TSR 2022/23. UNIT 04

DOCKER REFERENCE DOCUMENTATION

This text is referenced along the unit "Deployment Technologies", compiling reference information about Docker 20.10, including command line options and configuration directives. Swarm mode related information has been omitted.

SHORT[1] INDEX

MAIN SOURCES COMPILED:

1. Docker command line and options
   - `man` pages from `docker 20.10.18` installed on portal virtual machines
2. Docker main reference documentation
   - `https://docs.docker.com/reference/`
3. Dockerfile reference
   - `https://docs.docker.com/engine/reference/builder/`
4. Docker overview
   - `https://docs.docker.com/get-started/overview/`
5. `docker-compose` command line and options
   - `man` pages from `docker-compose 2.10.2` installed on portal virtual machines
6. `docker-compose.yml` reference
   - `https://docs.docker.com/compose/compose-file/`
   - `https://github.com/compose-spec/compose-spec/blob/master/spec.md`
   - Networking: `https://docs.docker.com/compose/networking/`

[1] You will find a more complete index at the end of this document

1 DOCKER COMMAND-LINE AND OPTIONS

From `docker --help`, and all its specializations, remarkably `docker network`, `docker node`, `docker service` and `docker swarm`.

A self-sufficient runtime for containers.

```
Usage: docker [OPTIONS] COMMAND
```

Options:

```
    --config string      Location of client config files
-c, --context string     Name of the context to use to connect to the daemon (overrides
                         DOCKER_HOST env var and default context set with "docker context use")
-D, --debug              Enable debug mode
-H, --host-list          Daemon socket(s) to connect to
-l, --log-level string   Set the logging level
    --tls                Use TLS; implied by --tlsverify
...
-v, --version            Print version information and quit
```

Commands:

```
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

Management Commands:

```
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```

1.1 Commands

1.1.1 attach

Attach local standard input, output, and error streams to a running container

```
docker attach [OPTIONS] CONTAINER
```

(Equivalent to `docker container attach`)

Options:

```
    --detach-keys string   Override the key sequence for detaching a container
    --no-stdin             Do not attach STDIN
    --sig-proxy            Proxy all received signals to the process (default true)
```

1.1.2 build

Build an image from a Dockerfile

```
docker build [OPTIONS] PATH | URL | -
```

(Equivalent to `docker image build`)

Options:

```
    --add-host list            Add a custom host-to-IP mapping (host:ip)
    --build-arg list           Set build-time variables
    --cache-from strings       Images to consider as cache sources
    --cgroup-parent string     Optional parent cgroup for the container
    --compress                 Compress the build context using gzip
    --cpu-period int           Limit the CPU CFS (Completely Fair Scheduler) period
    --cpu-quota int            Limit the CPU CFS (Completely Fair Scheduler) quota
-c, --cpu-shares int           CPU shares (relative weight)
    --cpuset-cpus string       CPUs in which to allow execution (0-3, 0,1)
    --cpuset-mems string       MEMs in which to allow execution (0-3, 0,1)
    --disable-content-trust    Skip image verification (default true)
-f, --file string              Name of the Dockerfile (Default is 'PATH/Dockerfile')
    --force-rm                 Always remove intermediate containers
    --iidfile string           Write the image ID to the file
    --isolation string         Container isolation technology
    --label list               Set metadata for an image
-m, --memory bytes             Memory limit
    --memory-swap bytes        Swap limit equal to memory plus swap:
                               '-1' to enable unlimited swap
    --network string           Set the networking mode for the RUN instructions during build
```

```
                                (default "default")
    --no-cache                  Do not use cache when building the image
    --pull                      Always attempt to pull a newer version of the image
-q, --quiet                     Suppress the build output and print image ID on success
    --rm                        Remove intermediate containers after a successful build
                                (default true)
    --security-opt strings      Security options
    --shm-size bytes            Size of /dev/shm
-t, --tag list                  Name and optionally a tag in the 'name:tag' format
    --target string             Set the target build stage to build.
    --ulimit ulimit             Ulimit options (default [])
```

1.1.3 commit

Create a new image from a container's changes

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

(Equivalent to `docker container commit`)

Options:

```
-a, --author string    Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
-c, --change value     Apply Dockerfile instruction to the created image (default [])
-m, --message string   Commit message
-p, --pause            Pause container during commit (default true)
```

1.1.4 cp

Copy files/folders between a container and the local filesystem

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
```

(Equivalent to `docker container cp`)

- Use `'-'` as the source to read a tar archive from stdin and extract it to a directory destination in a container.
- Use `'-'` as the destination to stream a tar archive of a container source to stdout.

Options:

```
-a, --archive       Archive mode (copy all uid/gid information)
-L, --follow-link   Always follow symbol link in SRC_PATH
```

1.1.5 create

Create a new container

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

(Equivalent to `docker container create`)

Options:

```
    --add-host list               Add a custom host-to-IP mapping (host:ip)
-a, --attach list                 Attach to STDIN, STDOUT or STDERR
    --blkio-weight uint16         Block IO (relative weight), between 10 and 1000, or 0 to
                                  disable (default 0)
    --blkio-weight-device list    Block IO weight (relative device weight) (default [])
    --cap-add list                Add Linux capabilities
    --cap-drop list               Drop Linux capabilities
    --cgroup-parent string        Optional parent cgroup for the container
    --cidfile string              Write the container ID to the file
```

```
    --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler) period
    --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler) quota
    --cpu-rt-period int              Limit CPU real-time period in microseconds
    --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
-c, --cpu-shares int                 CPU shares (relative weight)
    --cpus decimal                   Number of CPUs
    --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
    --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
    --device list                    Add a host device to the container
    --device-cgroup-rule list        Add a rule to the cgroup allowed devices list
    --device-read-bps list           Limit read rate (bytes per second) from a device
                                     (default [])
    --device-read-iops list          Limit read rate (IO per second) from a device (default [])
    --device-write-bps list          Limit write rate (bytes per second) to a device
                                     (default [])
    --device-write-iops list         Limit write rate (IO per second) to a device (default [])
    --disable-content-trust          Skip image verification (default true)
    --dns list                       Set custom DNS servers
    --dns-option list                Set DNS options
    --dns-search list                Set custom DNS search domains
    --domainname string              Container NIS domain name
    --entrypoint string              Overwrite the default ENTRYPOINT of the image
-e, --env list                       Set environment variables
    --env-file list                  Read in a file of environment variables
    --expose list                    Expose a port or a range of ports
    --gpus gpu-request               GPU devices to add to the container ('all' to pass all
                                     GPUs)
    --group-add list                 Add additional groups to join
    --health-cmd string              Command to run to check health
    --health-interval duration       Time between running the check (ms|s|m|h) (default 0s)
    --health-retries int             Consecutive failures needed to report unhealthy
    --health-start-period duration   Start period for the container to initialize before
                                     starting health-retries countdown (ms|s|m|h) (default 0s)
    --health-timeout duration        Maximum time to allow one check to run (ms|s|m|h)
                                     (default 0s)
-h, --hostname string                Container host name
    --init                           Run an init inside the container that forwards signals and
                                     reaps processes
-i, --interactive                    Keep STDIN open even if not attached
    --ip string                      IPv4 address (e.g., 172.30.100.104)
    --ip6 string                     IPv6 address (e.g., 2001:db8::33)
    --ipc string                     IPC namespace to use
    --isolation string               Container isolation technology
    --kernel-memory bytes            Kernel memory limit
-l, --label list                     Set meta data on a container
    --label-file list                Read in a line delimited file of labels
    --link list                      Add link to another container
    --link-local-ip list             Container IPv4/IPv6 link-local addresses
    --log-driver string              Logging driver for the container
    --log-opt list                   Log driver options
    --mac-address string             Container MAC address (e.g., 92:d0:c6:0a:29:33)
-m, --memory bytes                   Memory limit
    --memory-reservation bytes       Memory soft limit
    --memory-swap bytes              Swap limit equal to memory plus swap: '-1' to enable
                                     unlimited swap
    --memory-swappiness int          Tune container memory swappiness (0 to 100) (default -1)
    --mount mount                    Attach a filesystem mount to the container
    --name string                    Assign a name to the container
    --network network                Connect a container to a network
    --network-alias list             Add network-scoped alias for the container
    --no-healthcheck                 Disable any container-specified HEALTHCHECK
    --oom-kill-disable               Disable OOM Killer
    --oom-score-adj int              Tune host's OOM preferences (-1000 to 1000)
    --pid string                     PID namespace to use
    --pids-limit int                 Tune container pids limit (set -1 for unlimited)
    --privileged                     Give extended privileges to this container
```

```
-p, --publish list                  Publish a container's port(s) to the host
-P, --publish-all                   Publish all exposed ports to random ports
    --read-only                     Mount the container's root filesystem as read only
    --restart string                Restart policy to apply when a container exits
                                    (default "no")
    --rm                            Automatically remove the container when it exits
    --runtime string                Runtime to use for this container
    --security-opt list             Security Options
    --shm-size bytes                Size of /dev/shm
    --stop-signal string            Signal to stop a container (default "SIGTERM")
    --stop-timeout int              Timeout (in seconds) to stop a container
    --storage-opt list              Storage driver options for the container
    --sysctl map                    Sysctl options (default map[])
    --tmpfs list                    Mount a tmpfs directory
-t, --tty                           Allocate a pseudo-TTY
    --ulimit ulimit                 Ulimit options (default [])
-u, --user string                   Username or UID (format: <name|uid>[:<group|gid>])
    --userns string                 User namespace to use
    --uts string                    UTS namespace to use
-v, --volume list                   Bind mount a volume
    --volume-driver string          Optional volume driver for the container
    --volumes-from list             Mount volumes from the specified container(s)
-w, --workdir string                Working directory inside the container
```

1.1.6 diff

Inspect changes to files or directories on a container's filesystem

```
docker diff CONTAINER
```

(Equivalent to `docker container diff`)

1.1.7 events

Get real time events from the server

```
docker events [OPTIONS]
```

(Equivalent to `docker system events`)

Options:

```
-f, --filter filter   Filter output based on conditions provided
    --format string   Format the output using the given Go template
    --since string    Show all events created since timestamp
    --until string    Stream events until this timestamp
```

1.1.8 exec

Run a command in a running container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

(Equivalent to `docker container exec`)

Options:

```
-d, --detach              Detached mode: run command in the background
    --detach-keys string  Override the key sequence for detaching a container
-e, --env list            Set environment variables
-i, --interactive         Keep STDIN open even if not attached
    --privileged          Give extended privileges to the command
-t, --tty                 Allocate a pseudo-TTY
-u, --user string         Username or UID (format: <name|uid>[:<group|gid>])
-w, --workdir string      Working directory inside the container
```

1.1.9 export

Export a container's filesystem as a tar archive

```
docker export [OPTIONS] CONTAINER
```

(Equivalent to `docker container export`)

Options:

```
-o, --output string   Write to a file, instead of STDOUT
```

1.1.10 history

Show the history of an image

```
docker history [OPTIONS] IMAGE
```

(Equivalent to `docker image history`)

Options:

```
    --format string   Pretty-print images using a Go template
-H, --human           Print sizes and dates in human readable format (default true)
    --no-trunc        Don't truncate output
-q, --quiet           Only show numeric IDs
```

1.1.11 images

List images

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

(Equivalent to `docker image ls`)

Options:

```
-a, --all             Show all images (default hides intermediate images)
    --digests         Show digests
-f, --filter value    Filter output based on conditions provided
    --format string   Pretty-print images using a Go template
    --no-trunc        Don't truncate output
-q, --quiet           Only show numeric IDs
```

1.1.12 import

Import the contents from a tarball to create a filesystem image

```
docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]
```

(Equivalent to `docker image import`)

Options:

```
-c, --change value     Apply Dockerfile instruction to the created image
-m, --message string   Set commit message for imported image
```

1.1.13 info

Display system-wide information

```
docker info [OPTIONS]
```

(Equivalent to `docker system info`)

Options:

```
-f, --format string  Format the output using the given Go template
```

1.1.14 inspect

Return low-level information on Docker objects

```
docker inspect [OPTIONS] NAME|ID [NAME|ID...]
```

(Equivalent to `docker container inspect`)

Options:

```
-f, --format string  Format the output using the given Go template
-s, --size           Display total file sizes if the type is container
    --type string    Return JSON for specified type
```

1.1.15 kill

Kill one or more running containers

```
docker kill [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container kill`)

Options:

```
-s, --signal string   Signal to send to the container (default "KILL")
```

1.1.16 load

Load an image from a tar archive or STDIN

```
docker load [OPTIONS]
```

(Equivalent to `docker image load`)

Options:

```
-i, --input string   Read from tar archive file, instead of STDIN
-q, --quiet          Suppress the load output
```

1.1.17 login

Log in to a Docker registry.

If no server is specified, the default is defined by he daemon.

```
docker login [OPTIONS] [SERVER]
```

Options:

```
-p, --password string   Password
    --password-stdin    Take the password from stdin
-u, --username string   Username
```

1.1.18 logout

Log out from a Docker registry.

If no server is specified, the default is defined by he daemon.

```
docker logout [SERVER]
```

1.1.19 logs

Fetch the logs of a container

```
docker logs [OPTIONS] CONTAINER
```

(Equivalent to `docker container logs`)

Options:

```
    --details        Show extra details provided to logs
-f, --follow         Follow log output
    --since string   Show logs since timestamp (absolute or relative)
    --tail string    Number of lines to show from the end of the logs (default "all")
-t, --timestamps     Show timestamps
    --until string   Show logs before a timestamp (absolute or relative)
```

1.1.20 pause

Pause all processes within one or more containers

```
docker pause CONTAINER [CONTAINER...]
```

(Equivalent to `docker container pause`)

1.1.21 port

List port mappings or a specific mapping for the container

```
docker port CONTAINER [PRIVATE_PORT[/PROTO]]
```

(Equivalent to `docker container port`)

1.1.22 ps

List containers

```
docker ps [OPTIONS]
```

(Equivalent to `docker container ps`)

Options:

```
-a, --all            Show all containers (default shows just running)
-f, --filter filter  Filter output based on conditions provided
    --format string  Pretty-print containers using a Go template
-n, --last int       Show n last created containers (includes all states) (default -1)
-l, --latest         Show the latest created container (includes all states)
    --no-trunc       Don't truncate output
-q, --quiet          Only display numeric IDs
-s, --size           Display total file sizes
```

1.1.23 pull

Pull an image or a repository from a registry

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

(Equivalent to `docker image pull`)

Options:

```
-a, --all-tags                Download all tagged images in the repository
    --disable-content-trust   Skip image verification (default true)
-q, --quiet                   Suppress verbose output
```

1.1.24 push

Push an image or a repository to a registry

```
docker push [OPTIONS] NAME[:TAG]
```

(Equivalent to `docker image push`)

Options:

```
    --disable-content-trust   Skip image signing (default true)
```

1.1.25 rename

Rename a container

```
docker rename CONTAINER NEW_NAME
```

(Equivalent to `docker container rename`)

1.1.26 restart

Restart one or more containers

```
docker restart [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container restart`)

Options:

```
-t, --time int   Seconds to wait for stop before killing the container (default 10)
```

1.1.27 rm

Remove one or more containers

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container rm`)

Options:

```
-f, --force     Force the removal of a running container (uses SIGKILL)
-l, --link      Remove the specified link
-v, --volumes   Remove the volumes associated with the container
```

1.1.28 rmi

Remove one or more images

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

(Equivalent to: `docker image rm`)

Options:

```
-f, --force      Force removal of the image
    --no-prune   Do not delete untagged parents
```

1.1.29 run

Run a command in a new container

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

(Equivalent to `docker container run`)

Options:

```
    --add-host list                Add a custom host-to-IP mapping (host:ip)
-a, --attach list                  Attach to STDIN, STDOUT or STDERR
    --blkio-weight uint16          Block IO (relative weight), between 10 and 1000, or 0 to
                                   disable (default 0)
```

```
    --blkio-weight-device list       Block IO weight (relative device weight) (default [])
    --cap-add list                   Add Linux capabilities
    --cap-drop list                  Drop Linux capabilities
    --cgroup-parent string           Optional parent cgroup for the container
    --cidfile string                 Write the container ID to the file
    --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler) period
    --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler) quota
    --cpu-rt-period int              Limit CPU real-time period in microseconds
    --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
-c, --cpu-shares int                 CPU shares (relative weight)
    --cpus decimal                   Number of CPUs
    --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
    --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
-d, --detach                         Run container in background and print container ID
    --detach-keys string             Override the key sequence for detaching a container
    --device list                    Add a host device to the container
    --device-cgroup-rule list        Add a rule to the cgroup allowed devices list
    --device-read-bps list           Limit read rate (bytes per second) from a device
                                     (default [])
    --device-read-iops list          Limit read rate (IO per second) from a device (default [])
    --device-write-bps list          Limit write rate (bytes per second) to a device
                                     (default [])
    --device-write-iops list         Limit write rate (IO per second) to a device (default [])
    --disable-content-trust          Skip image verification (default true)
    --dns list                       Set custom DNS servers
    --dns-option list                Set DNS options
    --dns-search list                Set custom DNS search domains
    --domainname string              Container NIS domain name
    --entrypoint string              Overwrite the default ENTRYPOINT of the image
-e, --env list                       Set environment variables
    --env-file list                  Read in a file of environment variables
    --expose list                    Expose a port or a range of ports
    --gpus gpu-request               GPU devices to add to the container ('all' to pass all
                                     GPUs)
    --group-add list                 Add additional groups to join
    --health-cmd string              Command to run to check health
    --health-interval duration       Time between running the check (ms|s|m|h) (default 0s)
    --health-retries int             Consecutive failures needed to report unhealthy
    --health-start-period duration   Start period for the container to initialize before
                                     starting health-retries countdown (ms|s|m|h) (default 0s)
    --health-timeout duration        Maximum time to allow one check to run (ms|s|m|h)
                                     (default 0s)
-h, --hostname string                Container host name
    --init                           Run an init inside the container that forwards signals
                                     and reaps processes
-i, --interactive                    Keep STDIN open even if not attached
    --ip string                      IPv4 address (e.g., 172.30.100.104)
    --ip6 string                     IPv6 address (e.g., 2001:db8::33)
    --ipc string                     IPC mode to use
    --isolation string               Container isolation technology
    --kernel-memory bytes            Kernel memory limit
-l, --label list                     Set meta data on a container
    --label-file list                Read in a line delimited file of labels
    --link list                      Add link to another container
    --link-local-ip list             Container IPv4/IPv6 link-local addresses
    --log-driver string              Logging driver for the container
    --log-opt list                   Log driver options
    --mac-address string             Container MAC address (e.g., 92:d0:c6:0a:29:33)
-m, --memory bytes                   Memory limit
    --memory-reservation bytes       Memory soft limit
    --memory-swap bytes              Swap limit equal to memory plus swap: '-1' to enable
                                     unlimited swap
    --memory-swappiness int          Tune container memory swappiness (0 to 100) (default -1)
    --mount mount                    Attach a filesystem mount to the container
    --name string                    Assign a name to the container
    --network network                Connect a container to a network
```

