Assignment 03

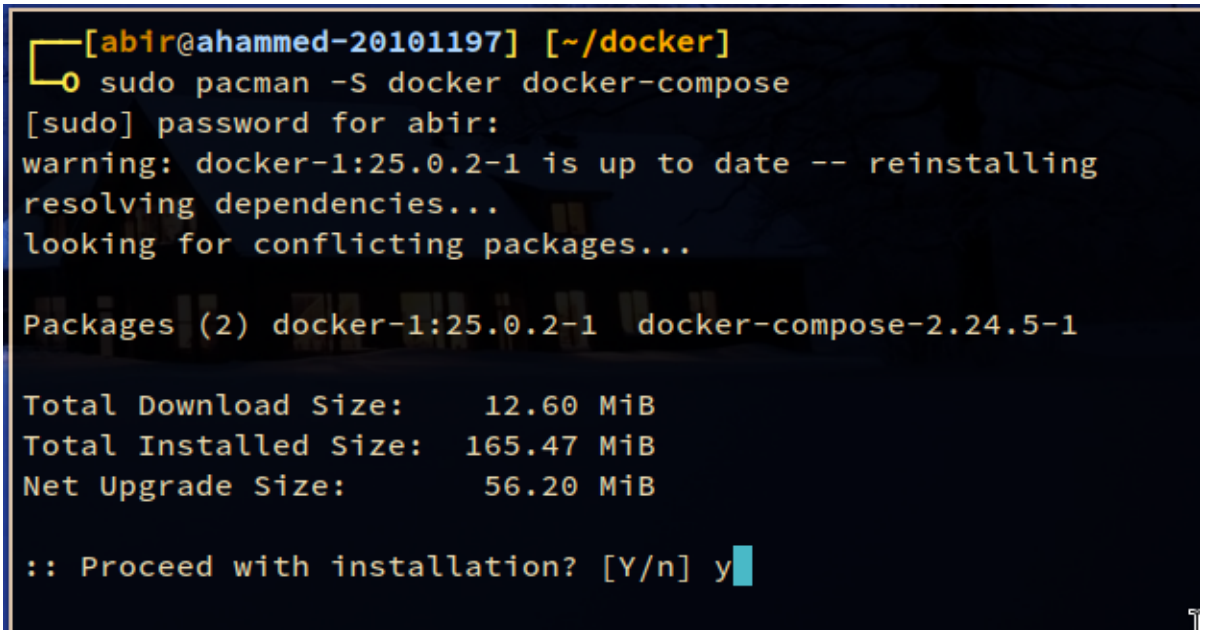
Abir Ahammed Bhuiyan
ID: 20101197
Dept. of Computer
Science and Engineering

Section : 01
Subject : CSE484
Semester : Spring-2024
Date : 22.02.2024

1. Install docker

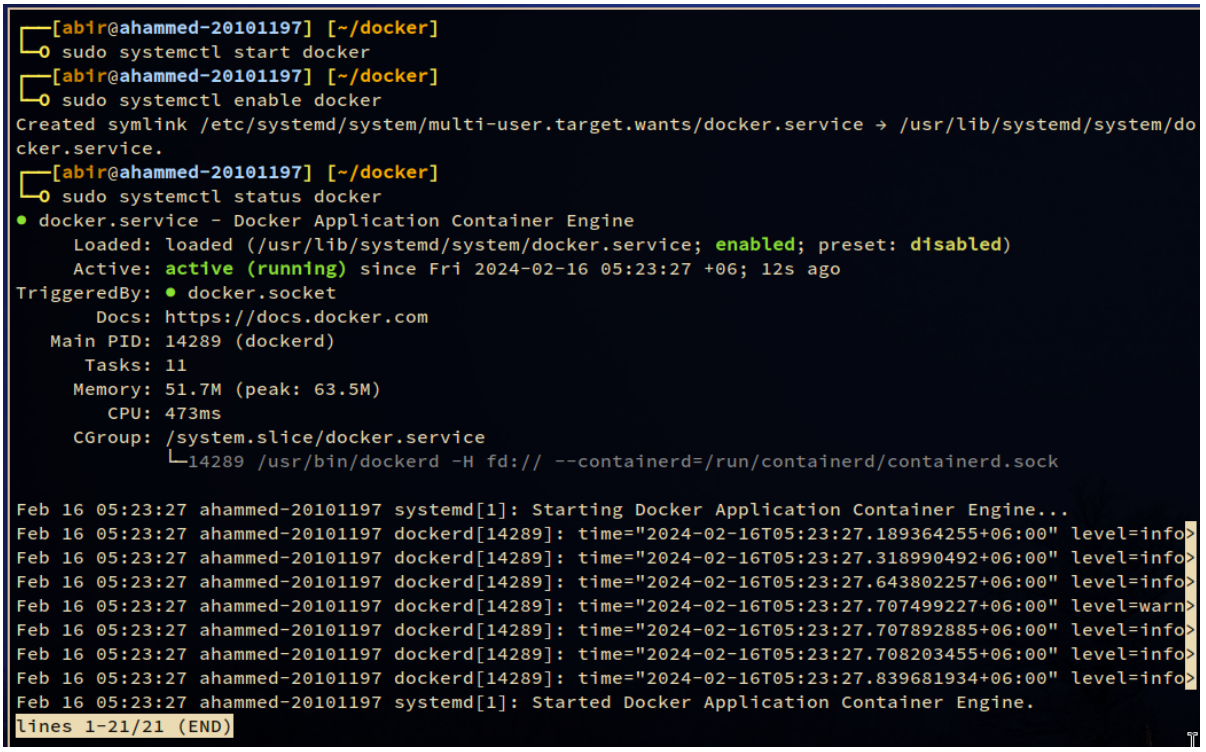
We will install docker in `Arch Linux` so we will be using `pacman` to install docker.

```
sudo pacman -S docker docker-compose
```



Install has been completed, let's `start` and `enable` docker service and check the status if the service is running.

```
sudo systemctl start docker
sudo systemctl enable docker
sudo systemctl status docker
```



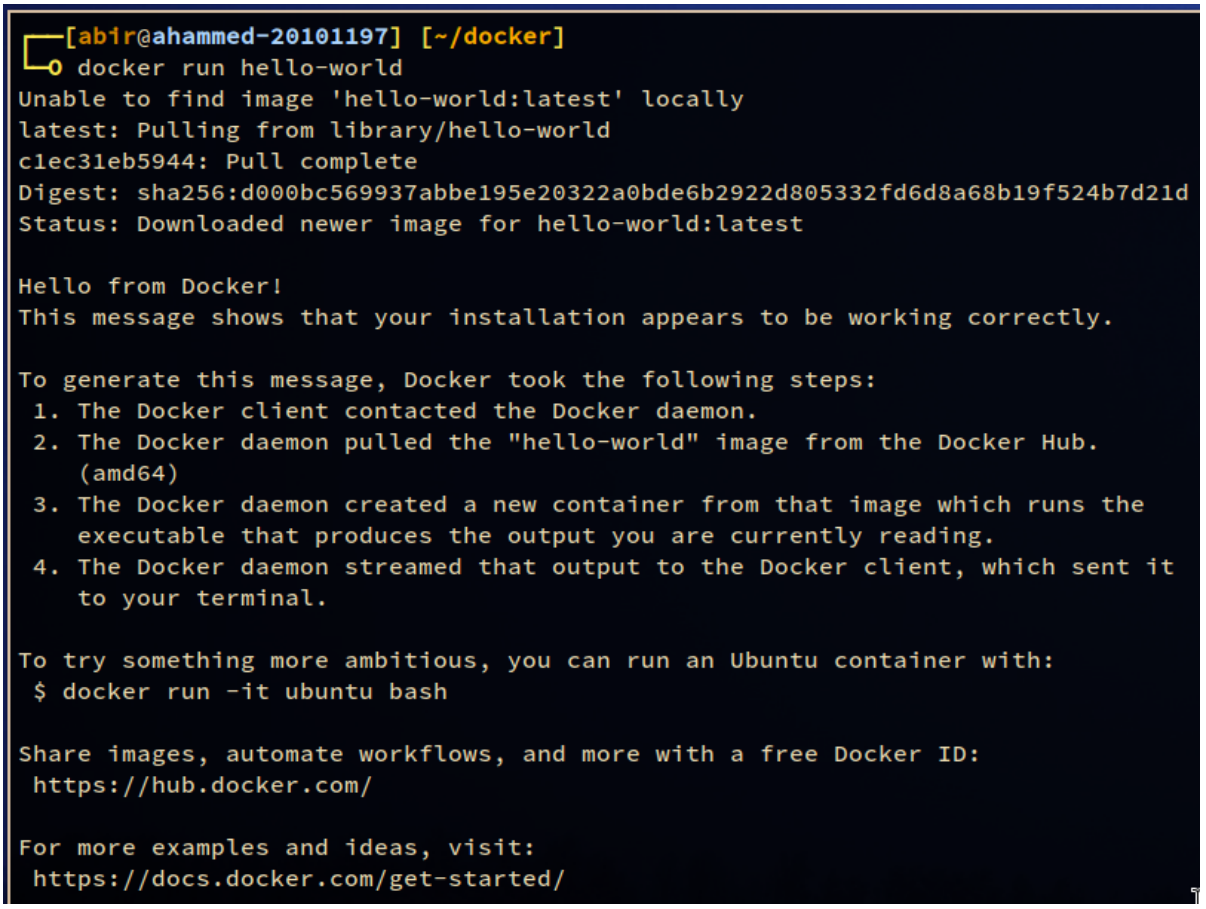
To avoid using `sudo` with each docker command, we can add our account (in this case `abir`) to the docker group like this:

```
sudo usermod -aG docker abir
```

After changing user group setting we must `logout` for the above change to take effect.

