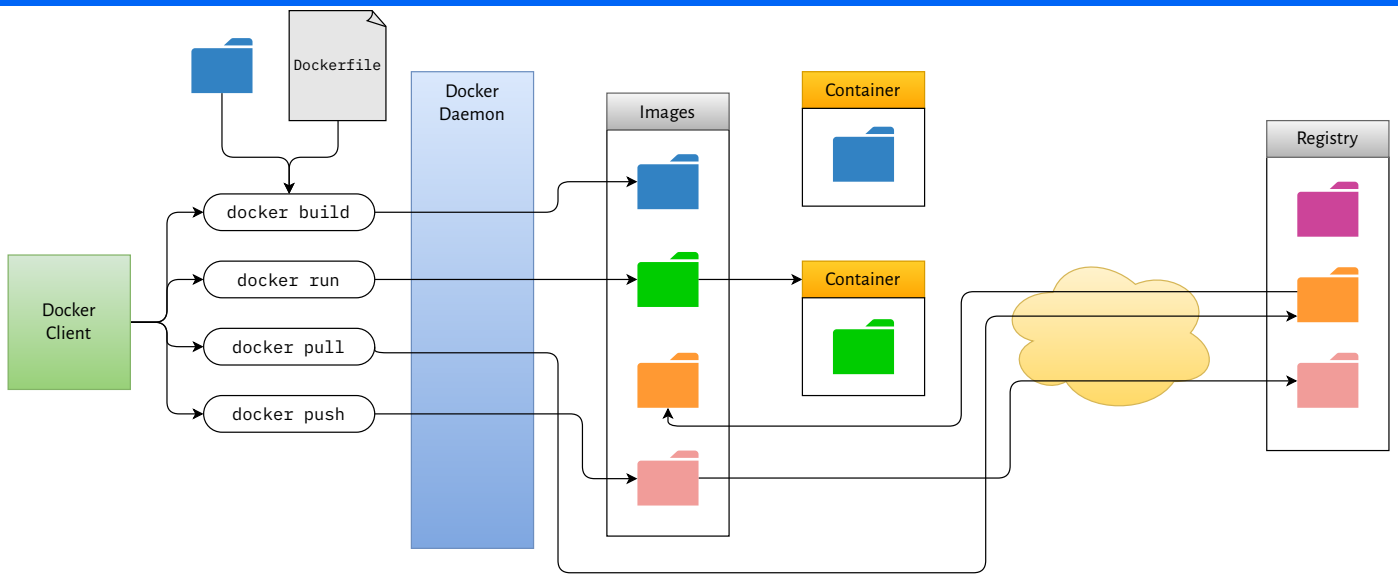
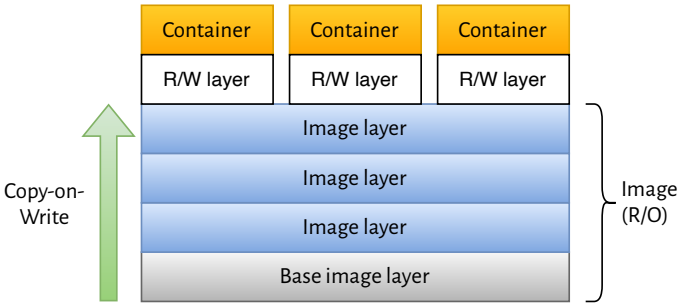
Docker Cheat Sheet

Last modified: **2024-04-13**

Architecture





Storage Drivers

Driver	FS	Level
`overlay2`	ext4, XFS (ftype=1)	File
`fuse-overlayfs`	Any	File
`btrfs`	BTRFS	Block
`zfs`	ZFS on Linux (ZoL)	Block
`vfs`	Any (no CoW)	File

Images

`docker images <repo>:<tag>`	List all images in `<repo>` filtered by `<tag>` (both are optional)
… `-a`	Show all images (i.e. include intermediate images)
… `-f "dangling=true"`	Leaf images with no tags attached (e.g. if removed by later build)
… `-f "label=<value>"`	Show images with attached label
… `--no-trunc`	Don't truncate image IDs
… `--digests`	Show digests
`docker inspect <image>`	Display detailed information about `<image>`
`docker rmi <image>`	Deletes local image `<image>`
`docker image prune`	Deletes all dangling local images (i.e. those not used by a named image)
… `-a`	Also delete unused local images (i.e. those not referenced by a container)
`docker history <image>`	Show the layers used to build a specified image

Last modified: **2024-04-13**

Building

`docker build -t <image> <dir>`	Builds `<image>` using build context `<dir>` containing `Dockerfile`
`docker build -t <image>:<tag> .`	Builds image `<image>` with tag `<tag>` using current dir as build context
… `--builder <name>`	Use alternative building instance `<name>`
… `--build-arg <var>=<value>`	Specify value for parameter declared with `ARG` in `Dockerfile`
… `-f <path>`	Look for `Dockerfile` at `<path>` instead of `<dir>/Dockerfile`
… `--no-cache`	Bypass the build cache
… `--cache-from=<src>`	Use `<src>` as source for cached images
… `--cache-to=<dst>`	After build, exports intermediate images to cache `<dst>`
… `--rm`	Remove any intermediate containers after successful build
… `--secret id=<id>,src=<file>`	Exposes `<file>` as secret `<id>` to match a `type=secret` mount
`docker buildx create <opts> <name>`	Creates a new builder instance
… `--name <name> --append`	Instead of creating a new builder, append new node to builder `<name>`
… `--driver <name>`	Use build driver `<name>`
… `--driver-opt "<opt>=<val>,…"`	Pass driver-specific options as comma-separated `<opt>=<val>` list
… `--use`	Also switch to newly-created building, as with `docker buildx use`
`docker buildx use <name>`	Use specified builder from now on
… `--default`	Set building as default for current context
… `--global`	Builder persists even after context is changed
`docker buildx ls`	Lists existing builder instances, with current one marked with `*`
`docker buildx rm <name>`	Removes the specified (or current if omitted) builder instance