```
    --network-alias list            Add network-scoped alias for the container
    --no-healthcheck                Disable any container-specified HEALTHCHECK
    --oom-kill-disable              Disable OOM Killer
    --oom-score-adj int             Tune host's OOM preferences (-1000 to 1000)
    --pid string                    PID namespace to use
    --pids-limit int                Tune container pids limit (set -1 for unlimited)
    --privileged                    Give extended privileges to this container
-p, --publish list                  Publish a container's port(s) to the host
-P, --publish-all                   Publish all exposed ports to random ports
    --read-only                     Mount the container's root filesystem as read only
    --restart string                Restart policy to apply when a container exits
                                    (default "no")
    --rm                            Automatically remove the container when it exits
    --runtime string                Runtime to use for this container
    --security-opt list             Security Options
    --shm-size bytes                Size of /dev/shm
    --sig-proxy                     Proxy received signals to the process (default true)
    --stop-signal string            Signal to stop a container (default "SIGTERM")
    --stop-timeout int              Timeout (in seconds) to stop a container
    --storage-opt list              Storage driver options for the container
    --sysctl map                    Sysctl options (default map[])
    --tmpfs list                    Mount a tmpfs directory
-t, --tty                           Allocate a pseudo-TTY
    --ulimit ulimit                 Ulimit options (default [])
-u, --user string                   Username or UID (format: <name|uid>[:<group|gid>])
    --userns string                 User namespace to use
    --uts string                    UTS namespace to use
-v, --volume list                   Bind mount a volume
    --volume-driver string          Optional volume driver for the container
    --volumes-from list             Mount volumes from the specified container(s)
-w, --workdir string                Working directory inside the container
```

1.1.30 save

Save one or more images to a tar archive (streamed to STDOUT by default)

```
docker save [OPTIONS] IMAGE [IMAGE...]
```

(Equivalent to `docker image save`)

Options:

```
-o, --output string   Write to a file, instead of STDOUT
```

1.1.31 search

Search the Docker Hub for images

```
docker search [OPTIONS] TERM
```

Options:

```
-f, --filter filter  Filter output based on conditions provided
    --format string  Pretty-print search using a Go template
    --limit int      Max number of search results (default 25)
    --no-trunc       Don't truncate output
```

1.1.32 start

Start one or more stopped containers

```
docker start [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container start`)

Options:

```
-a, --attach              Attach STDOUT/STDERR and forward signals
    --detach-keys string  Override the key sequence for detaching a container
-i, --interactive         Attach container's STDIN
```

1.1.33 stats

Display a live stream of container(s) resource usage statistics

```
docker stats [OPTIONS] [CONTAINER...]
```

(Equivalent to `docker container stats`)

Options:

```
-a, --all            Show all containers (default shows just running)
    --format string  Pretty-print images using a Go template
    --no-stream      Disable streaming stats and only pull the first result
    --no-trunc       Do not truncate output
```

1.1.34 stop

Stop one or more running containers

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container stop`)

Options:

```
-t, --time int   Seconds to wait for stop before killing it (default 10)
```

1.1.35 tag

Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

(Equivalent to `docker image tag`)

1.1.36 top

Display the running processes of a container

```
docker top CONTAINER [ps OPTIONS]
```

(Equivalent to `docker container top`)

1.1.37 unpause

Unpause all processes within one or more containers

```
docker unpause CONTAINER [CONTAINER...]
```

(Equivalent to `docker container unpause`)

1.1.38 update

Update configuration of one or more containers

```
docker update [OPTIONS] CONTAINER [CONTAINER...]
```

(Equivalent to `docker container update`)

Options:

```
    --blkio-weight uint16        Block IO (relative weight), between 10 and 1000, or 0 to
                                 disable (default 0)
```

```
    --cpu-period int              Limit CPU CFS (Completely Fair Scheduler) period
    --cpu-quota int               Limit CPU CFS (Completely Fair Scheduler) quota
    --cpu-rt-period int           Limit the CPU real-time period in microseconds
    --cpu-rt-runtime int          Limit the CPU real-time runtime in microseconds
-c, --cpu-shares int              CPU shares (relative weight)
    --cpus decimal                Number of CPUs
    --cpuset-cpus string          CPUs in which to allow execution (0-3, 0,1)
    --cpuset-mems string          MEMs in which to allow execution (0-3, 0,1)
    --kernel-memory bytes         Kernel memory limit
-m, --memory bytes                Memory limit
    --memory-reservation bytes    Memory soft limit
    --memory-swap bytes           Swap limit equal to memory plus swap: '-1' to enable
                                  unlimited swap
    --pids-limit int              Tune container pids limit (set -1 for unlimited)
    --restart string              Restart policy to apply when a container exits
```

1.1.39 version

Show the Docker version information

```
docker version [OPTIONS]
```

Options:

```
-f, --format string       Format the output using the given Go template
    --kubeconfig string   Kubernetes config file
```

1.1.40 wait

Block until one or more containers stop, then print their exit codes

```
docker wait CONTAINER [CONTAINER...]
```

(Equivalent to `docker container wait`)

1.2 Management commands

Underlined commands have been discussed before in detail.

1.2.1 builder

Manage builds

```
docker builder COMMAND
```

Commands:

```
build       Build an image from a Dockerfile
prune       Remove build caches
```

1.2.2 config

Manage Docker configs

```
docker config COMMAND
```

Commands:

```
  create      Create a config from a file or STDIN
  inspect     Display detailed information on one or more configs
  ls          List configs
  rm          Remove one or more configs
```

1.2.3 container

Manage containers

```
docker container COMMAND
```

Commands: Most of these `container` commands are **identical to** `docker` **full commands** from section 1.1

```
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes
```

1.2.4 context

Manage contexts

```
docker context COMMAND
```

Commands:

```
  create      Create a context
  export      Export a context to a tar or kubeconfig file
  import      Import a context from a tar or zip file
  inspect     Display detailed information on one or more contexts
  ls          List contexts
  rm          Remove one or more contexts
  update      Update a context
  use         Set the current docker context
```

1.2.5 engine

Command availability limited to an existing Docker enterprise license

```
docker engine COMMAND
```

Commands:

```
  activate    Create a context
  check       Export a context to a tar or kubeconfig file
```

```
  update      Import a context from a tar or zip file
  inspect     Display detailed information on one or more contexts
```

1.2.6 image

Manage images

```
docker image COMMAND
```

Commands:

```
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

Many of these `container` commands are **identical to `docker` full commands** from section 1.1, so only new ones will be described here.

1.2.6.1 image inspect

Display detailed information on one or more images

```
docker image inspect [OPTIONS] IMAGE [IMAGE...]
```

Options:

```
-f, --format string   Format the output using the given Go template
```

1.2.6.2 image prune

Remove unused images

```
docker image prune [OPTIONS]
```

Options:

```
-a, --all             Remove all unused images, not just dangling ones
    --filter filter   Provide filter values (e.g. 'until=<timestamp>')
-f, --force           Do not prompt for confirmation
```

1.2.7 network

Manage networks

```
docker network COMMAND
```

Commands:

```
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
```

```
  rm          Remove one or more networks
```

1.2.7.1 network connect

Connect a container to a network

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

Options:

```
    --alias strings            Add network-scoped alias for the container
    --driver-opt strings       driver options for the network
    --ip string                IP Address (e.g., 172.30.100.104)
    --ip6 string               IPv6 Address (e.g., 2001:db8::33)
    --link list                Add link to another container
    --link-local-ip strings    Add a link-local address for the container
```

1.2.7.2 network create

Create a network

```
docker network create [OPTIONS] NETWORK
```

Options:

```
    --attachable           Enable manual container attachment
    --aux-address value    Auxiliary IPv4 or IPv6 addresses used by Network driver
                           (default map[])
    --config-from string   The network from which copying the configuration
    --config-only          Create a configuration only network
-d, --driver string        Driver to manage the Network (default "bridge")
    --gateway strings      IPv4 or IPv6 Gateway for the master subnet
    --ingress              Create swarm routing-mesh network
    --internal             Restrict external access to the network
    --ip-range strings     Allocate container ip from a sub-range
    --ipam-driver string   IP Address Management Driver (default "default")
    --ipam-opt value       Set IPAM driver specific options (default map[])
    --ipv6                 Enable IPv6 networking
    --label list           Set metadata on a network
-o, --opt map              Set driver specific options (default map[])
    --scope string         Control the network's scope
    --subnet strings       Subnet in CIDR format that represents a network segment
```

1.2.7.3 network disconnect

Disconnect a container from a network

```
docker network disconnect [OPTIONS] NETWORK CONTAINER
```

Options:

```
-f, --force   Force the container to disconnect from a network
```

1.2.7.4 network inspect

Display detailed information on one or more networks

```
docker network inspect [OPTIONS] NETWORK [NETWORK...]
```

Options:

```
-f, --format string   Format the output using the given Go template
-v, --verbose         Verbose output for diagnostics
```

1.2.7.5 network ls

List networks

```
docker network ls [OPTIONS]
```

Aliases:

```
ls, list
```

Options:

```
-f, --filter filter  Provide filter values (e.g. 'driver=bridge')
    --format string  Pretty-print networks using a Go template
    --no-trunc       Do not truncate the output
-q, --quiet          Only display network IDs
```

1.2.7.6 network prune

Remove all unused networks

```
docker network network prune [OPTIONS]
```

Options:

```
    --filter filter   Provide filter values (e.g. 'until=<timestamp>')
-f, --force           Do not prompt for confirmation
```

1.2.7.1 network rm

Remove one or more networks

```
docker network rm NETWORK [NETWORK...]
```

Aliases:

```
rm, remove
```

~~1.2.8 node~~

Manage Swarm nodes

```
docker node COMMAND
```

*(**`docker node`** **subcommands have been excluded** from this text because swarm mode is beyond our targets)*

Commands:

```
  demote      Demote one or more nodes from manager in the swarm
  inspect     Display detailed information on one or more nodes
  ls          List nodes in the swarm
  promote     Promote one or more nodes to manager in the swarm
  ps          List tasks running on one or more nodes, defaults to current node
  rm          Remove one or more nodes from the swarm
  update      Update a node
```

~~1.2.9 plugin~~

Manage plugins

```
docker plugin COMMAND
```

*(**`docker plugin`** **subcommands have been excluded** from this text)*

Commands:

```
  create      Create a plugin from a rootfs and configuration. Plugin data directory must
              contain config.json and rootfs directory.
  disable     Disable a plugin
  enable      Enable a plugin
  inspect     Display detailed information on one or more plugins
  install     Install a plugin
  ls          List plugins
  push        Push a plugin to a registry
  rm          Remove one or more plugins
  set         Change settings for a plugin
  upgrade     Upgrade an existing plugin
```

~~1.2.10 secret~~

Manage Docker secrets

```
docker secret COMMAND
```

*(`docker secret` **subcommands have been excluded** from this text)*

Commands:

```
  create      Create a secret from a file or STDIN as content
  inspect     Display detailed information on one or more secrets
  ls          List secrets
  rm          Remove one or more secrets
```

1.2.11 service

Manage services

```
docker service COMMAND
```

Commands:

```
  create      Create a new service
  inspect     Display detailed information on one or more services
  logs        Fetch the logs of a service or task
  ls          List services
  ps          List the tasks of a service
  rm          Remove one or more services
  rollback    Revert changes to a service's configuration
  scale       Scale one or multiple replicated services
  update      Update a service
```

1.2.11.1 service create

Create a new service

```
docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Options:

```
    --config config                  Specify configurations to expose to the service
    --constraint list                Placement constraints
    --container-label list           Container labels
    --credential-spec credential-spec  Credential spec for managed service account (Windows
                                     only)
-d, --detach                         Exit immediately instead of waiting for the service to
                                     converge (default true)
    --dns list                       Set custom DNS servers
    --dns-option list                Set DNS options
    --dns-search list                Set custom DNS search domains
    --endpoint-mode string           Endpoint mode (vip or dnsrr) (default "vip")
    --entrypoint command             Overwrite the default ENTRYPOINT of the image
-e, --env list                       Set environment variables
```

```
    --env-file list                  Read in a file of environment variables
    --generic-resource list          User defined resources
    --group list                     Set one or more supplementary user groups for the
                                     container
    --health-cmd string              Command to run to check health
    --health-interval duration       Time between running the check (ms|s|m|h)
    --health-retries int             Consecutive failures needed to report unhealthy
    --health-start-period duration   Start period for the container to initialize before
                                     counting retries towards unstable (ms|s|m|h)
    --health-timeout duration        Maximum time to allow one check to run (ms|s|m|h)
    --host list                      Set one or more custom host-to-IP mappings (host:ip)
    --hostname string                Container hostname
    --init                           Use an init inside each service container to forward
                                     signals and reap processes
    --isolation string               Service container isolation mode
-l, --label list                     Service labels
    --limit-cpu decimal              Limit CPUs
    --limit-memory bytes             Limit Memory
    --log-driver string              Logging driver for service
    --log-opt list                   Logging driver options
    --mode string                    Service mode (replicated or global)
                                     (default "replicated")
    --mount mount                    Attach a filesystem mount to the service
    --name string                    Service name
    --network network                Network attachments
    --no-healthcheck                 Disable any container-specified HEALTHCHECK
    --no-resolve-image               Do not query the registry to resolve image digest and
                                     supported platforms
    --placement-pref pref            Add a placement preference
-p, --publish port                   Publish a port as a node port
-q, --quiet                          Suppress progress output
    --read-only                      Mount the container's root filesystem as read only
    --replicas uint                  Number of tasks
    --replicas-max-per-node uint     Maximum number of tasks per node (default 0 = unlimited)
    --reserve-cpu decimal            Reserve CPUs
    --reserve-memory bytes           Reserve Memory
    --restart-condition string       Restart when condition is met ("none"|"on-failure"|"any")
                                     (default "any")
    --restart-delay duration         Delay between restart attempts (ns|us|ms|s|m|h)
                                     (default 5s)
    --restart-max-attempts uint      Maximum number of restarts before giving up
    --restart-window duration        Window used to evaluate the restart policy
                                     (ns|us|ms|s|m|h)
    --rollback-delay duration        Delay between task rollbacks (ns|us|ms|s|m|h)
                                     (default 0s)
    --rollback-failure-action string Action on rollback failure ("pause"|"continue")
                                     (default "pause")
    --rollback-max-failure-ratio float Failure rate to tolerate during a rollback (default 0)
    --rollback-monitor duration      Duration after each task rollback to monitor for
                                     failure (ns|us|ms|s|m|h) (default 5s)
    --rollback-order string          Rollback order ("start-first"|"stop-first")
                                     (default "stop-first")
    --rollback-parallelism uint      Maximum number of tasks rolled back simultaneously
                                     (0 to roll back all at once) (default 1)
    --secret secret                  Specify secrets to expose to the service
    --stop-grace-period duration     Time to wait before force killing a container
                                     (ns|us|ms|s|m|h) (default 10s)
    --stop-signal string             Signal to stop the container
-t, --tty                            Allocate a pseudo-TTY
    --update-delay duration          Delay between updates (ns|us|ms|s|m|h) (default 0s)
    --update-failure-action string   Action on update failure
                                     ("pause"|"continue"|"rollback") (default "pause")
    --update-max-failure-ratio float Failure rate to tolerate during an update (default 0)
    --update-monitor duration        Duration after each task update to monitor for failure
                                     (ns|us|ms|s|m|h) (default 5s)
    --update-order string            Update order ("start-first"|"stop-first")
```

```
                                  (default "stop-first")
    --update-parallelism uint     Maximum number of tasks updated simultaneously
                                  (0 to update all at once) (default 1)
-u, --user string                 Username or UID (format: <name|uid>[:<group|gid>])
    --with-registry-auth          Send registry authentication details to swarm agents
-w, --workdir string              Working directory inside the Container
```

1.2.11.2 service inspect

Display detailed information on one or more services

```
docker service inspect [OPTIONS] SERVICE [SERVICE...]
```

Options:

```
-f, --format string   Format the output using the given Go template
    --pretty          Print the information in a human friendly format.
```

1.2.11.3 service logs

Fetch the logs of a service or task

```
docker service logs [OPTIONS] SERVICE|TASK
```

Options:

```
    --details         Show extra details provided to logs
-f, --follow          Follow log output
    --no-resolve      Do not map IDs to Names in output
    --no-task-ids     Do not include task IDs in output
    --no-trunc        Do not truncate output
    --raw             Do not neatly format logs
    --since string    Show logs since timestamp (e.g. 2013-01-02T13:23:37) or relative
                      (e.g. 42m for 42 minutes)
    --tail string     Number of lines to show from the end of the logs (default "all")
-t, --timestamps      Show timestamps
```

1.2.11.4 service ls

List services

```
docker service ls [OPTIONS]
```

Aliases:

```
ls, list
```

Options:

```
-f, --filter value    Filter output based on conditions provided
    --format string   Pretty-print services using a Go template
-q, --quiet           Only display IDs
```

1.2.11.5 service ps

List the tasks of one or more services

```
docker service ps [OPTIONS] SERVICE
```

Options:

```
-f, --filter filter   Filter output based on conditions provided
    --format string   Pretty-print tasks using a Go template
    --no-resolve      Do not map IDs to Names
    --no-trunc        Do not truncate output
```

```
-q, --quiet          Only display task IDs
```

1.2.11.6 service rm

Remove one or more services

```
docker service rm SERVICE [SERVICE...]
```

Aliases:

```
rm, remove
```

1.2.11.7 service rollback

Revert changes to a service's configuration

```
docker service rollback [OPTIONS] SERVICE
```

Options:

```
-d, --detach   Exit immediately instead of waiting for the service to converge
-q, --quiet    Suppress progress output
```

1.2.11.8 service scale

Scale one or multiple replicated services

```
docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]
```

Options:

```
-d, --detach   Exit immediately instead of waiting for the service to converge
```

