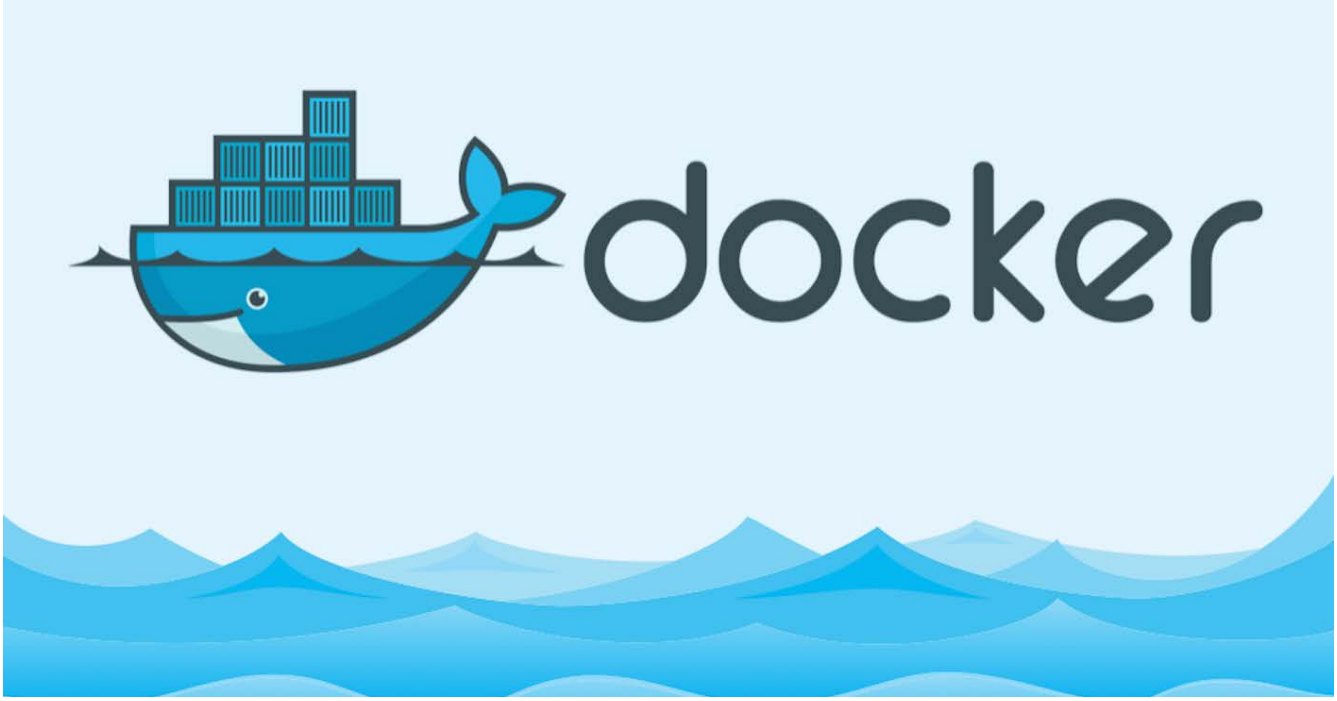


Hi,
This is AmirHossein, In this tutorial series I'm going to tell you about docker and deploy web applications with it, also I'll cover this concepts:

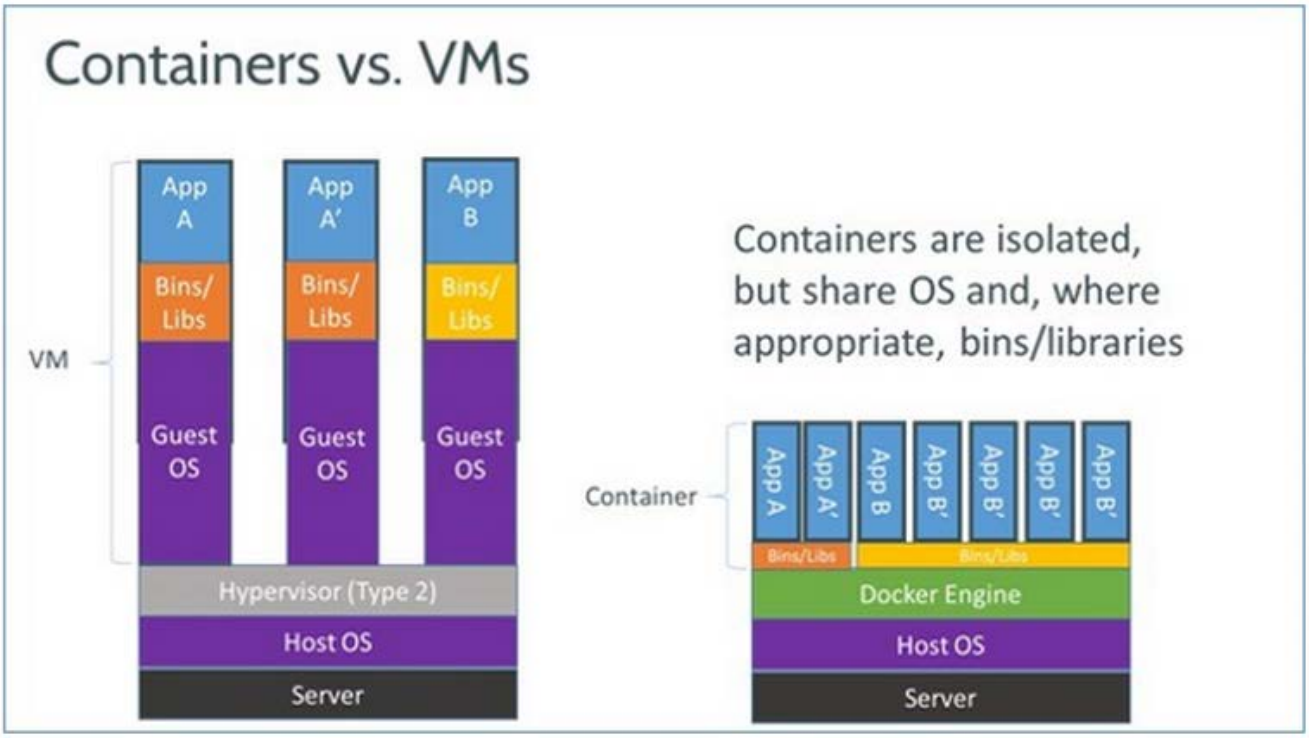
1- What is docker and Docker vs vm's
2- Docker Images and Docker Containers and Commands
3- DockerFiles and Deploy simple web application without database
4- Docker Volumes and Docker Links
5- Deploy simple web application with database (mysql)
6- Docker data containers
7- Deploy simple web application with database(mysql) and data containers
8- Deploy web application with weblogic and oracle as database
9- Docker swarm and load balancing
10- Docker Security
Hope you

What is Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

BUILD ------> SHIP ------> RUN

Its like VM's right !? so how it work and what is the advantage of this approach?



Unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application. Also all containers are isolated to each other so you can run containers separately.

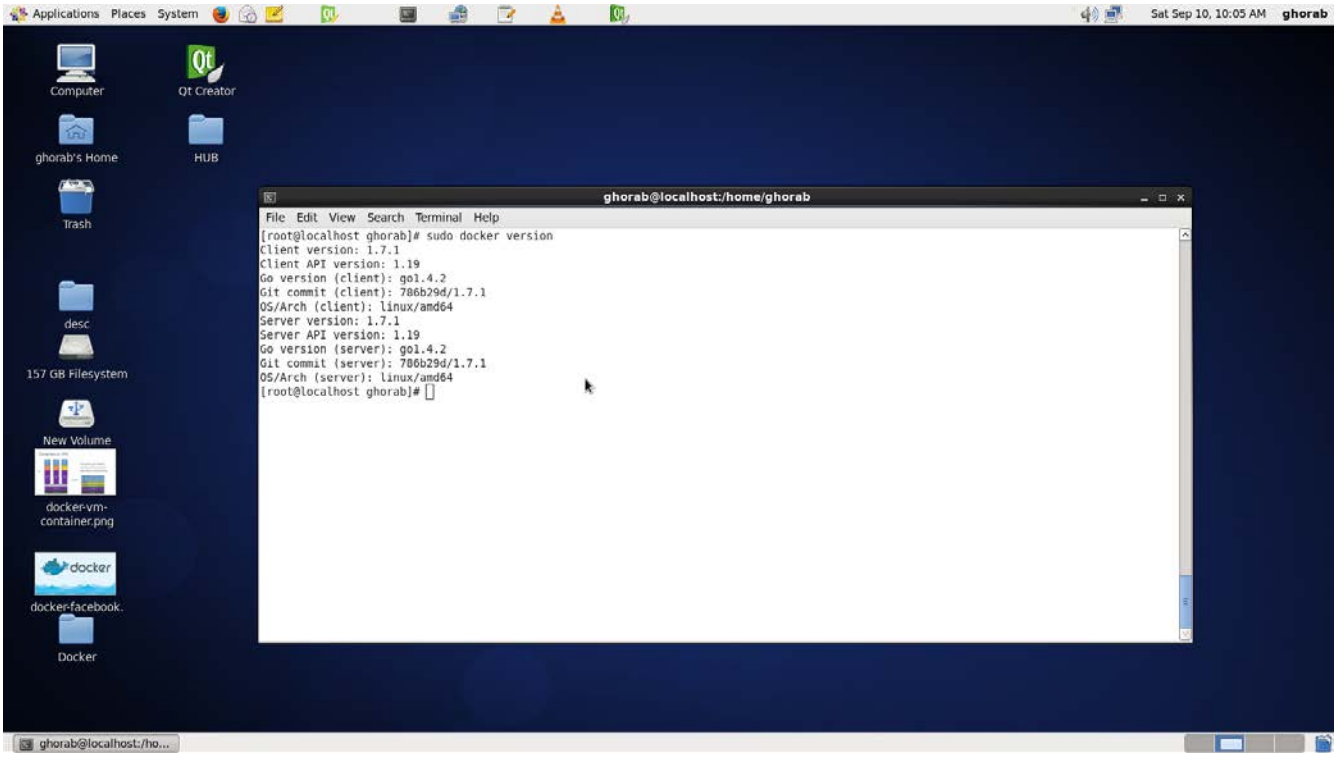
As mentioned before containers are separated from each other so its better to separate the jobs between containers (each container do one job) exactly same as multithreading. As an example think about a web application: A web application with Tomcat (or weblogic) is one container, Database app is one container and also data storage is one.

Docker Images and Docker Containers and Commands

I skipped installation part cause its pretty easy and its fully documented on www.docker.com .

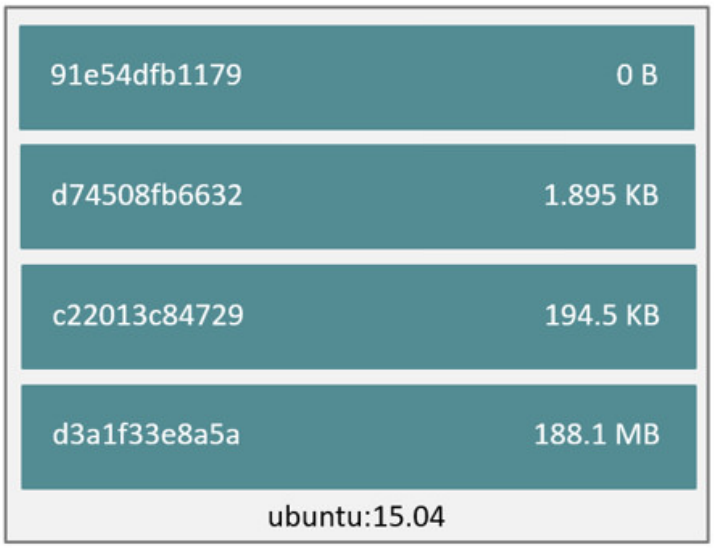
After installation you can check it with this command :

