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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'podman-image-build.1' command

$ man podman-image-build.1

```
podman-build(1)          General Commands Manual          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 Con? tainerfiles or Dockerfiles 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 ei? ther 'Containerfile' or 'Dockerfile'.

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

If no context directory is specified, then Podman will assume the cur? rent working directory 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 a Containerfile, the Containerfile is downloaded to a
temporary location.

When a Git repository is set as the URL, the repository is cloned lo?
cally 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 pod?
man build crashes or users kill the podman build process, these exter?
nal 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:ip

Add a custom host-to-IP mapping (host:ip)

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

--all-platforms

Instead of building for a set of platforms specified using the --plat?
form option, inspect the build's base images, and build for all of the
platforms for which they are all available. Stages that use scratch as
a starting point can not be inspected, so at least one non-scratch
stage must be present for detection to work usefully.

--annotation=annotation

Add an image annotation (e.g. annotation=value) to the image metadata.
Can be used multiple 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 im?

age to be pulled, if the build uses one, to the provided value instead of using the architecture of the build host. Unless overridden, subse?quent lookups of the same image in the local storage will match this architecture, regardless of the host. (Examples: arm, arm64, 386, amd64, ppc64le, s390x)

--authfile=path

Path of the authentication file. Default is ${XDG_RUNTIME_DIR}/contain?ers/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: There is also the option to override the default path of the au?thentication file by setting the REGISTRY_AUTH_FILE environment vari?able. This can be done with 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 environ?ment variables are, but which will not be added to environment variable list in the resulting image's configuration.

--build-context=name=value

Specify an additional build context using its short name and its loca?tion. Additional build contexts can be referenced in the same manner as we access different stages in COPY instruction.

Valid values could be:

? Local directory ? e.g. --build-context project2=../path/to/project2/src (This option is not available with the remote Podman client. On Podman machine setup (i.e macOS and Winows) path must exists on the machine VM)

? HTTP URL to a tarball ? e.g. --build-context src=https://exam?ple.org/releases/src.tar

? Container image ? specified with a container-image:// prefix, e.g. --build-context alpine=container-image://alpine:3.15, (also accepts docker://, docker-image://)

On the Containerfile side, reference the build context on all commands that accept the ?from? parameter. Here?s how that might look:

```
FROM [name]
COPY --from=[name] ...
RUN --mount=from=[name] ?
```

The value of [name] is matched with the following priority order:

? Named build context defined with --build-context [name]=..

? Stage defined with AS [name] inside Containerfile

? Image [name], either local or in a remote registry

--cache-from

Repository to utilize as a potential cache source. When specified, Buildah will try to look for cache images in the specified repository and will attempt to pull cache images instead of actually executing the build steps locally. Buildah will only attempt to pull previously cached images if they are considered as valid cache hits.

Use the --cache-to option to populate a remote repository with cache content.

Example

```
# populate a cache and also consult it
buildah build -t test --layers --cache-to registry/myrepo/cache --cache-from registry/myrepo/cache .
```

Note: --cache-from option is ignored unless --layers is specified.

--cache-to

Set this flag to specify a remote repository that will be used to store cache images. Buildah will attempt to push newly built cache image to the remote repository.

Note: Use the --cache-from option in order to use cache content in a remote repository.

Example

```
# populate a cache and also consult it
buildah build -t test --layers --cache-to registry/myrepo/cache --cache-from registry/myrepo/cache .
```

Note: --cache-to option is ignored unless --layers is specified.

--cache-ttl

Limit the use of cached images to only consider images with created timestamps less than duration ago. For example if --cache-ttl=1h is

specified, Buildah will only consider intermediate cache images which
are created under the duration of one hour, and intermediate cache im?
ages outside this duration will be ignored.

Note: Setting --cache-ttl=0 manually is equivalent to using --no-cache
in the implementation since this would effectively mean that user is
not willing to use cache at all.

--cap-add=CAP_xxx

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

--cap-drop=CAP_xxx

When executing RUN instructions, run the command specified in the in?
struction with the specified capability removed from its capability
set. The CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID,
CAP_KILL, CAP_NET_BIND_SERVICE, CAP_SETFCAP, CAP_SETGID, CAP_SETPCAP,
and CAP_SETUID 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 op?
tions, 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 reg?
istry. (Default: /etc/containers/certs.d) Please refer to containers-
certs.d(5) for details. (This option is not available with the remote
Podman client, including Mac and Windows (excluding WSL2) machines)

--cgroup-parent=path

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

--cgroupns=how

Sets the configuration for cgroup namespaces when handling RUN instruc?

tions. The configured value can be "" (the empty string) or "private" to indicate that a new cgroup namespace should be created, or it can be "host" to indicate that the cgroup namespace in which buildah itself is being run should be reused.