Now, verfiy if the docker is working or not by using the below command.

```
docker run hello-world
```



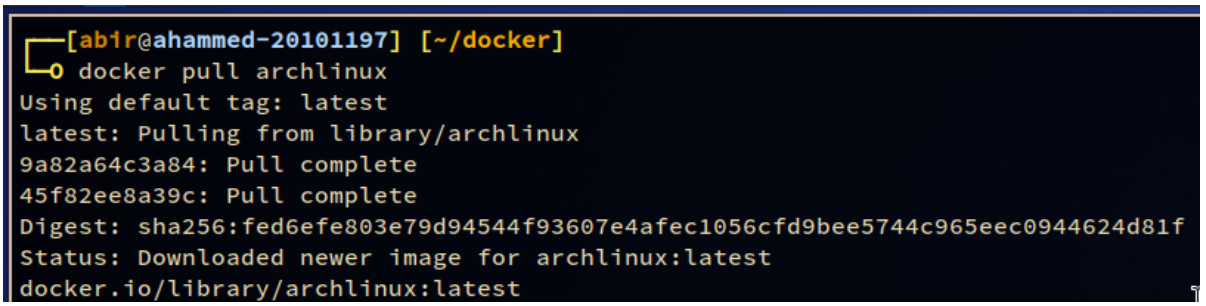
Docker has been successfully installed and working in our machine.

2. Show outputs of basic Docker commands (i.e pull, search, run, build, commit, rm, rmi, etc.. find more from Google)

pull

This command use to pull docker images from docker hub via CLI.

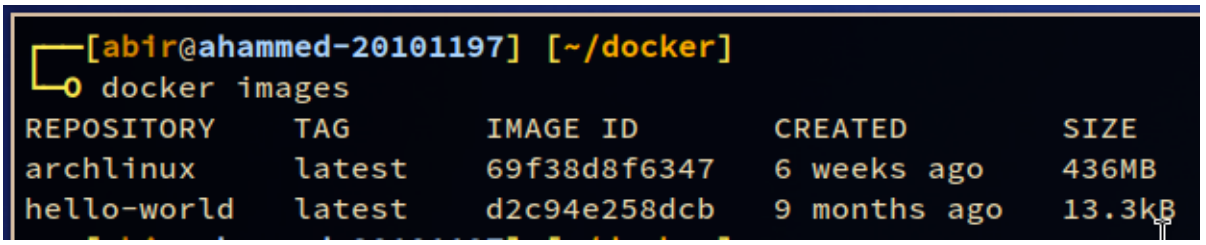
```
docker pull archlinux
```



images

By using `images` command we can see what images have been stored in our local machine.

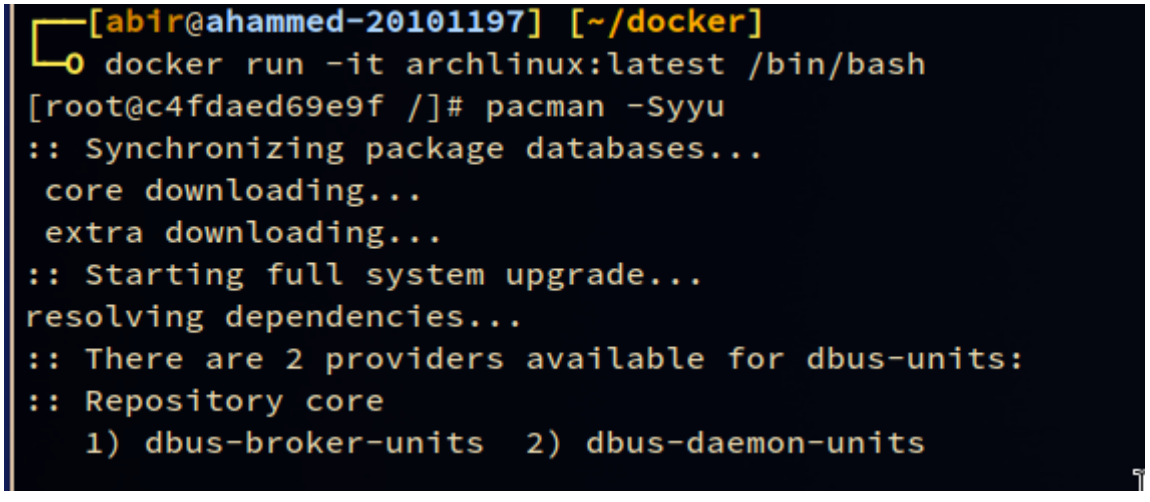
```
docker images
```



run

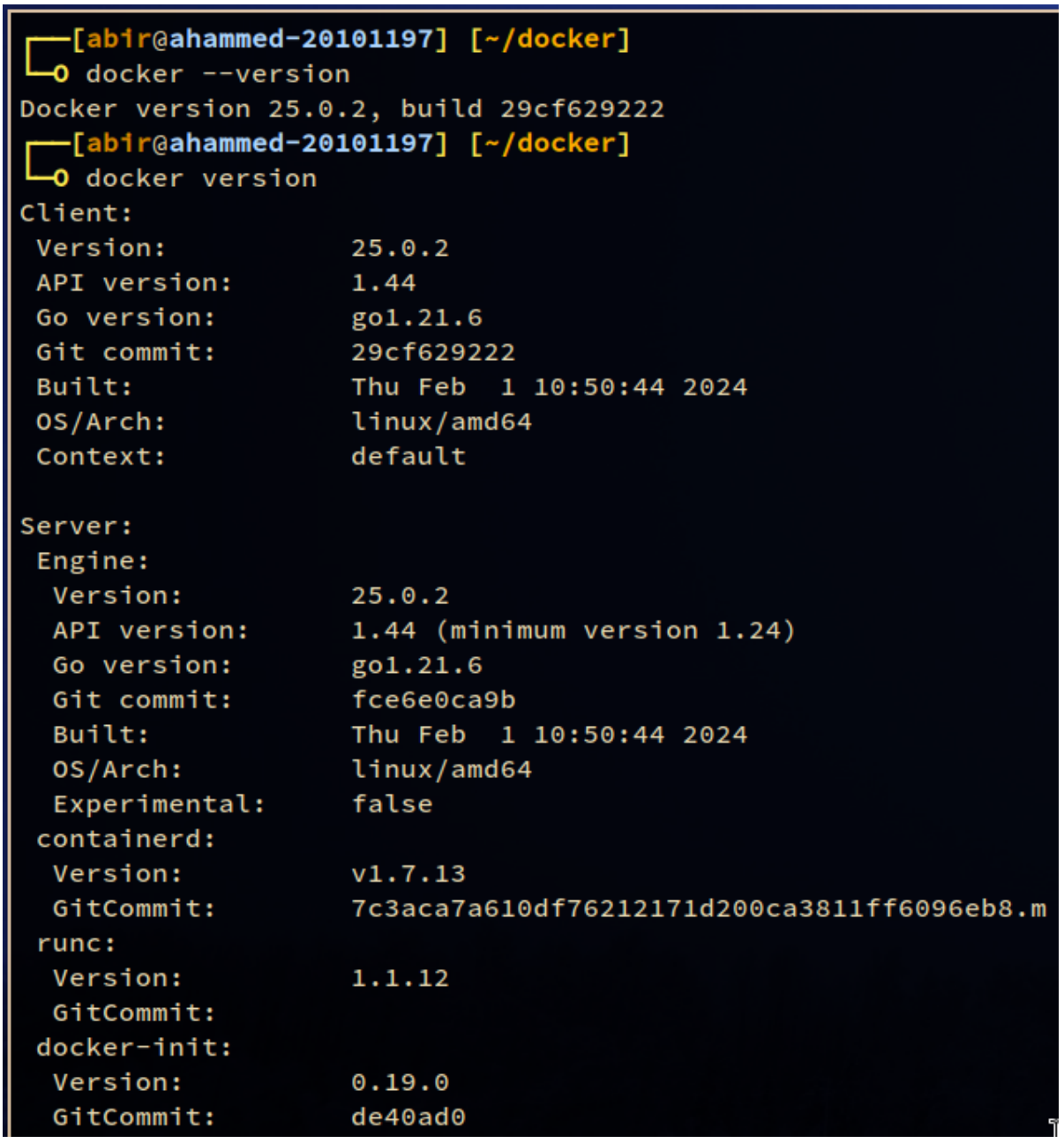
To run a docker image we can use `run` command. We can specify a flag `-it`, which just stands for `interactive terminal`.

```
docker run -it <image_name>
```



version

Using `--version` or `version` command we can see all the version information of docker.