1.2.11.9 service update

Update a service

```
docker service update [OPTIONS] SERVICE
```

Options:

```
    --args command                    Service command args
    --config-add config               Add or update a config file on a service
    --config-rm list                  Remove a configuration file
    --constraint-add list             Add or update a placement constraint
    --constraint-rm list              Remove a constraint
    --container-label-add list        Add or update a container label
    --container-label-rm list         Remove a container label by its key
    --credential-spec credential-spec Credential spec for managed service account
                                      (Windows only)
-d, --detach                          Exit immediately instead of waiting for the service to
                                      converge
    --dns-add list                    Add or update a custom DNS server
    --dns-option-add list             Add or update a DNS option
    --dns-option-rm list              Remove a DNS option
    --dns-rm list                     Remove a custom DNS server
    --dns-search-add list             Add or update a custom DNS search domain
    --dns-search-rm list              Remove a DNS search domain
    --endpoint-mode string            Endpoint mode (vip or dnsrr)
    --entrypoint command              Overwrite the default ENTRYPOINT of the image
    --env-add list                    Add or update an environment variable
    --env-rm list                     Remove an environment variable
    --force                           Force update even if no changes require it
    --generic-resource-add list       Add a Generic resource
    --generic-resource-rm list        Remove a Generic resource
```

```
    --group-add list                  Add an additional supplementary user group to
                                      the container
    --group-rm list                   Remove a previously added supplementary user group from
                                      the container
    --health-cmd string               Command to run to check health
    --health-interval duration        Time between running the check (ms|s|m|h)
    --health-retries int              Consecutive failures needed to report unhealthy
    --health-start-period duration    Start period for the container to initialize before
                                      counting retries towards unstable (ms|s|m|h)
    --health-timeout duration         Maximum time to allow one check to run (ms|s|m|h)
    --host-add list                   Add or update a custom host-to-IP mapping (host:ip)
    --host-rm list                    Remove a custom host-to-IP mapping (host:ip)
    --hostname string                 Container hostname
    --image string                    Service image tag
    --init                            Use an init inside each service container to forward
                                      signals and reap processes
    --isolation string                Service container isolation mode
    --label-add list                  Add or update a service label
    --label-rm list                   Remove a label by its key
    --limit-cpu decimal               Limit CPUs
    --limit-memory bytes              Limit Memory
    --log-driver string               Logging driver for service
    --log-opt list                    Logging driver options
    --mount-add mount                 Add or update a mount on a service
    --mount-rm list                   Remove a mount by its target path
    --network-add network             Add a network
    --network-rm list                 Remove a network
    --no-healthcheck                  Disable any container-specified HEALTHCHECK
    --no-resolve-image                Do not query the registry to resolve image digest and
                                      supported platforms
    --placement-pref-add pref         Add a placement preference
    --placement-pref-rm pref          Remove a placement preference
    --publish-add port                Add or update a published port
    --publish-rm port                 Remove a published port by its target port
-q, --quiet                           Suppress progress output
    --read-only                       Mount the container's root filesystem as read only
    --replicas uint                   Number of tasks
    --replicas-max-per-node uint      Maximum number of tasks per node (default 0 = unlimited)
    --reserve-cpu decimal             Reserve CPUs
    --reserve-memory bytes            Reserve Memory
    --restart-condition string        Restart when condition is met ("none"|"on-failure"|"any")
    --restart-delay duration          Delay between restart attempts (ns|us|ms|s|m|h)
    --restart-max-attempts uint       Maximum number of restarts before giving up
    --restart-window duration         Window used to evaluate the restart policy
                                      (ns|us|ms|s|m|h)
    --rollback                        Rollback to previous specification
    --rollback-delay duration         Delay between task rollbacks (ns|us|ms|s|m|h)
    --rollback-failure-action string  Action on rollback failure ("pause"|"continue")
    --rollback-max-failure-ratio float Failure rate to tolerate during a rollback
    --rollback-monitor duration       Duration after each task rollback to monitor for
                                      failure (ns|us|ms|s|m|h)
    --rollback-order string           Rollback order ("start-first"|"stop-first")
    --rollback-parallelism uint       Maximum number of tasks rolled back simultaneously
                                      (0 to roll back all at once)
    --secret-add secret               Add or update a secret on a service
    --secret-rm list                  Remove a secret
    --stop-grace-period duration      Time to wait before force killing a container
                                      (ns|us|ms|s|m|h)
    --stop-signal string              Signal to stop the container
    --sysctl-add list                 Add or update a Sysctl option
    --sysctl-rm list                  Remove a Sysctl option
-t, --tty                             Allocate a pseudo-TTY
    --update-delay duration           Delay between updates (ns|us|ms|s|m|h)
    --update-failure-action string    Action on update failure ("pause"|"continue"|"rollback")
    --update-max-failure-ratio float  Failure rate to tolerate during an update
    --update-monitor duration         Duration after each task update to monitor for failure
```

```
                                  (ns|us|ms|s|m|h)
    --update-order string         Update order ("start-first"|"stop-first")
    --update-parallelism uint     Maximum number of tasks updated simultaneously
                                  (0 to update all at once)
-u, --user string                 Username or UID (format: <name|uid>[:<group|gid>])
    --with-registry-auth          Send registry authentication details to swarm agents
-w, --workdir string              Working directory inside the container
```

1.2.12 stack

Manage Docker stacks

```
docker stack [OPTIONS] COMMAND
```

Options:

```
    --orchestrator string     Orchestrator to use (swarm|kubernetes|all)
```

Commands:

```
  deploy      Deploy a new stack or update an existing stack
  ls          List stacks
  ps          List the tasks in the stack
  rm          Remove one or more stacks
  services    List the services in the stack
```

1.2.12.1 stack deploy

Deploy a new stack or update an existing stack

```
docker stack deploy [OPTIONS] STACK
```

Aliases:

```
deploy, up
```

Options:

```
    --bundle-file string     Path to a Distributed Application Bundle file
 c, --compose-file string    Path to a Compose file, or "-" to read from stdin
    --orchestrator string    Orchestrator to use (swarm|kubernetes|all)
    --prune                  Prune services that are no longer referenced
    --resolve-image string   Query the registry to resolve image digest and supported platforms
                             ("always"|"changed"|"never") (default "always")
    --with-registry-auth     Send registry authentication details to Swarm agents
```

1.2.12.2 stack ls

List stacks

```
docker stack ls
```

Aliases:

```
ls, list
```

Options:

```
    --format string         Pretty-print stacks using a Go template
    --orchestrator string   Orchestrator to use (swarm|kubernetes|all)
```

1.2.12.3 stack ps

List the tasks in the stack

```
docker stack ps [OPTIONS] STACK
```

Options:

```
-f, --filter filter         Filter output based on conditions provided
    --format string         Pretty-print tasks using a Go template
    --no-resolve            Do not map IDs to Names
    --no-trunc              Do not truncate output
    --orchestrator string   Orchestrator to use (swarm|kubernetes|all)
-q, --quiet                 Only display task IDs
```

1.2.12.4 stack rm

Remove one or more stacks

```
docker stack rm [OPTIONS] STACK [STACK...]
```

Aliases:

```
rm, remove, down
```

Options:

```
    --orchestrator string   Orchestrator to use (swarm|kubernetes|all)
```

1.2.12.5 stack services

List the services in the stack

```
docker stack services [OPTIONS] STACK
```

Options:

```
-f, --filter filter         Filter output based on conditions provided
    --format string         Pretty-print services using a Go template
    --orchestrator string   Orchestrator to use (swarm|kubernetes|all)
-q, --quiet                 Only display IDs
```

~~1.2.13 swarm~~

Manage Swarm

```
docker swarm COMMAND
```

*(**swarm commands have been excluded** from this text because swarm mode is beyond our targets)*

Commands:

```
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm
```

1.2.14 system

Manage Docker

```
docker system COMMAND
```

Commands:

```
  df          Show docker disk usage
  events      Get real time events from the server
  info        Display system-wide information
  prune       Remove unused data
```

1.2.14.1 system df

Show docker disk usage

```
docker system df [OPTIONS]
```

Options:

```
    --format string   Pretty-print images using a Go template
-v, --verbose         Show detailed information on space usage
```

1.2.14.2 system events

Get real time events from the server

```
docker system events [OPTIONS]
```

(Equivalent to `docker events`)

Options:

```
-f, --filter filter   Filter output based on conditions provided
    --format string   Format the output using the given Go template
    --since string    Show all events created since timestamp
    --until string    Stream events until this timestamp
```

1.2.14.3 system info

Display system-wide information

```
docker system info [OPTIONS]
```

(Equivalent to `docker info`)

Options:

```
-f, --format string   Format the output using the given Go template
```

1.2.14.4 system prune

Remove unused data

```
docker system prune [OPTIONS]
```

Options:

```
-a, --all             Remove all unused images not just dangling ones
    --filter filter   Provide filter values (e.g. 'label=<key>=<value>')
-f, --force           Do not prompt for confirmation
    --volumes         Prune volumes
```

1.2.15 volume

Manage volumes

```
docker volume COMMAND
```

Commands:

```
  create      Create a volume
  inspect     Display detailed information on one or more volumes
```

```
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

1.2.15.1 volume create

Create a volume

```
docker volume create [OPTIONS] [VOLUME]
```

Options:

```
-d, --driver string   Specify volume driver name (default "local")
    --label list      Set metadata for a volume
-o, --opt map         Set driver specific options (default map[])
```

1.2.15.2 volume inspect

Display detailed information on one or more volumes

```
docker volume inspect [OPTIONS] VOLUME [VOLUME...]
```

Options:

```
-f, --format string   Format the output using the given Go template
```

1.2.15.3 volume ls

List volumes

```
docker volume ls [OPTIONS]
```

Aliases:

```
ls, list
```

Options:

```
-f, --filter value   Provide filter values (i.e. 'dangling=true')
    --format string  Format the output using a Go template
-q, --quiet          Only display volume names
```

1.2.15.1 volume prune

Remove all unused local volumes

```
docker volume prune [OPTIONS]
```

Options:

```
    --filter filter   Provide filter values (e.g. 'label=<label>')
-f, --force           Do not prompt for confirmation
```

1.2.15.2 volume rm

Remove one or more volumes

```
docker volume rm [OPTIONS] VOLUME [VOLUME...]
```

Aliases:

```
rm, remove
```

Options:

```
-f, --force          Force the removal of one or more volumes
```

2 DOCKERFILE REFERENCE

From https://docs.docker.com/engine/reference/builder/

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a Dockerfile. When you are done reading this page, refer to the *Dockerfile Best Practices* for a tip-oriented guide.

2.1 Usage

The docker build command builds an image from a Dockerfile and a *context*. The build's context is the files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a Git repository location.

The build context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. This example shows a build command that uses the current directory (.) as build context:

```
$ docker build .
Sending build context to Docker daemon  6.51 MB
...
```

The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it's best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.

***Warning**: Do not use your root directory, /, as the PATH as it causes the build to transfer the entire contents of your hard drive to the Docker daemon.*

To use a file in the build context, the Dockerfile refers to the file specified in an instruction, for example, a COPY instruction. To increase the build's performance, exclude files and directories by adding a .dockerignore file to the context directory. For information about how to create a .dockerignore file see the documentation on this page.

Traditionally, the Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

```
$ docker build -f /path/to/a/Dockerfile .
```

You can specify a repository and tag at which to save the new image if the build succeeds:

```
$ docker build -t shykes/myapp .
```

To tag the image into multiple repositories after the build, add multiple -t parameters when you run the build command:

```
$ docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .
```

Before the Docker daemon runs the instructions in the Dockerfile, it performs a preliminary validation of the Dockerfile and returns an error if the syntax is incorrect:

```
$ docker build -t test/myapp .

[+] Building 0.3s (2/2) FINISHED
 => [internal] load build definition from Dockerfile                    0.1s
 => => transferring dockerfile: 60B                                     0.0s
 => [internal] load .dockerignore                                       0.1s
 => => transferring context: 2B                                         0.0s
error: failed to solve: rpc error: code = Unknown desc = failed to solve with frontend
dockerfile.v0: failed to create LLB definition:
dockerfile parse error line 2: unknown instruction: RUNCMD
```

The Docker daemon runs the instructions in the `Dockerfile` one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image. The Docker daemon will automatically clean up the context you sent.

Note that each instruction is run independently, and causes a new image to be created - so `RUN cd /tmp` will not have any effect on the next instructions.

Whenever possible, Docker will re-use the intermediate images (cache), to accelerate the `docker build` process significantly. This is indicated by the `Using cache` message in the console output:

```
$ docker build -t svendowideit/ambassador .

[+] Building 0.7s (6/6) FINISHED
 => [internal] load build definition from Dockerfile                    0.1s
 => => transferring dockerfile: 286B                                    0.0s
 => [internal] load .dockerignore                                       0.1s
 => => transferring context: 2B                                         0.0s
 => [internal] load metadata for docker.io/library/alpine:3.2           0.4s
 => CACHED [1/2] FROM docker.io/library/alpine:3.2@sha256:e9a2035f9d0d7ce  0.0s
 => CACHED [2/2] RUN apk add --no-cache socat                           0.0s
 => exporting to image                                                  0.0s
 => => exporting layers                                                 0.0s
 => => writing image sha256:1affb80ca37018ac12067fa2af38cc5bcc2a8f09963de  0.0s
 => => naming to docker.io/svendowideit/ambassador                      0.0s
```

Build cache is only used from images that have a local parent chain. This means that these images were created by previous builds or the whole chain of images was loaded with `docker load`. If you wish to use build cache of a specific image you can specify it with `--cache-from` option. Images specified with `--cache-from` do not need to have a parent chain and may be pulled from other registries.

By default, the build cache is based on results from previous builds on the machine on which you are building. The `--cache-from` option also allows you to use a build-cache that's distributed through an image registry refer to the specifying external cache sources section in the docker build command reference.

When you're done with your build, you're ready to look into scanning your image with `docker scan`, and pushing your image to Docker Hub.

2.2 Buildkit

Starting with version 18.09, Docker supports a new backend for executing your builds that is provided by the `moby/buildkit` project. The BuildKit backend provides many benefits compared to the old implementation. For example, BuildKit can:

- Detect and skip executing unused build stages
- Parallelize building independent build stages
- Incrementally transfer only the changed files in your build context between builds
- Detect and skip transferring unused files in your build context
- Use external Dockerfile implementations with many new features
- Avoid side-effects with rest of the API (intermediate images and containers)
- Prioritize your build cache for automatic pruning

To use the BuildKit backend, you need to set an environment variable `DOCKER_BUILDKIT=1` on the CLI before invoking `docker build`

2.3 Format

Here is the format of the `Dockerfile`:

```
# Comment
INSTRUCTION arguments
```

The instruction is not case-sensitive. However, convention is for them to be UPPERCASE to distinguish them from arguments more easily.

Docker runs instructions in a `Dockerfile` in order. A `Dockerfile` **must begin with a "FROM" instruction**. This may be after parser directives, comments, and globally scoped ARGs. The `FROM` instruction specifies the *Parent Image* from which you are building. `FROM` may only be preceded by one or more `ARG` instructions, which declare arguments that are used in `FROM` lines in the `Dockerfile`.

Docker treats lines that *begin* with # as a comment, unless the line is a valid parser directive. A # marker anywhere else in a line is treated as an argument. This allows statements like:

```
# Comment
RUN echo 'we are running some # of cool things'
```

Comment lines are removed before the Dockerfile instructions are executed, which means that the comment in the following example is not handled by the shell executing the `echo` command, and both examples below are equivalent:

```
RUN echo hello \
# comment
world
```

```
RUN echo hello \
world
```

Line continuation characters are not supported in comments.

Note on whitespace

For backward compatibility, leading whitespace before comments (`#`) and instructions (such as RUN) are ignored, but discouraged. Leading whitespace is not preserved in these cases, and the following examples are therefore equivalent:

```
        # this is a comment-line
    RUN echo hello
RUN echo world
```

```
# this is a comment-line
RUN echo hello
RUN echo world
```

Note however, that whitespace in instruction *arguments*, such as the commands following RUN, are preserved, so the following example prints `hello  world` with leading whitespace as specified:

```
RUN echo "\
     hello\
     world"
```

2.4 Parser directives

Parser directives are optional, and affect the way in which subsequent lines in a `Dockerfile` are handled. Parser directives do not add layers to the build, and will not be shown as a build step. Parser directives are written as a special type of comment in the form `# directive=value`. A single directive may only be used once.

Once a comment, empty line or builder instruction has been processed, Docker no longer looks for parser directives. Instead it treats anything formatted as a parser directive as a comment and does not attempt to validate if it might be a parser directive. Therefore, all parser directives must be at the very top of a `Dockerfile`.

Parser directives are not case-sensitive. However, convention is for them to be lowercase. Convention is also to include a blank line following any parser directives. Line continuation characters are not supported in parser directives.

Due to these rules, the following examples are all invalid:

- Invalid due to line continuation:

```
# direc \
tive=value
```

- Invalid due to appearing twice:

```
# directive=value1
# directive=value2

FROM ImageName
```

- Treated as a comment due to appearing after a builder instruction:

```
FROM ImageName
# directive=value
```

- Treated as a comment due to appearing after a comment which is not a parser directive:

```
# About my dockerfile
# directive=value
FROM ImageName
```

The unknown directive is treated as a comment due to not being recognized. In addition, the known directive is treated as a comment due to appearing after a comment which is not a parser directive.

```
# unknowndirective=value
# knowndirective=value
```

- Non line-breaking whitespace is permitted in a parser directive. Hence, the following lines are all treated identically:

```
#directive=value
# directive =value
#       directive= value
# directive = value
#         dIrEcTiVe=value
```

The following parser directive is supported:

- `syntax`
- `escape`

2.4.1 syntax

```
# syntax=[remote image reference]
```

For example:

```
# syntax=docker/dockerfile:1
# syntax=docker.io/docker/dockerfile:1
# syntax=example.com/user/repo:tag@sha256:abcdef...
```

This feature is only available when using the BuildKit backend, and is ignored when using the classic builder backend.

The `syntax` directive defines the location of the Dockerfile builder that is used for building the current Dockerfile. The BuildKit backend allows to seamlessly use external implementations of builders that are distributed as Docker images and execute inside a container sandbox environment.

Custom Dockerfile implementation allows you to:

- Automatically get bugfixes without updating the Docker daemon
- Make sure all users are using the same implementation to build your Dockerfile
- Use the latest features without updating the Docker daemon
- Try out new features or third-party features before they are integrated in the Docker Daemon

- Use alternative build definitions, or create your own

Official releases

Docker distributes official versions of the images that can be used for building Dockerfiles under `docker/dockerfile` repository on Docker Hub. There are two channels where new images are released: stable and labs.

Stable channel follows semantic versioning. For example:

- `docker/dockerfile:1` - kept updated with the latest `1.x.x` minor *and* patch release
- `docker/dockerfile:1.2` - kept updated with the latest `1.2.x` patch release, and stops receiving updates once version `1.3.0` is released.
- `docker/dockerfile:1.2.1` - immutable: never updated

We recommend using `docker/dockerfile:1`, which always points to the latest stable release of the version 1 syntax, and receives both "minor" and "patch" updates for the version 1 release cycle. BuildKit automatically checks for updates of the syntax when performing a build, making sure you are using the most current version.

If a specific version is used, such as `1.2` or `1.2.1`, the Dockerfile needs to be updated manually to continue receiving bugfixes and new features. Old versions of the Dockerfile remain compatible with the new versions of the builder.

labs channel

The "labs" channel provides early access to Dockerfile features that are not yet available in the stable channel. Labs channel images are released in conjunction with the stable releases, and follow the same versioning with the `-labs` suffix, for example:

- `docker/dockerfile:labs` - latest release on labs channel
- `docker/dockerfile:1-labs` - same as `dockerfile:1` in the stable channel, with labs features enabled
- `docker/dockerfile:1.2-labs` - same as `dockerfile:1.2` in the stable channel, with labs features enabled
- `docker/dockerfile:1.2.1-labs` - immutable: never updated. Same as `dockerfile:1.2.1` in the stable channel, with labs features enabled

Choose a channel that best fits your needs; if you want to benefit from new features, use the labs channel. Images in the labs channel provide a superset of the features in the `stable` channel; note that stable features in the labs channel images follow semantic versioning, but "labs" features do not, and newer releases may not be backwards compatible, so it is recommended to use an immutable full version variant.

For documentation on "labs" features, master builds, and nightly feature releases, refer to the description in the BuildKit source repository on GitHub. For a full list of available images, visit

the image repository on Docker Hub, and the docker/dockerfile-upstream image repository for development builds.

2.4.2 escape

```
# escape=\ (backslash)
```

Or

```
# escape=` (backtick)
```

The `escape` directive sets the character used to escape characters in a `Dockerfile`. If not specified, the default escape character is `\`.

The escape character is used both to escape characters in a line, and to escape a newline. This allows a `Dockerfile` instruction to span multiple lines. Note that regardless of whether the `escape` parser directive is included in a `Dockerfile`, *escaping is not performed in a RUN command, except at the end of a line.*

Setting the escape character to ` is especially useful on Windows, where `\` is the directory path separator. ` is consistent with Windows PowerShell.

Consider the following example which would fail in a non-obvious way on Windows. The second `\` at the end of the second line would be interpreted as an escape for the newline, instead of a target of the escape from the first `\`. Similarly, the `\` at the end of the third line would, assuming it was actually handled as an instruction, cause it be treated as a line continuation. The result of this dockerfile is that second and third lines are considered a single instruction:

```
FROM microsoft/nanoserver
COPY testfile.txt c:\\
RUN dir c:\
```

Results in:

```
PS E:\myproject> docker build -t cmd .