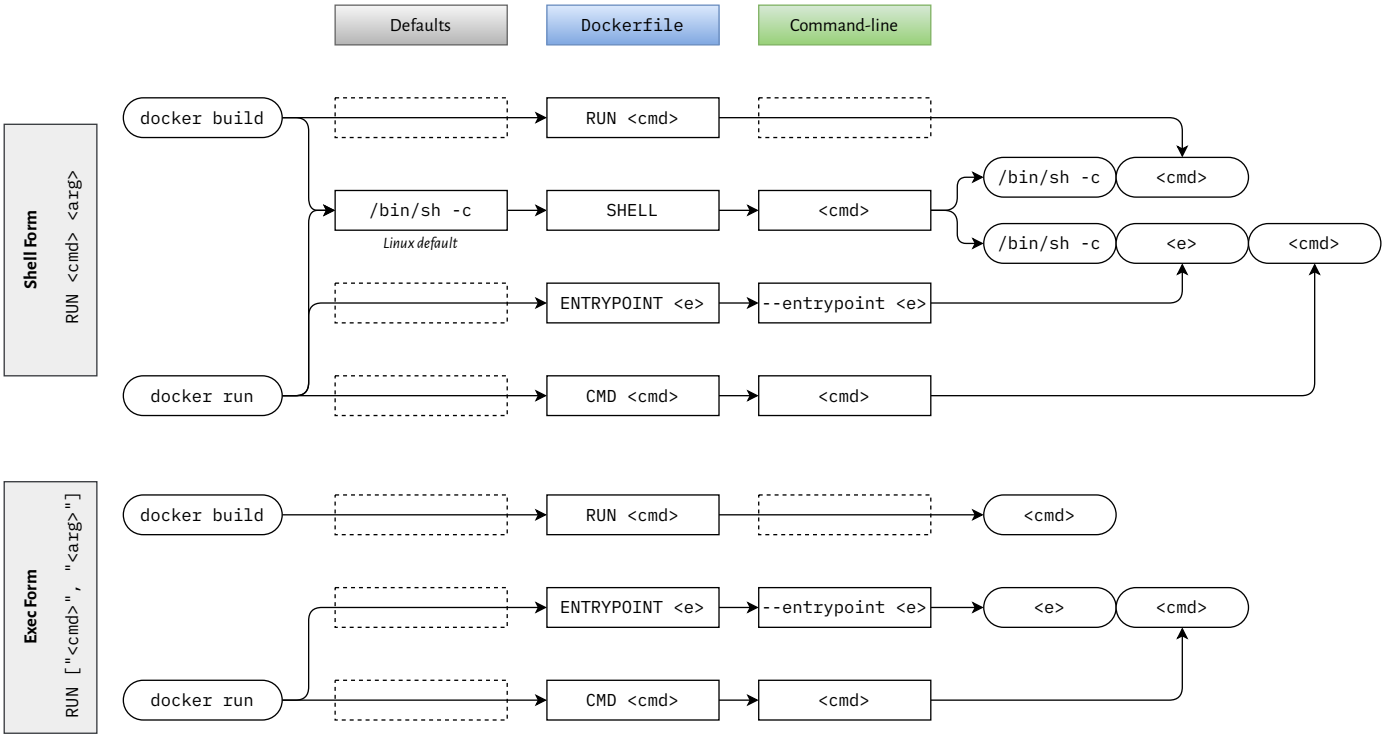
Driver	Notes	Auto load image	Cache export	Tarball output	Multi-arch images	BuiltKit configuration
`docker`	Uses bundled BuildKit (default)	✅				
`docker-container`	Creates BuildKit in container		✅	✅	✅	✅
`kubernetes`	Creates BuildKit in Kubernetes cluster		✅	✅	✅	✅
`remote`	Connects to remote BuiltKit daemon		✅	✅	✅	External

Driver Options: `docker-container`

`image`	Sets BuildKit image to use in container
`memory`	Set maximum memory usage of container
`memory-swap`	Sets swap memory limit for container
`cpu-quota`	Imposes CPU CFS quota on container
`cpu-period`	Sets CPU CFS scheduler period on container
`cpu-shares`	Configures CPU share (relative) of container
`cpuset-cpus`	Limits the set of CPU cores used by container
`cpuset-mems`	Limits set of CPU memory nodes available
`network`	Set network mode for the container
`cgroup-parent`	Only when using `cgroupfs` driver
`restart-policy`	See `restart` in compose reference
`env.<var>`	Sets value of environment `<var>`

Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Shell Form vs. Exec Form



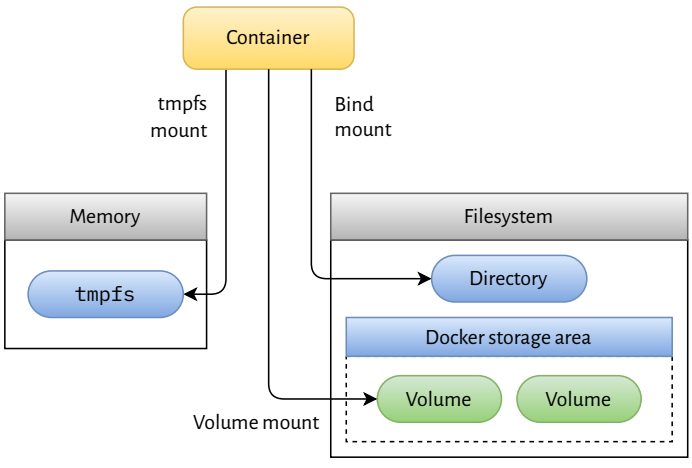
Containers

`docker run <image>`	Starts container with `<image>` and runs `CMD` from `Dockerfile`
`docker run <image> <cmd>`	Starts container with `<image>` and runs `<cmd>`
… `-a <stream>`	Attach only to `<stream>` (default is `stdout` and `stderr`)
… … `-d` *or* `--detach`	Detached mode: run container in background, print container ID
… `-e <var>=<value>`	Set environment variable `<var>` to `<value>` (or current value if omitted)
… `--entrypoint <cmd>`	Override the `ENTRYPOINT` from `Dockerfile`
… `-h <hostname>`	Set the hostname in the container to `<hostname>`
… `-i -t`	Interactive mode (keep `stdin` open) and allocate pseudo-TTY
… `--ip <addr4> --ip6 <addr6>`	Sets IPv4 address to `<addr4>` and IPv6 address to `<addr6>`
… `-m <size>`	Set memory limit to `<size>` (e.g. `512MB` or `2GB`)
… `--mount type=bind,src=<s>,dst=<d>`	Bind mount `<s>` on host into `<d>` within container
… `--mount type=volume,src=<n>,dst=<d>`	Bind volume named `<n>` on host into `<d>` within container
… `--name <name>`	Set container name to `<name>`
… `--network=<net>`	Connect container to `<net>`, created with `docker network create`
… `-p <ip>:<hport>:<cport>/tcp`	Bind container TCP port `<cport>` to `<hport>` of host interface `<ip>`
… `--read-only`	Mount container's root filesystem as read-only
… `-w <dir>`	Set current working directory in the container to `<dir>`

Containers (cont.)

`docker attach <container>`	Attach terminal to standard in/out/error of command in `<container>`
`docker cp <container>:<src> <dst>`	Copy file `<src>` in `<container>` to `<dst>` on host
`docker commit <container> <image>`	Write `<container>` current filesystem to `<image>`
`docker create <image>`	As `docker run`, but doesn't start the container
`docker diff <container>`	Show changes to filesystem in `<container>` since it was created
`docker exec <container> <cmd>`	Run additional command `<cmd>` in `<container>`
`docker inspect <container>`	Show detailed information about `<container>` in JSON
`docker kill -s SIGHUP <container>`	Send signal `SIGHUP` to command in `<container>` (default `SIGKILL`)
`docker logs --follow <container>`	Show logged `stdout` in `stderr` data, and continue to follow them
… `--since <datetime>`	Show all logs captured after `<datetime>` (either ISO, or `15m`, `30s`, etc.)
… `--tail <lines>`	Show only the most recent `<lines>` lines
`docker ps -a`	Show all containers (or just running ones without `-a`)
`docker ps --last <n>`	Show only `<n>` most recently created containers (also implicitly sets `-a`)
`docker pause <container>`	Pause execution of all processes in `<container>` as if with `SIGSTOP`.
`docker unpause <container>`	Restarts execution previously stopped with `docker pause`
`docker port <container>`	Display port mappings for `<container>`
`docker rename <old> <new>`	Rename container `<old>` to `<new>`
`docker restart <container>`	Restart `<container>`
`docker rm <container>`	Delete `<container>`
`docker stop <container>`	Stop `<container>` with `SIGTERM`, then `SIGKILL` after timeout
`docker start <container>`	Start stopped `<container>`
`docker stats`	Display a live data string for running containers
`docker top <container>`	Display the running processes in `<container>`