rmi

Using `rmi` command we can remove an image. In the picture, we have tried to remove an image name `myarch`.

```
[abir@ahammed-20101197] [~]
o docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
image-serve       latest    d8b5452dfa33   33 hours ago    42.6MB
eniac00/myarch    latest    4991cc415eaf   41 hours ago    436MB
myarch            latest    4991cc415eaf   41 hours ago    436MB
nginx             alpine    6913ed9ec8d0   7 days ago      42.6MB
mongo             latest    b8df2163f9aa   2 weeks ago     755MB
archlinux         latest    69f38d8f6347   7 weeks ago     436MB
hello-world       latest    d2c94e258dcb   9 months ago    13.3kB
[abir@ahammed-20101197] [~]
o docker rmi myarch:latest
Untagged: myarch:latest
[abir@ahammed-20101197] [~]
o docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
image-serve       latest    d8b5452dfa33   33 hours ago    42.6MB
eniac00/myarch    latest    4991cc415eaf   41 hours ago    436MB
nginx             alpine    6913ed9ec8d0   7 days ago      42.6MB
mongo             latest    b8df2163f9aa   2 weeks ago     755MB
archlinux         latest    69f38d8f6347   7 weeks ago     436MB
hello-world       latest    d2c94e258dcb   9 months ago    13.3kB
[abir@ahammed-20101197] [~]
o
```

ps

Using `ps -a` we can view process status of all the containers.

```
[abir@ahammed-20101197] [~]
o docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                     PORTS     NAMES
6c0db7707365   469ae7bf5468   "/docker-entrypoint.…"   21 hours ago    Exited (0) 21 hours ago              ahnaf-web
27bdf39819f6   image-serve    "/docker-entrypoint.…"   33 hours ago    Exited (0) 32 hours ago              myserver
[abir@ahammed-20101197] [~]
o
```

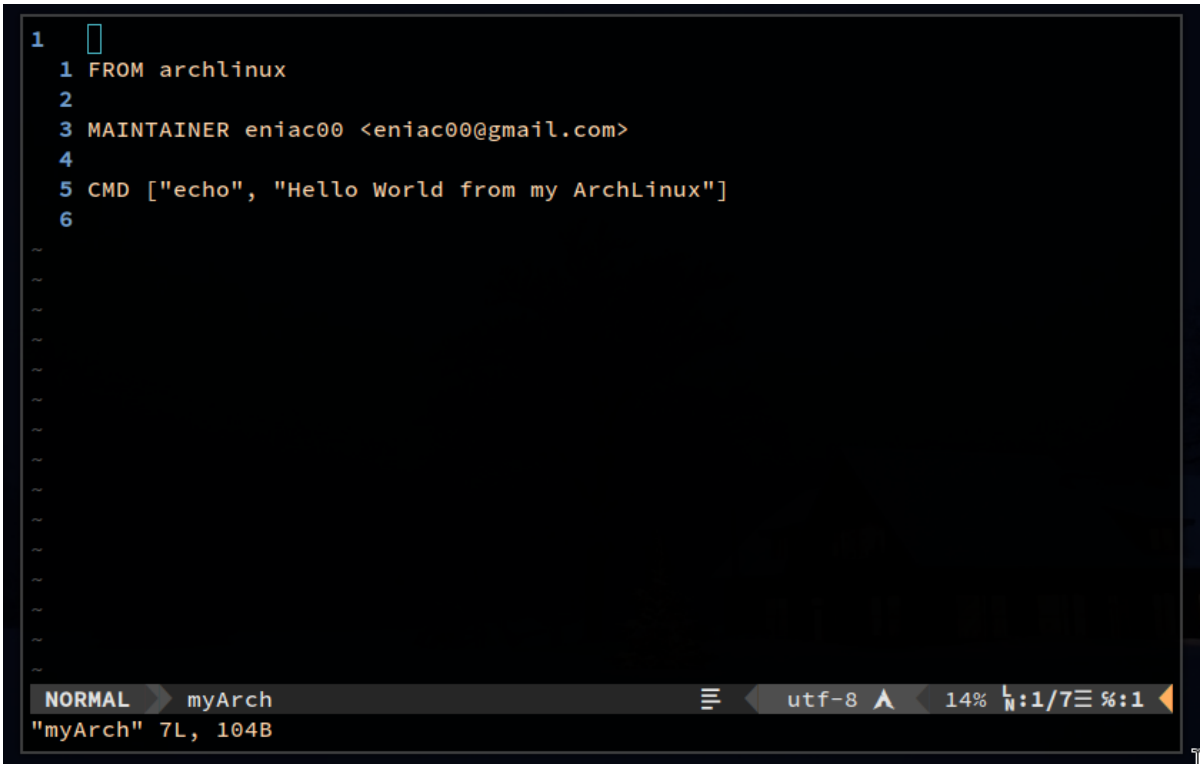
3. Create a Docker image using Dockerfile.

We are going to create our own version of `Arch Linux` that will print `Hello World from my ArchLinux`.

First we have to create a directory and open a new file with any name in our case we have created a directory called `Docker` and inside that directory we have created a file named `myArch`.

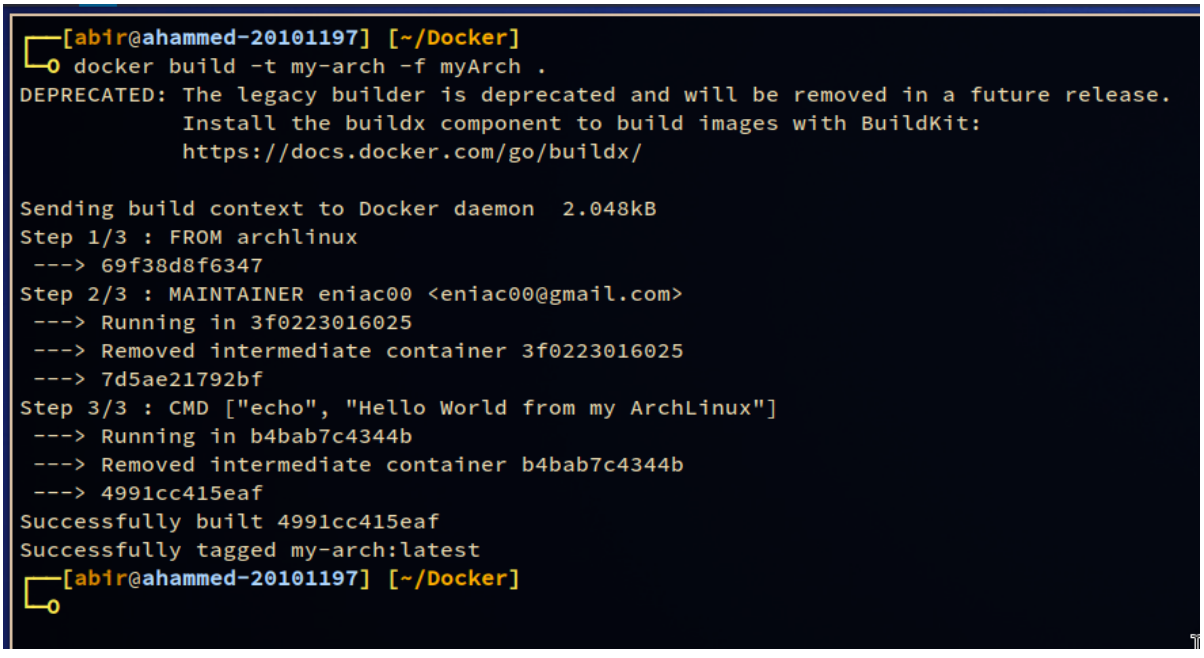
```
[abir@ahammed-20101197] [~]
o mkdir Docker
[abir@ahammed-20101197] [~]
o cd Docker/
[abir@ahammed-20101197] [~/Docker]
o touch myArch
[abir@ahammed-20101197] [~/Docker]
o
```

Then write the following in the file and save it.



