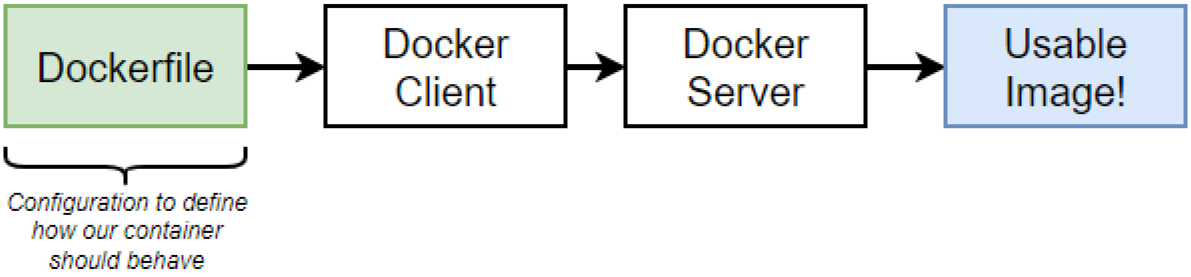
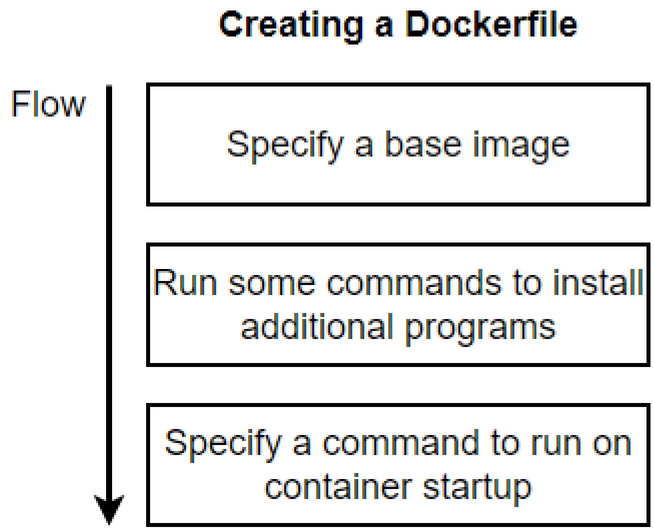
3. Building Custom image

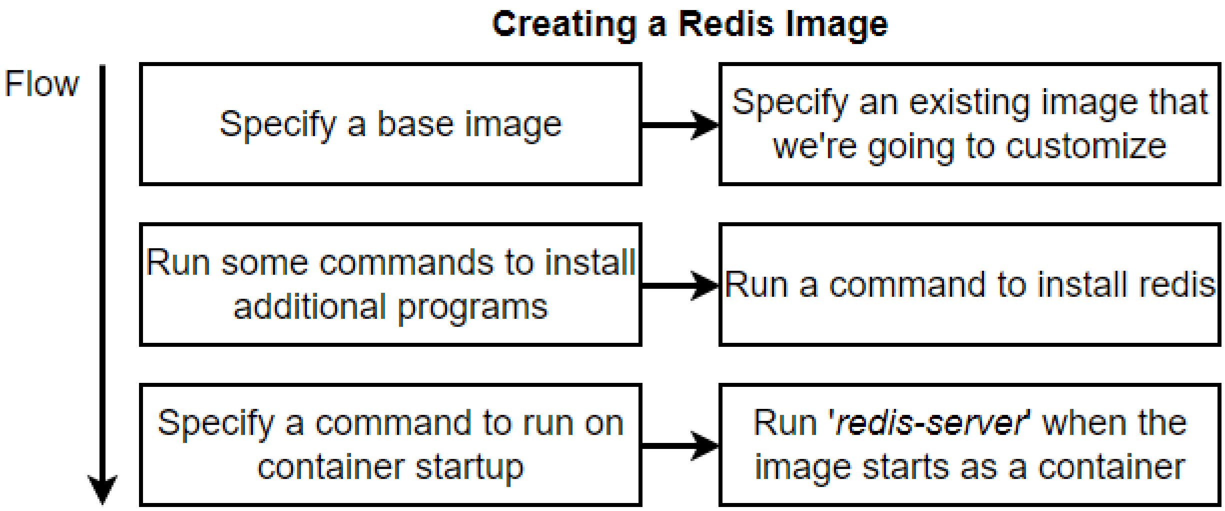
3.1 Creating docker Images



- General steps to create images:


Creating a Dockerfile
Flow
Specify a base image
Run some commands to install additional programs
Specify a command to run on container startup

Example: creating redis image


Creating a Redis Image
Flow
Specify a base image
Specify an existing image that we're going to customize
Run some commands to install additional programs
Run a command to install redis
Specify a command to run on container startup
Run 'redis-server' when the image starts as a container

DockerFile:

```
#Step 1: Use an existing docker image as a base

FROM alpine

# Step 2: Download and install dependency

RUN apk add --update redis

# Step 3: Tell the image what to do when it starts as container

CMD ["redis-server"]
```

3.2 Building Dockerfile

$docker build .

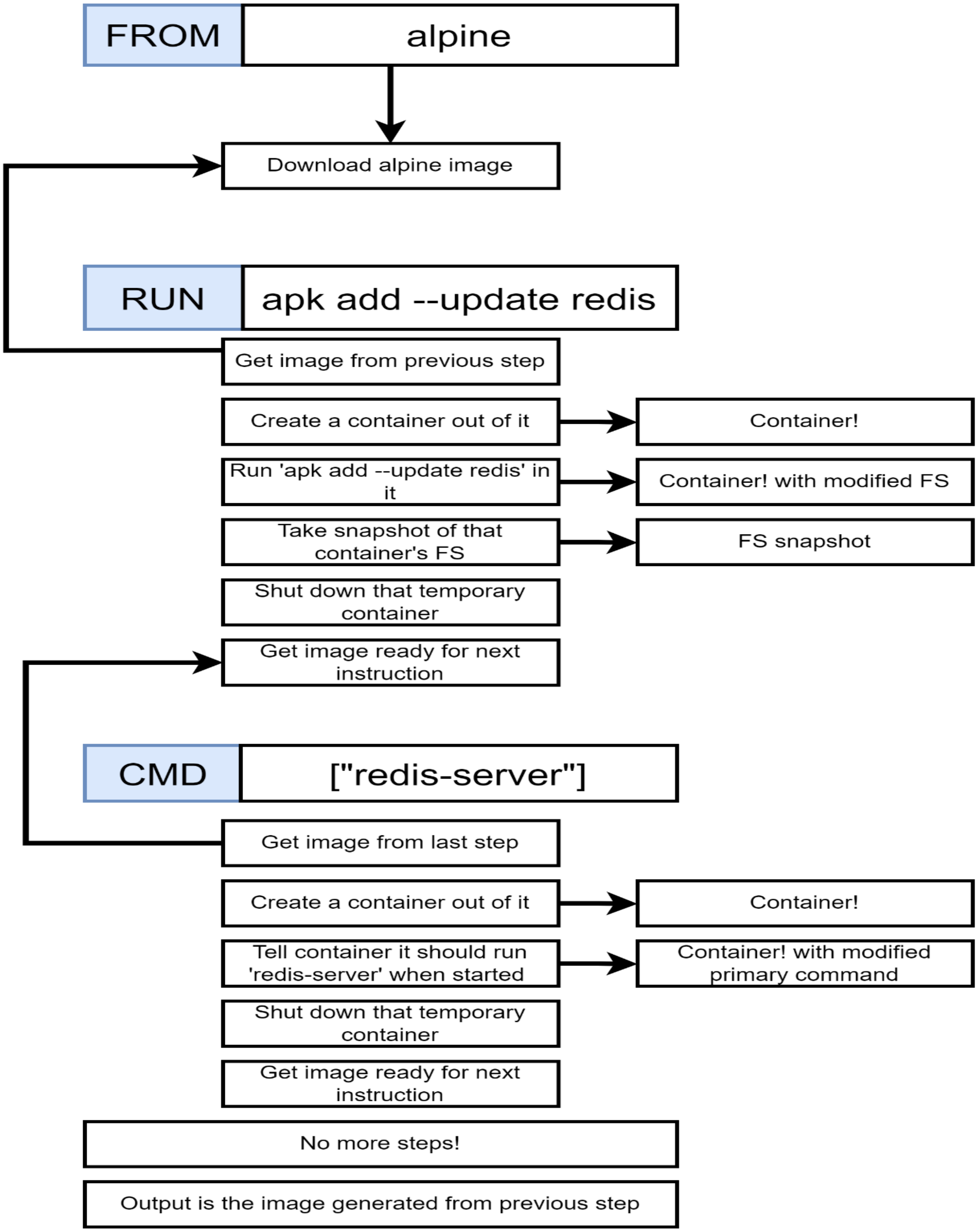
For example steps in building a redis image

```
$docker build .
Deploying '<unknown> Dockerfile: Dockerfile'...
Building image...
Preparing build context archive...
[=================================================>]9/9 files
Done
Sending build context to Docker daemon...
[=================================================>] 1.729kB
Done

Step 1/3 : FROM alpine
latest: Pulling from library/alpine
213ec9aee27d: Already exists
Digest: sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
Status: Downloaded newer image for alpine:latest
 ---> 9c6f07244728
Step 2/3 : RUN apk add --update redis
 ---> Running in c43dc497a40f
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/x86_64/APKINDEX.tar.gz
(1/1) Installing redis (7.0.4-r1)
Executing redis-7.0.4-r1.pre-install
Executing redis-7.0.4-r1.post-install
Executing busybox-1.35.0-r17.trigger
```

```
OK: 9 MiB in 15 packages
Removing intermediate container c43dc497a40f
 ---> 644163bf2308
Step 3/3 : CMD ["redis-server"]
