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

$ man buildah-build.1

buildah-build(1)   General Commands Manual   buildah-build(1)

NAME

buildah-build - Build an image using instructions from Containerfiles

SYNOPSIS

buildah build [options] [context]

buildah bud [options] [context]

buildah build-using-dockerfile [options] [context]

build has aliases bud and build-using-dockerfile.

DESCRIPTION

Builds an image using instructions from one or more Containerfiles 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 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 Buildah will assume the current working direc? tory as build context, which should contain a 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. Any comments (Lines beginning with #) in in? cluded Containerfile(s) that are not preprocess commands, will be printed as warnings dur? ing builds.

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 file 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.

OPTIONS

--add-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 ARCH 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 host. (Ex?

amples: arm, arm64, 386, amd64, ppc64le, s390x)

--authfile path

Path of the authentication file. Default is ${XDG_\RUNTIME_DIR}/containers/auth.json. If

XDG_RUNTIME_DIR is not set, the default is /run/containers/$UID/auth.json. This file is

created using using buildah 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.

Please refer to the BUILD TIME VARIABLES ?#build-time-variables? section for the list of

variables that can be overridden within the Containerfile at run time.

--cache-from

Images to utilise as potential cache sources. Buildah does not currently support

--cache-from so this is a NOOP.

--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. The default

certificates directory is /etc/containers/certs.d.

--cgroup-parent=""

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. Buildah doesn't send a

copy of the context directory to a daemon or a remote server. Thus, compressing the data

before sending it is irrelevant to Buildah.

--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=0

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/main/troubleshooting.md#26-run?
ning-containers-with-cpu-limits-fails-with-a-permissions-error

--cpu-quota=0

Limit the CPU CFS (Completely Fair Scheduler) quota

Limit the container's CPU usage. By default, containers run with the full CPU resource.
This flag tell the kernel to restrict the container's CPU usage to the quota you specify.

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

--cpu-shares, -c=0

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=""

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

--cpuset-mems=""

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=device

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 Buildah. This flag is a NOOP and provided solely for scripting compati?

bility.

--dns=[]

Set custom DNS servers

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 Buildah. The /etc/resolv.conf file in the image will be used without changes.

--dns-option=[]

Set custom DNS options

--dns-search=[]

Set custom DNS search domains

--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 local file is specified as the Containerfile and it does not exist, the context di?

rectory will be prepended to the local file value.

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

--force-rm bool-value

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

false).

--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=true

By default proxy environment variables are passed into the container if set for the buil?dah process. This can be disabled by setting the --http-proxy option to false. The envi?ronment variables passed in include http_proxy, https_proxy, ftp_proxy, no_proxy, and also the upper case versions of those.

--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 file

Path to an alternative .containerignore (.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 buildah 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, with --no-new-keyring added to its create invocation, reusing the host's network and UTS namespaces, and creating private IPC, PID, mount, and user namespaces; the default for unprivileged users), and chroot (an internal wrapper that leans more toward chroot(1) than container technology, reusing the host's control group, network, IPC, and PID namespaces, and creating private mount and UTS namespaces, and creating user namespaces only when they're required for ID mapping).

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

--jobs N

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 on 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 container engines, like Podman, to run the

container with just these capabilities. The container engine 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, container engines should print

an error message and will run the container with the default capabilities.

--layers bool-value

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

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 "listName"

Name of the manifest list to which the built image will be added. Creates the manifest

list if it does not exist. This option is useful for building multi architecture images.

--memory, -m=""

Memory limit (format: [], where unit = b, k, m or g)

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, --net=mode

Sets the configuration for network namespaces when handling RUN instructions.

Valid mode values are:

? none: no networking;

? host: use the host network stack. Note: the host mode gives the container full access to local system services such as D-bus and is therefore considered inse?
cure;

? 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="OS"

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 host.