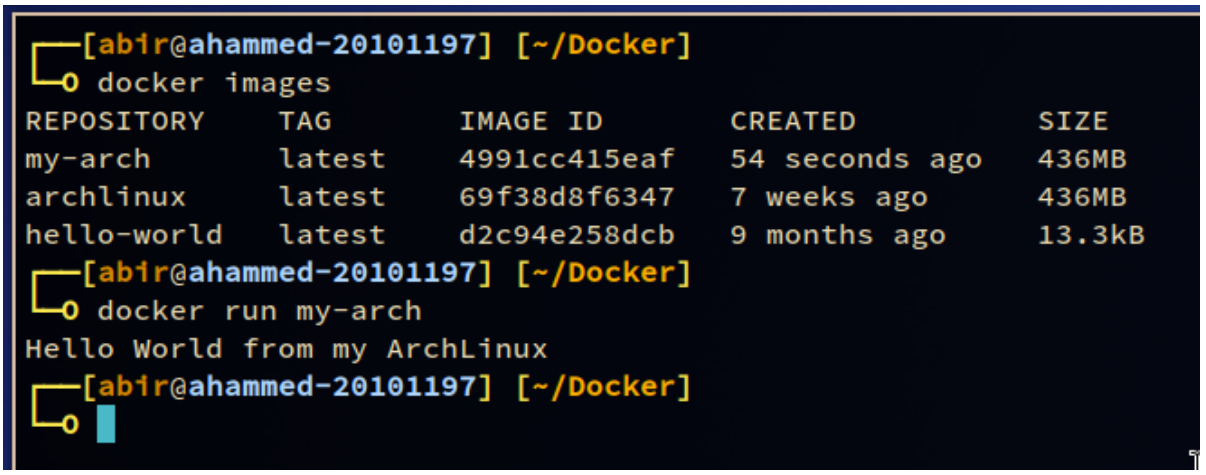
After that we have to build the the docker image from the docker file. To do that simply run the below command.

```
docker build -t my-arch -f myArch .
```



Our docker image is ready to go in our case its name is `my-arch`. Let's run it.

```
docker run my-arch
```



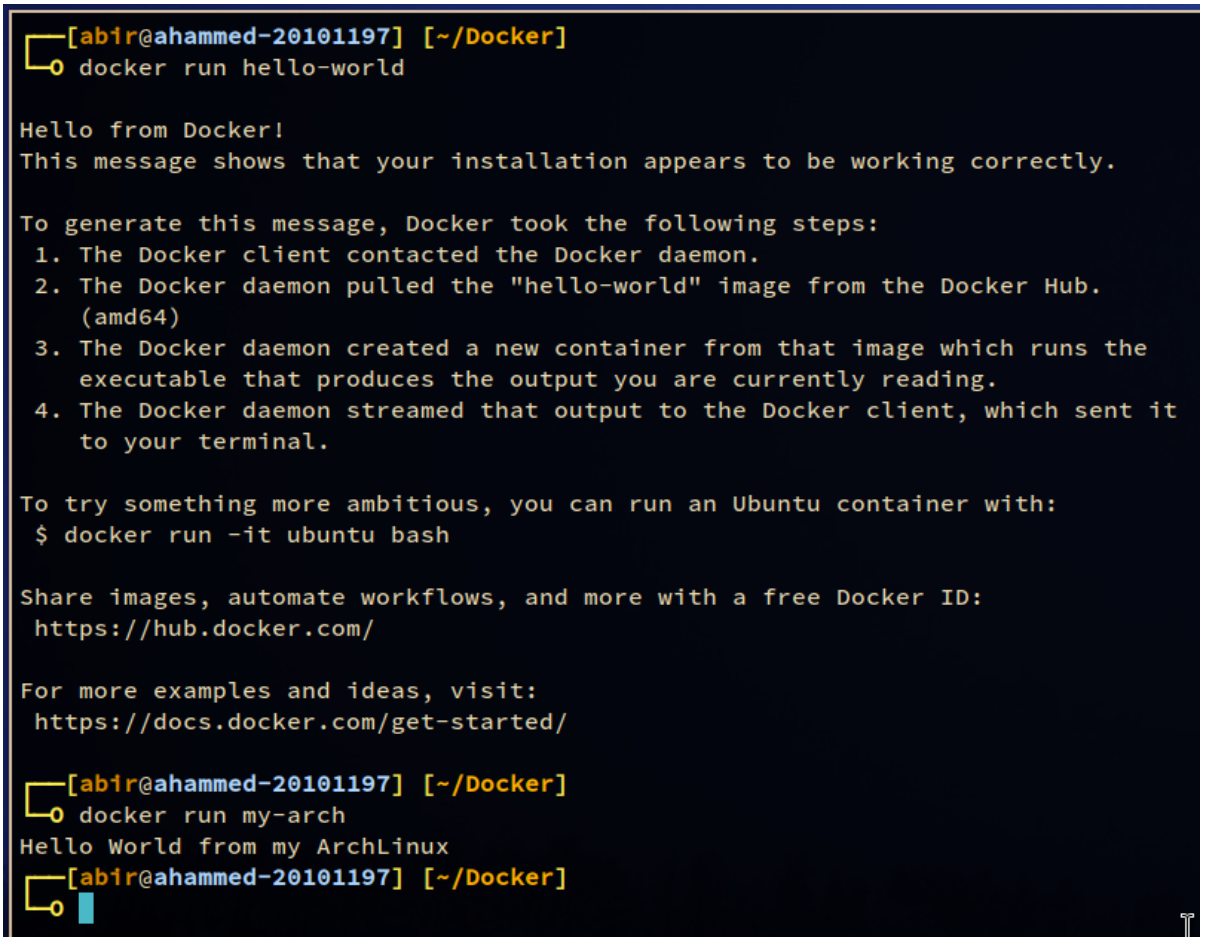
After running it we can see our desired output like the picture above.

4. Run a container as a single task, show outputs, and show the status of all containers (using docker ps -a)

To run a docker image (later container) as a single task we can use the `run` command like below.

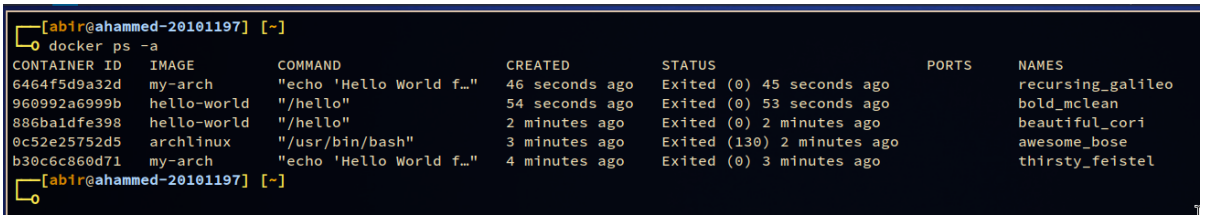
```
docker run hello-world
```

Here `hello-world` is the image name.



To see the all the containers status run a command like below.

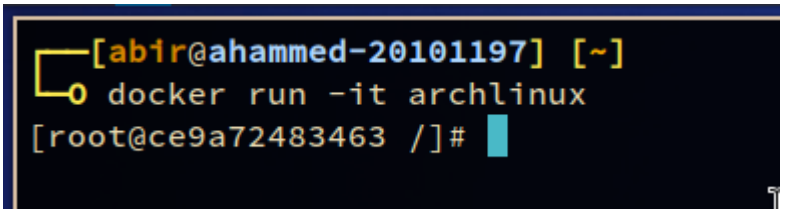
```
docker ps -a
```



5. Run a container in iterative mode and install different packages in the container. Show each step.

Using the below command we can run a docker image in iterative mode (here `archlinux` is the image name).