 ---> Running in e6c6f79cd6fe
Removing intermediate container e6c6f79cd6fe
 ---> e5ce217762cd

Successfully built e5ce217762cd
'<unknown> Dockerfile: Dockerfile' has been deployed successfully.
```

```
$docker run e5ce217762cd
1:C 04 Sep 2022 16:07:23.061 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 04 Sep 2022 16:07:23.061 # Redis version=7.0.4, bits=64, commit=4bd17450, modified=0,
pid=1, just started
1:C 04 Sep 2022 16:07:23.061 # Warning: no config file specified, using the default config. In
order to specify a config file use redis-server /path/to/redis.conf
1:M 04 Sep 2022 16:07:23.063 * monotonic clock: POSIX clock_gettime
1:M 04 Sep 2022 16:07:23.065 * Running mode=standalone, port=6379.
1:M 04 Sep 2022 16:07:23.065 # Server initialized
1:M 04 Sep 2022 16:07:23.065 # WARNING overcommit_memory is set to 0! Background save
may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to
/etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for
this to take effect.
1:M 04 Sep 2022 16:07:23.067 * Ready to accept connections
```

3.2.1 Detailed brief out of how the instruction in docker file executes:


FROM
alpine
Download alpine image
RUN
apk add --update redis
Get image from previous step
Create a container out of it
Container!
Run 'apk add --update redis' in it
Container! with modified FS
Take snapshot of that container's FS
FS snapshot
Shut down that temporary container
Get image ready for next instruction
CMD
["redis-server"]
Get image from last step
Create a container out of it
Container!
Tell container it should run 'redis-server' when started
Container! with modified primary command
Shut down that temporary container
Get image ready for next instruction
No more steps!
Output is the image generated from previous step