Sending build context to Docker daemon 3.072 kB
Step 1/2 : FROM microsoft/nanoserver
 ---> 22738ff49c6d
Step 2/2 : COPY testfile.txt c:\RUN dir c:
GetFileAttributesEx c:RUN: The system cannot find the file specified.
PS E:\myproject>
```

One solution to the above would be to use `/` as the target of both the `COPY` instruction, and `dir`. However, this syntax is, at best, confusing as it is not natural for paths on Windows, and at worst, error prone as not all commands on Windows support `/` as the path separator.

By adding the `escape` parser directive, the following `Dockerfile` succeeds as expected with the use of natural platform semantics for file paths on Windows:

```
# escape=`

FROM microsoft/nanoserver
COPY testfile.txt c:\
RUN dir c:\
```

Results in:

```
PS E:\myproject> docker build -t succeeds --no-cache=true .

Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM microsoft/nanoserver
 ---> 22738ff49c6d
Step 2/3 : COPY testfile.txt c:\
 ---> 96655de338de
Removing intermediate container 4db9acbb1682
Step 3/3 : RUN dir c:\
 ---> Running in a2c157f842f5
 Volume in drive C has no label.
 Volume Serial Number is 7E6D-E0F7

 Directory of c:\

10/05/2016  05:04 PM             1,894 License.txt
10/05/2016  02:22 PM    <DIR>          Program Files
10/05/2016  02:14 PM    <DIR>          Program Files (x86)
10/28/2016  11:18 AM                62 testfile.txt
10/28/2016  11:20 AM    <DIR>          Users
10/28/2016  11:20 AM    <DIR>          Windows
           2 File(s)          1,956 bytes
           4 Dir(s)  21,259,096,064 bytes free
 ---> 01c7f3bef04f
Removing intermediate container a2c157f842f5
Successfully built 01c7f3bef04f
PS E:\myproject>
```

2.5 Environment replacement

Environment variables (declared with the `ENV` statement) can also be used in certain instructions as variables to be interpreted by the `Dockerfile`. Escapes are also handled for including variable-like syntax into a statement literally.

Environment variables are notated in the `Dockerfile` either with `$variable_name` or `${variable_name}`. They are treated equivalently and the brace syntax is typically used to address issues with variable names with no whitespace, like `${foo}_bar`.

The `${variable_name}` syntax also supports a few of the standard `bash` modifiers as specified below:

- `${variable:-word}` indicates that if `variable` is set then the result will be that value. If `variable` is not set then `word` will be the result.
- `${variable:+word}` indicates that if `variable` is set then `word` will be the result, otherwise the result is the empty string.

In all cases, word can be any string, including additional environment variables.

Escaping is possible by adding a `\` before the variable: `\$foo` or `\${foo}`, for example, will translate to `$foo` and `${foo}` literals respectively.

Example (parsed representation is displayed after the #):

```
FROM busybox
ENV foo /bar
WORKDIR ${foo}   # WORKDIR /bar
ADD . $foo       # ADD . /bar
COPY \$foo /quux # COPY $foo /quux
```

Environment variables are supported by the following list of instructions in the Dockerfile:

- ADD
- COPY
- ENV
- EXPOSE
- FROM
- LABEL
- STOPSIGNAL
- USER
- VOLUME
- WORKDIR
- ONBUILD (when combined with one of the supported instructions above)

Environment variable substitution will use the same value for each variable throughout the entire command. In other words, in this example:

```
ENV abc=hello
ENV abc=bye def=$abc
ENV ghi=$abc
```

will result in `def` having a value of `hello`, not `bye`. However, `ghi` will have a `value` of `bye` because it is not part of the same command that set `abc` to `bye`.

2.6 .dockerignore file

Before the docker CLI sends the context to the docker daemon, it looks for a file named `.dockerignore` in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using `ADD` or `COPY`.

The CLI interprets the `.dockerignore` file as a newline-separated list of patterns similar to the file globs of Unix shells. For the purposes of matching, the root of the context is considered to be both the working and the root directory. For example, the patterns `/foo/bar` and `foo/bar` both exclude a file or directory named `bar` in the `foo` subdirectory of `PATH` or in the root of the git repository located at `URL`. Neither excludes anything else.

If a line in `.dockerignore` file starts with `#` in column 1, then this line is considered as a comment and is ignored before interpreted by the CLI.

Here is an example `.dockerignore` file:

```
# comment
*/temp*
*/*/temp*
temp?
```

This file causes the following build behavior:

Rule	Behavior
`# comment`	Ignored.
`*/temp*`	Exclude files and directories whose names start with `temp` in any immediate subdirectory of the root. For example, the plain file `/somedir/temporary.txt` is excluded, as is the directory `/somedir/temp`.
`*/*/temp*`	Exclude files and directories starting with `temp` from any subdirectory that is two levels below the root. For example, `/somedir/subdir/temporary.txt` is excluded.
`temp?`	Exclude files and directories in the root directory whose names are a one-character extension of `temp`. For example, `/tempa` and `/tempb` are excluded.

Matching is done using Go's `filepath.Match` rules. A preprocessing step removes leading and trailing whitespace and eliminates `.` and `..` elements using Go's `filepath.Clean`. Lines that are blank after preprocessing are ignored.

Beyond Go's `filepath.Match` rules, Docker also supports a special wildcard string ** that matches any number of directories (including zero). For example, `**/*.go` will exclude all files that end with `.go` that are found in all directories, including the root of the build context.

Lines starting with `!` (exclamation mark) can be used to make exceptions to exclusions. The following is an example `.dockerignore` file that uses this mechanism:

```
*.md
!README.md
```

All markdown files *except* `README.md` are excluded from the context.

The placement of `!` exception rules influences the behavior: the last line of the `.dockerignore` that matches a particular file determines whether it is included or excluded. Consider the following example:

```
*.md
!README*.md
README-secret.md
```

No markdown files are included in the context except README files other than `README-secret.md`.

Now consider this example:

```
*.md
README-secret.md
!README*.md
```

All of the README files are included. The middle line has no effect because `!README*.md` matches `README-secret.md` and comes last.

You can even use the `.dockerignore` file to exclude the `Dockerfile` and `.dockerignore` files. These files are still sent to the daemon because it needs them to do its job. But the `ADD` and `COPY` commands do not copy them to the image.

Finally, you may want to specify which files to include in the context, rather than which to exclude. To achieve this, specify `*` as the first pattern, followed by one or more `!` exception patterns.

***Note**: For historical reasons, the pattern `.` is ignored.*

2.7 FROM

```
FROM [--platform=<platform>] <image> [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

The `FROM` instruction initializes a new build stage and sets the *Base Image* for subsequent instructions. As such, a valid `Dockerfile` must start with a `FROM` instruction. The image can be any valid image – it is especially easy to start by **pulling an image** from the *Public Repositories*.

- `ARG` is the only instruction that may precede `FROM` in the `Dockerfile`.
- `FROM` can appear multiple times within a single `Dockerfile` to create multiple images or use one build stage as a dependency for another. Simply make a note of the last image ID output by the commit before each new `FROM` instruction. Each `FROM` instruction clears any state created by previous instructions.
- Optionally a name can be given to a new build stage by adding `AS name` to the `FROM` instruction. The name can be used in subsequent `FROM` and `COPY --from=<name|index>` instructions to refer to the image built in this stage.
- The `tag` or `digest` values are optional. If you omit either of them, the builder assumes a `latest` by default. The builder returns an error if it cannot find the `tag` value.

The optional `--platform` flag can be used to specify the platform of the image in case `FROM` references a multi-platform image. For example, `linux/amd64`, `linux/arm64`, or `windows/amd64`. By default, the target platform of the build request is used. Global build arguments can be used in the value of this flag, for example automatic platform `ARG`s allow you to force a stage to native build platform (`--platform=$BUILDPLATFORM`), and use it to cross-compile to the target platform inside the stage.

2.7.1 Understand how ARG and FROM interact

`FROM` instructions support variables that are declared by any `ARG` instructions that occur before the first `FROM`.

```
ARG  CODE_VERSION=latest
FROM base:${CODE_VERSION}
CMD  /code/run-app
FROM extras:${CODE_VERSION}
CMD  /code/run-extras
```

An `ARG` declared before a `FROM` is outside of a build stage, so it can't be used in any instruction after a `FROM`. To use the default value of an `ARG` declared before the first `FROM` use an `ARG` instruction without a value inside of a build stage:

```
ARG VERSION=latest
FROM busybox:$VERSION
ARG VERSION
RUN echo $VERSION > image_version
```

2.8 RUN

RUN has 2 forms:

```
RUN <command>
```

(*shell* form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)

```
RUN ["executable", "param1", "param2"]
```

(*exec* form)

The `RUN` instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the `Dockerfile`.

Layering `RUN` instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image's history, much like source control.

The *exec* form makes it possible to avoid shell string munging, and to `RUN` commands using a base image that does not contain the specified shell executable.

The default shell for the *shell* form can be changed using the `SHELL` command.

In the *shell* form you can use a `\` (backslash) to continue a single RUN instruction onto the next line. For example, consider these two lines:

```
RUN /bin/bash -c 'source $HOME/.bashrc ;\
echo $HOME'
```

Together they are equivalent to this single line:

```
RUN /bin/bash -c 'source $HOME/.bashrc ; echo $HOME'
```

To use a different shell, other than '/bin/sh', use the exec form passing in the desired shell. For example:

```
RUN ["/bin/bash", "-c", "echo hello"]
```

***Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').*

Unlike the *shell* form, the *exec* form does not invoke a command shell. This means that normal shell processing does not happen. For example, `RUN [ "echo", "$HOME" ]` will not do variable substitution on `$HOME`. If you want shell processing then either use the *shell* form or execute a

shell directly, for example: `RUN [ "sh", "-c", "echo $HOME" ]`. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

***Note**: In the JSON form, it is necessary to escape backslashes. This is particularly relevant on Windows where the backslash is the path separator. The following line would otherwise be treated as shell form due to not being valid JSON, and fail in an unexpected way:*

```
RUN ["c:\windows\system32\tasklist.exe"]
```

The correct syntax for this example is:

```
RUN ["c:\\windows\\system32\\tasklist.exe"]
```

The cache for `RUN` instructions isn't invalidated automatically during the next build. The cache for an instruction like `RUN apt-get dist-upgrade -y` will be reused during the next build. The cache for `RUN` instructions can be invalidated by using the `--no-cache` flag, for example `docker build --no-cache .`

The cache for `RUN` instructions can be invalidated by `ADD` and `COPY` instructions.

2.9 CMD

The CMD instruction has three forms:

```
CMD ["executable","param1","param2"]
```

(*exec* form, this is the preferred form)

```
CMD ["param1","param2"]
```

(as *default parameters to ENTRYPOINT*)

```
CMD command param1 param2
```

(*shell* form)

There can only be one `CMD` instruction in a `Dockerfile`. If you list more than one `CMD` then only the last `CMD` will take effect.

The main purpose of a `CMD` is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an `ENTRYPOINT` instruction as well.

If `CMD` is used to provide default arguments for the `ENTRYPOINT` instruction, both the `CMD` and `ENTRYPOINT` instructions should be specified with the JSON array format.

***Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').*

Unlike the *shell* form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, `CMD [ "echo", "$HOME" ]` will not do variable substitution on `$HOME`. If you want shell processing then either use the shell form or execute a shell directly, for example: `CMD [ "sh", "-c", "echo $HOME" ]`. When using the exec form

and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

When used in the shell or exec formats, the `CMD` instruction sets the command to be executed when running the image.

If you use the *shell* form of the `CMD`, then the `<command>` will execute in `/bin/sh -c`:

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

If you want to **run your** `<command>` **without a shell** then you must express the command as a JSON array and give the full path to the executable. **This array form is the preferred format of** `CMD`**.** Any additional parameters must be individually expressed as strings in the array:

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

If you would like your container to run the same executable every time, then you should consider using `ENTRYPOINT` in combination with `CMD`. See *ENTRYPOINT*.

If the user specifies arguments to `docker run` then they will override the default specified in `CMD`.

***Note**: don't confuse RUN with CMD. RUN actually runs a command and commits the result; CMD does not execute anything at build time, but specifies the intended command for the image.*

2.10 LABEL

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

The `LABEL` instruction adds metadata to an image. A `LABEL` is a key-value pair. To include spaces within a `LABEL` value, use quotes and backslashes as you would in command-line parsing. A few usage examples:

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

An image can have more than one label. You can specify multiple labels on a single line. Prior to Docker 1.10, this decreased the size of the final image, but this is no longer the case. You may choose to specify multiple labels in a single instruction, in one of the following two ways:

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
```

```
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

Labels included in base or parent images (images in the `FROM` line) are inherited by your image. If a label already exists but with a different value, the most-recently-applied value overrides any previously-set value.

To view an image's labels, use the `docker inspect` command. You can use the `--format` option to show just the labels

```
docker image inspect --format='{{json .Config.Labels}}' myimage
```

```
{
    "com.example.vendor": "ACME Incorporated"
    "com.example.label-with-value": "foo",
    "version": "1.0",
    "description": "This text illustrates that label-values can span multiple lines.",
    "multi.label1": "value1",
    "multi.label2": "value2",
    "other": "value3"
}
```

2.11 MAINTAINER*

```
MAINTAINER <name>
```

(*deprecated*)

The `MAINTAINER` instruction allows you to set the *Author* field of the generated images. The `LABEL` instruction is a much more flexible version of this and you should use it instead, as it enables setting any metadata you require, and can be viewed easily, for example with `docker inspect`. To set a label corresponding to the `MAINTAINER` field you could use:

```
LABEL org.opencontainers.image.authors="SvenDowideit@home.org.au"
```

This will then be visible from `docker inspect` with the other labels.

2.12 EXPOSE

```
EXPOSE <port> [<port>/<protocol>...]
```

The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

The `EXPOSE` instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. To actually publish the port when running the container, use the `-p` flag on `docker run` to publish and map one or more ports, or the `-P` flag to publish all exposed ports and map them to high-order ports.

By default, `EXPOSE` assumes TCP. You can also specify UDP:

```
EXPOSE 80/udp
```

To expose on both TCP and UDP, include two lines:

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

In this case, if you use `-P` with `docker run`, the port will be exposed once for TCP and once for UDP. Remember that `-P` uses an ephemeral high-ordered host port on the host, so the port will not be the same for TCP and UDP.

Regardless of the `EXPOSE` settings, you can override them at runtime by using the `-p` flag. For example

```
docker run -p 80:80/tcp -p 80:80/udp ...
```

To set up port redirection on the host system, see *using the -P flag*. The `docker network` command supports creating networks for communication among containers without the need to expose or publish specific ports, because the containers connected to the network can communicate with each other over any port.

2.13 ENV

```
ENV <key>=<value> ...
```

The `ENV` instruction sets the environment variable `<key>` to the value `<value>`. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well. The value will be interpreted for other environment variables, so quote characters will be removed if they are not escaped. Like command line parsing, quotes and backslashes can be used to include spaces within values.

Example:

```
ENV MY_NAME="John Doe"
ENV MY_DOG=Rex\ The\ Dog
ENV MY_CAT=fluffy
```

The `ENV` instruction allows for multiple `<key>=<value> ...` variables to be set at one time, and the example below will yield the same net results in the final image:

```
ENV MY_NAME="John Doe" MY_DOG=Rex\ The\ Dog \
    MY_CAT=fluffy
```

The environment variables set using `ENV` will persist when a container is run from the resulting image. You can view the values using `docker inspect`, and change them using `docker run --env <key>=<value>`.

Environment persistence can cause unexpected side effects. For example, setting `ENV DEBIAN_FRONTEND=noninteractive` changes the behaviour of `apt-get`, and may confuse users of your image.

If an environment variable is only needed during build, and not in the final image, consider setting a value for a single command instead:

```
RUN DEBIAN_FRONTEND=noninteractive apt-get update && apt-get install -y ...
```

Or using `ARG`, which is not persisted in the final image:

```
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y ...
```

Alternative syntax

The `ENV` instruction also allows an alternative syntax `ENV <key> <value>`, omitting the `=`. The alternative syntax is supported for backward compatibility, but discouraged, and may be removed in a future release.

2.14 ADD

ADD has two forms:

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

(the latter form is required for paths containing whitespace)

***Note**: The `--chown` feature is only supported on Dockerfiles used to build Linux containers, and will not work on Windows containers. Since user and group ownership concepts do not translate between Linux and Windows, the use of `/etc/passwd` and `/etc/group` for translating user and group names to IDs restricts this feature to only be viable for Linux OS-based containers*

The `ADD` instruction copies new files, directories or remote file URLs from `<src>` and adds them to the filesystem of the container at the path `<dest>`.

Multiple `<src>` resource may be specified but if they are files or directories, their paths are interpreted as relative to the source of the context of the build.

Each `<src>` may contain wildcards and matching will be done using Go's `filepath.Match` rules. For example:

To add all files starting with "hom":

```
ADD hom* /mydir/
```

In the example below, `?` is replaced with any single character, e.g., "home.txt".

```
ADD hom?.txt /mydir/
```

The `<dest>` is an absolute path, or a path relative to `WORKDIR`, into which the source will be copied inside the destination container.

The example below uses a relative path, and adds "test.txt" to `<WORKDIR>/relativeDir/`:

```
ADD test.txt relativeDir/
```

Whereas this example uses an absolute path, and adds "test.txt" to `/absoluteDir/`:

```
ADD test.txt /absoluteDir/
```

When adding files or directories that contain special characters (such as `[` and `]`), you need to escape those paths following the Golang rules to prevent them from being treated as a matching pattern. For example, to add a file named `arr[0].txt`, use the following;

```
ADD arr[[]0].txt /mydir/
```

All new files and directories are created with a UID and GID of 0, unless the optional `--chown flag` specifies a given username, groupname, or UID/GID combination to request specific ownership of the content added. The format of the `--chown` flag allows for either username and groupname strings or direct integer UID and GID in any combination. Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container's root filesystem `/etc/passwd` and `/etc/group` files will be used to perform the translation from name to integer UID or GID respectively. The following examples show valid definitions for the `--chown` flag:

```
ADD --chown=55:mygroup files* /somedir/
ADD --chown=bin files* /somedir/
ADD --chown=1 files* /somedir/
ADD --chown=10:11 files* /somedir/
```

If the container root filesystem does not contain either `/etc/passwd` or `/etc/group` files and either user or group names are used in the `--chown` flag, the build will fail on the `ADD` operation. Using numeric IDs requires no lookup and will not depend on container root filesystem content.

In the case where `<src>` is a remote file URL, the destination will have permissions of 600. If the remote file being retrieved has an `HTTP Last-Modified` header, the timestamp from that header will be used to set the `mtime` on the destination file. However, like any other file processed during an `ADD`, `mtime` will not be included in the determination of whether or not the file has changed and the cache should be updated.

***Note**: If you build by passing a Dockerfile through STDIN (`docker build - < somefile`), there is no build context, so the Dockerfile can only contain a URL based ADD instruction. You can also pass a compressed archive through STDIN: (`docker build - < archive.tar.gz`), the Dockerfile at the root of the archive and the rest of the archive will get used at the context of the build.*

If your URL files are protected using authentication, you will need to use `RUN wget`, `RUN curl` or use another tool from within the container as the `ADD` instruction does not support authentication.

***Note**: The first encountered `ADD` instruction will invalidate the cache for all following instructions from the Dockerfile if the contents of `<src>` have changed. This includes invalidating the cache for `RUN` instructions.*

`ADD` obeys the following rules:

- The `<src>` path must be inside the *context* of the build; you cannot `ADD ../something /something`, because the first step of a `docker build` is to send the context directory (and subdirectories) to the docker daemon.
- If `<src>` is a URL and `<dest>` does not end with a trailing slash, then a file is downloaded from the URL and copied to `<dest>`.
- If `<src>` is a URL and `<dest>` does end with a trailing slash, then the filename is inferred from the URL and the file is downloaded to `<dest>/<filename>`. For instance, `ADD http://example.com/foobar /` would create the file `/foobar`. The URL must have a nontrivial path so that an appropriate filename can be discovered in this case (`http://example.com` will not work).
- If `<src>` is a directory, the entire contents of the directory are copied, including filesystem metadata.