$ sudo docker version



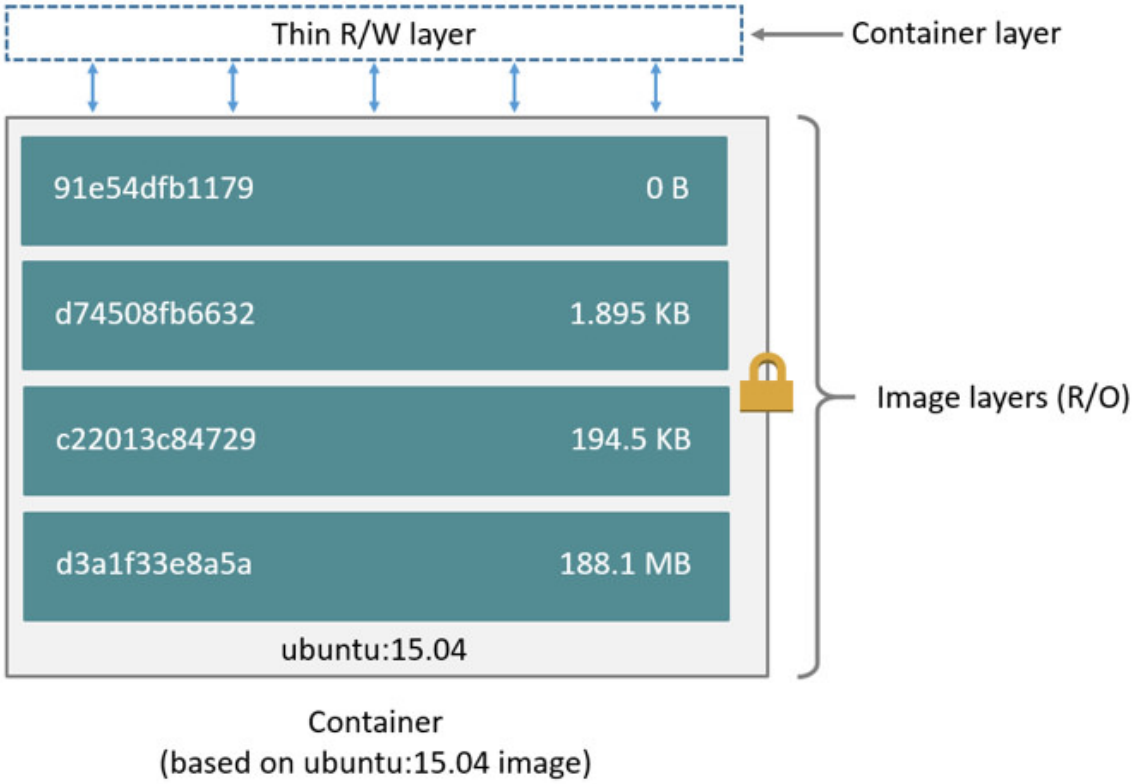
Docker Images and Docker Containers

Each Docker image references a list of read-only layers that represent filesystem differences. Layers are stacked on top of each other to form a base for a container's root filesystem. The diagram below shows the Ubuntu 15.04 image comprising 4 stacked image layers.



Image

When you create a new container, you add a new, thin, writable layer on top of the underlying stack. This layer is often called the “container layer”. All changes made to the running container - such as writing new files, modifying existing files, and deleting files - are written to this thin writable container layer. The diagram below shows a container based on the Ubuntu 15.04 image.

