, Originally compiled by Tom Sweeney <tsweeney@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 fol? low suit immediately.