  ***Note**: The directory itself is not copied, just its contents.*

- If `<src>` is a *local* tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory. Resources from *remote* URLs are **not** decompressed. When a directory is copied or unpacked, it has the same behavior as `tar -x`, the result is the union of:

  1. Whatever existed at the destination path and

2. The contents of the source tree, with conflicts resolved in favor of "2." on a file-by-file basis.

***Note**: Whether a file is identified as a recognized compression format or not is done solely based on the contents of the file, not the name of the file. For example, if an empty file happens to end with `.tar.gz` this will not be recognized as a compressed file and **will not** generate any kind of decompression error message, rather the file will simply be copied to the destination.*

- If `<src>` is any other kind of file, it is copied individually along with its metadata. In this case, if `<dest>` ends with a trailing slash `/`, it will be considered a directory and the contents of `<src>` will be written at `<dest>/base(<src>)`.
- If multiple `<src>` resources are specified, either directly or due to the use of a wildcard, then `<dest>` must be a directory, and it must end with a slash `/`.
- If `<dest>` does not end with a trailing slash, it will be considered a regular file and the contents of `<src>` will be written at `<dest>`.
- If `<dest>` doesn't exist, it is created along with all missing directories in its path.

2.15 COPY

COPY has two forms:

```
COPY [--chown=<user>:<group>] <src>... <dest>
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

(This latter form is required for paths containing whitespace)

***Note**: The `--chown` feature is only supported on Dockerfiles used to build Linux containers, and will not work on Windows containers. Since user and group ownership concepts do not translate between Linux and Windows, the use of `/etc/passwd` and `/etc/group` for translating user and group names to IDs restricts this feature to only be viable for Linux OS-based containers*

The `COPY` instruction copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>`.

Multiple `<src>` resource may be specified but they must be relative to the source directory that is being built (the context of the build).

Each `<src>` may contain wildcards and matching will be done using Go's filepath.Match rules. For example:

To add all files starting with "hom":

```
COPY hom* /mydir/
```

In the example below, `?` is replaced with any single character, e.g., "home.txt".

```
COPY hom?.txt /mydir/
```

The `<dest>` is an absolute path, or a path relative to `WORKDIR`, into which the source will be copied inside the destination container.

The example below uses a relative path, and adds "test.txt" to `<WORKDIR>/relativeDir/`:

```
COPY test.txt relativeDir/
```

Whereas this example uses an absolute path, and adds “test.txt” to `/absoluteDir/`:

```
COPY test.txt /absoluteDir/
```

When copying files or directories that contain special characters (such as `[` and `]`), you need to escape those paths following the Golang rules to prevent them from being treated as a matching pattern. For example, to copy a file named `arr[0].txt`, use the following;

```
COPY arr[[]0].txt /mydir/
```

All new files and directories are created with a UID and GID of 0, unless the optional `--chown flag` specifies a given username, groupname, or UID/GID combination to request specific ownership of the content added. The format of the `--chown` flag allows for either username and groupname strings or direct integer UID and GID in any combination. Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container’s root filesystem `/etc/passwd` and `/etc/group` files will be used to perform the translation from name to integer UID or GID respectively. The following examples show valid definitions for the `--chown` flag:

```
COPY --chown=55:mygroup files* /somedir/
COPY --chown=bin files* /somedir/
COPY --chown=1 files* /somedir/
COPY --chown=10:11 files* /somedir/
```

If the container root filesystem does not contain either `/etc/passwd` or `/etc/group` files and either user or group names are used in the `--chown` flag, the build will fail on the `COPY` operation. Using numeric IDs requires no lookup and will not depend on container root filesystem content.

***Note**: If you build using STDIN (`docker build - < somefile`), there is no build context, so COPY can’t be used.*

Optionally `COPY` accepts a flag `--from=<name|index>` that can be used to set the source location to a previous build stage (created with `FROM .. AS <name>`) that will be used instead of a build context sent by the user. The flag also accepts a numeric index assigned for all previous build stages started with `FROM` instruction. In case a build stage with a specified name can’t be found an image with the same name is attempted to be used instead.

`COPY` obeys the following rules:

- The `<src>` path must be inside the *context* of the build; you cannot `COPY ../something /something`, because the first step of a `docker build` is to send the context directory (and subdirectories) to the docker daemon.
- If `<src>` is a directory, the entire contents of the directory are copied, including filesystem metadata.

***Note**: The directory itself is not copied, just its contents.*

- If `<src>` is any other kind of file, it is copied individually along with its metadata. In this case, if `<dest>` ends with a trailing slash `/`, it will be considered a directory and the contents of `<src>` will be written at `<dest>/base(<src>)`.
- If multiple `<src>` resources are specified, either directly or due to the use of a wildcard, then `<dest>` must be a directory, and it must end with a slash `/`.

- If `<dest>` does not end with a trailing slash, it will be considered a regular file and the contents of `<src>` will be written at `<dest>`.
- If `<dest>` doesn't exist, it is created along with all missing directories in its path.

***Note**: The first encounered `COPY` instruction will invalidate the cache for all following instructions from the Dockerfile if the contents of `<src>` have changed. This includes invalidating the cache for `RUN` instructions.*

2.16 ENTRYPOINT

`ENTRYPOINT` has two forms:

The *exec* form, which is the preferred form:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

The *shell* form:

```
ENTRYPOINT command param1 param2
```

An `ENTRYPOINT` allows you to configure a container that will run as an executable.

For example, the following will start nginx with its default content, listening on port 80:

```
docker run -i -t --rm -p 80:80 nginx
```

Command line arguments to `docker run <image>` will be appended after all elements in an *exec* form `ENTRYPOINT`, and will override all elements specified using `CMD`. This allows arguments to be passed to the entry point, i.e., `docker run <image> -d` will pass the `-d` argument to the entry point. You can override the `ENTRYPOINT` instruction using the `docker run --entrypoint` flag.

The *shell* form prevents any `CMD` or `run` command line arguments from being used, but has the disadvantage that your `ENTRYPOINT` will be started as a subcommand of `/bin/sh -c`, which does not pass signals. This means that the executable will not be the container's `PID 1` - and will *not* receive Unix signals - so your executable will not receive a `SIGTERM` from `docker stop <container>`.

Only the last `ENTRYPOINT` instruction in the `Dockerfile` will have an effect.

2.16.1 Exec form ENTRYPOINT example

You can use the *exec* form of `ENTRYPOINT` to set fairly stable default commands and arguments and then use either form of `CMD` to set additional defaults that are more likely to be changed.

```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD ["-c"]
```

When you run the container, you can see that `top` is the only process:

```
$ docker run -it --rm --name test  top -H
top - 08:25:00 up  7:27,  0 users,  load average: 0.00, 0.01, 0.05
Threads:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2056668 total,  1616832 used,   439836 free,    99352 buffers
KiB Swap:  1441840 total,        0 used,  1441840 free.  1324440 cached Mem
```

```
  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   19744   2336   2080 R  0.0  0.1   0:00.04 top
```

To examine the result further, you can `use docker exec`:

```
$ docker exec -it test ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  2.6  0.1  19752  2352 ?        Ss+  08:24   0:00 top -b -H
root         7  0.0  0.1  15572  2164 ?        R+   08:25   0:00 ps aux
```

And you can gracefully request top to shut down using `docker stop test` .

The following `Dockerfile` shows using the `ENTRYPOINT` to run Apache in the foreground (i.e., as `PID 1`):

```
FROM debian:stable
RUN apt-get update && apt-get install -y --force-yes apache2
EXPOSE 80 443
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

If you need to write a starter script for a single executable, you can ensure that the final executable receives the Unix signals by using `exec` and `gosu` commands:

```
#!/usr/bin/env bash
set -e

if [ "$1" = 'postgres' ]; then
    chown -R postgres "$PGDATA"

    if [ -z "$(ls -A "$PGDATA")" ]; then
        gosu postgres initdb
    fi

    exec gosu postgres "$@"
fi

exec "$@"
```

Lastly, if you need to do some extra cleanup (or communicate with other containers) on shutdown, or are co-ordinating more than one executable, you may need to ensure that the `ENTRYPOINT` script receives the Unix signals, passes them on, and then does some more work:

```
#!/bin/sh
# Note: I've written this using sh so it works in the busybox container too

# USE the trap if you need to also do manual cleanup after the service is stopped,
#     or need to start multiple services in the one container
trap "echo TRAPed signal" HUP INT QUIT TERM

# start service in background here
/usr/sbin/apachectl start

echo "[hit enter key to exit] or run 'docker stop <container>'"
read

# stop service and clean up here
echo "stopping apache"
/usr/sbin/apachectl stop

echo "exited $0"
```

If you run this image with `docker run -it --rm -p 80:80 --name test apache`, you can then examine the container's processes with `docker exec`, or `docker top`, and then ask the script to stop Apache:

```
$ docker exec -it test ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.0   4448   692 ?        Ss+  00:42   0:00 /bin/sh /run.sh 123 cmd cmd2
root        19  0.0  0.2  71304  4440 ?        Ss   00:42   0:00 /usr/sbin/apache2 -k start
www-data    20  0.2  0.2 360468  6004 ?        Sl   00:42   0:00 /usr/sbin/apache2 -k start
www-data    21  0.2  0.2 360468  6000 ?        Sl   00:42   0:00 /usr/sbin/apache2 -k start
root        81  0.0  0.1  15572  2140 ?        R+   00:44   0:00 ps aux

$ docker top test
PID                 USER                COMMAND
10035               root                {run.sh} /bin/sh /run.sh 123 cmd cmd2
10054               root                /usr/sbin/apache2 -k start
10055               33                  /usr/sbin/apache2 -k start
10056               33                  /usr/sbin/apache2 -k start

$ /usr/bin/time docker stop test
test
real    0m 0.27s
user    0m 0.03s
sys     0m 0.03s
```

Note: *you can override the ENTRYPOINT setting using `--entrypoint`, but this can only set the binary to exec (no sh -c will be used).*

Note*: The exec form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').*

Unlike the *shell* form, the *exec* form does not invoke a command shell. This means that normal shell processing does not happen. For example, `ENTRYPOINT [ "echo", "$HOME" ]` will not do variable substitution on `$HOME`. If you want shell processing then either use the *shell* form or execute a shell directly, for example: `ENTRYPOINT [ "sh", "-c", "echo $HOME" ]`. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

2.16.2 Shell form ENTRYPOINT example

You can specify a plain string for the `ENTRYPOINT` and it will execute in `/bin/sh -c`. This form will use shell processing to substitute shell environment variables, and will ignore any `CMD` or `docker run` command line arguments. To ensure that `docker stop` will signal any long running `ENTRYPOINT` executable correctly, you need to remember to start it with `exec`:

```
FROM ubuntu
ENTRYPOINT exec top -b
```

When you run this image, you'll see the single `PID 1` process:

```
$ docker run -it --rm --name test top
Mem: 1704520K used, 352148K free, 0K shrd, 0K buff, 140368121167873K cached
CPU:   5% usr   0% sys   0% nic  94% idle   0% io   0% irq   0% sirq
Load average: 0.08 0.03 0.05 2/98 6
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
    1     0 root     R     3164   0%   0% top -b
```

Which will exit cleanly on `docker stop`:

```
$ /usr/bin/time docker stop test
test
real    0m 0.20s
user    0m 0.02s
sys     0m 0.04s
```

If you forget to add `exec` to the beginning of your `ENTRYPOINT`:

```
FROM ubuntu
ENTRYPOINT top -b
CMD --ignored-param1
```

You can then run it (giving it a name for the next step):

```
$ docker run -it --name test top --ignored-param2
Mem: 1704184K used, 352484K free, 0K shrd, 0K buff, 140621524238337K cached
CPU:   9% usr   2% sys   0% nic  88% idle   0% io   0% irq   0% sirq
Load average: 0.01 0.02 0.05 2/101 7
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
    1     0 root     S     3168   0%   0% /bin/sh -c top -b cmd cmd2
    7     1 root     R     3164   0%   0% top -b
```

You can see from the output of `top` that the specified `ENTRYPOINT` is not `PID 1`.

If you then run `docker stop test`, the container will not exit cleanly - the `stop` command will be forced to send a `SIGKILL` after the timeout:

```
$ docker exec -it test ps aux
PID   USER     COMMAND
    1 root     /bin/sh -c top -b cmd cmd2
    7 root     top -b
    8 root     ps aux

$ /usr/bin/time docker stop test
test
real    0m 10.19s
user    0m 0.04s
sys     0m 0.03s
```

2.16.3 Understand how CMD and ENTRYPOINT interact

Both `CMD` and `ENTRYPOINT` instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of `CMD` or `ENTRYPOINT` commands.
2. `ENTRYPOINT` should be defined when using the container as an executable.
3. `CMD` should be used as a way of defining default arguments for an `ENTRYPOINT` command or for executing an ad-hoc command in a container.
4. `CMD` will be overridden when running the container with alternative arguments.

The table below shows what command is executed for different `ENTRYPOINT` / `CMD` combinations:

	No `ENTRYPOINT`	`ENTRYPOINT exec_entry p1_entry`	`ENTRYPOINT ["exec_entry", "p1_entry"]`
No `CMD`	*error, not allowed*	`/bin/sh -c exec_entry p1_entry`	`exec_entry p1_entry`

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

***Note**: If `CMD` is from the base image, setting `ENTRYPOINT` will reset `CMD` to an empty value. In this scenario, `CMD` must be defined in the current image to have a value.*

2.17 VOLUME

```
VOLUME ["/data"]
```

The `VOLUME` instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, `VOLUME ["/var/log/"]`, or a plain string with multiple arguments, such as `VOLUME /var/log` or `VOLUME /var/log /var/db`.

The `docker run` command initializes the newly created volume with any data that exists at the specified location within the base image. For example, consider the following Dockerfile snippet:

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

This Dockerfile results in an image that causes `docker run`, to create a new mount point at `/myvol` and copy the `greeting` file into the newly created volume.

2.17.1 Notes about specifying volumes

Keep the following things in mind about volumes in the `Dockerfile`.

- **Volumes on Windows-based containers**: When using Windows-based containers, the destination of a volume inside the container must be one of:
  - a non-existing or empty directory
  - a drive other than `C:`
- **Changing the volume from within the Dockerfile**: If any build steps change the data within the volume after it has been declared, those changes will be discarded.
- **JSON formatting**: The list is parsed as a JSON array. You must enclose words with double quotes (") rather than single quotes (').
- **The host directory is declared at container run-time**: The host directory (the mountpoint) is, by its nature, host-dependent. This is to preserve image portability. since a given host directory can't be guaranteed to be available on all hosts. For this reason, you can't mount a host directory from within the Dockerfile. The `VOLUME`

instruction does not support specifying a `host-dir` parameter. You must specify the mountpoint when you create or run the container

2.18 USER

```
USER <user>[:<group>]
```

or

```
USER <UID>[:<GID>]
```

The `USER` instruction sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any `RUN`, `CMD` and `ENTRYPOINT` instructions that follow it in the `Dockerfile`.

Note that when specifying a group for the user, the user will have *only* the specified group membership. Any other configured group memberships will be ignored.

Warning: *When the user doesn't have a primary group then the image (or the next instructions) will be run with the `root` group*

2.19 WORKDIR

```
WORKDIR /path/to/workdir
```

The `WORKDIR` instruction sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY` and `ADD` instructions that follow it in the `Dockerfile`. If the `WORKDIR` doesn't exist, it will be created even if it's not used in any subsequent `Dockerfile` instruction.

The `WORKDIR` instruction can be used multiple times in a `Dockerfile`. If a relative path is provided, it will be relative to the path of the previous `WORKDIR` instruction. For example:

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

The output of the final `pwd` command in this `Dockerfile` would be `/a/b/c`.

The `WORKDIR` instruction can resolve environment variables previously set using `ENV`. You can only use environment variables explicitly set in the `Dockerfile`. For example:

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

The output of the final `pwd` command in this `Dockerfile` would be `/path/$DIRNAME`

2.20 ARG

```
ARG <name>[=<default value>]
```

The `ARG` instruction defines a variable that users can pass at build-time to the builder with the `docker build` command using the `--build-arg <varname>=<value>` flag. If a user specifies a build argument that was not defined in the Dockerfile, the build outputs an error.

```
[Warning] One or more build-args [foo] were not consumed.
```

A Dockerfile may include one or more `ARG` instructions. For example, a valid Dockerfile:

```
FROM busybox
ARG user1
ARG buildno
# ...
```

Warning: *It is not recommended to use build-time variables for passing secrets like github keys, user credentials etc. Build-time variable values are visible to any user of the image with the* `docker history` *command.*

2.20.1 Default values

An `ARG` instruction can optionally include a default value:

```
FROM busybox
ARG user1=someuser
ARG buildno=1
# ...
```

If an `ARG` instruction has a default value and if there is no value passed at build-time, the builder uses the default.

2.20.2 Scope

An `ARG` variable definition comes into effect from the line on which it is defined in the `Dockerfile` not from the argument's use on the command-line or elsewhere. For example, consider this Dockerfile:

```
01: FROM busybox
02: USER ${user:-some_user}
03: ARG user
04: USER $user
05: # ...
```

A user builds this file by calling:

```
$ docker build --build-arg user=what_user .
```

The `USER` at line 2 evaluates to `some_user` as the `user` variable is defined on the subsequent line 3. The `USER` at line 4 evaluates to `what_user` as `user` is defined and the `what_user` value was passed on the command line. Prior to its definition by an `ARG` instruction, any use of a variable results in an empty string.

An `ARG` instruction goes out of scope at the end of the build stage where it was defined. To use an arg in multiple stages, each stage must include the `ARG` instruction.

```
FROM busybox
ARG SETTINGS
RUN ./run/setup $SETTINGS

FROM busybox
ARG SETTINGS
RUN ./run/other $SETTINGS
```

2.20.3 Using ARG variables

You can use an `ARG` or an `ENV` instruction to specify variables that are available to the `RUN` instruction. Environment variables defined using the `ENV` instruction always override an `ARG` instruction of the same name. Consider this Dockerfile with an `ENV` and `ARG` instruction.

```
01: FROM ubuntu
02: ARG CONT_IMG_VER
03: ENV CONT_IMG_VER v1.0.0
04: RUN echo $CONT_IMG_VER
```

Then, assume this image is built with this command:

```
$ docker build --build-arg CONT_IMG_VER=v2.0.1 .
```

In this case, the `RUN` instruction uses `v1.0.0` instead of the `ARG` setting passed by the user: `v2.0.1` This behavior is similar to a shell script where a locally scoped variable overrides the variables passed as arguments or inherited from environment, from its point of definition.

Using the example above but a different `ENV` specification you can create more useful interactions between `ARG` and `ENV` instructions:

```
01: FROM ubuntu
02: ARG CONT_IMG_VER
03: ENV CONT_IMG_VER ${CONT_IMG_VER:-v1.0.0}
04: RUN echo $CONT_IMG_VER
```

Unlike an `ARG` instruction, `ENV` values are always persisted in the built image. Consider a docker build without the `--build-arg` flag:

```
$ docker build .
```

Using this Dockerfile example, `CONT_IMG_VER` is still persisted in the image but its value would be `v1.0.0` as it is the default set in line 3 by the `ENV` instruction.

The variable expansion technique in this example allows you to pass arguments from the command line and persist them in the final image by leveraging the `ENV` instruction. Variable expansion is only supported for a limited set of Dockerfile instructions.