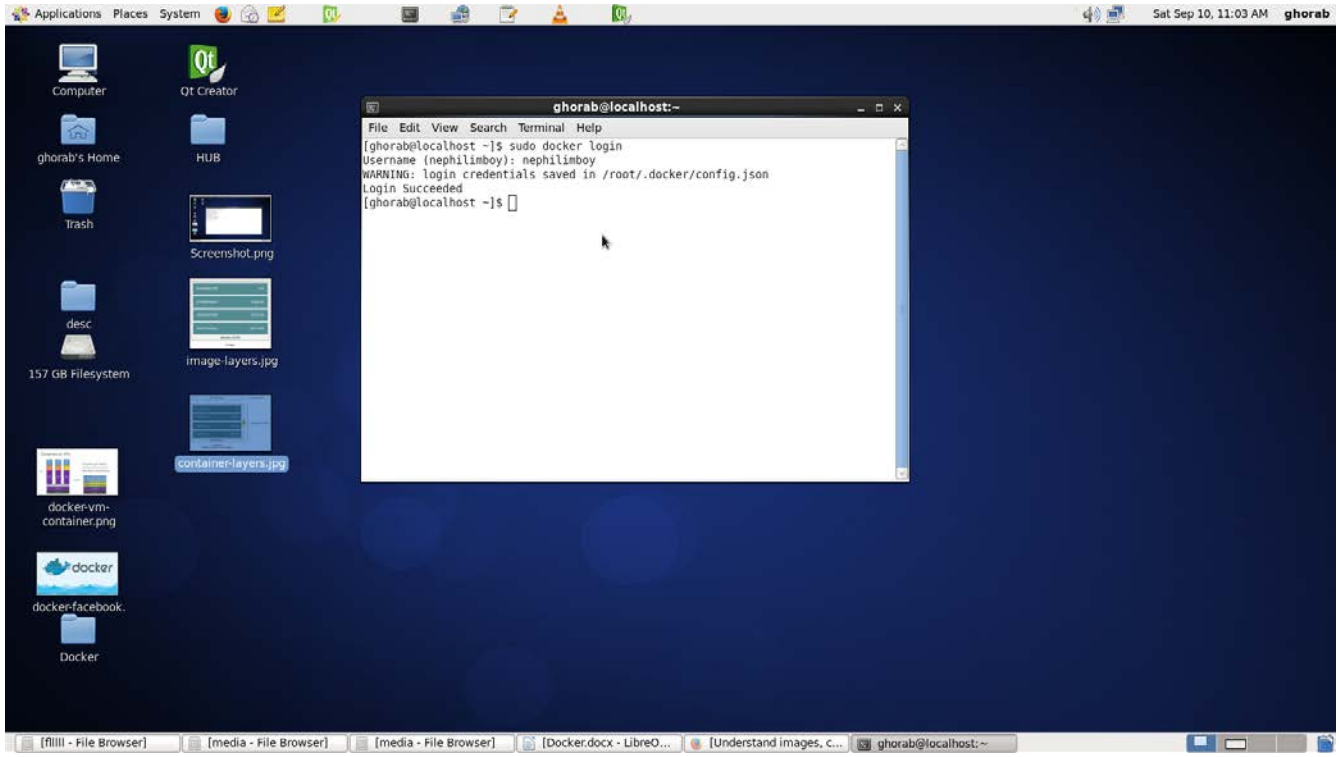


Its like : Image is a Class and a container is an Instance of that Class

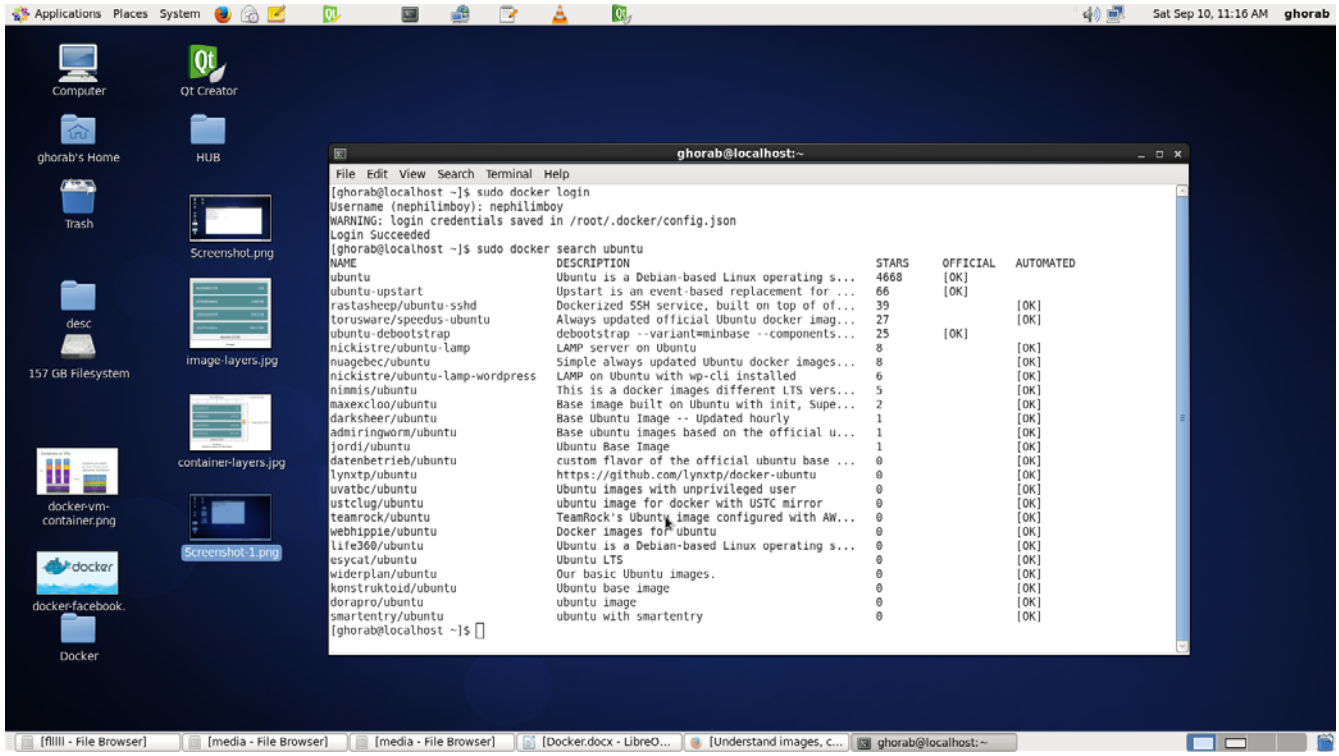
You can download (Pull) all images from “docker hub” by pull command like :

$ sudo docker pull “image name”

First lets Loin to docker hub :

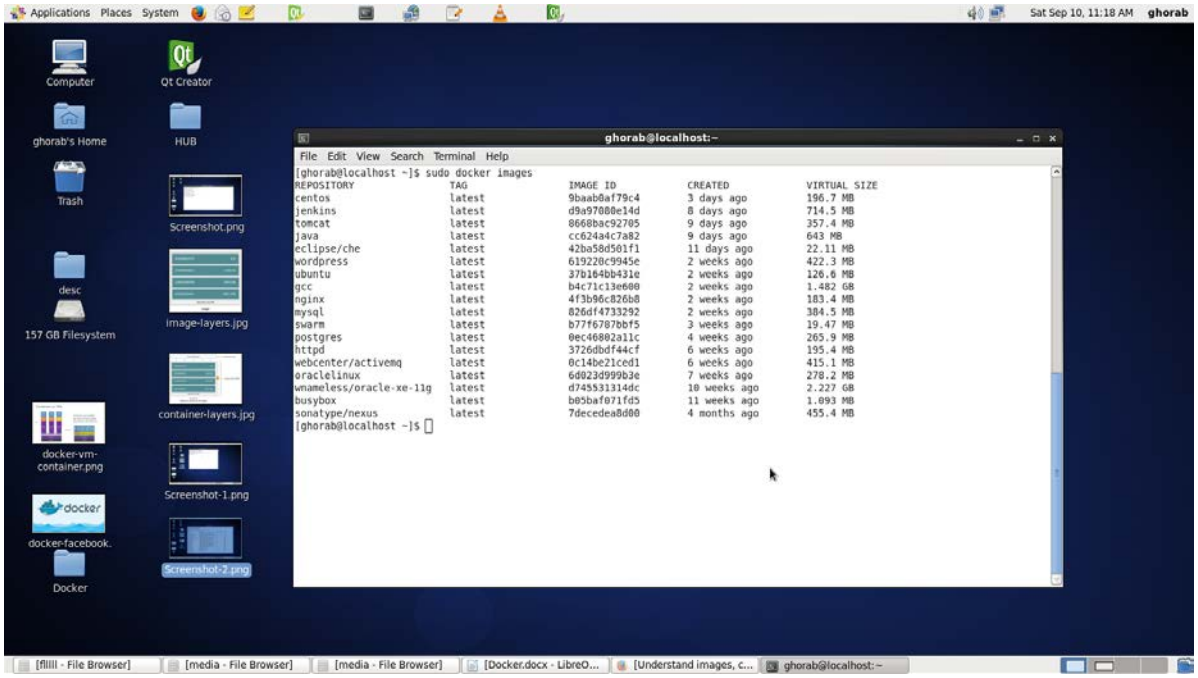


OK lets search Ubuntu images :



And you can pull them. I already pulled some of them.

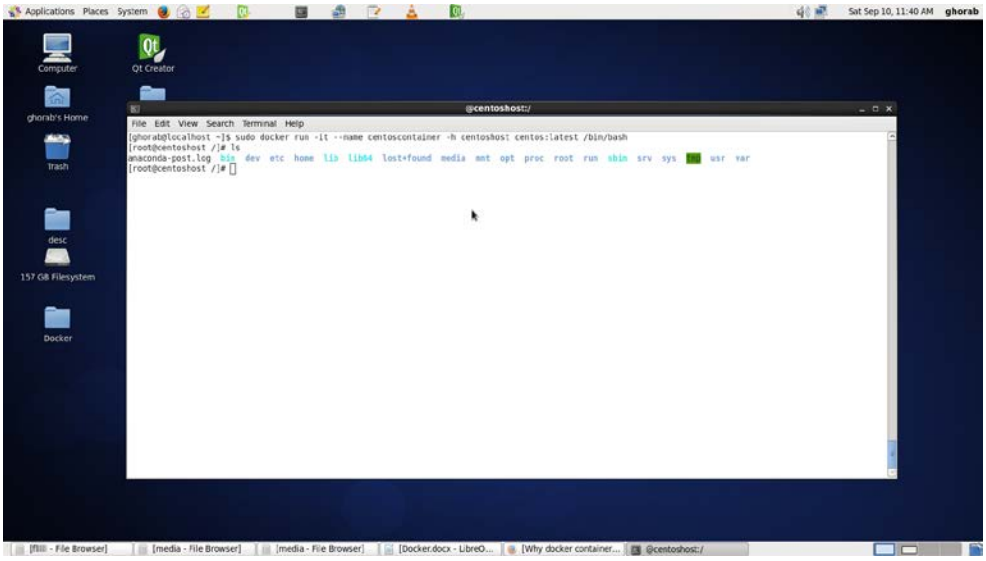
$ sudo docker images



Let’s make a container from CentOs image and run it with “RUN” command :