Mounts



- Volumes use a **volume driver** to store volumes
- Default volume driver is `local`
  - Takes no options on Windows
  - On Linux, takes options similar to `mount` command

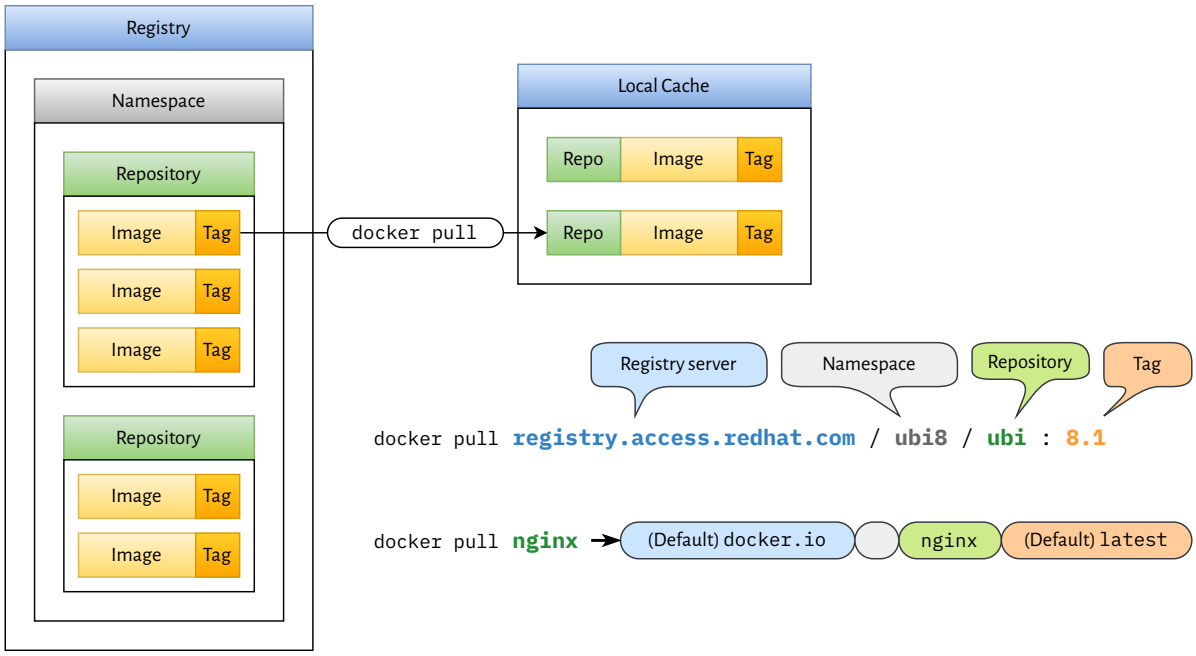
```
docker volume create --driver local \
--opt type=nfs \
--opt o=addr=1.2.3.4,rw \
--opt device=:/remote/path
```

- Further volume drivers are added by volume plugins
- Run `docker info` and look for `Plugins:` / `Volume:`

Volumes

`docker volume create <vol>`	Create volume `<vol>` (if omitted, Docker generates a random name)
… `-d <driver>`	Use alternative driver `<driver>` (typically requires volume plugins)
… `-o <option>=<value>`	Pass options directly to the volume driver (see **Mounts** section above)
`docker volume inspect <vol>`	Displays information about `<vol>` as JSON
`docker volume ls`	List all currently known volumes
`docker volume prune -a`	Remove volumes not used by any containers (without `-a`, only unnamed)
`docker volume rm <vol>`	Delete volume `<vol>`, will fail if used by at least one container

Registries



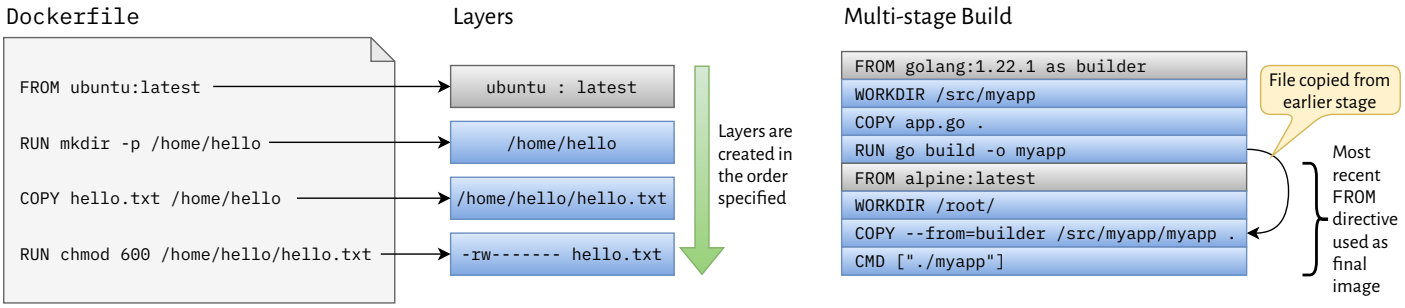
`docker pull <name>:<tag>`	Download image from repository `<name>` at tag `<tag>`
… `-a`	Pull all tags within the repository
`docker login <host>`	Log in to a specified registry
`docker tag <i> <host>/<repo>:<tag>`	Tag image `<i>` for pushing to `<repo>` on `<host>` with tag `<tag>`
`docker push <host>/<repo>:<tag>`	Upload image `<repo>:<tag>` to same repository on registry `<host>`
`docker search <host>/<term>`	Search for `<term>` on registry `<host>` (Docker Hub if not specified)

Cache Source/Destination Specifications

`type=local,src=<path>,tag=<tag>`		Push / pull from local directory (`s/src/dest/` on export)	
`type=registry,ref=<host>/<repo>:<tag>`		Push / pull from remote registry	
`type=inline`		Embed the cache in the image, and push them both together	
`mode=max`	Include intermediate layers (export only)	`ignore-error=true`	Ignore errors (export only)

Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Dockerfile



`FROM <image>:<tag> AS <name>`	Start new stage named `<name>` with `<image>:<tag>` as base image
… `--platform=<platform>`	Select `<platform>` from a multi-platform image
`RUN ["<cmd>", "<arg>", … ]`	Execute `<cmd> <arg> …` when building image (**exec** form)
`RUN <cmd> <arg>`	Execute `<cmd> <arg> …` when building image (**shell** form)
`RUN <<EOF`	Execute multiple commands in one step (**shell** form**, heredoc**)
`#!/usr/bin/env python`	*Optionally*, a shebang line can define an alternative shell
`<cmd>`	Each command is run with the specified shell
`<cmd>`	The entire block is considered one "command" and creates one layer
`EOF`	Terminate with a line containing *only* delimiter specified on the first line

`RUN --mount=type=bind,from=<stage>,source=<src>,target=<dst> <cmd>`

Bind mounts `<src>` in earlier stage `<stage>` (or build context if omitted) at `<dst>` in container for running `<cmd>`

Tip: *Bind mounted files only persist for a single instruction and so the file doesn't exist in the final image, which can be more efficient.*

`RUN --mount=type=cache,target=<dst>,sharing=locked <cmd>`

Mount empty directory for caching at `<dst>` in container for running `<cmd>` using single-writer locking (default is `shared`)