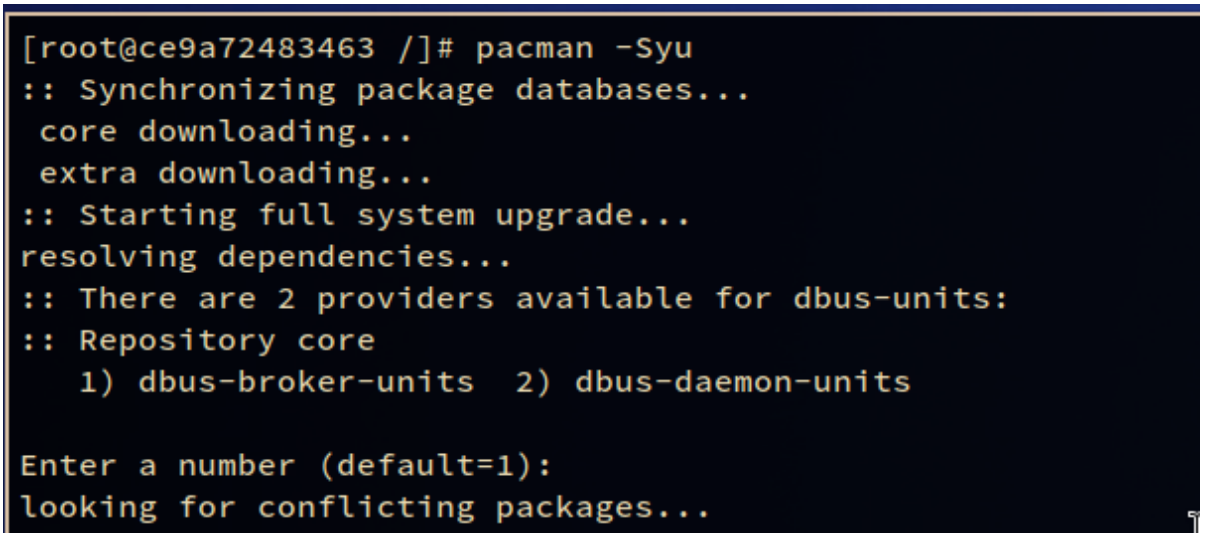
```
docker run -it archlinux
```



After running we can see that our prompt has been changed, that indicates we are currently in the shell of our newly created container. we can install different packages but first we have to update the package manager in our case that is `pacman`.

```
pacman -Syu
```

```
pacman -S neofetch
pacman -S vim
```



We have tried to install two packages `neofetch` and `vim`. After installing we have tried to run them and they ran successfully.

```
[root@ce9a72483463 /]# pacman -S neofetch
resolving dependencies...
looking for conflicting packages...

Package (1)     New Version  Net Change  Download Size

extra/neofetch  7.1.0-2        0.33 MiB       0.08 MiB

Total Download Size:   0.08 MiB
Total Installed Size:  0.33 MiB

:: Proceed with installation? [Y/n] y
:: Retrieving packages...
 neofetch-7.1.0-2-any downloading...
checking keyring...
checking package integrity...
loading package files...
checking for file conflicts...
:: Processing package changes...
installing neofetch...
Optional dependencies for neofetch
    catimg: Display Images
    chafa: Image to text support
    feh: Wallpaper Display
    imagemagick: Image cropping / Thumbnail creation / Take a screenshot
    jp2a: Display Images
    libcaca: Display Images
    nitrogen: Wallpaper Display
    w3m: Display Images
    xdotool: See https://github.com/dylanaraps/neofetch/wiki/Images-in-the-terminal
    xorg-xdpyinfo: Resolution detection (Single Monitor)
    xorg-xprop: Desktop Environment and Window Manager
    xorg-xrandr: Resolution detection (Multi Monitor + Refresh rates)
    xorg-xwininfo: See https://github.com/dylanaraps/neofetch/wiki/Images-in-the-terminal
:: Running post-transaction hooks...
(1/1) Arming ConditionNeedsUpdate...
```

```
[root@ce9a72483463 /]# neofetch
                    -`                    root@ce9a72483463
                   .o+`                   -----------------
                  `ooo/                   OS: Arch Linux x86_64
                 `+oooo:                  Host: B365M GAMING HD
                `+oooooo:                 Kernel: 6.7.5-arch1-1
                -+oooooo+:                Uptime: 1 hour, 52 mins
              `/:-:++oooo+:               Packages: 118 (pacman)
             `/++++/+++++++:              Shell: bash 5.2.26
            `/++++++++++++++:             Resolution: 1920x1080
           `/+++ooooooooooooo/`           CPU: Intel i5-9400F (6) @ 4.100GHz
          ./ooosssso++osssssso+`          GPU: NVIDIA GeForce GT 1030
         .oossssso-````/ossssss+`         Memory: 3166MiB / 7889MiB
        -osssssso.      :ssssssso.
       :osssssss/        osssso+++.
      /ossssssss/        +ssssooo/-
    `/ossssso+/:-        -:/+osssso+-
   `+sso+:-`                 `.-/+oso:
  `++:.                           `-/+/
  .`                                 `/

[root@ce9a72483463 /]#
```

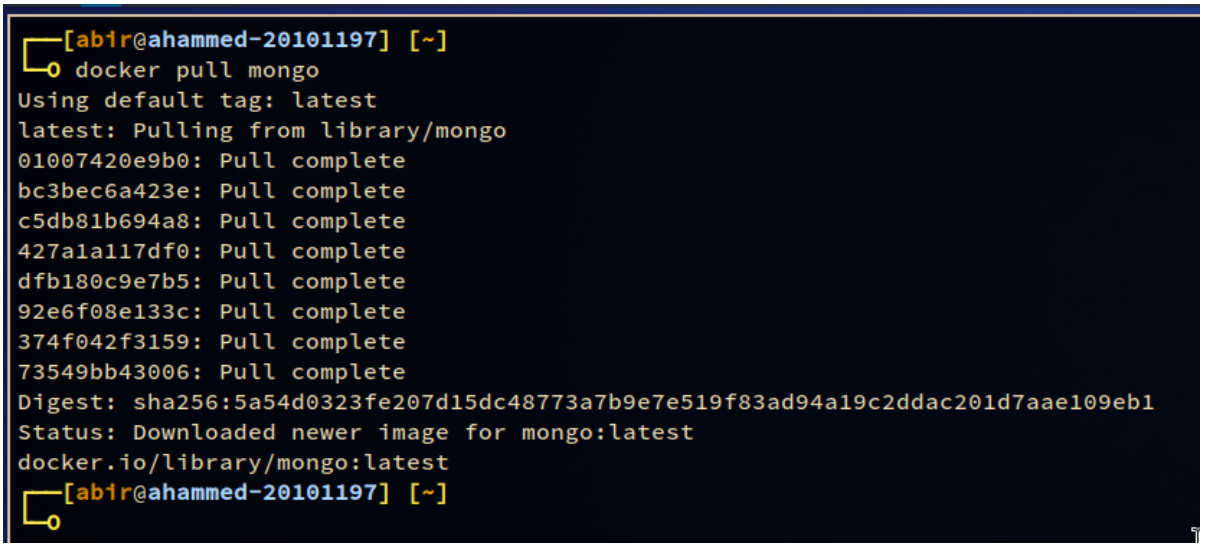
```
[root@ce9a72483463 /]# pacman -S vim
resolving dependencies...
looking for conflicting packages...

Package (3)        New Version            Net Change  Download Size

core/gpm           1.20.7.r38.ge82d1a6-5    0.40 MiB       0.14 MiB
extra/vim-runtime  9.1.0080-1              36.44 MiB       7.01 MiB
extra/vim          9.1.0080-1               4.91 MiB       2.18 MiB

Total Download Size:    9.33 MiB
Total Installed Size:  41.75 MiB

:: Proceed with installation? [Y/n] y
:: Retrieving packages...
 vim-runtime-9.1.0080-1-x86_64 downloading...
 vim-9.1.0080-1-x86_64 downloading...
 gpm-1.20.7.r38.ge82d1a6-5-x86_64 downloading...
checking keyring...
checking package integrity...
loading package files...
checking for file conflicts...
:: Processing package changes...
installing vim-runtime...
Optional dependencies for vim-runtime
    sh: support for some tools and macros [installed]
    python: demoserver example tool
    gawk: mve tools upport [installed]
installing gpm...
installing vim...
Optional dependencies for vim
    python: Python language support
    ruby: Ruby language support
    lua: Lua language support
    perl: Perl language support
    tcl: Tcl language support
:: Running post-transaction hooks...
(1/2) Reloading system manager configuration...
  Skipped: Current root is not booted.
(2/2) Arming ConditionNeedsUpdate...
[root@ce9a72483463 /]#
```

```
[root@ce9a72483463 /]# vim --version
VIM - Vi IMproved 9.1 (2024 Jan 02, compiled Feb 06 2024 15:30:35)
Included patches: 1-80
Compiled by Arch Linux
Huge version without GUI.  Features included (+) or not (-):
+acl               +file_in_path      +mouse_urxvt       -tag_any_white
+arabic            +find_in_path      +mouse_xterm       +tcl/dyn
+autocmd           +float             +multi_byte        +termguicolors
+autochdir         +folding           +multi_lang        +terminal
-autoservername    -footer            -mzscheme          +terminfo
-balloon_eval      +fork()            +netbeans_intg     +termresponse
+balloon_eval_term +gettext           +num64             +textobjects
```

6. Run a database container in the background. Then show logs of the running container. After, access the container in interactive mode. show some SQL queries inside the container.