$sudo docker run -it –-name “container name” -h “container host name” “image name : image tag” “command that you want to run in the container “

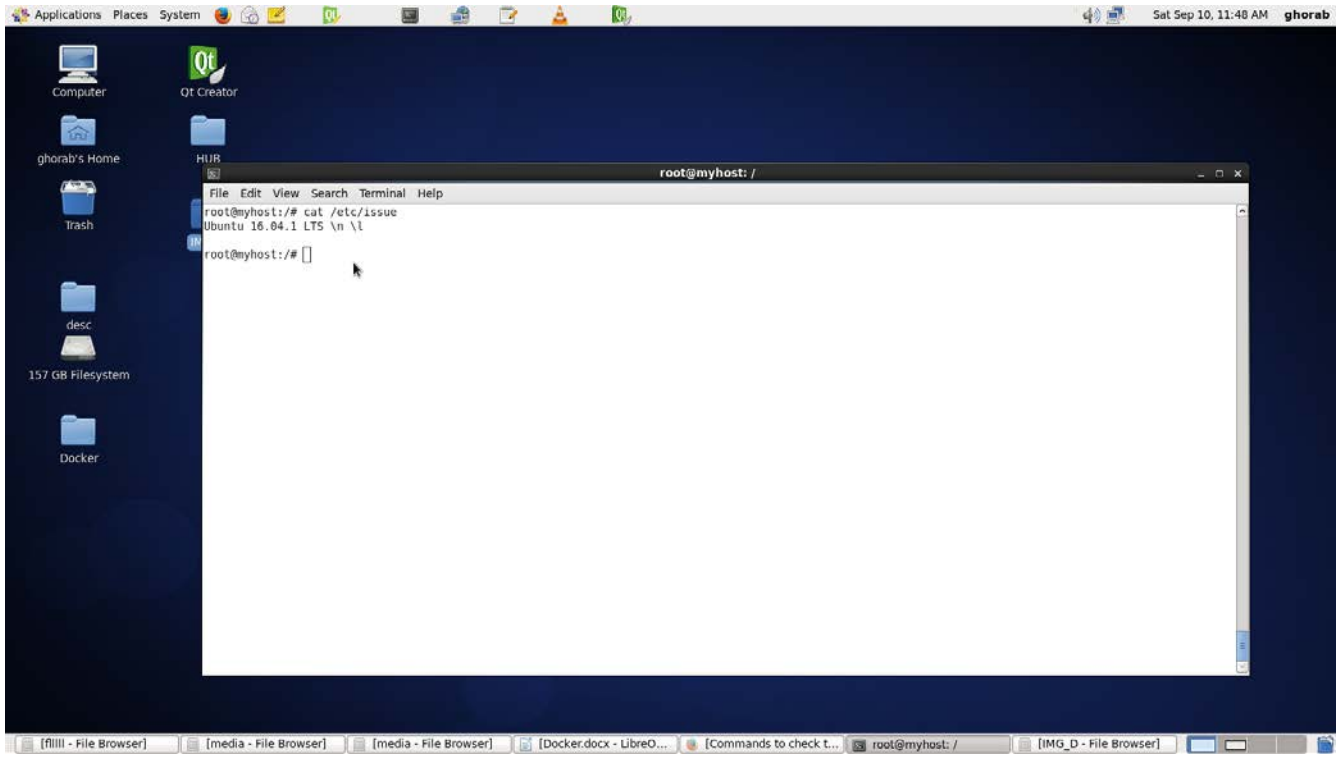
You can find all Flags in www.docker.com , let’s execute the run command :



Now we are in Centos container and we can run command there. Pretty cool right,,,
with “control p + q “ we can exit from container without stopping it, OK now let switch to Ubuntu container :

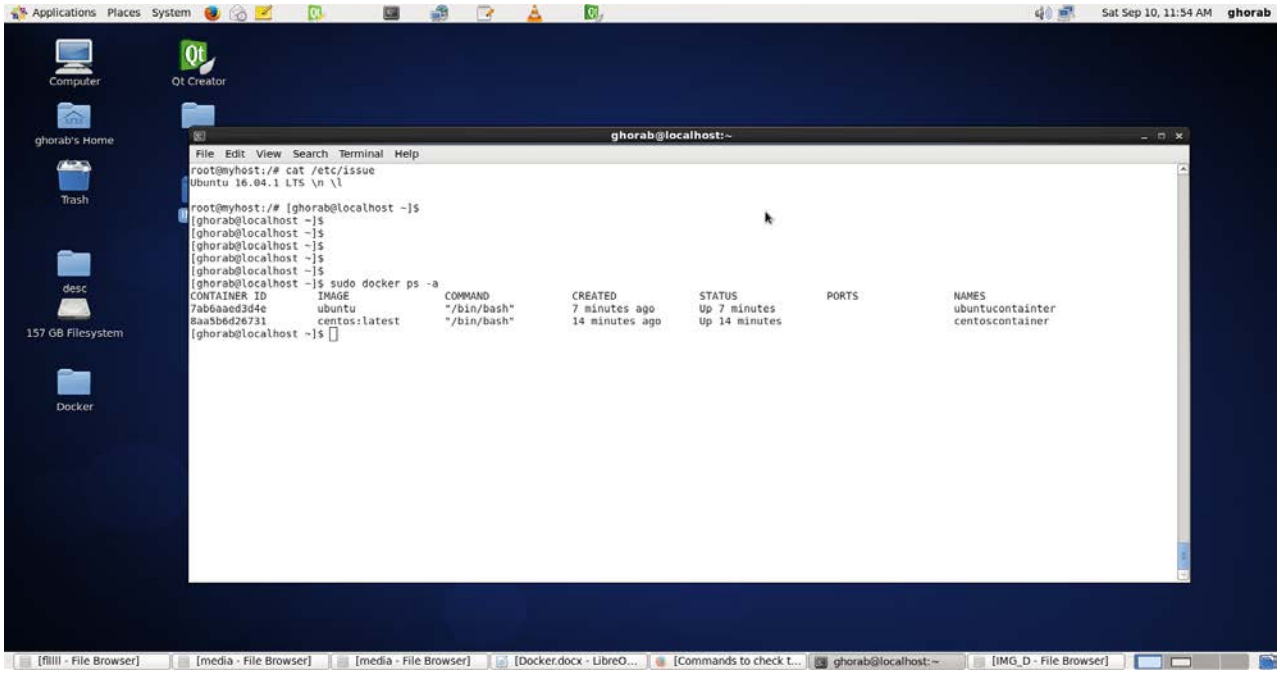
$sudo docker run -it –-name ubuntucontainter -h myhost ubuntu /bin/bash

now we are in Ubuntu Linux let’s check it with “cat /etc/issue” command...