`RUN --mount=type=tmpfs,target=<dst>,size=<size> <cmd>`

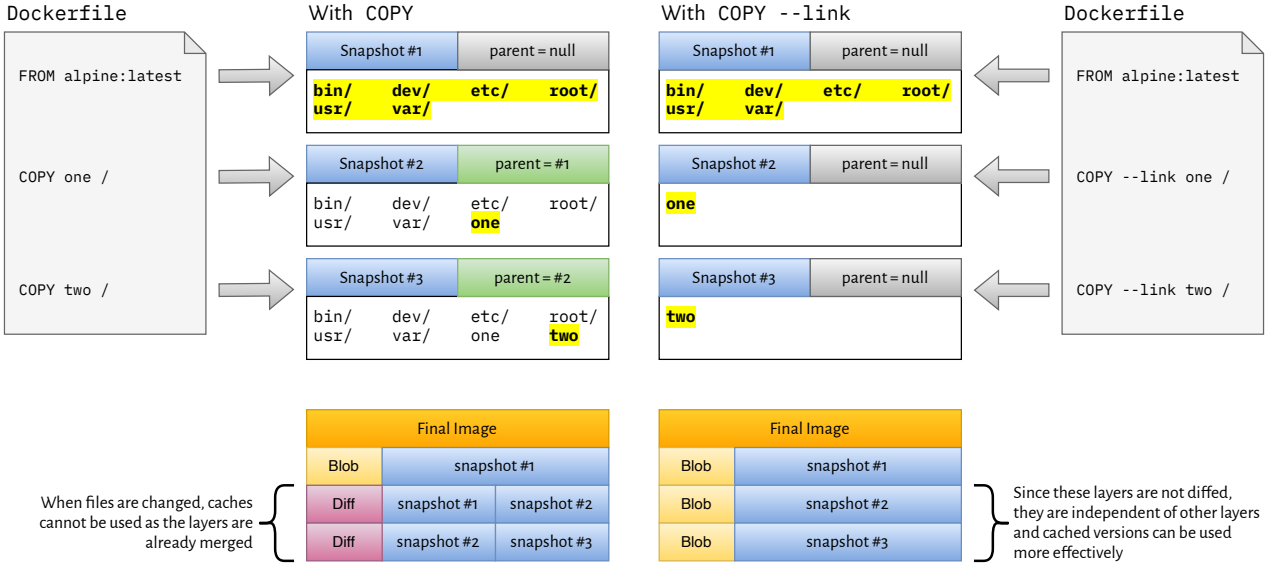
Mount a `tmpfs` at `<dst>` limited to `<size>` in container for running `<cmd>`

`RUN --mount=type=secret,id=<id>,target=<dst> <cmd>`

Mount secret `<id>` as `<dst>` in container, without including secret in the image (see `--secret` option to `docker build`)

`RUN --mount=type=ssh <cmd>`	Allow container to access keys from host SSH agent for running `<cmd>`
… `--network=<type>`	Specify `<type>` as `none` for no network, `host` for host's network
`CMD ["<cmd>", "<arg>", … ]`	Set default run command to `<cmd>` (**exec** form)
`CMD <cmd> <arg>`	Set default run command to `<cmd>` (**shell** form)
`ENTRYPOINT <cmd>`	Set prefix for run commands (**exec** and **shell** forms, see diagram p.2)
`SHELL ["<cmd>", … ]`	Override platform's default shell (**exec** form only, see diagram p.2)
`COPY <src> … <dst>`	Copies file(s) `<src>` (relative to context) into `<dst>` (relative to PWD)
… `--from=<src>`	Instead of build context, copy from specified stage or other image
… `--chown=<u>:<g> --chmod=<p>`	Specify owner user `<u>`, group `<g>` and permissions `<p>` of target
… `--link`	Places copied files into their own snapshot layer for better build caching
`WORKDIR <dir>`	Sets PWD, if not absolute then relative to existing PWD

Dockerfile (cont.)



`ADD <src> … <dir>`	Similar to `COPY` but supports fetching from remote URLs as well
`ADD http://eg.com/x.txt dst/`	Due to trailing slash, creates files `dst/x.txt`
`ADD foo.tar <dir>`	Will be unpacked in `<dir>` (also if compressed with **gzip**, **bzip2** and **xz**)
… `--link --chown --chmod`	Same meaning as for `COPY`
… `--checksum=<algo>:<hash>`	Validate hash of fetched file with `<algo>` (e.g. `sha256`) matches `<hash>`
`ARG <var>=<default>`	Declare build arg `<var>` with default, reference later with `$<var>`
`docker build --build-arg <var>=<value>`	… to override the value later, at build time
`ENV <var>=<value>`	Sets environment variable `<var>` to `<value>` both at build- *and* runtime

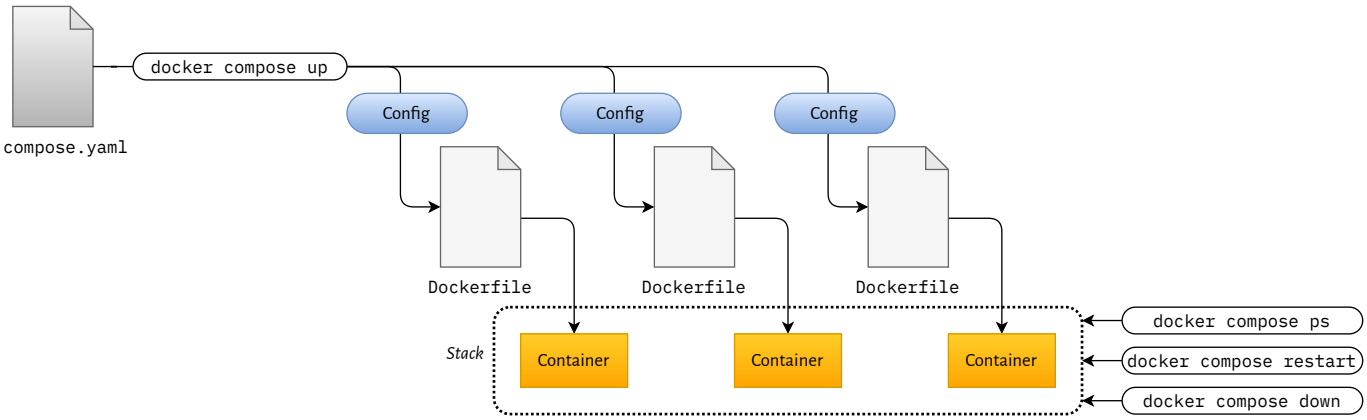
Tip: *If only required at build time, consider using* `ARG`*, or just setting for just one command, such as:* `RUN VAR=value cmd`

`EXPOSE <port>/<proto>`	Documents `<port>` as a listen port, `<proto>` is `tcp` if omitted

Tip: *This doesn't actually publish ports to the host, but passing* `-P` *to* `docker run` *will publish all* ***exposed*** *ports to random host ports.*

`HEALTHCHECK CMD <cmd>`	Specifies health check command, or `NONE` instead of `CMD` to disable
… `--interval=<duration>`	Run this time after start, and again at each interval (default `30s`)
… `--timeout=<duration>`	A check taking longer than this is considered failed (default `30s`)
… `--retries=<n>`	Consider container `unhealthy` after `<n>` consecutive failures (default `3`)
`LABEL <key>=<value> …`	Adds metadata to image, use double quotes as needed
`MAINTAINER <name>`	Sets the **Author** field, but in general `LABEL` should be used instead
`ONBUILD <instruction>`	Adds a trigger to be executed as if just after `FROM` in a derived build
`STOPSIGNAL <signal>`	Override default `SIGTERM` sent by `docker stop`
`USER <user>:<group>`	Sets default user and (optionally) group for remainder of stage
`VOLUME ["<dir>"]`	Creates specified mount point linked to new anonymous volume on host