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

--cpp-flag=flags

Set additional flags to pass to the C Preprocessor cpp(1). Containerfiles ending with a ".in" suffix will be preprocessed via cpp(1). This option can be used to pass additional flags to cpp.Note: You can also set default CPPFLAGS by setting the BUILDAH_CPPFLAGS environment variable (e.g., export BUILDAH_CPPFLAGS="-DDEBUG").

--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 will be 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/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.

--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/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.

--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/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.

--cpuset-mems=nodes

Memory nodes (MEMs) in which to allow execution (0-3, 0,1). Only effec?tive 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/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.

--creds=[username[:password]]

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 entered. The password is en?tered 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 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 container 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: if the user only has access rights via a group, accessing the de? vice 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 location where the image is being written. This is the default setting, because image layers are compressed automatically when they are pushed to registries, and images being written to local stor? age would only need to be decompressed again to be stored. Compression 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 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/re?
solv.conf in the container by Podman. The /etc/resolv.conf file in the image will be used without changes.

This option cannot be combined with --network that is set to none.

Note: this option takes effect only during RUN instructions in the build. It does not affect /etc/resolv.conf in the final image.

--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.

--env=env[=value]

Add a value (e.g. env=value) to the built image. Can be used multiple times. If neither = nor a *value* are specified, but env is set in the current environment, the value from the current environment will be added to the image. To remove an environment variable from the built image, use the --unsetenv option.

--file, -f=Containerfile

Specifies a Containerfile which contains instructions for building the image, either a local file or an http or https URL. If more than one Containerfile is specified, FROM instructions will only be accepted from the last 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.

Specifying the option -f - causes the Containerfile contents to be read from stdin.

--force-rm

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 formats include oci (OCI image-spec v1.0, the de?fault) and docker (version 2, using schema format 2 for the manifest).

Note: You can also override the default format by setting the BUIL?DAH_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.

With the remote podman client, not all container transports will work as expected. For example, oci-archive:/x.tar will reference /x.tar on the remote machine instead of on the client. When using podman remote clients it is best to restrict use to containers-storage, and docker:// transports.

--group-add=group | keep-groups

Assign additional groups to the primary user running within the con?tainer process.

? keep-groups is a special value that tells Buildah 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 con?tainer. Currently only available with the crun OCI runtime. Note: keep-groups is exclusive, other groups cannot be specified with this flag.

--help, -h

Print usage statement

--hooks-dir=path

Each *.json file in the path configures a hook for buildah build con? tainers. For more details on the syntax of the JSON files and the se? mantics of hook injection. Buildah currently support both the 1.0.0 and 0.1.0 hook schemas, although the 0.1.0 schema is deprecated.

This option may be set multiple times; paths from later options have higher precedence.

For the annotation conditions, buildah uses any annotations set in the generated OCI configuration.

For the bind-mount conditions, only mounts explicitly requested by the caller via --volume are considered. Bind mounts that buildah inserts by default (e.g. /dev/shm) are not considered.

If --hooks-dir is unset for root callers, Buildah will currently de? fault to /usr/share/containers/oci/hooks.d and /etc/contain? ers/oci/hooks.d in order of increasing precedence. Using these defaults is deprecated, and callers should migrate to explicitly setting --hooks-dir.

--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 should not use any proxy. Proxy environment variables specified for the container in any other way will override the values that would have been passed through from the host. (Other ways to spec? ify 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 will use the proxy environ? ment variables that are set on the server process.

Defaults to true.

--identity-label

Adds default identity label io.buildah.version if set. (default true).

--ignorefile

Path to an alternative .containerignore file.

--iidfile=ImageIDfile

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

--ipc=how

Sets the configuration for IPC namespaces when handling RUN instruc? tions. The configured value can be "" (the empty string) or "con? tainer" 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 name? space 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 run? time, the default), rootless (OCI-compatible runtime invoked using a modified configuration and its --rootless option enabled, with --no-new-keyring --no-pivot added to its create invocation, with network and UTS namespaces 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 specifies the list of Linux capabilities re?

quired for the container to run properly. This label specified in a container image tells Podman to run the container with just these capa? bilities. Podman launches the container with just the specified capa? bilities, as long as this list of capabilities is a subset of the de? fault list.

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

--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 environment 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.

This option is not supported on the remote client, including Mac and Windows (excluding WSL2) machines.

--logsplit=bool-value

If --logfile and --platform are specified, the --logsplit option allows end-users to split the log file for each platform into different files in the following format: ${logfile}_${platform-os}_${platform-arch}.

This option is not supported on the remote client, including Mac and Windows (excluding WSL2) machines.

--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=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.

--network=mode, --net

Sets the configuration for network namespaces when handling RUN in? structions.

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)

? <network name|ID>: Join the network with the given name or ID, e.g. use --network mynet to join the network with the name mynet. Only supported for rootful users.

