Docker

1. Creating Docker Images

- Dockerfile will be passed to Docker CLI
- Which is fed to Docker server and creates image
- Flow of creating Dockerfile:
  - Specify base image
  - Run commands to install additional programs
  - Specify command to run on container startup

2. Building a Docker file

- **docker build .**

This command creates the image by running the Dockerfile in the provided directory.

E.g. Creating an image that runs redis server:

```
# Use an existing docker image as a base
FROM alpine
# Download and install a dependency
RUN apk add --update redis
# Tell the image what to do when it starts
# as a container.
CMD ["redis-server"]
```

```
harshs-MacBook-Pro:redis-image harshnasit$ docker build .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM alpine
 ---> f70734b6a266
Step 2/3 : RUN apk add --update redis
 ---> Running in e07388939372
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/community/x86_64/APKINDEX.tar.g
z
(1/1) Installing redis (5.0.7-r0)
Executing redis-5.0.7-r0.pre-install
Executing redis-5.0.7-r0.post-install
Executing busybox-1.31.1-r9.trigger
OK: 7 MiB in 15 packages
Removing intermediate container e07388939372
 ---> 791a09c54a8a
Step 3/3 : CMD ["redis-server"]
 ---> Running in b37bd4248933
Removing intermediate container b37bd4248933
 ---> e7d4b0195f1e
Successfully built e7d4b0195f1e
```

Since I already have alpine downloaded, it directly accesses the image from the local cache and can be used for creating containers from the alpine image.
Copy the id and execute **docker run <image id>**

```
harshs-MacBook-Pro:redis-image harshnasit$ docker run e7d4b0195f1e
1:C 16 Jun 2020 00:17:49.021 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 16 Jun 2020 00:17:49.021 # Redis version=5.0.7, bits=64, commit=bed89672, mo
dified=0, pid=1, just started
1:C 16 Jun 2020 00:17:49.021 # Warning: no config file specified, using the defa
ult config. In order to specify a config file use redis-server /path/to/redis.co
nf
1:M 16 Jun 2020 00:17:49.023 * Running mode=standalone, port=6379.
1:M 16 Jun 2020 00:17:49.023 # WARNING: The TCP backlog setting of 511 cannot be
 enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:M 16 Jun 2020 00:17:49.023 # Server initialized
1:M 16 Jun 2020 00:17:49.023 # WARNING you have Transparent Huge Pages (THP) sup
port enabled in your kernel. This will create latency and memory usage issues wi
th Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transpa
rent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to ret
ain the setting after a reboot. Redis must be restarted after THP is disabled.
1:M 16 Jun 2020 00:17:49.023 * Ready to accept connections
```

3. Docker file teardown

Every step represents an instruction for the docker server. The instruction **FROM** is used to specify a base image which contains pre-installed commands/setups. The **RUN** instruction is used to execute some command to setup the pre-requisites. The **CMD** command is used to specify the primary command for the

image to be built(or the command that runs when containers are created from this image).

4. Whats a Base Image

When we create an image, there is no infrastructure and the image is empty. Base Image is used to have an initial setup and further configure dependencies. We used Alpine as a base image because it comes with default set of programs such as apk that can be used to install and run redis.

5. Build process in detail

The build command is what we use to take a docker file and generate an image out of it. The . is specifying the build context. It is a set of files and folders encapsulated in this container. If the base image is not in the local cache, it downloads it from the docker hub. The flow of image creation is step by step, meaning the temporary image created in 1 step is used in the next step.

E.g. Explanation for the redis-server docker file
When the **RUN** step is run, it gets the image from the previous step and creates a new container out of it. Then the apk command is run inside this container modifying the file system by creating a new folder called redis. After that, it takes the file system snapshot and creates a new temporary image by removing the running container.

Next, the **CMD** command takes the previously created temporary image and creates a container out of it. **CMD** sets the primary command for the container which is then saved as a permanent image in the cache. The temporary container gets removed. Now the new image can be used to create and run containers.

6. Rebuilds with Cache

If the build command is executed again, it uses the cached temporary images to run the similar steps. E.g. If I add another RUN command to install gcc after redis, until step 2 it will use the cached images. Note: if the order of steps is changed, the image will not be built from the cached images, instead it will create new temporary images. This way, using cache helps Docker get speed and performance. To sum up, if a Docker file needs to be changed, always make changes as far down as possible.

7. Tagging an image

After building a custom image, it can be tedious to create containers by using the image id. E.g. docker run 94095c54fd6d. Instead images can be tagged with names to make creating of containers readable. Syntax:

docker build -t <docker_id>/<name of project>:<version> .

Version is traditionally going to be latest or a number.
E.g. **docker build -t harshn12/redis:latest .** Now containers can be created by **docker run harshn12/redis** and by default it runs the latest version. So, the image is being tagged by the version number provided.

8. Manual image Generation, Docker Commit

Generally, containers are created from images, but images can be created from containers manually as well. Syntax:

docker commit -c 'CMD ["redis-server"]' <container_id>

This command creates an image out of the provided container id. But this approach is generally avoided.