Docker Compose

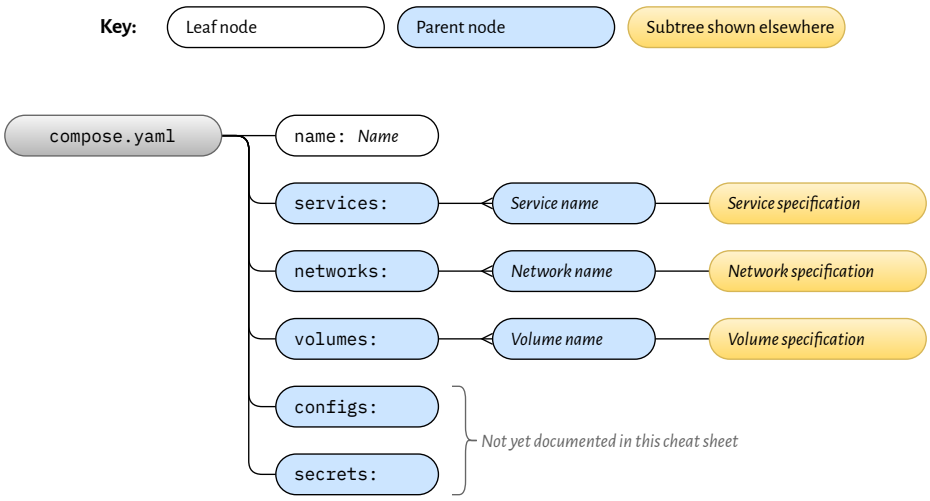


`docker compose up`	Build, (re)create, start containers, and attach to the merged output of them
… `--abort-on-container-exit`	Stops all containers once any container exits (cannot be used with `-d`)
… `-d` *or* `--detach`	Detached mode: run containers in the background
… `--force-recreate`	Re-create containers even if the image and configuration are the same
… `--no-recreate`	Don't re-create containers which already exist
… `--wait`	Used with detached mode, wait for containers to be running and healthy
… `-w` *or* `--watch`	Watch sources and rebuild/refresh containers on changes
`docker compose down`	Stops containers and removes containers, networks, volumes and images
… `--remove-orphans`	Remove containers for services no longer defined in Compose file
… `--rmi`	Remove images used by services
… `-v` *or* `--volumes`	Remove volumes names in Compose file & attached anonymous volumes
`docker compose build`	Runs only the build step
`docker compose config`	Display final configuration that will be applied
… `--format=json`	Renders in JSON instead of YAML
… `-o <file>`	Outputs to `<file>` instead of `stdout`
… `--images`	Just display list of images, each on one line
… `--services`	Just display list of service names, each on one line
… `--volumes`	Just display list of volume names, each on one line
`docker compose cp <svc>:<src> <dst>`	Copy files from `<src>` in container for `<svc>` to local `<dst>`
`docker compose create`	Runs only the container creation step
`docker compose events`	Stream events for all containers in the stack
… `--json`	Render events in JSON format
`docker compose exec <svc> <cmd>`	Equivalent to `docker exec` in container of specified `<svc>`
`docker compose images`	Lists images used by the containers in the stack
`docker compose kill`	Force stop containers

Docker Compose (cont.)

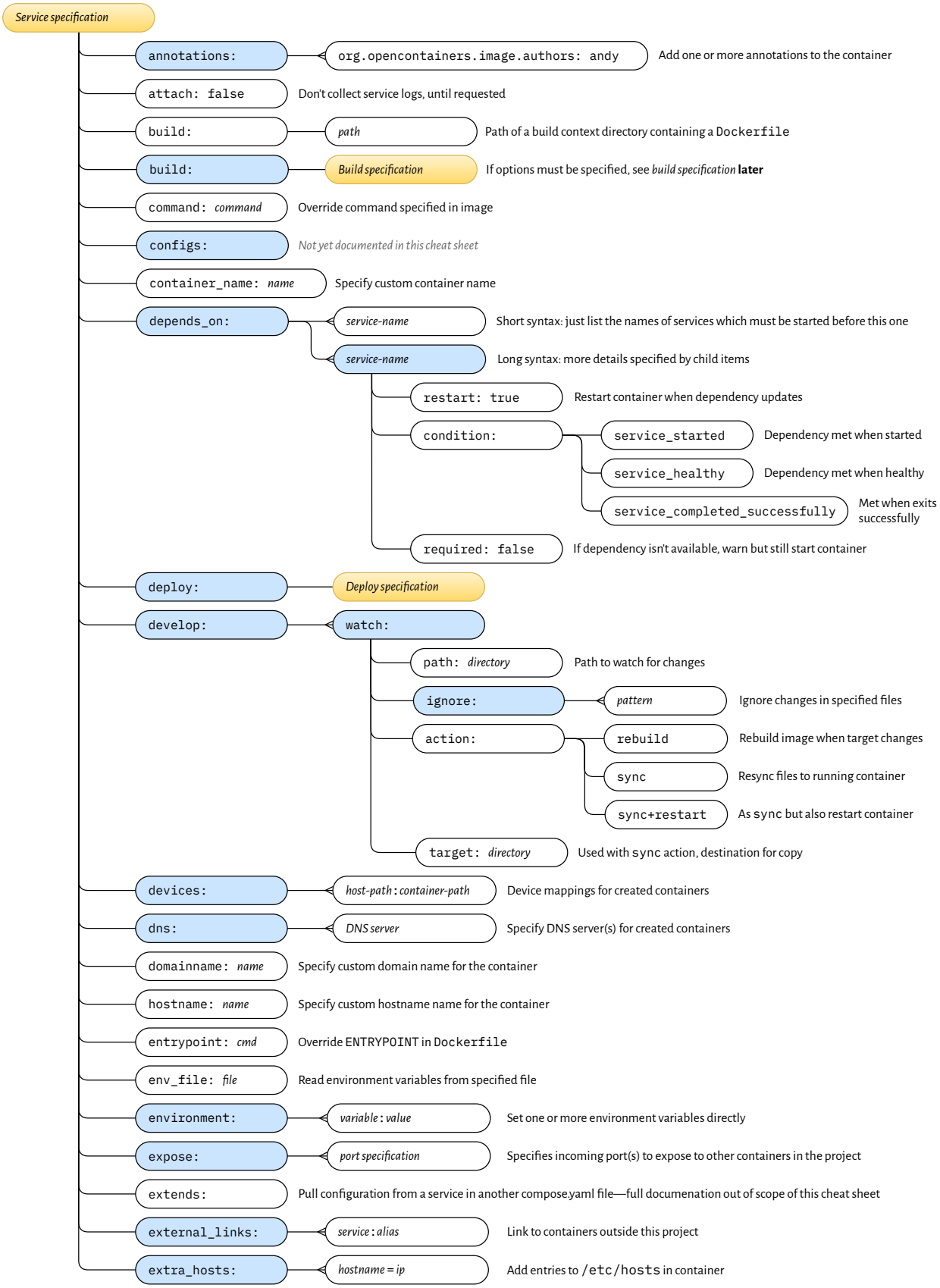
`docker compose logs`	Displays log output from all services
`docker compose ls`	Lists running Compose projects
`docker compose pause`	Pauses all containers in the project
`docker compose unpause`	Unpauses containers paused with `docker compose pause`
`docker compose port <svc> <port>`	Prints the public port bound to private `<port>` in `<svc>`
`docker compose ps`	Lists all containers for the project, including status and exposed ports
`docker compose pull`	Pulls images associated with services, but doesn't start containers
`docker compose push`	Pushes locally built images to their respective registries
`docker compose restart`	Restarts all stopped and running services in the project
`docker compose rm`	Removes stopped containers from the project
… `-s` *or* `--stop`	Also stops containers, if required, before removing
… `-v` *or* `--volumes`	Remove any anonymous volumes attached to the containers
`docker compose run <svc> <cmd>`	Starts specified `<svc>` from the project and runs `<cmd>` in it
… `-d` *or* `--detach`	Detached mode: run containers in the background
… `--rm`	Remove container when it exits
… `-P` *or* `--service-ports`	Also map all service's ports from the compose file to the host
… `-w <dir>`	Set current working directory in the container to `<dir>`
`docker compose start`	Starts existing containers for services in the project
`docker compose stop`	Stops running containers in the project without removing them
`docker compose top`	Displays running processes in all running containers in projects
`docker compose wait <svc>`	Blocks until the first of the specified services stops
`docker compose watch`	Watches build contexts for services and rebuild/restart when they change