--pid how

Sets the configuration for PID namespaces when handling RUN instructions. The configured value can be "" (the empty string) or "private" to indicate that a new PID namespace should be created, or it can be "host" to indicate that the PID namespace in which buildah 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 buildah bud 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 flag is enabled, attempt to pull the latest image from the registries listed in

registries.conf if a local image does not exist or the image is newer than the one in

storage. Raise an error if the image is not in any listed registry and is not present lo?

cally.

If the flag is disabled (with --pull=false), do not pull the image from the registry, un?

less there is no local image. Raise an error if the image is not in any registry and is

not present locally.

Defaults to true.

--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 bool-value

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. Default is runc, or crun when machine is configured to use cgroups V2.

Note: You can also override the default runtime by setting the BUILDAH_RUNTIME environment

variable. export BUILDAH_RUNTIME=/usr/bin/crun

--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 --runtime-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 flag in a RUN instruction within a Containerfile:

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

--security-opt=[]

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

  "label=role:ROLE"   : Set the label role for the container

  "label=type:TYPE"   : Set the label type for the container

  "label=level:LEVEL" : Set the label level for the container

  "label=disable"   : Turn off label confinement 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 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 built image using the GPG key that matches the specified fingerprint.

--squash

Squash all of the image's new layers into a single new layer; any preexisting layers are

not squashed.

--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=secret,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 component, the reg?

istry name localhost 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 to remain the same. All files committed to the layers of the image will be created

with the timestamp.

--tls-verify bool-value

Require HTTPS and verification of certificates when talking to container registries (de?

faults to true). TLS verification cannot be used when talking to an insecure registry.

--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)

  "nofile": maximum number of open files (1048576); when run by root

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

  "nproc": maximum number of processes (1048576); when run by root

  "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 "private" to indicate that a new user namespace should be created, it can be "host" to indicate that the user namespace in which buildah itself is being run should be reused, or it can be the path to an user namespace which is already in use by another process.

--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, buildah will assume that the specified group name is also a suitable user name to use as the default setting for this option.

Users can specify the maps directly using --userns-uid-map described in the buildah(1) man page.

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

--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, buildah will assume that the specified user name is also a suitable group name to use as the default setting for this option.

Users can specify the maps directly using --userns-gid-map described in the buildah(1) man page.

NOTE: When this option is specified by a rootless user, the specified mappings are rela?
tive to the rootless usernamespace 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 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 buildah 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 pulled.

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

Create a bind mount. If you specify, -v /HOST-DIR:/CONTAINER-DIR, Buildah
bind mounts /HOST-DIR in the host to /CONTAINER-DIR in the Buildah
container. The OPTIONS are a comma delimited list and can be: [1] ?#Footnote1?

? [rw|ro]

? [U]

? [z|Z|O]

? [[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. Buildah bind-mounts the HOST-DIR to the path you specify. For exam?
ple, if you supply /foo as the host path, Buildah copies the contents of /foo to the con?
tainer 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.

Write Protected Volume Mounts

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

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

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, Buildah 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 Buildah to relabel file objects on the shared vol?
umes. The z option tells Buildah that two containers share the volume content. As a re?
sult, Buildah labels the content with a shared content label. Shared volume labels allow
all containers to read/write content. The Z option tells Buildah to label the content
with a private unshared label. Only the current container can use a private volume.

Overlay Volume Mounts

The :O flag tells Buildah to mount the directory from the host as a temporary storage us?
ing 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
Overlay FS terms the source directory will be the lower, and the container storage direc?
tory will be the upper. Modifications to the mount point are destroyed when the RUN com?
mand 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:

- 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. It is recommended that you do not modify the directory until the container finishes running.

By default bind mounted volumes are private. That means any mounts done inside container
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 the 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 (the mount point where source dir is mounted on) it 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.

BUILD TIME VARIABLES

The ENV instruction in a Containerfile can be used to define variable values. When the image is built, the values will persist in the container image. At times it is more con?venient to change the values in the Containerfile via a command-line option rather than changing the values within the Containerfile itself.

The following variables can be used in conjunction with the --build-arg option to override the corresponding values set in the Containerfile using the ENV instruction.