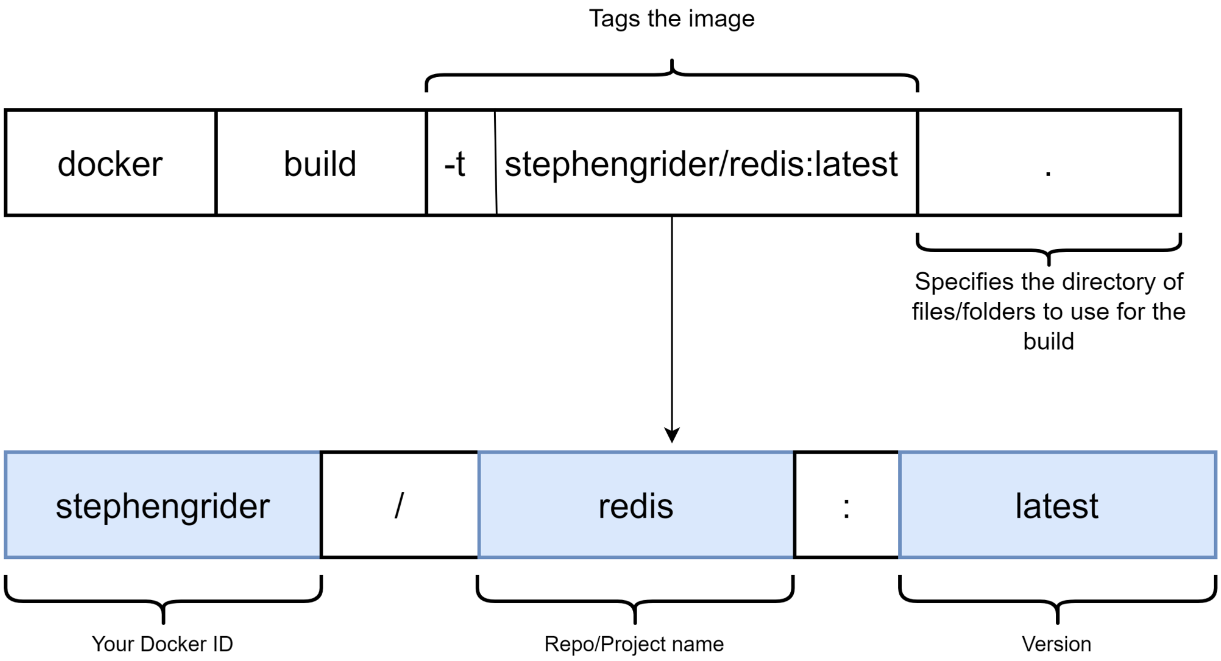
3.3 Rebuild using cache

- The concept of Docker images comes with immutable layers.
- Every command you execute results in a new layer that contains the changes compared to the previous layer.
- All previously built layers are cached and can be reused.
- But if the lines in docker are altered then no cached versions are used.

  For more about caching refer : https://www.baeldung.com/linux/docker-build-cache

3.4 Tagging an image

3.4.1 While building image:



3.4.2 Tag an image referenced by ID

- To tag a local image with ID “0e5574283393” into the “redd” repository with “version2.0”:

```
$ docker tag 0e5574283393 redd/httpd:version2.0
```

3.4.3 Tag an image referenced by Name

- To tag a local image with name “httpd” into the “redd” repository with “version2.0”:

```
$ docker tag httpd redd/httpd:version2.0
```

- Note that since the tag name is not specified, the alias is created for an existing local version httpd:latest.

3.4.4 Tag an image referenced by Name and Tag

- To tag a local image with name “httpd” and tag “test” into the “fedora” repository with “version1.0.test”:

```
$ docker tag httpd:test fedora/httpd:version1.0.test
```

3.4.5 Tag an image for a private repository

- To push an image to a private registry and not the central Docker registry you must tag it with the registry hostname and port (if needed).

```
$ docker tag 0e5574283393 myregistryhost:5000/fedora/httpd:version1.0
```

Refer : https://docs.docker.com/engine/reference/commandline/tag/#usage