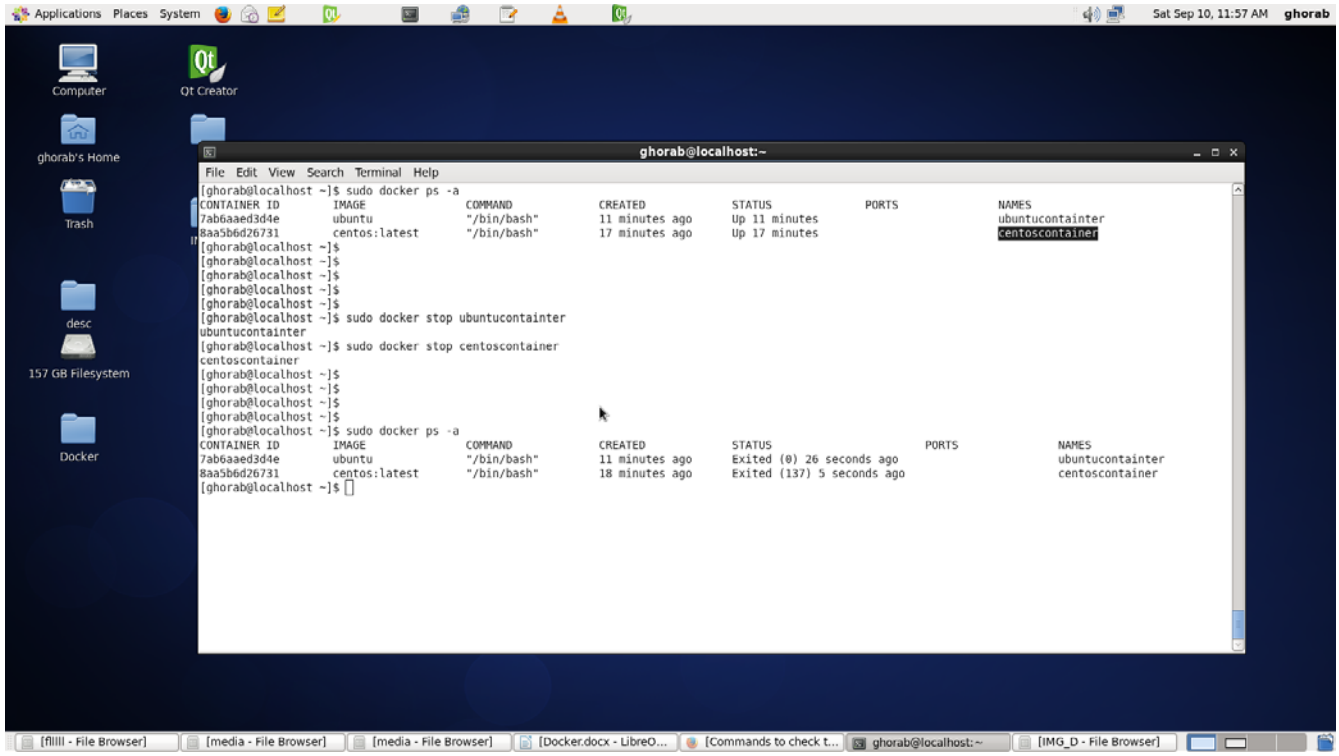


again “ control p + q “ back us to the host machine terminal ,,,

next command that I want to tell you is “ docker ps “ command. With this command you can see all running or stopping container, let’s use it with flag -a to see all containers :

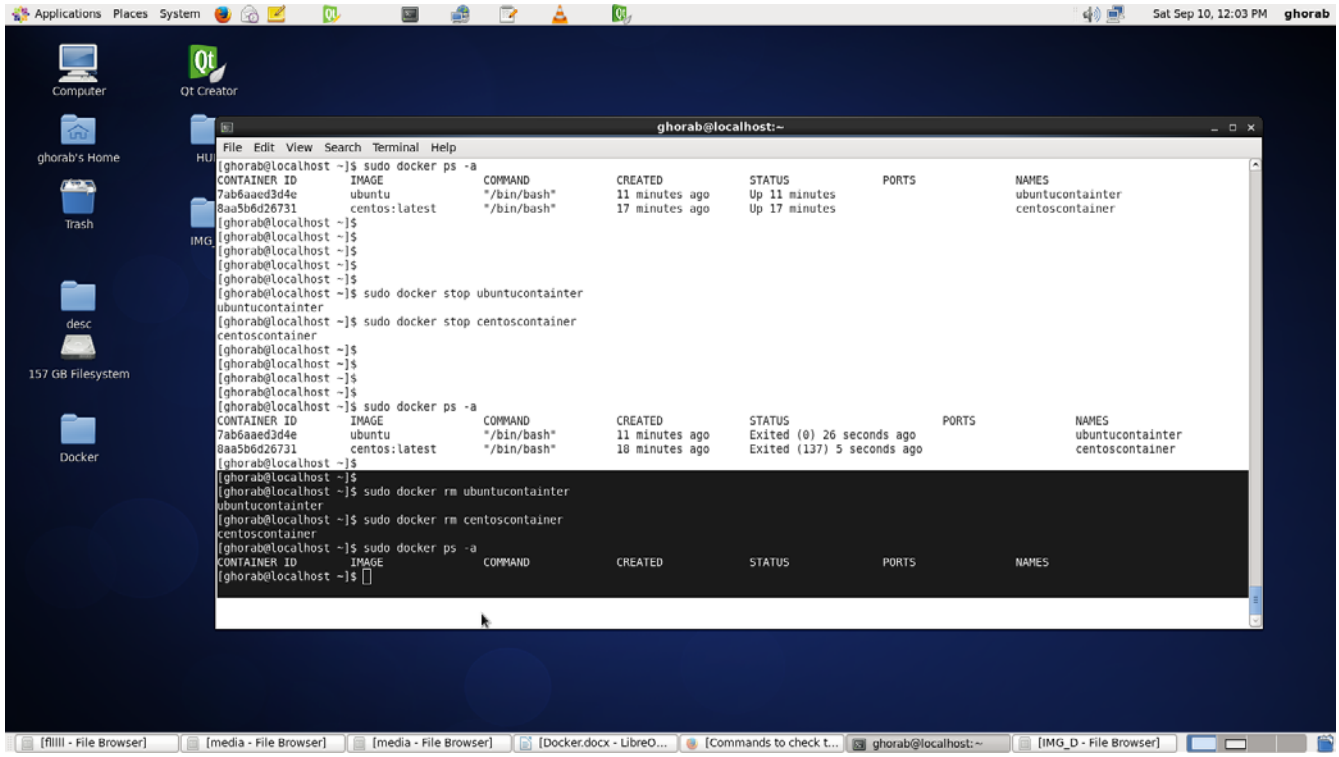


So we have 2 running containers, Let’s stop them with “docker stop “container name” “ command.....