? HTTP_PROXY

? HTTPS_PROXY

? FTP_PROXY

? NO_PROXY

Please refer to the Using Build Time Variables ?#using-build-time-variables? section of the Examples.

EXAMPLE

Build an image using local Containerfiles

```
buildah build .

buildah build -f Containerfile .

cat  /Dockerfile | buildah build -f - .

buildah build -f Dockerfile.simple -f Dockerfile.notsosimple .

buildah build --timestamp=$(date '+%s') -t imageName .

buildah build -t imageName .

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

buildah build --tls-verify=false -t imageName .

buildah build --runtime-flag log-format=json .

buildah build -f Containerfile --runtime-flag debug .

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

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

buildah build --security-opt label=level:s0:c100,c200 --cgroup-parent /path/to/cgroup/par?
```

```
    ent -t imageName .

    buildah build --arch=arm --variant v7 -t imageName .

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

    buildah build -v /home/test:/myvol:z,U -t imageName .

    buildah build -v /var/lib/dnf:/var/lib/dnf:O -t imageName .

    buildah build --layers -t imageName .

    buildah build --no-cache -t imageName .

    buildah build -f Containerfile --layers --force-rm -t imageName .

    buildah build --no-cache --rm=false -t imageName .

    buildah build --dns-search=example.com --dns=223.5.5.5 --dns-option=use-vc .

    buildah build -f Containerfile.in -t imageName .
```

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

```
    buildah build --arch arm --manifest myimage /tmp/mysrc

    buildah build --arch amd64 --manifest myimage /tmp/mysrc

    buildah build --arch s390x --manifest myimage /tmp/mysrc

    buildah bud --platform linux/s390x,linux/ppc64le,linux/amd64 --manifest myimage /tmp/mysrc

    buildah bud --platform linux/arm64 --platform linux/amd64 --manifest myimage /tmp/mysrc
```

Building an image using a URL

This will clone the specified GitHub repository from the URL and use it as context. The Containerfile or Dockerfile at the root of the repository is used as the context of the build. This only works if the GitHub repository is a dedicated repository.

```
    buildah build github.com/scollier/purpletest
```

Note: You can set an arbitrary Git repository via the git:// scheme.

Building an image using a URL to a tarball'ed context

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

```
    buildah build -f dev/Containerfile https://10.10.10.1/docker/context.tar.gz
```

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

Using Build Time Variables

Replace the value set for the HTTP_PROXY environment variable within the Containerfile.

```
buildah build --build-arg=HTTP_PROXY="http://127.0.0.1:8321"
```

ENVIRONMENT

BUILD_REGISTRY_SOURCES

BUILD_REGISTRY_SOURCES, if set, is treated as a JSON object which contains lists of reg?
istry names under the keys insecureRegistries, blockedRegistries, and allowedRegistries.
When pulling an image from a registry, if the name of the registry matches any of the
items in the blockedRegistries list, the image pull attempt is denied. If there are reg?
istries in the allowedRegistries list, and the registry's name is not in the list, the
pull attempt is denied.

TMPDIR The TMPDIR environment variable allows the user to specify where temporary files
are stored while pulling and pushing images. Defaults to '/var/tmp'.

Files

.containerignore/.dockerignore

If the .containerignore/.dockerignore file exists in the context directory, buildah build
reads its contents. If both exist, then .containerignore is used. Use the --ignorefile
flag to override the ignore file path location. Buildah 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

Buildah 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 .containerignore 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 .containerignore/.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.

policy.json (/etc/containers/policy.json)

Signature policy file. This defines the trust policy for container images. Controls which container registries can be used for image, and whether or not the tool should trust the images.

SEE ALSO

buildah(1), cpp(1), buildah-login(1), docker-login(1), namespaces(7), pid_namespaces(7), containers-policy.json(5), containers-registries.conf(5), user_namespaces(7), crun(1), runc(8)

FOOTNOTES

1: The Buildah project is committed to inclusivity, a core value of open source. The mas?
ter 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, Buildah will follow suit immediately.