--no-cache

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

--no-hosts

Do not create /etc/hosts for the container. By default, Podman will manage /etc/hosts, adding the container's own IP address and any hosts from --add-host. --no-hosts disables this, and the image's /etc/hosts will be preserved unmodified.

This option conflicts with --add-host.

--omit-history

Omit build history information in the built image. (default false).

This option is useful for the cases where end users explicitly want to set --omit-history to omit the optional History from built images or when working with images built using build tools that do not include History information in their images.

--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. Unless overridden, subsequent lookups of the same image in the local storage will match this OS, regardless of the host.

--os-feature=feature

Set the name of a required operating system feature for the image which will be built. By default, if the image is not based on scratch, the base image's required OS feature list is kept, if the base image speci?fied any. This option is typically only meaningful when the image's OS is Windows.

If feature has a trailing -, then the feature is removed from the set of required features which will be listed in the image.

--os-version=version

Set the exact required operating system version for the image which will be built. By default, if the image is not based on scratch, the base image's required OS version is kept, if the base image specified one. This option is typically only meaningful when the image's OS is Windows, and is typically set in Windows base images, so using this op?tion is usually unnecessary.

--output, -o=output-opts

Output destination (format: type=local,dest=path)

The --output (or -o) option extends the default behavior of building a container image by allowing users to export the contents of the image as files on the local filesystem, which can be useful for generating local binaries, code generation, etc. (This option is not available with the remote Podman client, including Mac and Windows (excluding WSL2) machines)

The value for --output is a comma-separated sequence of key=value pairs, defining the output type and options.

Supported keys are: - dest: Destination path for exported output. Valid value is absolute or relative path, - means the standard output. - type: Defines the type of output to be used. Valid values is documented below.

Valid type values are: - local: write the resulting build files to a directory on the client-side. - tar: write the resulting files as a single tarball (.tar).

If no type is specified, the value defaults to local. Alternatively, instead of a comma-separated sequence, the value of --output can be just a destination (in the **dest** format) (e.g.--output some-path,--output -) where--output some-pathis treated as if **type=local** and--output -` is treated as if type=tar.

--pid=pid

Sets the configuration for PID namespaces when handling RUN instruc? tions. The configured value can be "" (the empty string) or "con? tainer" 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, when using one) to the provided value instead of using the current operating system and architecture of the host (for example linux/arm). Unless overridden, subsequent lookups of the same image in the local storage will match this platform, regardless of the host.

If --platform is set, then the values of the --arch, --os, and --vari? ant options will be overridden.

The --platform option 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 sev?eral 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 exists, RUN instructions will not be able to succeed without the help of emulation provided by packages like qemu-user-static.

--pull=policy

Pull image policy. The default is always.

? always, true: Always pull the image and throw an error if the pull fails.

? missing: Pull the image only if it could not be found in the local containers storage. Throw an error if no image could be found and the pull fails.

? never, false: Never pull the image but use the one from the local containers storage. Throw an error if no image could be 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 messages which indicate which instruction is being pro?cessed, and of progress when pulling images from a registry, and when writing the output image.

--retry=attempts

Number of times to retry in case of failure when performing pull of im?
ages from registry. Default is 3.

--retry-delay=duration

Duration of delay between retry attempts in case of failure when per?
forming pull of images from registry. Default is 2s.

--rm

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 specified by the RUN instruction.

Note: You can also override the default runtime by setting the BUIL?
DAH_RUNTIME environment variable. export BUILDAH_RUNTIME=/usr/lo?
cal/bin/runc

--runtime-flag=flag

Adds global flags for the container rutime. To list the supported
flags, please consult the manpages of the selected container runtime.

Note: Do not pass the leading -- to the flag. To pass the runc flag
--log-format json to buildah build, the option given would be --run?
time-flag log-format=json.

--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 option 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=alternate-profile : Set the apparmor confinement pro?
file 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 con?
tainer processes

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

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

? label=disable : Turn off label separation for the con?
tainer

? no-new-privileges : Not supported

? seccomp=unconfined : Turn off seccomp confinement for the con?
tainer

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

--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 con?
tainer. This option conflicts with --ipc=host.

--sign-by=fingerprint

Sign the image using a GPG key with the specified FINGERPRINT. (This
option is not available with the remote Podman client, including Mac
and Windows (excluding WSL2) machines,)

--skip-unused-stages

Skip stages in multi-stage builds which don't affect the target stage.
(Default: true).

--squash

Squash all of the image's new layers into a single new layer; any pre?
existing 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 option in a RUN instruction
within a Containerfile:
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 completes successfully. If imageName does not include a
registry name, the registry name localhost will be prepended to the im?
age 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 interme?
diate 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 determin?
istic builds (defaults to current time). By default, the created time?
stamp 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 commit?
ted to the layers of the image will be created with the timestamp.