2.20.4 Predefined ARGs

Docker has a set of predefined `ARG` variables that you can use without a corresponding `ARG` instruction in the Dockerfile.

- `HTTP_PROXY`
- `http_proxy`
- `HTTPS_PROXY`
- `https_proxy`
- `FTP_PROXY`
- `ftp_proxy`
- `NO_PROXY`
- `no_proxy`

To use these, pass them on the command line using the `--build-arg` flag, for example:

```
$ docker build --build-arg HTTPS_PROXY=https://my-proxy.example.com .
```

By default, these pre-defined variables are excluded from the output of `docker history`. Excluding them reduces the risk of accidentally leaking sensitive authentication information in an `HTTP_PROXY` variable.

For example, consider building the following Dockerfile using `--build-arg HTTP_PROXY=http://user:pass@proxy.lon.example.com`

```
FROM ubuntu
RUN echo "Hello World"
```

In this case, the value of the `HTTP_PROXY` variable is not available in the `docker history` and is not cached. If you were to change location, and your proxy server changed to `http://user:pass@proxy.sfo.example.com`, a subsequent build does not result in a cache miss.

If you need to override this behaviour then you may do so by adding an `ARG` statement in the Dockerfile as follows:

```
FROM ubuntu
ARG HTTP_PROXY
RUN echo "Hello World"
```

When building this Dockerfile, the `HTTP_PROXY` is preserved in the `docker history`, and changing its value invalidates the build cache.

~~2.20.5 Automatic platform ARGs in the global scope~~

This feature is only available when using the BuildKit backend.

(*See `https://docs.docker.com/engine/reference/builder/#automatic-platform-args-in-the-global-scope` for information about this topic*)

~~2.20.6 Impact on build caching~~

(*See `https://docs.docker.com/engine/reference/builder/#impact-on-build-caching` for information about this topic*)

2.21 ONBUILD

```
ONBUILD [INSTRUCTION]
```

The `ONBUILD` instruction adds to the image a *trigger* instruction to be executed at a later time, when the image is used as the base for another build. The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the `FROM` instruction in the downstream Dockerfile.

Any build instruction can be registered as a trigger.

This is useful if you are building an image which will be used as a base to build other images, for example an application build environment or a daemon which may be customized with user-specific configuration.

- For example, if your image is a reusable Python application builder, it will require application source code to be added in a particular directory, and it might require a build script to be called *after* that. You can't just call `ADD` and `RUN` now, because you don't yet have access to the application source code, and it will be different for each application build. You could simply provide application developers with a boilerplate `Dockerfile`

to copy-paste into their application, but that is inefficient, error-prone and difficult to update because it mixes with application-specific code.

The solution is to use `ONBUILD` to register advance instructions to run later, during the next build stage.

Here's how it works:

1. When it encounters an `ONBUILD` instruction, the builder adds a trigger to the metadata of the image being built. The instruction does not otherwise affect the current build.
2. At the end of the build, a list of all triggers is stored in the image manifest, under the key `OnBuild`. They can be inspected with the `docker inspect` command.
3. Later the image may be used as a base for a new build, using the `FROM` instruction. As part of processing the `FROM` instruction, the downstream builder looks for `ONBUILD` triggers, and executes them in the same order they were registered. If any of the triggers fail, the `FROM` instruction is aborted which in turn causes the build to fail. If all triggers succeed, the `FROM` instruction completes and the build continues as usual.
4. Triggers are cleared from the final image after being executed. In other words they are not inherited by "grand-children" builds.

For example you might add something like this:

```
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
```

***Warning**: Chaining `ONBUILD` instructions using `ONBUILD` `ONBUILD` isn't allowed.*

***Warning**: The `ONBUILD` instruction may not trigger `FROM` or `MAINTAINER` instructions.*

2.22 STOPSIGNAL

```
STOPSIGNAL signal
```

The `STOPSIGNAL` instruction sets the system call signal that will be sent to the container to exit. This signal can be a signal name in the format SIGNAME, for instance SIGKILL, or an unsigned number that matches a position in the kernel's syscall table, for instance 9. The default is SIGTERM if not defined.

The image's default stopsignal can be overridden per container, using the `--stop-signal` flag on `docker run` and `docker create`.

2.23 HEALTHCHECK

The `HEALTHCHECK` instruction has two forms:

```
HEALTHCHECK [OPTIONS] CMD command
```

(check container health by running a command inside the container)

```
HEALTHCHECK NONE
```

(disable any healthcheck inherited from the base image)

The `HEALTHCHECK` instruction tells Docker how to test a container to check that it is still working. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

When a container has a healthcheck specified, it has a *health status* in addition to its normal status. This status is initially `starting`. Whenever a health check passes, it becomes `healthy` (whatever state it was previously in). After a certain number of consecutive failures, it becomes `unhealthy`.

The options that can appear before `CMD` are:

`--interval=DURATION` (default: 30s)
`--timeout=DURATION` (default: 30s)
`--start-period=DURATION` (default: 0s)
`--retries=N` (default: 3)

The health check will first run **interval** seconds after the container is started, and then again **interval** seconds after each previous check completes.

- If a single run of the check takes longer than **timeout** seconds then the check is considered to have failed.
- It takes **retries** consecutive failures of the health check for the container to be considered `unhealthy`.

`start period` provides initialization time for containers that need time to bootstrap. Probe failure during that period will not be counted towards the maximum number of retries. However, if a health check succeeds during the start period, the container is considered started and all consecutive failures will be counted towards the maximum number of retries.

There can only be one `HEALTHCHECK` instruction in a Dockerfile. If you list more than one then only the last `HEALTHCHECK` will take effect.

The command after the `CMD` keyword can be either a shell command (e.g. `HEALTHCHECK CMD /bin/check-running`) or an *exec* array (as with other Dockerfile commands; see e.g. `ENTRYPOINT` for details).

The command's exit status indicates the health status of the container. The possible values are:

- `0`: success - the container is healthy and ready for use
- `1`: unhealthy - the container is not working correctly
- `2`: reserved - do not use this exit code

For example, to check every five minutes or so that a web-server is able to serve the site's main page within three seconds:

```
HEALTHCHECK --interval=5m --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

To help debug failing probes, any output text (UTF-8 encoded) that the command writes on stdout or stderr will be stored in the health status and can be queried with `docker inspect`. Such output should be kept short (only the first 4096 bytes are stored currently).

When the health status of a container changes, a `health_status` event is generated with the new status.

2.24 SHELL

```
SHELL ["executable", "parameters"]
```

The `SHELL` instruction allows the default shell used for the *shell* form of commands to be overridden. The default shell on Linux is `["/bin/sh", "-c"]`, and on Windows is `["cmd", "/S", "/C"]`. The SHELL instruction *must* be written in JSON form in a Dockerfile.

The `SHELL` instruction is particularly useful on Windows where there are two commonly used and quite different native shells: `cmd` and `powershell`, as well as alternate shells available including `sh`.

The `SHELL` instruction can appear multiple times. Each `SHELL` instruction overrides all previous `SHELL` instructions, and affects all subsequent instructions. For example:

```
FROM microsoft/windowsservercore

# Executed as cmd /S /C echo default
RUN echo default

# Executed as cmd /S /C powershell -command Write-Host default
RUN powershell -command Write-Host default

# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello

# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S"", "/C"]
RUN echo hello
```

The following instructions can be affected by the `SHELL` instruction when the *shell* form of them is used in a Dockerfile: `RUN`, `CMD` and `ENTRYPOINT`.

The following example is a common pattern found on Windows which can be streamlined by using the `SHELL` instruction:

```
RUN powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
```

The command invoked by docker will be:

```
cmd /S /C powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
```

This is inefficient for two reasons. First, there is an un-necessary cmd.exe command processor (aka shell) being invoked. Second, each `RUN` instruction in the *shell* form requires an extra `powershell -command` prefixing the command.

To make this more efficient, one of two mechanisms can be employed. One is to use the JSON form of the `RUN` command such as:

```
RUN ["powershell", "-command", "Execute-MyCmdlet", "-param1 \"c:\\foo.txt\""]
```

While the JSON form is unambiguous and does not use the un-necessary cmd.exe, it does require more verbosity through double-quoting and escaping. The alternate mechanism is to use the `SHELL` instruction and the *shell* form, making a more natural syntax for Windows users, especially when combined with the `escape` parser directive:

```
# escape=`

FROM microsoft/nanoserver
SHELL ["powershell","-command"]
RUN New-Item -ItemType Directory C:\Example
ADD Execute-MyCmdlet.ps1 c:\example\
RUN c:\example\Execute-MyCmdlet -sample 'hello world'
```

Resulting in:

```
PS E:\myproject> docker build -t shell .
Sending build context to Docker daemon 4.096 kB
Step 1/5 : FROM microsoft/nanoserver
 ---> 22738ff49c6d
Step 2/5 : SHELL powershell -command
 ---> Running in 6fcdb6855ae2
 ---> 6331462d4300
Removing intermediate container 6fcdb6855ae2
Step 3/5 : RUN New-Item -ItemType Directory C:\Example
 ---> Running in d0eef8386e97


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/28/2016  11:26 AM                Example


 ---> 3f2fbf1395d9
Removing intermediate container d0eef8386e97
Step 4/5 : ADD Execute-MyCmdlet.ps1 c:\example\
 ---> a955b2621c31
Removing intermediate container b825593d39fc
Step 5/5 : RUN c:\example\Execute-MyCmdlet 'hello world'
 ---> Running in be6d8e63fe75
hello world
 ---> 8e559e9bf424
Removing intermediate container be6d8e63fe75
Successfully built 8e559e9bf424
PS E:\myproject>
```

The `SHELL` instruction could also be used to modify the way in which a shell operates. For example, using `SHELL cmd /S /C /V:ON|OFF` on Windows, delayed environment variable expansion semantics could be modified.

The `SHELL` instruction can also be used on Linux should an alternate shell be required such `zsh`, `csh`, `tcsh` and others.

2.25 Dockerfile examples

Below you can see some examples of Dockerfile syntax.

 # Nginx

```
# VERSION               0.0.1

FROM ubuntu
LABEL Description="This image is used to start the foobar executable" Vendor="ACME Products" Version="1.0"
RUN apt-get update && apt-get install -y inotify-tools nginx apache2 openssh-server
```

 # Firefox over VNC (https://github.com/philstenning/Firefox-in-docker-with-VNC)

```
# Firefox over VNC
#
# VERSION               0.3

FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir ~/.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]
```

 # Multiple images example

```
# VERSION               0.1

FROM ubuntu
RUN echo foo > bar
# Will output something like ===> 907ad6c2736f

FROM ubuntu
RUN echo moo > oink
# Will output something like ===> 695d7793cbe4

# You'll now have two images, 907ad6c2736f with /bar, and 695d7793cbe4 with /oink.
```

3 DOCKER-COMPOSE COMMAND LINE AND OPTIONS

From `docker-compose –help`

Define and run multi-container applications with Docker.

Usage:

```
docker-compose [OPTIONS] COMMAND
docker-compose -h|--help
```

Options:

```
    --ansi string              Control when to print ANSI control characters
                               ("never"|"always"|"auto") (default "auto")
    --compatibility            Run compose in backward compatibility mode
    --env-file string          Specify an alternate environment file.
-f, --file stringArray         Compose configuration files
    --profile stringArray      Specify a profile to enable
    --project-directory string Specify an alternate working directory
                               (default: the path of the, first specified, Compose file)
-p, --project-name string      Project name
```

Commands:

```
build       Build or rebuild services
convert     Converts the compose file to platform's canonical format
cp          Copy files/folders between a service container and the local filesystem
create      Creates containers for a service.
down        Stop and remove containers, networks
events      Receive real time events from containers.
exec        Execute a command in a running container.
images      List images used by the created containers
kill        Force stop service containers.
logs        View output from containers
ls          List running compose projects
pause       Pause services
port        Print the public port for a port binding.
ps          List containers
pull        Pull service images
push        Push service images
restart     Restart service containers
rm          Removes stopped service containers
run         Run a one-off command on a service.
start       Start services
stop        Stop services
top         Display the running processes
unpause     Unpause services
up          Create and start containers
version     Show the Docker Compose version information
```

3.1 build

Build or rebuild services.

```
docker-compose build [OPTIONS] [SERVICE...]
```

Services are built once and then tagged as `project_service`, e.g. `composetest_db`. If you change a service's `Dockerfile` or the contents of its build directory, you can run `docker-compose build` to rebuild it.

Options:

```
    --build-arg stringArray   Set build-time variables for services.
    --no-cache                Do not use cache when building the image.
    --progress string         Set type of progress output (auto, tty, plain, quiet)
                              (default "auto")
    --pull                    Always attempt to pull a newer version of the image.
-q, --quiet                   Don't print anything to STDOUT
    --ssh string              Set SSH authentications used when building service images.
                              (use 'default' for using your default SSH Agent)
```

3.2 convert

Converts the compose file to platform's canonical format.

```
docker-compose convert [OPTIONS] [SERVICE...]
```

Aliases:

```
convert, config
```

Options:

```
    --format string          Format the output. Values: [yaml | json] (default "yaml")
    --hash string            Print the service config hash, one per line.
    --images                 Print the image names, one per line.
    --no-interpolate         Don't interpolate environment variables.
    --no-normalize           Don't normalize compose model.
-o, --output string          Save to file (default to stdout)
    --profiles               Print the profile names, one per line.
-q, --quiet                  Only validate the configuration, don't print anything.
    --resolve-image-digests  Pin image tags to digests.
    --services               Print the service names, one per line.
    --volumes                Print the volume names, one per line.
```

3.3 create

Creates containers for a service.

```
docker-compose create [OPTIONS] [SERVICE...]
```

Options:

```
  --build           Build images before starting containers.
  --force-recreate  Recreate containers even if their configuration and image haven't changed.
  --no-build        Don't build an image, even if it's missing.
  --no-recreate     If containers already exist, don't recreate them. Incompatible with
                    --force-recreate.
  --pull string     Pull image before running ("always"|"missing"|"never")
                    (default "missing")
```

3.4 down

Stop and remove containers, networks.

```
docker-compose down [OPTIONS]
```

Options:

```
    --remove-orphans    Remove containers for services not defined in the Compose file.
    --rmi string        Remove images used by services. "local" remove only images that
                        don't have a custom tag ("local"|"all")
-t, --timeout int       Specify a shutdown timeout in seconds (default 10)
-v, --volumes volumes   Remove named volumes declared in the volumes section of the Compose
                        file and anonymous volumes attached to containers.
```

3.5 events

Receive real time events from containers.

```
docker-compose events [OPTIONS] [--] [SERVICE...]
```

Options:

```
    --json      Output events as a stream of json objects
```

3.6 exec

Execute a command in a running container

```
docker-compose exec [OPTIONS] SERVICE COMMAND [ARGS...]
```

Options:

```
-d, --detach                    Detached mode: Run command in the background.
-e, --env stringArray           Set environment variables
    --index int                 index of the container if there are multiple instances of a
                                service [default: 1]. (default 1)
-T, --no-TTY docker compose exec Disable pseudo-TTY allocation. By default docker compose exec
                                allocates a TTY. (default true)
    --privileged                Give extended privileges to the process.
-u, --user string               Run the command as this user.
-w, --workdir string            Path to workdir directory for this command..
```

3.7 help

Get help on a command.

```
docker-compose help [COMMAND]
```

3.8 images

List images used by the created containers.

```
docker-compose images [OPTIONS] [--] [SERVICE...]
```

Options:

```
-q, --quiet      Only display IDs
```

3.9 kill

Force stop service containers.

```
docker-compose kill [OPTIONS] [SERVICE...]
```

Options:

```
    --remove-orphans        Remove containers for services not defined in the Compose file..
-s, --signal string         SIGNAL to send to the container. Default signal is SIGKILL.
```

3.10 logs

View output from containers.

```
docker-compose logs [OPTIONS] [SERVICE...]
```

Options:

```
-f, --follow          Follow log output.
    --no-color        Produce monochrome output.
    --no-log-prefix   Don't print prefix in logs
    --since string    Show logs since a timestamp (e.g. 2013-01-02T13:23:37Z) or relative
                      (e.g. 42m for 42 minutes)
    --tail string     Number of lines to show from the end of the logs for each container.
-t, --timestamps      Show timestamps.
    --until string    Show logs before a timestamp (e.g. 2013-01-02T13:23:37Z) or relative
                      (e.g. 42m for 42 minutes)
```

3.11 ls

List running compose projects

```
docker-compose ls [OPTIONS]
```

Options:

```
-a, --all             Show all stopped Compose projects
    --filter filter   Filter output based on conditions provided.
    --format string   Format the output. Values: [pretty | json]. (default "pretty")
-q, --quiet           Only display IDs.
```

3.12 pause

Pause services.

```
docker-compose pause [SERVICE...]
```

3.13 port

Print the public port for a port binding.

```
docker-compose port [OPTIONS] SERVICE PRIVATE_PORT
```

Options:

```
    --index int       index of the container if service has multiple replicas (default 1)
    --protocol string  tcp or udp (default "tcp")
```

3.14 ps

List containers.

```
docker-compose ps [OPTIONS] [SERVICE...]
```

Options:

```
-a, --all                 Show all stopped containers (including those created by the run
                          command)
    --filter string       Filter services by a property (supported filters: status).
    --format string       Format the output. Values: [pretty | json] (default "pretty")
-q, --quiet               Only display IDs
    --services            Display services
    --status stringArray  Filter services by status. Values: [paused | restarting | removing |
                          running | dead | created | exited]
```

3.15 pull

Pulls images for services defined in a Compose file, but does not start the containers.

```
docker-compose pull [OPTIONS] [SERVICE...]
```

Options:

```
    --ignore-pull-failures   Pull what it can and ignores images with pull failures.
    --include-deps           Also pull services declared as dependencies
-q, --quiet                  Pull without printing progress information
```

3.16 push

Pushes service images

```
docker-compose push [OPTIONS] [SERVICE...]
```

Options:

```
    --ignore-push-failures  Push what it can and ignores images with push failures
```

3.17 restart

Restart service containers

```
docker-compose restart [OPTIONS] [SERVICE...]
```

Options:

```
  -t, --timeout TIMEOUT      Specify a shutdown timeout in seconds (default 10)
```

3.18 rm

Removes stopped service containers.

```
docker-compose rm [OPTIONS] [SERVICE...]
```

By default, anonymous volumes attached to containers will not be removed. You can override this with `-v`. To list all volumes, use `docker volume ls`.

Any data which is not in a volume will be lost.

Options:

```
-f, --force   Don't ask to confirm removal
-s, --stop    Stop the containers, if required, before removing
-v, --volumes Remove any anonymous volumes attached to containers
```

3.19 run

Run a one-off command on a service.

```
docker-compose run [OPTIONS] SERVICE [COMMAND] [ARGS...]
```

Options:

```
-d, --detach              Detached mode: Run container in the background, print new container
                          name.
    --entrypoint string   Override the entrypoint of the image.
-e, --env stringArray     Set environment variables
-i, --interactive         Keep STDIN open even if not attached. (default true)
-l, --label stringArray   Add or override a label
    --name string         Assign a name to the container
-T, --no-TTY              Disable pseudo-TTY allocation (default: auto-detected).
                          (default true)
    --no-deps             Don't start linked services.
-p, --publish stringArray Publish a container's port(s) to the host.
```

```
    --quiet-pull           Pull without printing progress information.
    --rm                   Automatically remove the container when it exits
    --service-ports        Run command with the service's ports enabled and mapped to the host.
    --use-aliases          Use the service's network useAliases in the network(s) the container
                           connects to.
-u, --user string          Run as specified username or uid
-v, --volume stringArray   Bind mount a volume.
-w, --workdir string       Working directory inside the container
```

3.20 start

Start services

```
docker-compose start [SERVICE...]
```

3.21 stop

Stop services

```
docker-compose stop [OPTIONS] [SERVICE...]
```

Options:

```
-t, --timeout int   Specify a shutdown timeout in seconds (default 10)
```

3.22 top

Display the running processes

```
docker-compose top [SERVICE...]
```

3.23 unpause

Unpause services.