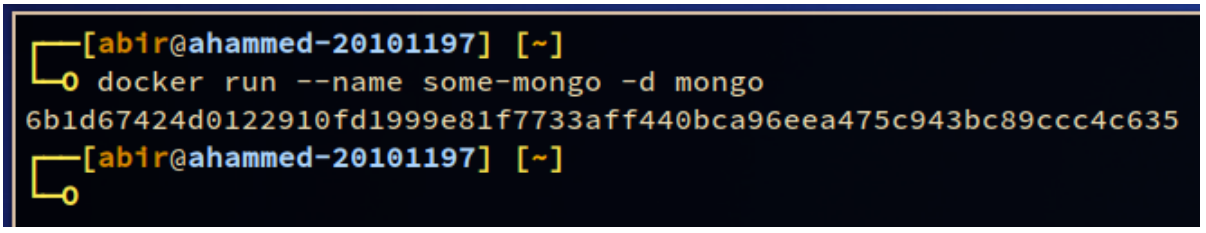
For this we will be using a popular `NoSQL` database known as `MongoDB`. To pull the docker image from the remote to our local machine we can try this command,

```
docker pull mongo
```



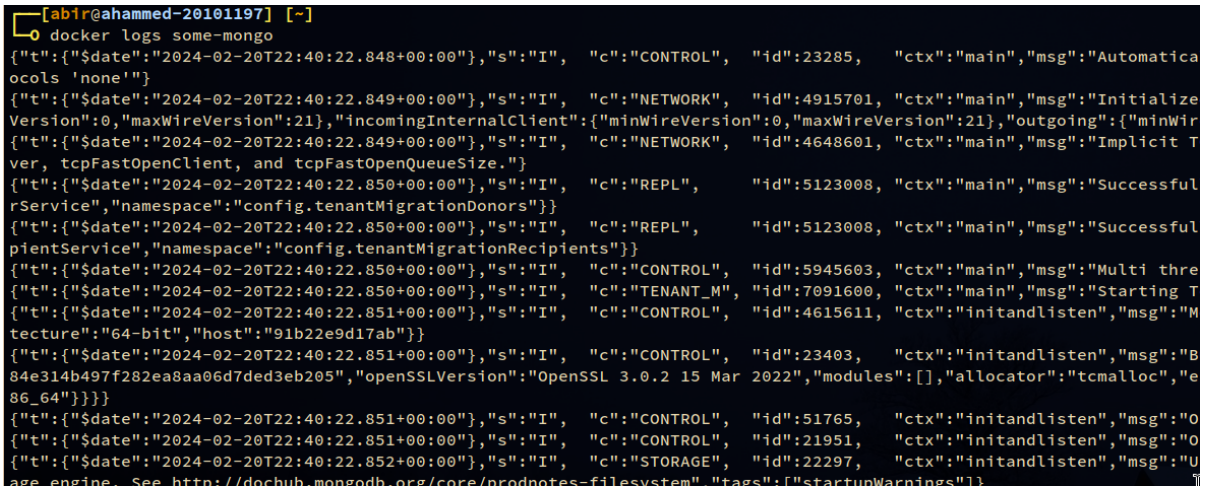
After that we can run the container in the background.

```
docker run --name some-mongo -d mongo
```



To see the logs this command can be use,

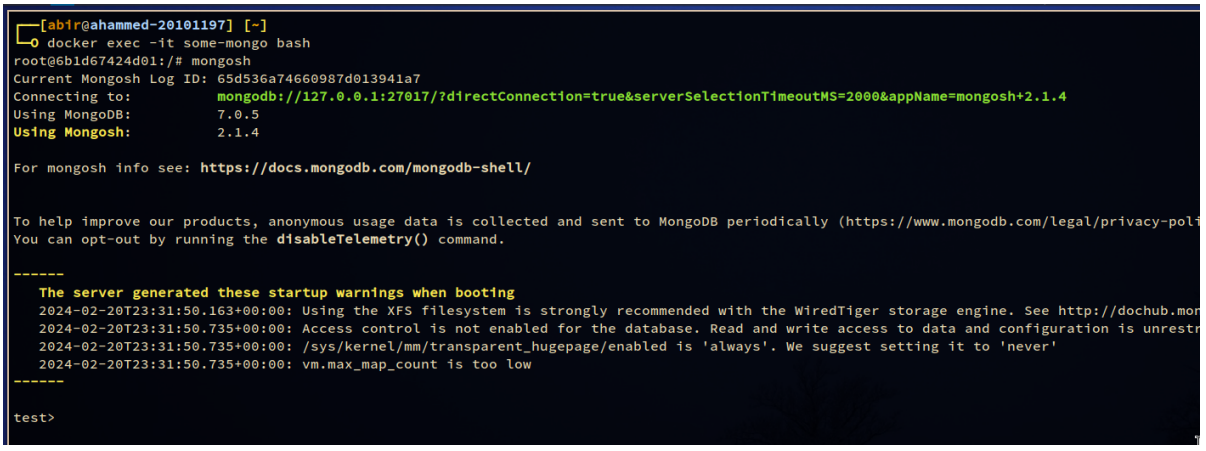
```
docker logs some-mongo
```



To access the running database container in interective mode we can use `exec` command.

```
docker exec -it some-mongo bash
```

Also, after accessing the `bash` shell inside the container we can use `mongosh` command to run the `MongoDB` shell .



Some queries inside the container

To see the version information of the current `MongoDB` use `db.version()` inside `mongosh` shell. Moreover, we can see all the databases list with `show dbs` and to create a new database we can use the `use <database_name>` command.

```
db.version()
show dbs
use docker_test
```

To create a new collection and insert some data inside the database we can use `insert` command.

```
db.products.insert( { item: "card", qty: 15 } )
```

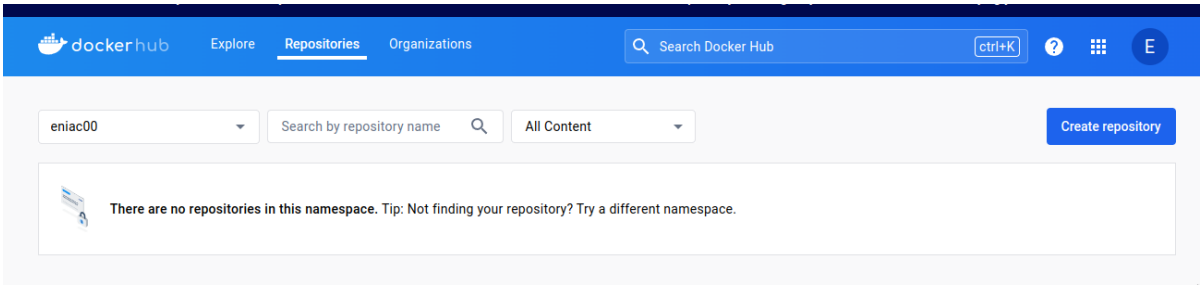
Above command will create a collection named `products` and insert data `{ item: "card", qty: 15 }`.

Moreover, we can use `find()` to list all the data present in a collection.