If the only instruction in a Containerfile is FROM, this flag has no effect.

--tls-verify

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

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

Specifies resource limits to apply to processes launched when process? ing 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)

--unsetenv=env

Unset environment variables from the final image.

--userns=how

Sets the configuration for user namespaces when handling RUN instruc? tions. The configured value can be "" (the empty string) or "con?

tainer" 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-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 num?ber of consecutive IDs which the map entry represents.

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

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

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

--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 be?ing 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 relative to the rootless user namespace in the container, rather than being relative to the host as it would be when run rootful.

--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 entry represents.

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

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

If none of --userns-uid-map-user, --userns-gid-map-group, or --userns-uid-map are specified, but --userns-gid-map is specified, the UID map will be set to use the same numeric values as the GID 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 relative to the rootless user namespace in the container, rather than being relative to the host as it would be when run rootful.

--uts=how

Sets the configuration for UTS namespaces when handling RUN instructions. The configured value can be "" (the empty string) or "container" to indicate that a new UTS namespace 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=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. Specifying the -v /HOST-DIR:/CONTAINER-DIR option, Podman bind mounts /HOST-DIR from the host to /CONTAINER-DIR in the Podman container.

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 absolute path as well. Podman bind-mounts the HOST-DIR to the specified path. For example, when specifying the host path /foo, 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 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 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, add one of these two suf? fixes :z or :Z to the volume mount. These suffixes tell Podman to rela? bel file objects on the shared volumes. The z option tells Podman that two containers share the volume content. As a result, Podman labels the content with a shared content label. Shared volume labels allow all containers to read/write content. The Z option tells Podman to label the content with a private 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 the host 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 tem? porary storage using the Overlay file system. The RUN command contain? ers are allowed to modify contents within the mountpoint and are stored in the container storage in a separate directory. In Overlay FS terms the source directory will be the lower, and the container storage di? rectory will be the upper. Modifications to the mount point are de? stroyed when the RUN command finishes executing, similar to a tmpfs mount point.

Any subsequent execution of RUN commands sees the original source di? rectory content, any changes from previous RUN commands no longer ex?

ists.

One use case of the overlay mount is sharing the package cache from the

host into the container 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 container 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. Do not modify the

directory 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 prop?

erty.

When the mount propagation policy is set to shared, any mounts com?

pleted inside the container on that volume will be visible to both the

host and container. When the mount propagation policy is set to slave,

one way mount propagation is enabled and any mounts completed on the

host for that volume will be visible only inside of the container. To

control the mount propagation property of volume use the :[r]shared,

:[r]slave or :[r]private propagation flag. 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,PROPAGATION <source-mount-dir> to determine propagation prop?

erties 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 propaga?

tion properties 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 com? mand. 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/Containerfile | podman build -f - .

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

$ podman build -f Containerfile.in $HOME

$ podman build -t imageName .

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

$ podman build --network mynet .
```

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 Container? file, a Git repository, or URL to an archive. If the URL is a Contain? erfile, it is downloaded to a temporary location and used as the con? text. 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 loca? tion 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 https://github.com/scollier/purpletest
```

Note: Github does not support using git:// for performing clone opera? tion due to recent changes in their security guidance (https://github.blog/2021-09-01-improving-git-protocol-security-github/). Use an https:// URL if the source repository is hosted on Github.

Building an image using a URL to an archive

Podman will fetch the archive file, decompress it, and use its contents as the build context. The Containerfile at the root of the archive and the rest of the archive will get used as the context of the build.

Passing the -f PATH/Containerfile option as well tells the system to 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 compression).

Files

.containerignore/.dockerignore

If the file .containerignore or .dockerignore exists in the context di?rectory, podman build reads its contents. Use the --ignorefile option to override the Podman uses the content to exclude files and directo?ries from the context directory, when executing COPY and ADD directives in the Containerfile/Dockerfile

The .containerignore and .dockerignore files use the same syntax; if both are in the context directory, podman build will only use .con?tainerignore.

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

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

Example .containerignore file:

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

/.c Excludes files and directories whose names ends with .c in any top level subdirectory. For example, the source file include/root?less.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 con?tent in it.

Lines starting with ! (exclamation mark) can be used to make exceptions to exclusions. The following is an example .containerignore 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 .containerignore files described here:

https://github.com/containers/common/blob/main/docs/containerignore.5.md

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

Using a useradd command within a Containerfile with a large UID/GID 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 being created in the container image.

When using the useradd command within the build script, 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(1), runc(8), useradd(8), podman-ps(1), podman-rm(1), Containerfile(5), containerignore(5)

HISTORY

Aug 2020, Additional options and .containerignore 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 follow suit immediately.

podman-build(1)