```
docker-compose unpause [SERVICE...]
```

3.24 up

Create and start containers

```
docker-compose up [OPTIONS] [SERVICE...]
```

Options:

```
    --abort-on-container-exit   Stops all containers if any container was stopped. Incompatible
                                with -d
    --always-recreate-deps      Recreate dependent containers. Incompatible with --no-recreate.
    --attach stringArray        Attach to service output.
    --attach-dependencies       Attach to dependent containers.
    --build                     Build images before starting containers.
-d, --detach                    Detached mode: Run containers in the background
    --exit-code-from string     Return the exit code of the selected service container. Implies
                                --abort-on-container-exit
    --force-recreate            Recreate containers even if their configuration and image
                                haven't changed.
    --no-build                  Don't build an image, even if it's missing.
    --no-color                  Produce monochrome output.
    --no-deps                   Don't start linked services.
    --no-log-prefix             Don't print prefix in logs.
    --no-recreate               If containers already exist, don't recreate them. Incompatible
                                with --force-recreate.
    --no-start                  Don't start the services after creating them.
    --pull string               Pull image before running ("always"|"missing"|"never") (default
```

```
                                "missing")
    --quiet-pull                Pull without printing progress information.
    --remove-orphans            Remove containers for services not defined in the Compose file.
-V, --renew-anon-volumes        Recreate anonymous volumes instead of retrieving data from the
                                previous containers.
    --scale scale               Scale SERVICE to NUM instances. Overrides the scale setting in
                                the Compose file if present.
-t, --timeout int               Use this timeout in seconds for container shutdown when attached
                                or when containers are already running. (default 10)
    --wait                      Wait for services to be running|healthy. Implies detached mode.
```

3.25 version

Show the Docker Compose version information

```
docker-compose version [OPTIONS]
```

Options:

```
-f, --format string   Format the output. Values: [pretty | json]. (Default: pretty)
    --short           Shows only Compose's version number.
```

4 DOCKER-COMPOSE.YML REFERENCE

From `https://docs.docker.com/compose/compose-file/`

The Compose file is a YAML file defining services, networks, and volumes for a Docker application. The latest and recommended version of the Compose file format is defined by the Compose Specification[2]. The Compose spec merges the legacy 2.x and 3.x versions, aggregating properties across these formats and is implemented by **Compose 1.27.0+**

The default path for a Compose file is `compose.yaml` (preferred) or `compose.yml` in working directory. Compose implementations SHOULD also support `docker-compose.yaml` and `docker-compose.yml` for backward compatibility. If both files exist, Compose implementations MUST prefer canonical `compose.yaml` one.

Multiple Compose files can be combined together to define the application model.

4.1 Profiles

Profiles allow to adjust the Compose application model for various usages and environments. A Compose implementation SHOULD allow the user to define a set of active profiles. The exact mechanism is implementation specific and MAY include command line flags, environment variables, etc.

The Services top-level element supports a `profiles` attribute to define a list of named profiles. Services without a `profiles` attribute set MUST always be enabled. A service MUST be ignored by the Compose implementation when none of the listed `profiles` match the active ones, unless the service is explicitly targeted by a command. In that case its `profiles` MUST be added to the set of active profiles. All other top-level elements are not affected by `profiles` and are always active.

[2] https://github.com/compose-spec/compose-spec/blob/master/spec.md

References to other services (by `links`, `extends` or shared resource syntax `service:xxx`) MUST not automatically enable a component that would otherwise have been ignored by active profiles. Instead the Compose implementation MUST return an error.

4.1.1 Illustrative example

```
services:
  foo:
    image: foo
  bar:
    image: bar
    profiles:
      - test
  baz:
    image: baz
    depends_on:
      - bar
    profiles:
      - test
  zot:
    image: zot
    depends_on:
      - bar
    profiles:
      - debug
```

- Compose application model parsed with no profile enabled only contains the `foo` service.
- If profile `test` is enabled, model contains the services `bar` and `baz` which are enabled by the `test` profile and service `foo` which is always enabled.
- If profile `debug` is enabled, model contains both `foo` and `zot` services, but not `bar` and `baz` and as such the model is invalid regarding the `depends_on` constraint of `zot`.
- If profiles `debug` and `test` are enabled, model contains all services: `foo`, `bar`, `baz` and `zot`.
- If Compose implementation is executed with `bar` as explicit service to run, it and the `test` profile will be active even if `test` profile is not enabled by the user.
- If Compose implementation is executed with `baz` as explicit service to run, the service `baz` and the profile `test` will be active and `bar` will be pulled in by the `depends_on` constraint.
- If Compose implementation is executed with `zot` as explicit service to run, again the model will be invalid regarding the `depends_on` constraint of `zot` since zot and `bar` have no common profiles listed.
- If Compose implementation is executed with `zot` as explicit service to run and profile `test` enabled, profile `debug` is automatically enabled and service `bar` is pulled in as a dependency starting both services `zot` and `bar`.

4.2 Services top-level element

A Service is an abstract definition of a computing resource within an application which can be scaled/replaced independently from other components. Services are backed by a set of containers, run by the platform according to replication requirements and placement constraints. Being backed by containers, Services are defined by a Docker image and set of runtime arguments. All containers within a service are identically created with these arguments.

A Compose file MUST declare a `services` root element as a map whose keys are string representations of service names, and whose values are service definitions. A service definition contains the configuration that is applied to each container started for that service.

Each service MAY also include a Build section, which defines how to create the Docker image for the service. Compose implementations MAY support building docker images using this service definition (`build` support is an OPTIONAL aspect of the Compose specification).

Each Service defines runtime constraints and requirements to run its containers. The `deploy` section groups these constraints and allows the platform to adjust the deployment strategy to best match containers' needs with available resources (`deploy`[3] support is an OPTIONAL aspect of the Compose specification).

4.3 build[4]

Compose Specification is extended to support an OPTIONAL `build` subsection on services. This section define the build requirements for service container image. Only a subset of Compose file services MAY define such a Build subsection, others being created based on `Image` attribute. When a Build subsection is present for a service, it is *valid* for a Compose file to miss an `Image` attribute for corresponding service, as Compose implementation can build image from source.
Build can be either specified as a single string defining a context path, or as a detailed build definition.
In the former case, the whole path is used as a Docker context to execute a docker build, looking for a canonical `Dockerfile` at context root. Context path can be absolute or relative, and if so relative path MUST be resolved from Compose file parent folder. As an absolute path prevent the Compose file to be portable, Compose implementation SHOULD warn user accordingly.
In the later case, build arguments can be specified, including an alternate Dockerfile location. This one can be absolute or relative path. If `Dockerfile` path is relative, it MUST be resolved from context path. As an absolute path prevent the Compose file to be portable, Compose implementation SHOULD warn user if an absolute alternate Dockerfile path is used.

- **Consistency with Image**

When service definition do include both `Image` attribute and a `Build` section, Compose implementation can't guarantee a pulled image is strictly equivalent to building the same image from sources. Without any explicit user directives, Compose implementation with Build support MUST first try to pull Image, then build from source if image was not found on registry. Compose implementation MAY offer options to customize this behaviour by user request.

- **Publishing built images**

Compose implementation with Build support SHOULD offer an option to push built images to a registry. Doing so, it MUST NOT try to push service images without an `Image` attribute. Compose implementation SHOULD warn user about missing `Image` attribute which prevent image being pushed.
Compose implementation MAY offer a mechanism to compute an `Image` attribute for service when not explicitly declared in yaml file. In such a case, the resulting Compose configuration is

[3] https://docs.docker.com/compose/compose-file/deploy/
[4] https://docs.docker.com/compose/compose-file/build/

considered to have a valid `Image` attribute, whenever the actual raw yaml file doesn't explicitly declare one.

- **Illustrative sample**

The following sample illustrates Compose specification concepts with a concrete sample application. The sample is non-normative.

```
services:
  frontend:
    image: awesome/webapp
    build: ./webapp

  backend:
    image: awesome/database
    build:
      context: backend
      dockerfile: ../backend.Dockerfile

  custom:
    build: ~/custom
```

When used to build service images from source, such a Compose file will create three docker images:

- `awesome/webapp` docker image is built using `webapp` sub-directory within Compose file parent folder as docker build context. Lack of a `Dockerfile` within this folder will throw an error.
- `awesome/database` docker image is built using `backend` sub-directory within Compose file parent folder. `backend.Dockerfile` file is used to define build steps, this file is searched relative to context path, which means for this sample .. will resolve to Compose file parent folder, so `backend.Dockerfile` is a sibling file.
- a docker image is built using `custom` directory within user's HOME as docker context. Compose implementation warn user about non-portable path used to build image.

On push, both `awesome/webapp` and `awesome/database` docker images are pushed to (default) registry. `custom` service image is skipped as no `Image` attribute is set and user is warned about this missing attribute.

- **Build definition**

The `build` element define configuration options that are applied by Compose implementations to build Docker image from source. `build` can be specified either as a string containing a path to the build context or a detailed structure:

```
services:
  webapp:
    build: ./dir
```

Using this string syntax, only the build context can be configured as a relative path to the Compose file's parent folder. This path MUST be a directory and contain a `Dockerfile`.
Alternatively build can be an object with fields defined as follow

4.3.1 context

`context` defines either a path to a directory containing a Dockerfile, or a url to a git repository.

When the value supplied is a relative path, it MUST be interpreted as relative to the location of the Compose file. Compose implementations MUST warn user about absolute path used to define build context as those prevent Compose file from being portable.

```
build:
  context: ./dir
```

4.3.2 dockerfile

`dockerfile` allows to set an alternate Dockerfile. A relative path MUST be resolved from the build context. Compose implementations MUST warn user about absolute path used to define Dockerfile as those prevent Compose file from being portable..

```
build:
  context: .
  dockerfile: webapp.Dockerfile
```

4.3.3 args

`args` define build arguments, i.e. Dockerfile `ARG` values.

Using following Dockerfile:

```
ARG GIT_COMMIT
RUN echo "Based on commit: $GIT_COMMIT"
```

Then specify the arguments under the `build` key. You can pass a mapping or a list:
`args` can be set in Compose file under the `build` key to define `GIT_COMMIT`. `args` can be set a mapping or a list:

```
build:
  context: .
  args:
    GIT_COMMIT: cdc3b19
```

```
build:
  context: .
  args:
    - GIT_COMMIT=cdc3b19
```

Value can be omitted when specifying a build argument, in which case its value at build time MUST be obtained by user interaction, otherwise build arg won't be set when building the Docker image.

```
args:
  - GIT_COMMIT
```

4.3.4 ssh

`ssh` defines SSH authentications that the image builder SHOULD use during image build (e.g., cloning private repository)

`ssh` property syntax can be either:

- `default` - let the builder connect to the ssh-agent.
- `ID=path` - a key/value definition of an ID and the associated path. Can be either a PEM file, or path to ssh-agent socket

Simple `default` sample

```
build:
  context: .
  ssh:
    - default   # mount the default ssh agent
```

or

```
build:
  context: .
  ssh: ["default"]   # mount the default ssh agent
```

Using a custom id `myproject` with path to a local SSH key:

```
build:
  context: .
  ssh:
    - myproject=~/.ssh/myproject.pem
```

Image builder can then rely on this to mount SSH key during build.

4.3.5 cache_from

`cache_from` defines a list of sources the Image builder SHOULD use for cache resolution.

Cache location syntax MUST follow the global format `[NAME|type=TYPE[,KEY=VALUE]]`. Simple `NAME` is actually a shortcut notation for `type=registry,ref=NAME`.

Compose Builder implementations MAY support custom types, the Compose Specification defines canonical types which MUST be supported:

- `registry` to retrieve build cache from an OCI image set by key `ref`

```
build:
  context: .
  cache_from:
    - alpine:latest
    - type=local,src=path/to/cache
    - type=gha
```

Unsupported caches MUST be ignored and not prevent user from building image.

4.3.6 cache_to

`cache_to` defines a list of export locations to be used to share build cache with future builds.

```
build:
  context: .
  cache_to:
   - user/app:cache
   - type=local,dest=path/to/cache
```

Cache target is defined using the same `type=TYPE[,KEY=VALUE]` syntax defined by `cache_from`.

Unsupported cache target MUST be ignored and not prevent user from building image.

4.3.7 extra_hosts

`extra_hosts` adds hostname mappings at build-time. Use the same syntax as `extra_hosts`.

```
extra_hosts:
  - "somehost:162.242.195.82"
  - "otherhost:50.31.209.229"
```

Compose implementations MUST create matching entry with the IP address and hostname in the container's network configuration, which means for Linux `/etc/hosts` will get extra lines:

```
162.242.195.82  somehost
50.31.209.229   otherhost
```

4.3.8 isolation

`isolation` specifies a build's container isolation technology. Like isolation supported values are platform-specific.

4.3.9 Labels

`labels` add metadata to the resulting image. `labels` can be set either as an array or a map.

Reverse-DNS notation SHOULD be used to prevent labels from conflicting with those used by other software.

```
build:
  context: .
  labels:
    com.example.description: "Accounting webapp"
    com.example.department: "Finance"
    com.example.label-with-empty-value: ""
```

```
build:
  context: .
  labels:
    - "com.example.description=Accounting webapp"
    - "com.example.department=Finance"
    - "com.example.label-with-empty-value"
```

4.3.10 no_cache

`no_cache` disables image builder cache and enforces a full rebuild from source for all image layers. This only applies to layers declared in the Dockerfile, referenced images COULD be retrieved from local image store whenever tag has been updated on registry (see pull).

4.3.11 pull

`pull` requires the image builder to pull referenced images (`FROM` Dockerfile directive), even if those are already available in the local image store.

4.3.12 shm_size

`shm_size` set the size of the shared memory (`/dev/shm` partition on Linux) allocated for building Docker image. Specify as an integer value representing the number of bytes or as a string expressing a byte value.

```
build:
  context: .
  shm_size: '2gb'
```

```
build:
  context: .
  shm_size: 10000000
```

4.3.13 target

`target` defines the stage to build as defined inside a multi-stage `Dockerfile`.

```
build:
  context: .
  target: prod
```

4.3.14 secrets

`secrets` grants access to sensitive data defined by secrets on a per-service build basis. Two different syntax variants are supported: the short syntax and the long syntax.

Compose implementations MUST report an error if the secret isn't defined in the `secrets` section of the Compose file.

SHORT SYNTAX

The short syntax variant only specifies the secret name. This grants the container access to the secret and mounts it as read-only to `/run/secrets/<secret_name>` within the container. The source name and destination mountpoint are both set to the secret name.

The following example uses the short syntax to grant the build of the `frontend` service access to the `server-certificate` secret. The value of `server-certificate` is set to the contents of the file `./server.cert`.

```
services:
  frontend:
    build:
      context: .
      secrets:
        - server-certificate
secrets:
  server-certificate:
    file: ./server.cert
```

LONG SYNTAX

The long syntax provides more granularity in how the secret is created within the service's containers.

- `source`: The name of the secret as it exists on the platform.
- `target`: The name of the file to be mounted in `/run/secrets/` in the service's task containers. Defaults to `source` if not specified.
- `uid` and `gid`: The numeric UID or GID that owns the file within `/run/secrets/` in the service's task containers. Default value is USER running container.

- `mode`: The permissions for the file to be mounted in `/run/secrets/` in the service's task containers, in octal notation. Default value is world-readable permissions (mode `0444`). The writable bit MUST be ignored if set. The executable bit MAY be set.

The following example sets name of the `my_secret` to `redis_secret` within the container, sets the mode to `0440` (group-readable) and sets the user and group to `103`. The `redis` service does not have access to the `my_other_secret` secret.

```
services:
  frontend:
    build:
      context: .
      secrets:
        - source: server-certificate
          target: server.cert
          uid: "103"
          gid: "103"
          mode: 0440
secrets:
  server-certificate:
    external: true
```

Service builds MAY be granted access to multiple secrets. Long and short syntax for secrets MAY be used in the same Compose file. Defining a secret in the top-level `secrets` MUST NOT imply granting any service build access to it. Such grant must be explicit within the service specification as a secrets service element.

4.3.15 tags

`tags` defines a list of tag mappings that MUST be associated to the build image. This list comes in addition of the `image` property defined in the service section

```
tags:
  - "myimage:mytag"
  - "registry/username/myrepos:my-other-tag"
```

4.3.16 platforms

`platforms` defines a list of target platforms.

```
build:
  context: "."
  platforms:
    - "linux/amd64"
    - "linux/arm64"
```

When the `platforms` attribute is omitted, Compose implementations MUST include the service's platform in the list of the default build target platforms.

Compose implementations SHOULD report an error in the following cases:

- when the list contains multiple platforms but the implementation is incapable of storing multi-platform images
- when the list contains an unsupported platform ```yml build: context: "." platforms:
    - "linux/amd64"
    - "unsupported/unsupported" ```
- when the list is non-empty and does not contain the service's platform

```
services:
frontend:
  platform: "linux/amd64"
  build:
    context: "."
    platforms:
      - "linux/arm64"
```

4.4 deploy[5]

Compose specification is a platform-neutral way to define multi-container applications. A Compose implementation supporting deployment of application model MAY require some additional metadata as the Compose application model is way too abstract to reflect actual infrastructure needs per service, or lifecycle constraints.

Compose Specification Deployment allows users to declare additional metadata on services so Compose implementations get relevant data to allocate adequate resources on platform and configure them to match user's needs.

Definitions

Compose Specification is extended to support an OPTIONAL `deploy` subsection on services. This section define runtime requirements for a service.

4.4.1 endpoint_mode

`endpoint_mode` specifies a service discovery method for external clients connecting to a service. Default and available values are platform specific, anyway the Compose specification define two canonical values:

- `endpoint_mode: vip`: Assigns the service a virtual IP (VIP) that acts as the front end for clients to reach the service on a network. Platform routes requests between the client and nodes running the service, without client knowledge of how many nodes are participating in the service or their IP addresses or ports.
- `endpoint_mode: dnsrr`: Platform sets up DNS entries for the service such that a DNS query for the service name returns a list of IP addresses (DNS round-robin), and the client connects directly to one of these.

```
services:
  frontend:
    image: awesome/webapp
    ports:
      - "8080:80"
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip
```

[5] https://docs.docker.com/compose/compose-file/deploy/

4.4.2 labels

`labels` specifies metadata for the service. These labels MUST *only* be set on the service and *not* on any containers for the service. This assumes the platform has some native concept of "service" that can match Compose application model.

```
services:
  frontend:
    image: awesome/webapp
    deploy:
      labels:
        com.example.description: "This label will appear on the web service"
```

4.4.3 mode

`mode` define the replication model used to run the service on platform. Either `global` (exactly one container per physical node) or `replicated` (a specified number of containers). The default is `replicated`.

```
services:
  frontend:
    image: awesome/webapp
    deploy:
      mode: global
```

4.4.4 placement

`placement` specifies constraints and preferences for platform to select a physical node to run service containers.

4.4.5 constraints

`constraints` defines a REQUIRED property the platform's node MUST fulfill to run service container. Can be set either by a list or a map with string values.