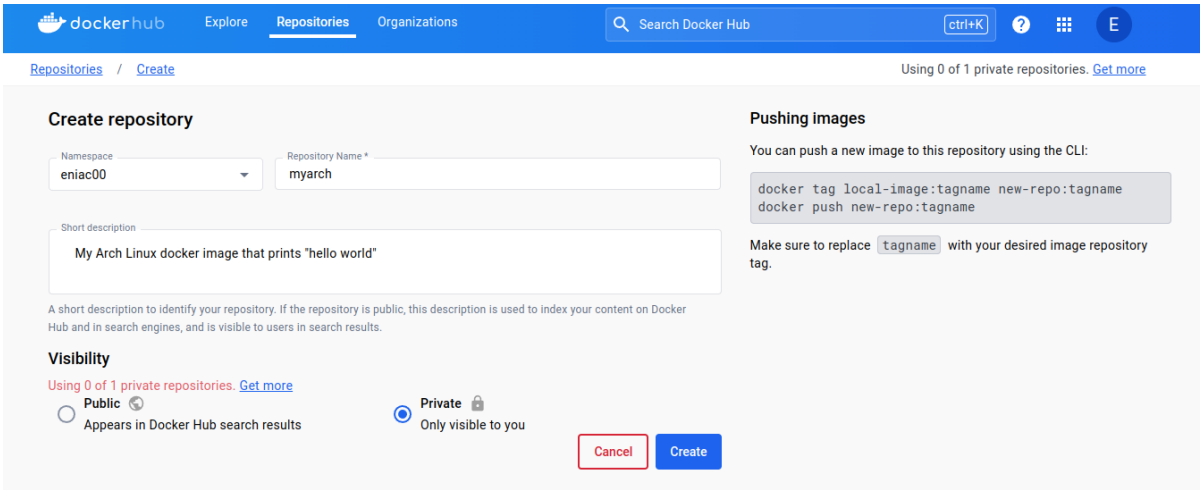
```
db.products.find().pretty()
```

8. How to make your own private registry? Show steps.

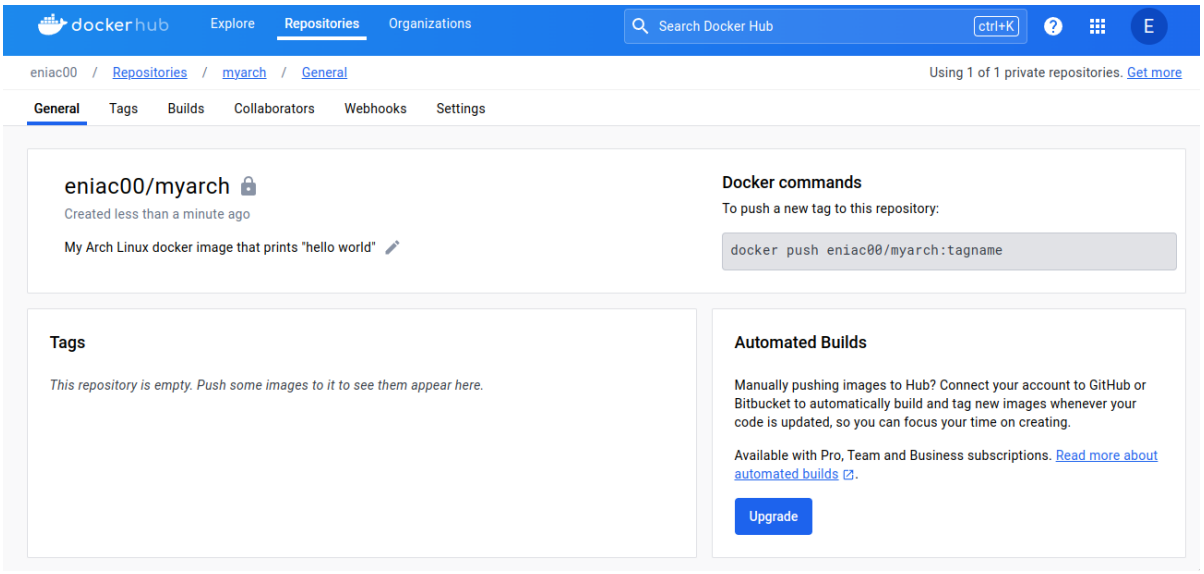
We can create a new account in `Docker Hub` after login into our account we can select `Repositories` tab.



From there we can tap on the `Create repository` button to create a new repository. Here, we are creating a new repository `myarch`. Then click on `Create`.



This is how our new repository name `myarch` has been created. Note that, we have created a `Private` repository.

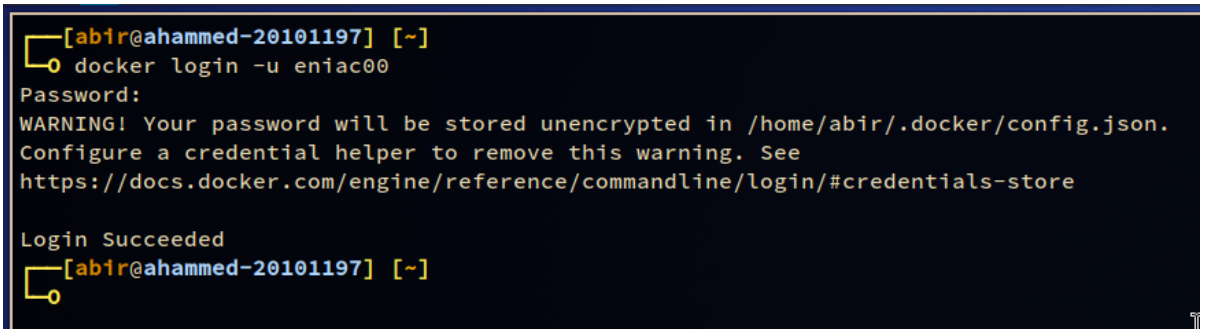


7. Push your own image into the Docker public registry/Hub.

For pushing our own image to our public/private repository we can use,

```
docker login -u <username>
```

Use password or token to authenticate your login.



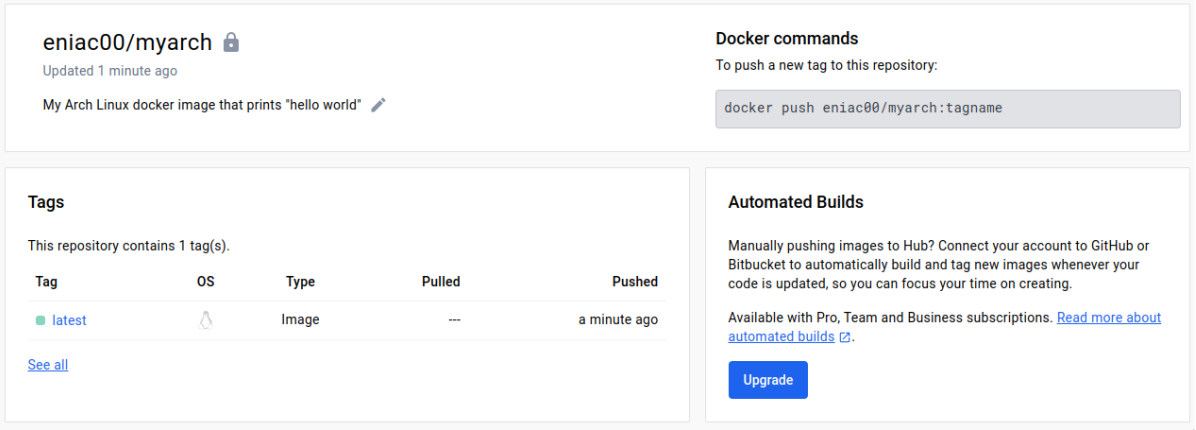
We will push our newly created docker image `myarch`.

```
docker tag myarch:latest eniac00/myarch:latest
```

```
docker push eniac00/myarch:latest
```

```
[abir@ahammed-20101197] [~]
o docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
myarch        latest    4991cc415eaf   2 hours ago    436MB
mongo         latest    b8df2163f9aa   13 days ago    755MB
archlinux     latest    69f38d8f6347   7 weeks ago    436MB
hello-world   latest    d2c94e258dcb   9 months ago   13.3kB
[abir@ahammed-20101197] [~]
o docker tag myarch:latest eniac00/myarch:latest
[abir@ahammed-20101197] [~]
o docker push eniac00/myarch:latest
The push refers to repository [docker.io/eniac00/myarch]
d822bba4f032: Pushed
6de0fb710365: Pushed
latest: digest: sha256:9c049bad6e06f49c8d73274f294190260af7c3016550d8db35f30003c00219e1 size: 738
[abir@ahammed-20101197] [~]
o
```

In the browser we can see that our image has been successfully pushed to `Docker Hub`.



9. Create a small website or app with minimal functionality (Could be a simple HTML website that has a button that opens a static image/file) inside a Docker container. Then run the application (inside the container) in the background of your HOST machine in any port. Browse the website from your host machine.

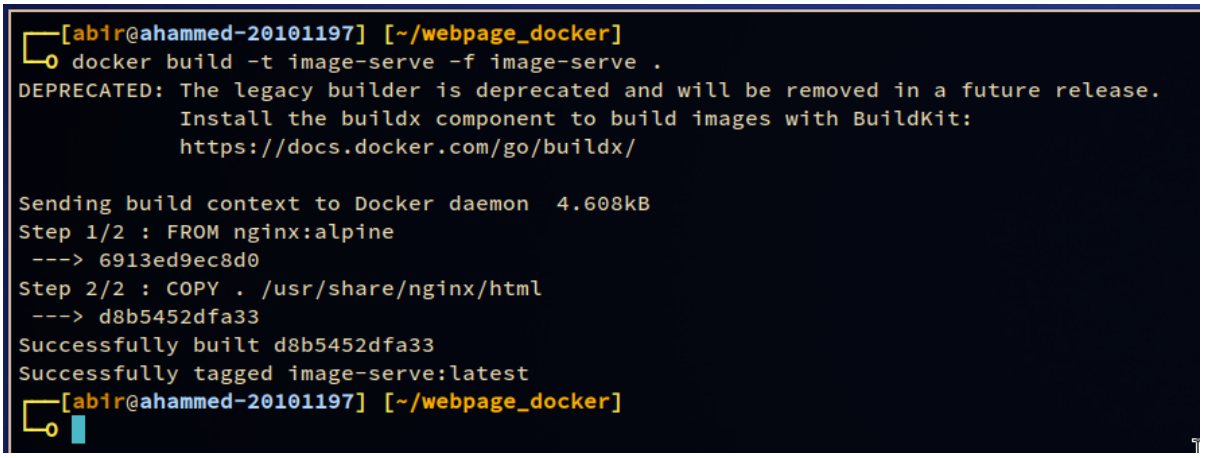
First, create a new directory and create a webpage along with a docker file there.

```
[abir@ahammed-20101197] [~/webpage_docker]
o ll
drwxr-xr-x  abir abir 4.0 KB Wed Feb 21 12:01:24 2024 .
drwx--x---+ abir abir 4.0 KB Wed Feb 21 13:28:09 2024 ..
.rw-r--r--  abir abir  48 B  Wed Feb 21 12:01:17 2024 image-serve
.rw-r--r--  abir abir 1.8 KB Wed Feb 21 12:00:16 2024 index.html
[abir@ahammed-20101197] [~/webpage_docker]
o
```

Insert necessary codes in the `*.html` file and update docker file (i.e. `image-serve`) like below image and save.