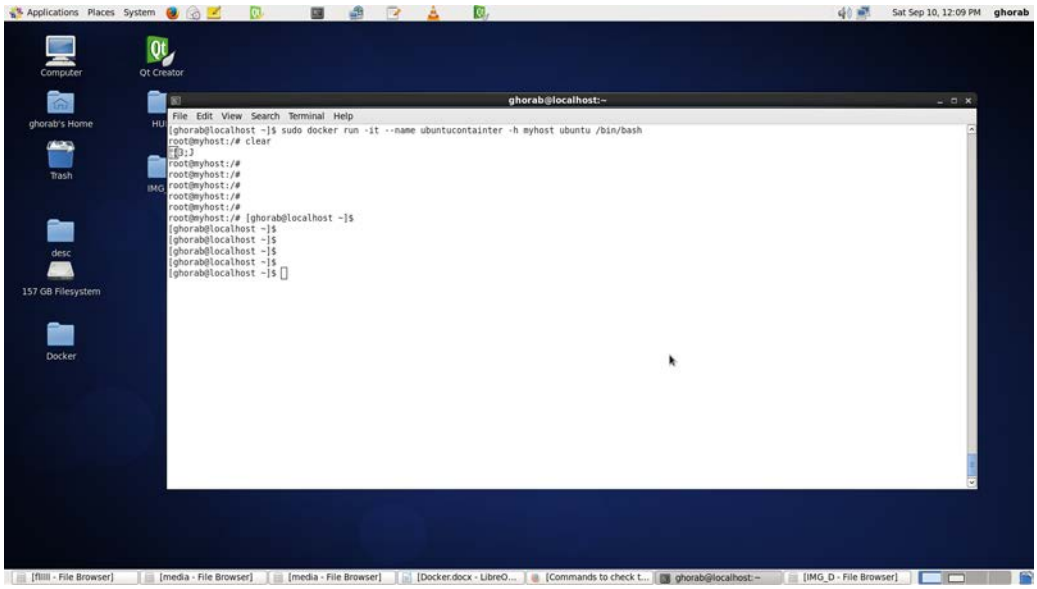
check out the STATUS column in the last “docker ps -a” command. Using “exit” keyword command in the container terminal is another way to stop the container.

Now let’s remove the containers with “ docker rm “container name” “ command.

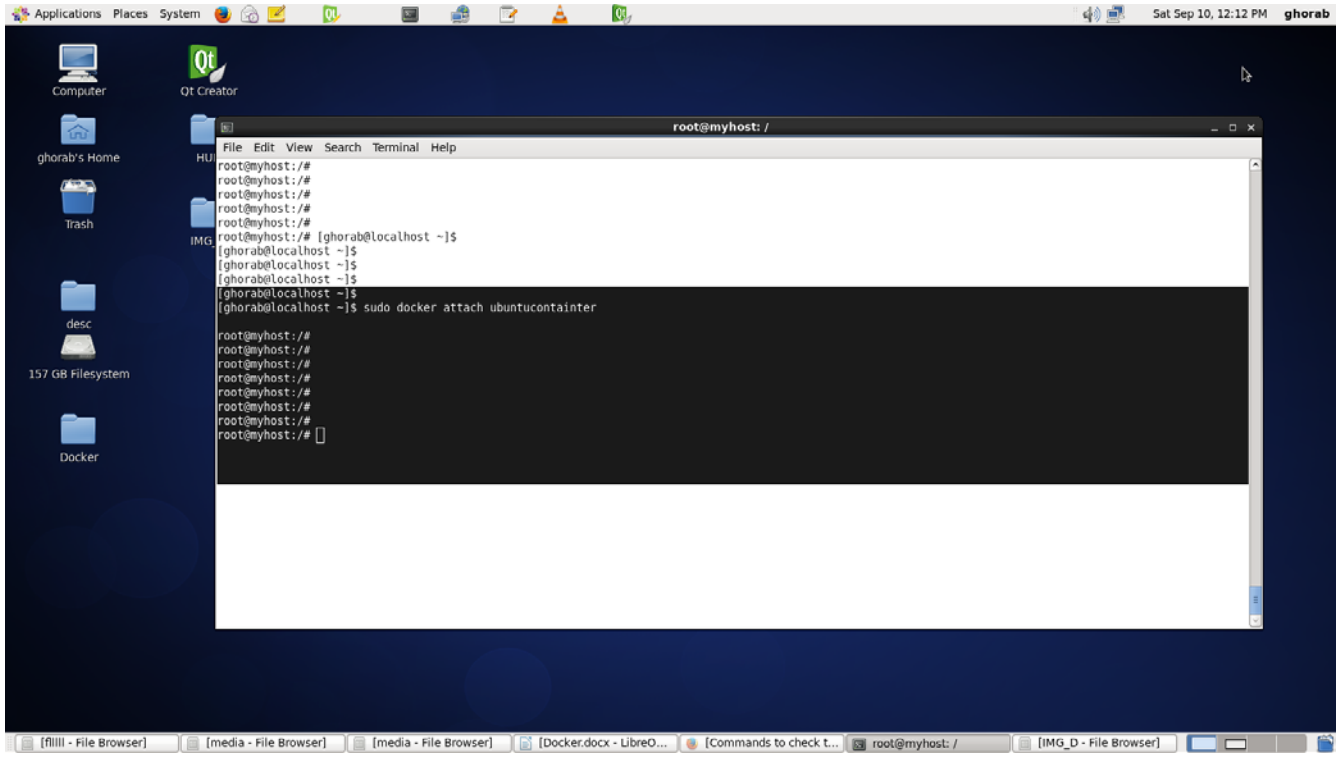


You can remove all container at once with “ docker rm $(docker ps -aq) “ command. Also you can execute “ docker rmi “image name” “ to remove image.

We can also attach to the container with attach command , let me exit from container without stopping it (control p + q).