Docker Compose File



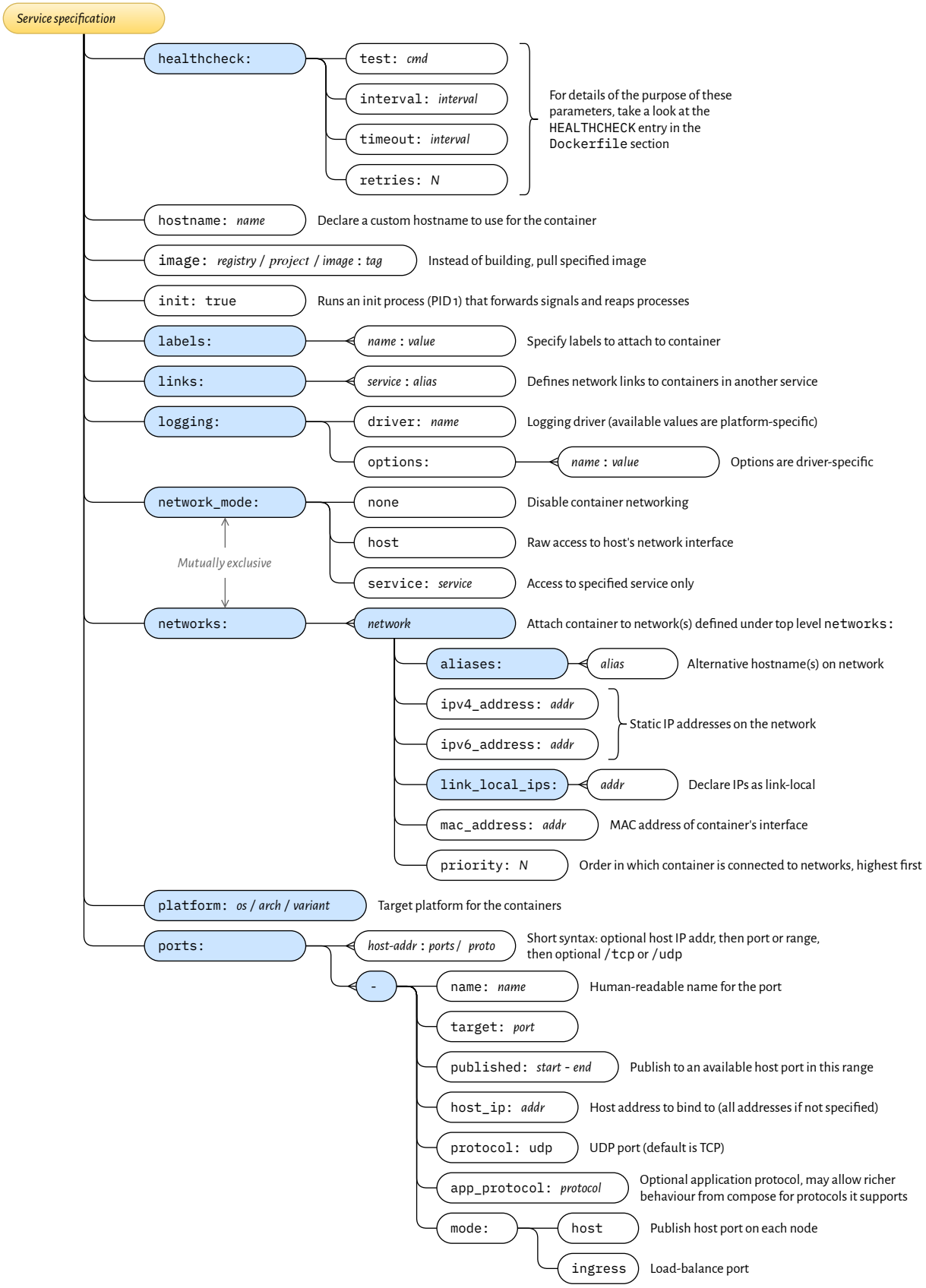
Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Docker Compose File (Service Specification)



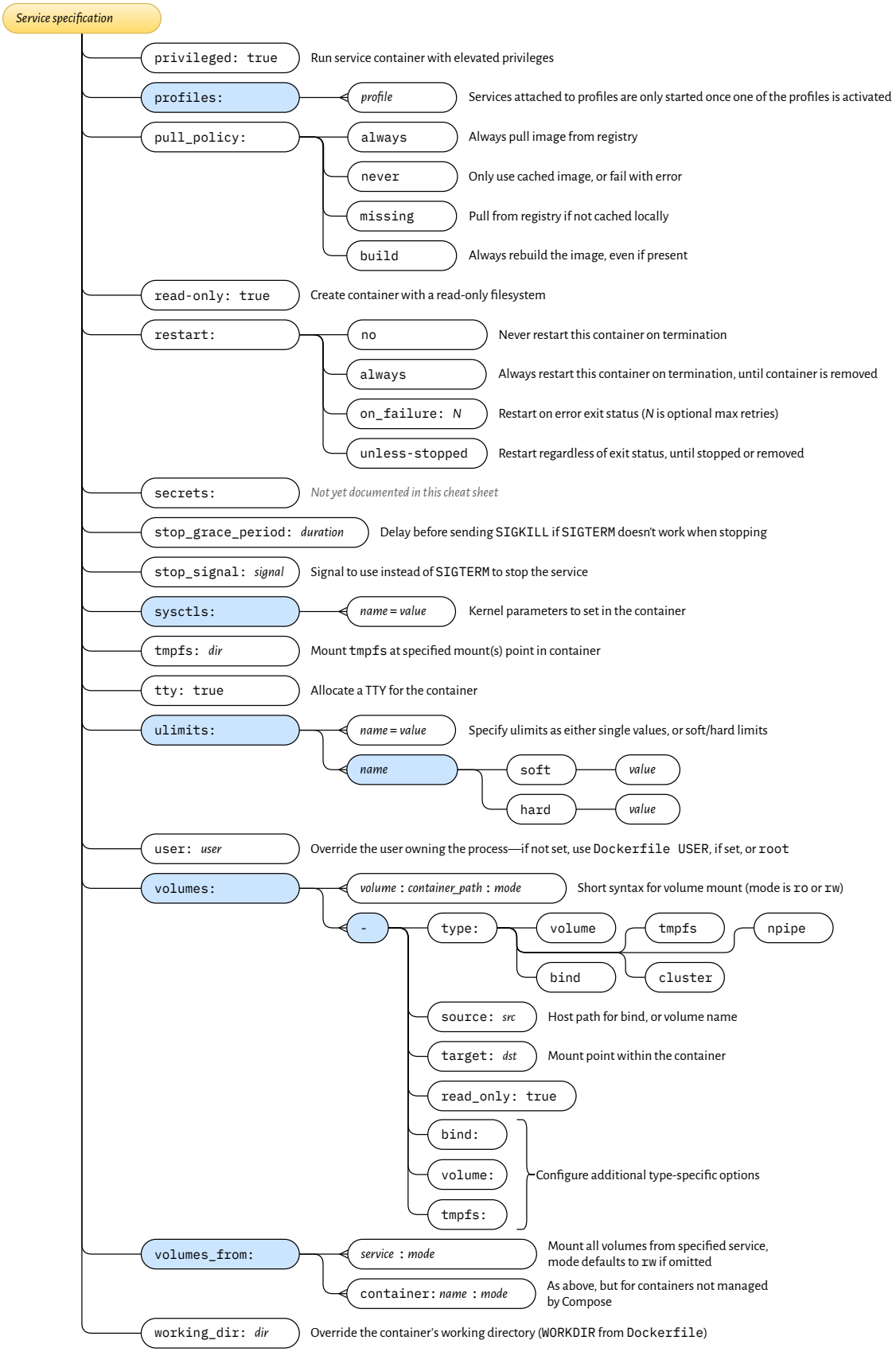
Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Docker Compose File (Service Specification, cont.)