Next build the image from the docker file.

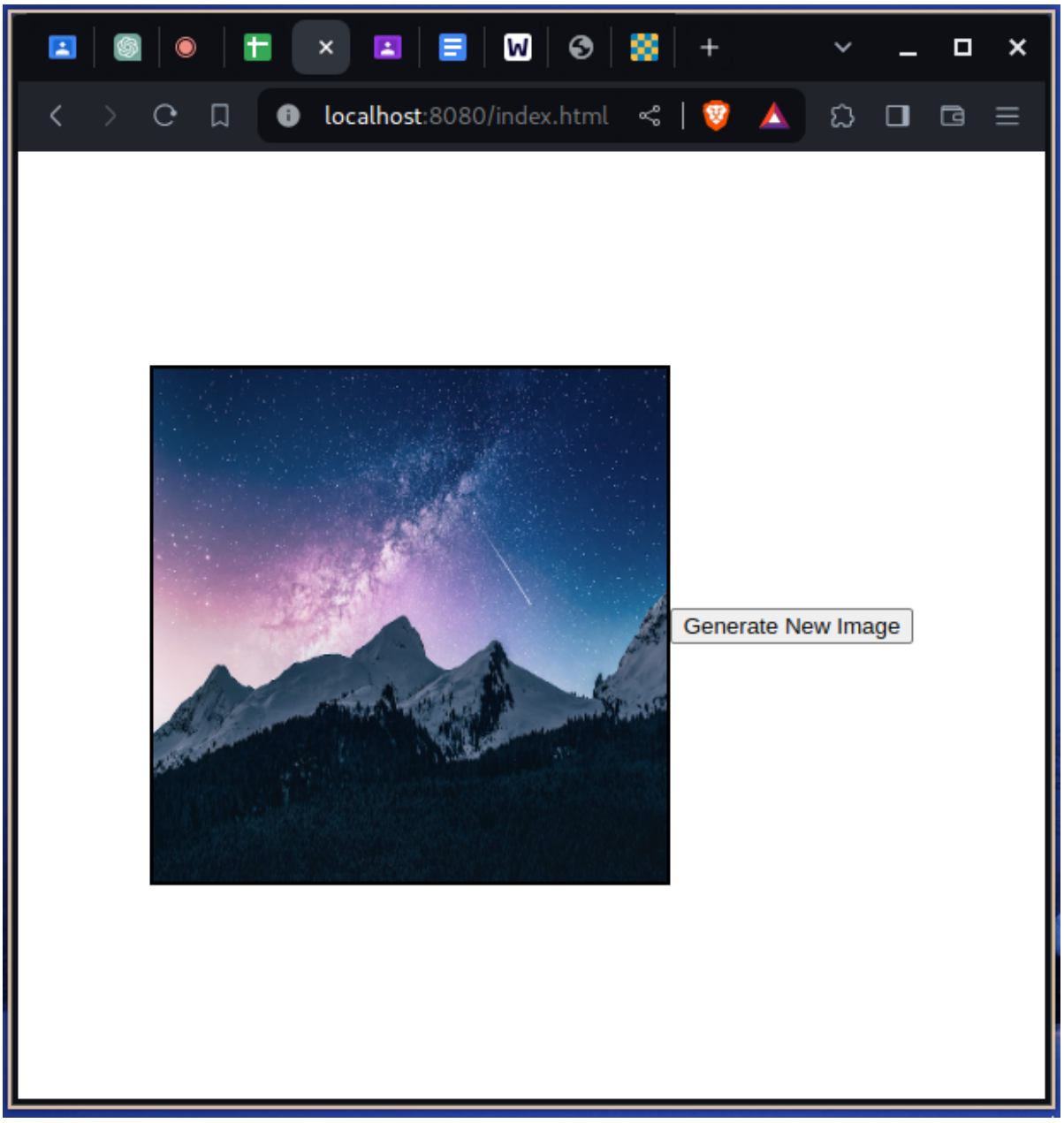
```
docker build -t image-serve -f image-serve .
```



Now, run the docker container in the background and map containers `port 80` to host machine's `port 8080` with the below command.

```
docker run -d -p 8080:80 --name myserver image-serve
```

We can visit the `localhost:8080/index.html` in any browser and our containerized webpage will be visible to us like the below image.



10. Migrate the new container having the application into another machine. Again run the container and browse the URL. It should work.

In my machine

To migrate a docker image we first we need to archive our image in a `*.tar` file. In this case we will be migrating our image `image-serve`, so we need to archive it.

```
docker save image-serve:latest > image_serve.tar
```

After that we can share that archived image in any machine. We will be using `scp` command to securly share the archived image over the secure shell (`ssh`).

```
scp image_serve.tar abir@192.168.0.111:~/docker_test/
```

In another machine

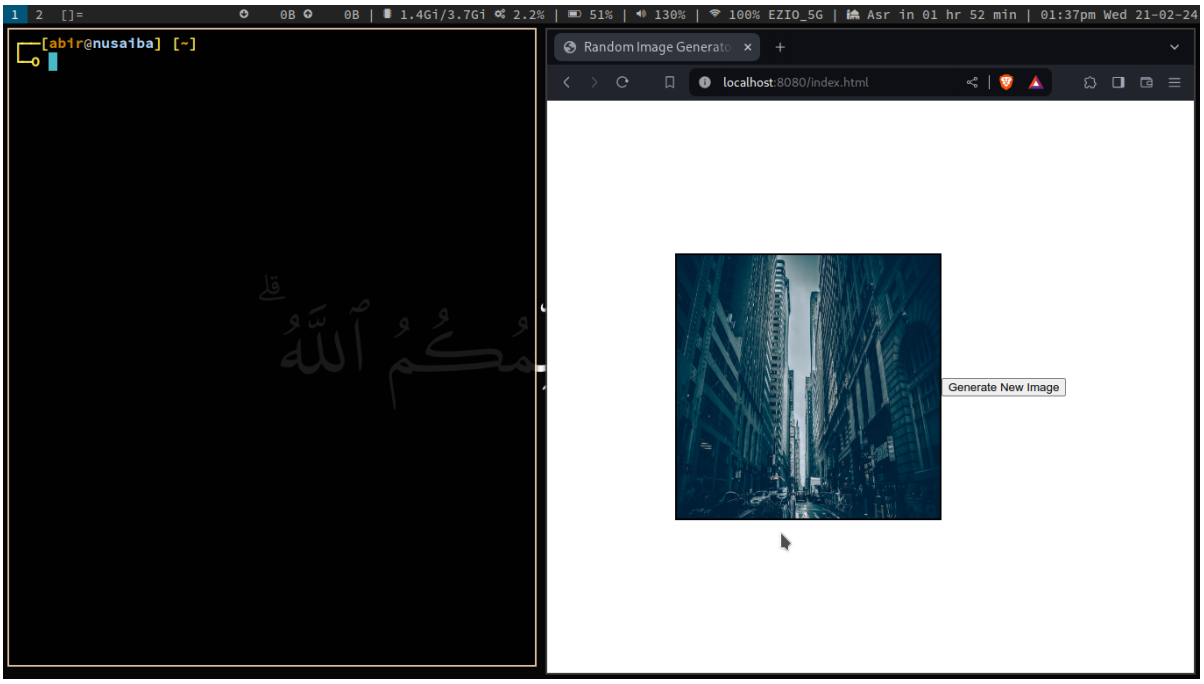
In another machine we can see that archived image. Now we just have to load the archived image using `load` command make sure that `docker` is installed in that machine.

```
docker load < image_serve.tar
```

We can run the newly created image (i.e. `image-serve`) using the above command we have seen in No. 9.

```
docker run -d -p 8080:80 --name myserver image-serve
```

Visit `localhost:8080/index.html` in any browser and we can see our webpage.


1 2 []= 0B 0B | 1.4Gi/3.7Gi 2.2% | 51% | 130% | 100% EZIO_5G | Asr in 01 hr 52 min | 01:37pm Wed 21-02-24
[abir@nusaiba] [~]
Random Image Generato
localhost:8080/index.html
Generate New Image