```
deploy:
  placement:
    constraints:
      - disktype=ssd
```

```
deploy:
  placement:
    constraints:
      disktype: ssd
```

4.4.6 preferences

`preferences` defines a property the platform's node SHOULD fulfill to run service container. Can be set either by a list or a map with string values.

```
deploy:
  placement:
    preferences:
      - datacenter=us-east
```

```
deploy:
  placement:
    preferences:
      datacenter: us-east
```

4.4.7 replicas

If the service is `replicated` (which is the default), `replicas` specifies the number of containers that SHOULD be running at any given time.

```
services:
  frontend:
    image: awesome/webapp
    deploy:
      mode: replicated
      replicas: 6
```

4.4.8 resources

`resources` configures physical resource constraints for container to run on platform. Those constraints can be configured as a:

- `limits`: The platform MUST prevent container to allocate more
- `reservations`: The platform MUST guarantee container can allocate at least the configured amount

```
services:
  frontend:
    image: awesome/webapp
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
          pids: 1
        reservations:
          cpus: '0.25'
          memory: 20M
```

- **cpus**

`cpus` configures a limit or reservation for how much of the available CPU resources (as number of cores) a container can use.

- **memory**

`memory` configures a limit or reservation on the amount of memory a container can allocate, set as a string expressing a byte value.

- **pids**

`pids` tunes a container's PIDs limit, set as an integer.

- **devices**

`devices` configures reservations of the devices a container can use. It contains a list of reservations, each set as an object with the following parameters: `capabilities`, `driver`, `count`, `device_ids` and `options`.

Devices are reserved using a list of `capabilities`, making capabilities the only required field. A device MUST satisfy all the requested capabilities for a successful reservation.

- **capabilities**

`capabilities` are set as a list of strings, expressing both generic and driver specific capabilities. The following generic capabilities are recognized today:

  - `gpu`: Graphics accelerator
  - `tpu`: AI accelerator

To avoid name clashes, driver specific capabilities MUST be prefixed with the driver name. For example, reserving an nVidia CUDA-enabled accelerator might look like this:

```
deploy:
  resources:
    reservations:
      devices:
        - capabilities: ["nvidia-compute"]
```

- **driver**

A different driver for the reserved device(s) can be requested using `driver` field. The value is specified as a string.

```
deploy:
  resources:
    reservations:
      devices:
        - capabilities: ["nvidia-compute"]
          driver: nvidia
```

- **count**

If `count` is set to `all` or not specified, Compose implementations MUST reserve all devices that satisfy the requested capabilities. Otherwise, Compose implementations MUST reserve at least the number of devices specified. The value is specified as an integer.

```
deploy:
  resources:
    reservations:
      devices:
        - capabilities: ["tpu"]
          count: 2
```

`count` and `device_ids` fields are exclusive. Compose implementations MUST return an error if both are specified.

- **device_ids**

If `device_ids` is set, Compose implementations MUST reserve devices with the specified IDs providing they satisfy the requested capabilities. The value is specified as a list of strings.

```
deploy:
  resources:
    reservations:
      devices:
        - capabilities: ["gpu"]
          device_ids: ["GPU-f123d1c9-26bb-df9b-1c23-4a731f61d8c7"]
```

`count` and `device_ids` fields are exclusive. Compose implementations MUST return an error if both are specified.

- **options**

Driver specific options can be set with `options` as key-value pairs.

```
deploy:
  resources:
    reservations:
      devices:
        - capabilities: ["gpu"]
          driver: gpuvendor
          options:
            virtualization: false
```

4.4.9 restart_policy

`restart_policy` configures if and how to restart containers when they exit. If `restart_policy` is not set, Compose implementations MUST consider `restart` field set by service configuration.

- `condition`: One of `none`, `on-failure` or `any` (default: any).
- `delay`: How long to wait between restart attempts, specified as a duration (default: 0).
- `max_attempts`: How many times to attempt to restart a container before giving up (default: never give up). If the restart does not succeed within the configured `window`, this attempt doesn't count toward the configured `max_attempts` value. For example, if `max_attempts` is set to '2', and the restart fails on the first attempt, more than two restarts MUST be attempted.
- `window`: How long to wait before deciding if a restart has succeeded, specified as a duration (default: decide immediately).

```
deploy:
  restart_policy:
    condition: on-failure
    delay: 5s
    max_attempts: 3
    window: 120s
```

4.4.10 rollback_config

`rollback_config` configures how the service should be rollbacked in case of a failing update.

- `parallelism`: The number of containers to rollback at a time. If set to 0, all containers rollback simultaneously.
- `delay`: The time to wait between each container group's rollback (default 0s).
- `failure_`action: What to do if a rollback fails. One of continue or pause (default pause)
- `monitor`: Duration after each task update to monitor for failure (`ns|us|ms|s|m|h`) (default 0s).
- `max_failure_ratio`: Failure rate to tolerate during a rollback (default 0).
- order: Order of operations during rollbacks. One of stop-first (old task is stopped before starting new one), or start-first (new task is started first, and the running tasks briefly overlap) (default stop-first).

4.4.11 update_config

`update_config` configures how the service should be updated. Useful for configuring rolling updates.

- `parallelism`: The number of containers to update at a time.
- `delay`: The time to wait between updating a group of containers.
- `failure_action`: What to do if an update fails. One of `continue`, `rollback`, or `pause` (default: `pause`).
- `monitor`: Duration after each task update to monitor for failure (ns|us|ms|s|m|h) (default 0s).
- `max_failure_ratio`: Failure rate to tolerate during an update.
- `order`: Order of operations during updates. One of stop-first (old task is stopped before starting new one), or start-first (new task is started first, and the running tasks briefly overlap) (default `stop-first`).

```
deploy:
  update_config:
    parallelism: 2
    delay: 10s
    order: stop-first
```

4.5 Networks top-level element[6]

Networks are the layer that allow services to communicate with each other. The networking model exposed to a service is limited to a simple IP connection with target services and external resources, while the Network definition allows fine-tuning the actual implementation provided by the platform.

Networks can be created by specifying the network name under a top-level `networks` section. Services can connect to networks by specifying the network name under the service `networks` subsection

In the following example, at runtime, networks `front-tier` and `back-tier` will be created and the `frontend` service connected to the `front-tier` network and the `back-tier` network.

```
services:
```

[6] https://docs.docker.com/compose/compose-file/#networks-top-level-element

```
  frontend:
    image: awesome/webapp
    networks:
      - front-tier
      - back-tier

networks:
  front-tier:
  back-tier:
```

4.5.1 driver

`driver` specifies which driver should be used for this network. Compose implementations MUST return an error if the driver is not available on the platform.

```
driver: overlay
```

Default and available values are platform specific. Compose specification MUST support the following specific drivers: `none` and `host`

- `host` use the host's networking stack
- `none` disable networking

- **host or none**

The syntax for using built-in networks such as `host` and `none` is different, as such networks implicitly exists outside the scope of the Compose implementation. To use them one MUST define an external network with the name `host` or `none` and an alias that the Compose implementation can use (`hostnet` or `nonet` in the following examples), then grant the service access to that network using its alias.

```
services:
  web:
    networks:
      hostnet: {}

networks:
  hostnet:
    external: true
    name: host
services:
  web:
    ...
    networks:
      nonet: {}

networks:
  nonet:
    external: true
    name: none
```

4.5.2 driver_opts

`driver_opts` specifies a list of options as key-value pairs to pass to the driver for this network. These options are driver-dependent - consult the driver's documentation for more information. Optional.

```
driver_opts:
  foo: "bar"
  baz: 1
```

4.5.3 attachable

If `attachable` is set to `true`, then standalone containers SHOULD be able attach to this network, in addition to services. If a standalone container attaches to the network, it can communicate with services and other standalone containers that are also attached to the network.

```
networks:
  mynet1:
    driver: overlay
    attachable: true
```

4.5.4 enable_ipv6

`enable_ipv6` enable IPv6 networking on this network.

4.5.5 ipam

`ipam` specifies custom a IPAM configuration. This is an object with several properties, each of which is optional:

- `driver`: Custom IPAM driver, instead of the default.
- `config`: A list with zero or more configuration elements, each containing:
  - `subnet`: Subnet in CIDR format that represents a network segment
  - `ip_range`: Range of IPs from which to allocate container IPs
  - `gateway`: IPv4 or IPv6 gateway for the master subnet
  - `aux_addresses`: Auxiliary IPv4 or IPv6 addresses used by Network driver, as a mapping from hostname to IP
- `options`: Driver-specific options as a key-value mapping.

A full example:

```
ipam:
  driver: default
  config:
    - subnet: 172.28.0.0/16
      ip_range: 172.28.5.0/24
      gateway: 172.28.5.254
      aux_addresses:
        host1: 172.28.1.5
        host2: 172.28.1.6
        host3: 172.28.1.7
  options:
    foo: bar
    baz: "0"
```

4.5.6 internal

By default, Compose implementations MUST provides external connectivity to networks. `internal` when set to `true` allow to create an externally isolated network.

4.5.7 labels

Add metadata to containers using Labels. Can use either an array or a dictionary.

Users SHOULD use reverse-DNS notation to prevent labels from conflicting with those used by other software.

```
labels:
  com.example.description: "Financial transaction network"
  com.example.department: "Finance"
  com.example.label-with-empty-value: ""
```

```
labels:
  - "com.example.description=Financial transaction network"
  - "com.example.department=Finance"
  - "com.example.label-with-empty-value"
```

Compose implementations MUST set `com.docker.compose.project` and `com.docker.compose.network labels`.

4.5.8 external

If set to `true`, `external` specifies that this network's lifecycle is maintained outside of that of the application. Compose Implementations SHOULD NOT attempt to create these networks, and raises an error if one doesn't exist.

If `external` is set to `true` , then the resource is not managed by Compose. If `external` is set to `true` and the network configuration has other attributes set besides `name`, then Compose Implementations SHOULD reject the Compose file as invalid.

In the example below, `proxy` is the gateway to the outside world. Instead of attempting to create a network, Compose implementations SHOULD interrogate the platform for an existing network simply called `outside` and connect the `proxy` service's containers to it.

```
services:
  proxy:
    image: awesome/proxy
    networks:
      - outside
      - default
  app:
    image: awesome/app
    networks:
      - default

networks:
  outside:
    external: true
```

4.5.9 name

`name` sets a custom name for this network. The name field can be used to reference networks which contain special characters. The name is used as is and will **not** be scoped with the project name.

```
networks:
  network1:
    name: my-app-net
```

It can also be used in conjunction with the `external` property to define the platform network that the Compose implementation should retrieve, typically by using a parameter so the Compose file doesn't need to hard-code runtime specific values:

```
networks:
  network1:
    external: true
    name: "${NETWORK_ID}"
```

4.6 Volumes top-level element[7]

`volumes` defines mount host paths or named volumes that MUST be accessible by service containers.

If the mount is a host path and only used by a single service, it MAY be declared as part of the service definition instead of the top-level `volumes` key.

To reuse a volume across multiple services, a named volume MUST be declared in the top-level `volumes` key.

This example shows a named volume (`db-data`) being used by the `backend` service, and a bind mount defined for a single service

```
services:
  backend:
    image: awesome/backend
    volumes:
      - type: volume
        source: db-data
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: /var/run/postgres/postgres.sock
        target: /var/run/postgres/postgres.sock

volumes:
  db-data:
```

- **Short syntax**

The short syntax uses a single string with colon-separated values to specify a volume mount (`VOLUME:CONTAINER_PATH`), or an access mode (`VOLUME:CONTAINER_PATH:ACCESS_MODE`).

- `VOLUME`: MAY be either a host path on the platform hosting containers (bind mount) or a volume name
- `CONTAINER_PATH`: the path in the container where the volume is mounted
- `ACCESS_MODE`: is a comma-separated `,` list of options and MAY be set to:
  - `rw`: read and write access (default)
  - `ro`: read-only access
  - `z`: SELinux option indicates that the bind mount host content is shared among multiple containers
  - `Z`: SELinux option indicates that the bind mount host content is private and unshared for other containers

[7] https://docs.docker.com/compose/compose-file/#volumes

***Note**: The SELinux re-labeling bind mount option is ignored on platforms without SELinux.*
***Note**: Relative host paths MUST only be supported by Compose implementations that deploy to a local container runtime...*

- **Long syntax**

The long form syntax allows the configuration of additional fields that can't be expressed in the short form.

- `type`: the mount type `volume`, `bind`, `tmpfs` or `npipe`
- `source`: the source of the mount, a path on the host for a bind mount, or the name of a volume defined in the top-level `volumes` key. Not applicable for a tmpfs mount.
- `target`: the path in the container where the volume is mounted
- `read_only`: flag to set the volume as read-only
- `bind`: configure additional bind options
  - `propagation`: the propagation mode used for the bind
  - `create_host_path`: create a directory at the source path on host if there is nothing present. Do nothing if there is something present at the path. This is automatically implied by short syntax for backward compatibility with docker-compose legacy.
  - `selinux`: the SELinux re-labeling option `z` (shared) or `Z` (private)
- `volume`: configure additional volume options
  - `nocopy`: flag to disable copying of data from a container when a volume is created
- `tmpfs`: configure additional tmpfs options
  - `size`: the size for the tmpfs mount in bytes (either numeric or as bytes unit)
  - `mode`: the filemode for the tmpfs mount as Unix permission bits as an octal number
- `consistency`: the consistency requirements of the mount. Available values are platform specific

4.6.1 volumes_from

`volumes_from` mounts all of the volumes from another service or container, optionally specifying read-only access (ro) or read-write (rw). If no access level is specified, then read-write MUST be used.

String value defines another service in the Compose application model to mount volumes from. The `container:` prefix, if supported, allows to mount volumes from a container that is not managed by the Compose implementation.

```
volumes_from:
  - service_name
  - service_name:ro
  - container:container_name
  - container:container_name:rw
```

4.6.2 working_dir

`working_dir` overrides the container's working directory from that specified by image (i.e. Dockerfile `WORKDIR`).

4.7 Configs configuration reference [* DESDE AQUÍ *]

The top-level `configs` declaration defines or references configs which can be granted to the services in this stack. The source of the config is either `file` or `external`.

- `file`: The config is created with the contents of the file at the specified path.
- `external`: If set to true, specifies that this config has already been created. Docker will not attempt to create it, and if it does not exist, a `config not found` error occurs.
- `name`: The name of the config object in Docker. This field can be used to reference configs that contain special characters. The name is used as is and will **not** be scoped with the stack name.
- `driver` and `driver_opts`: The name of a custom secret driver, and driver-specific options passed as key/value pairs. Only supported when using `docker stack`.
- `template_driver`: The name of the templating driver to use, which controls whether and how to evaluate the secret payload as a template. If no driver is set, no templating is used. Only supported when using `docker stack`.

In this example, `my_first_config` will be created (as `<stack_name>_my_first_config`) when the stack is deployed, and `my_second_config` already exists in Docker.

```
configs:
  my_first_config:
    file: ./config_data
  my_second_config:
    external: true
```

Another variant for external configs is when the name of the config in Docker is different from the name that will exist within the service. The following example modifies the previous one to use the external config called `redis_config`.

```
configs:
  my_first_config:
    file: ./config_data
  my_second_config:
    external:
      name: redis_config
```

You still need to grant access to the config to each service in the stack.

4.8 Secrets configuration reference

The top-level `secrets` declaration defines or references secrets which can be granted to the services in this stack. The source of the secret is either `file` or `external`.

- `file`: The secret is created with the contents of the file at the specified path.
- `external`: If set to true, specifies that this secret has already been created. Docker will not attempt to create it, and if it does not exist, a `secret not found` error occurs.
- `name`: The name of the secret object in Docker. This field can be used to reference secrets that contain special characters. The name is used as is and will **not** be scoped with the stack name.
- `template_driver`: The name of the templating driver to use, which controls whether and how to evaluate the secret payload as a template. If no driver is set, no templating is used. Only supported when using `docker stack`.

In this example, `my_first_secret` will be created (as `<stack_name>_my_first_secret`) when the stack is deployed, and `my_second_secret` already exists in Docker.

```
secrets:
  my_first_secret:
    file: ./secret_data
  my_second_secret:
    external: true
```

Another variant for external secrets is when the name of the secret in Docker is different from the name that will exist within the service. The following example modifies the previous one to use the external secret called `redis_secret`.

```
secrets:
  my_first_secret:
    file: ./secret_data
  my_second_secret:
    external: true
      name: redis_secret
```

You still need to grant access to the secrets to each service in the stack.

4.9 Variable substitution

Your configuration options can contain environment variables. Compose uses the variable values from the shell environment in which `docker-compose` is run. For example, suppose the shell contains `POSTGRES_VERSION=9.3` and you supply this configuration:

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```

When you run `docker-compose up` with this configuration, Compose looks for the `POSTGRES_VERSION` environment variable in the shell and substitutes its value in. For this example, Compose resolves the `image` to `postgres:9.3` before running the configuration.

If an environment variable is not set, Compose substitutes with an empty string. In the example above, if `POSTGRES_VERSION` is not set, the value for the `image` option is `postgres:`.

You can set default values for environment variables using a `.env` file, which Compose will automatically look for. Values set in the shell environment will override those set in the `.env` file.

Both `$VARIABLE` and `${VARIABLE}` syntax are supported. Additionally when using the 2.1 file format, it is possible to provide inline default values using typical shell syntax:

- `${VARIABLE:-default}` will evaluate to `default` if `VARIABLE` is unset or empty in the environment.
- `${VARIABLE-default}` will evaluate to `default` only if `VARIABLE` is unset in the environment.

Similarly, the following syntax allows you to specify mandatory variables:

- `${VARIABLE:?err}` exits with an error message containing `err` if `VARIABLE` is unset or empty in the environment.

- `${VARIABLE?err}` exits with an error message containing `err` if `VARIABLE` is unset in the environment.

Other extended shell-style features, such as `${VARIABLE/foo/bar}`, are not supported.

You can use a `$$` (double-dollar sign) when your configuration needs a literal dollar sign. This also prevents Compose from interpolating a value, so a `$$` allows you to refer to environment variables that you don't want processed by Compose.

```
web:
  build: .
  command: "$$VAR_NOT_INTERPOLATED_BY_COMPOSE"
```

If you forget and use a single dollar sign (`$`), Compose interprets the value as an environment variable and will warn you:

The `VAR_NOT_INTERPOLATED_BY_COMPOSE` is not set. Substituting an empty string.

4.10 Extension fields

It is possible to re-use configuration fragments using extension fields. Those special fields can be of any format as long as they are located at the root of your Compose file and their name start with the `x-` character sequence.

```
version: "3.9"
x-custom:
  items:
    - a
    - b
  options:
    max-size: '12m'
  name: "custom"
```

The contents of those fields are ignored by Compose, but they can be inserted in your resource definitions using YAML anchors. For example, if you want several of your services to use the same logging configuration:

```
logging:
  options:
    max-size: '12m'
    max-file: '5'
  driver: json-file
```

You may write your Compose file as follows:

```
version: "3.9"
x-logging:
  &default-logging
  options:
    max-size: '12m'
    max-file: '5'
  driver: json-file

services:
  web:
    image: myapp/web:latest
    logging: *default-logging
  db:
```

```
    image: mysql:latest
    logging: *default-logging
```

It is also possible to partially override values in extension fields using the YAML merge type. For example:

```
version: "3.9"
x-volumes:
  &default-volume
  driver: foobar-storage

services:
  web:
    image: myapp/web:latest
    volumes: ["vol1", "vol2", "vol3"]
volumes:
  vol1: *default-volume
  vol2:
    << : *default-volume
    name: volume02
  vol3:
    << : *default-volume
    driver: default
    name: volume-local
```

FULL INDEX