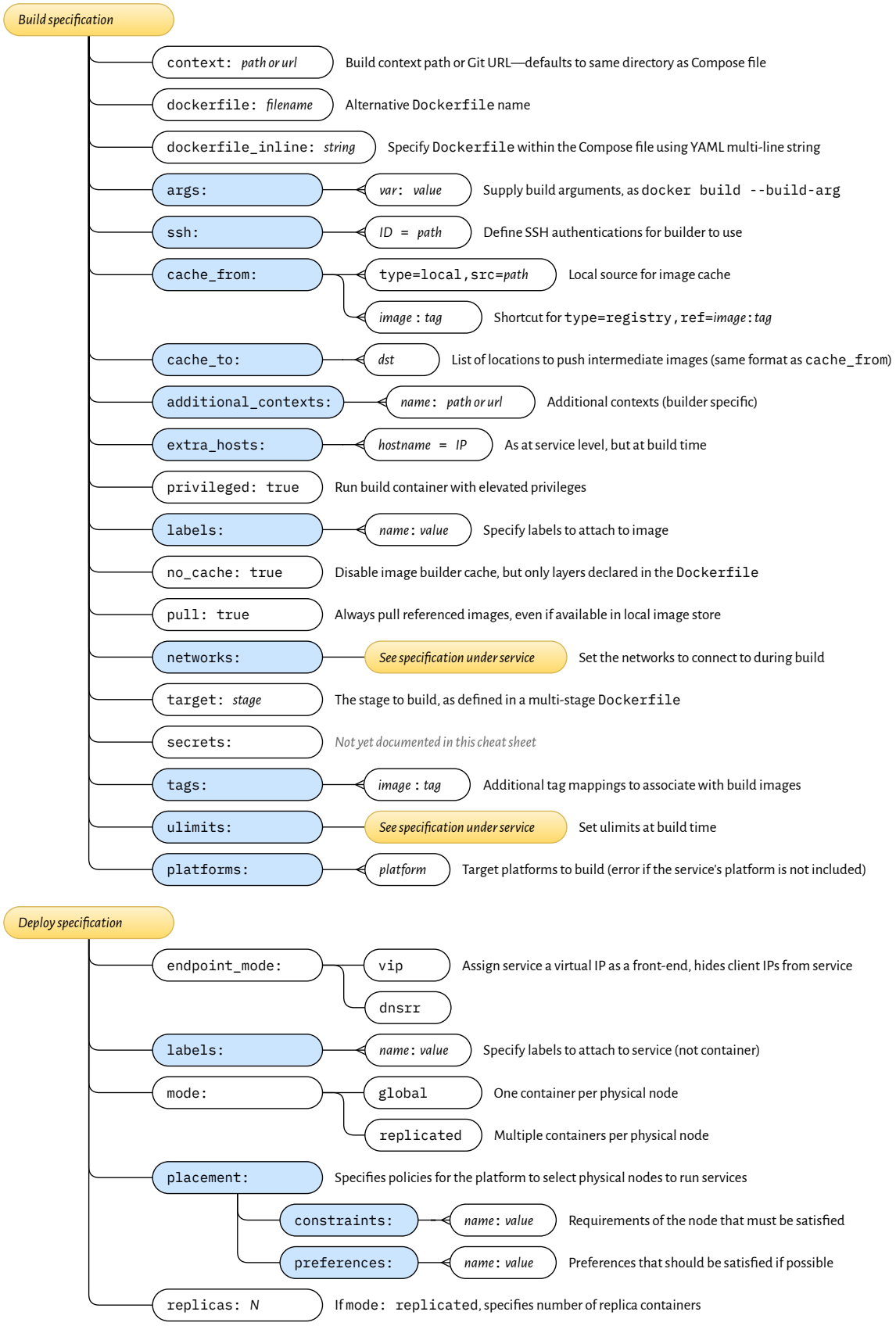
Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Docker Compose File (Service Specification, cont.)



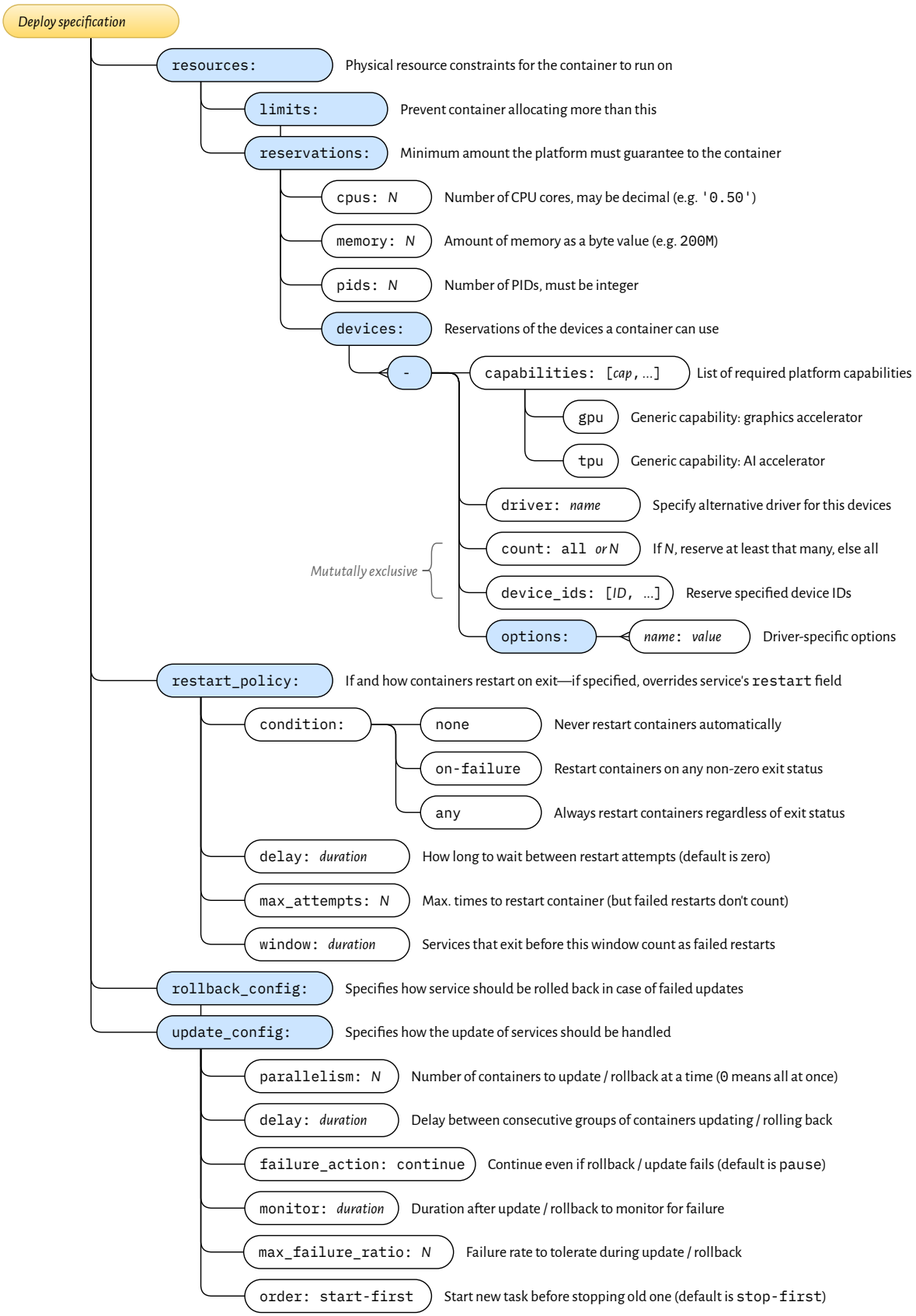
Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Docker Compose File (Service Specification, cont.)



Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

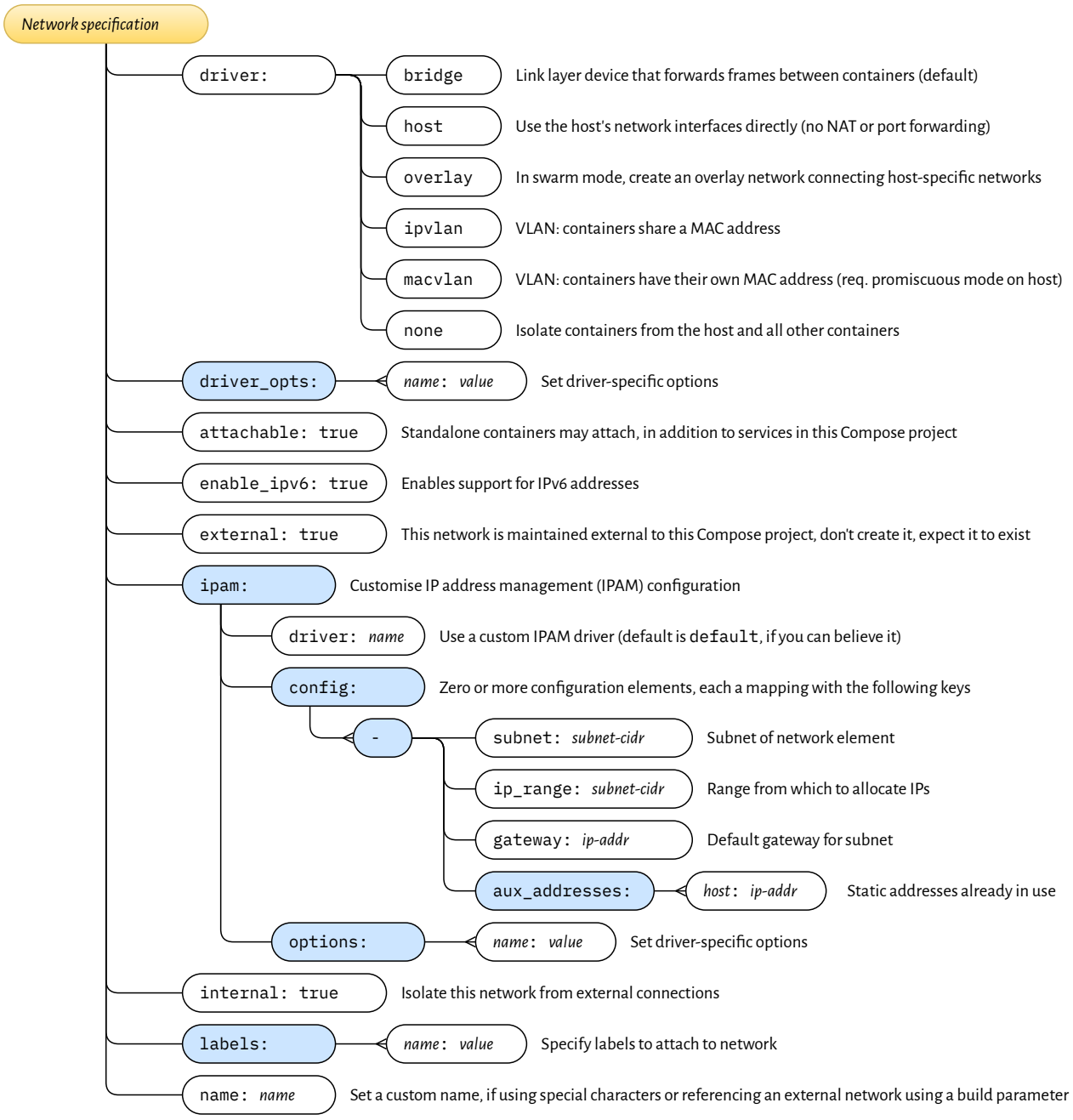
Docker Compose File (Service Specification, cont.)

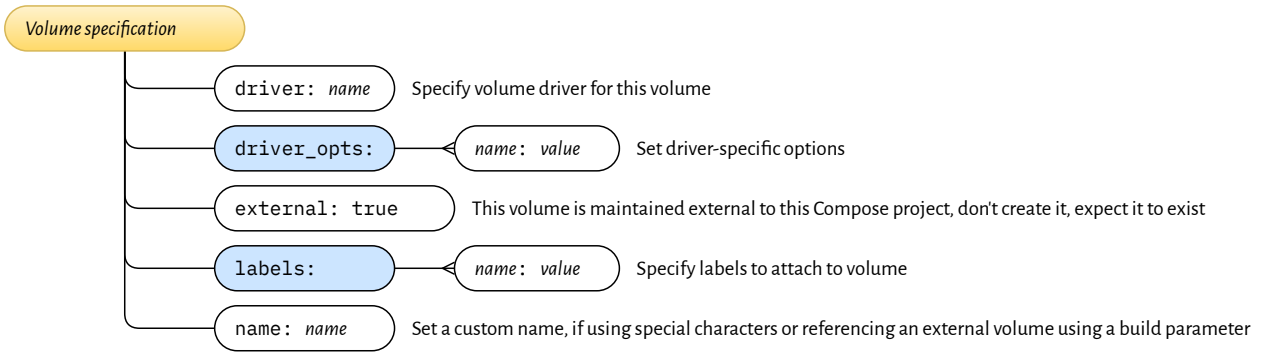


Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf

Last modified: **2024-04-13**

Docker Compose File (Network and Volume specifications)





Only some options are shown, this is not a comprehensive reference. Latest version: https://www.andy-pearce.com/docker-cheat-sheet.pdf