Let’s back to the container with attach :

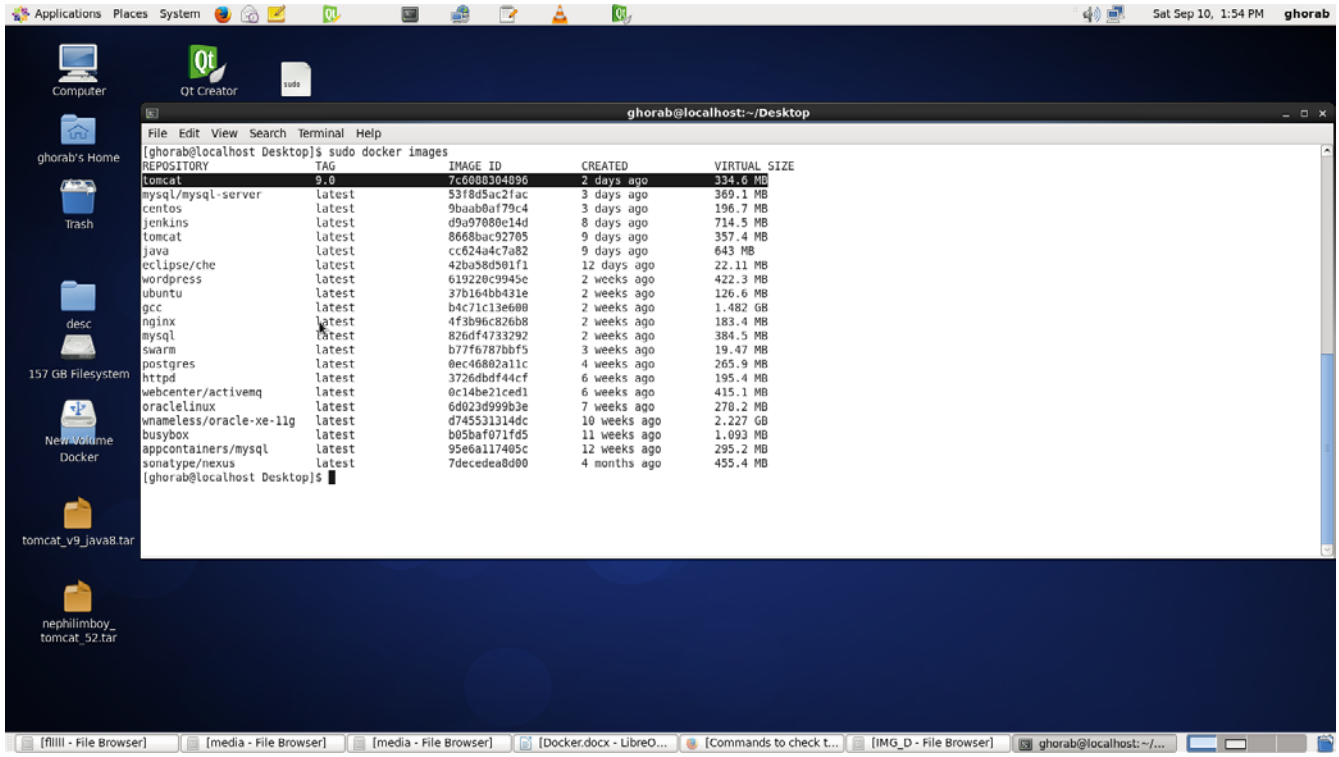


and now I'm in container again.

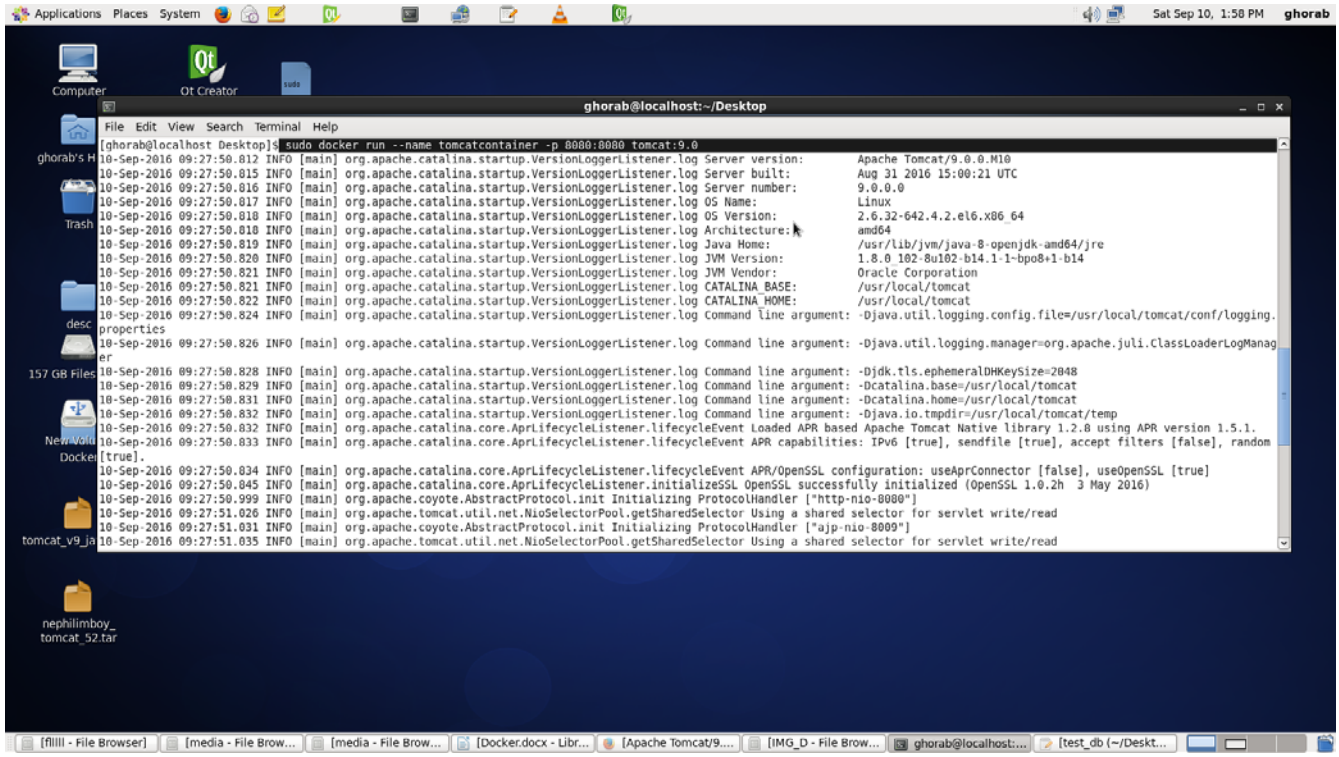
That's it for Starter part I'll show you more command in the next parts when we need it, or let me say it again every thing is fully documented in www.docker.com .

DockerFiles and Deploy web application without database

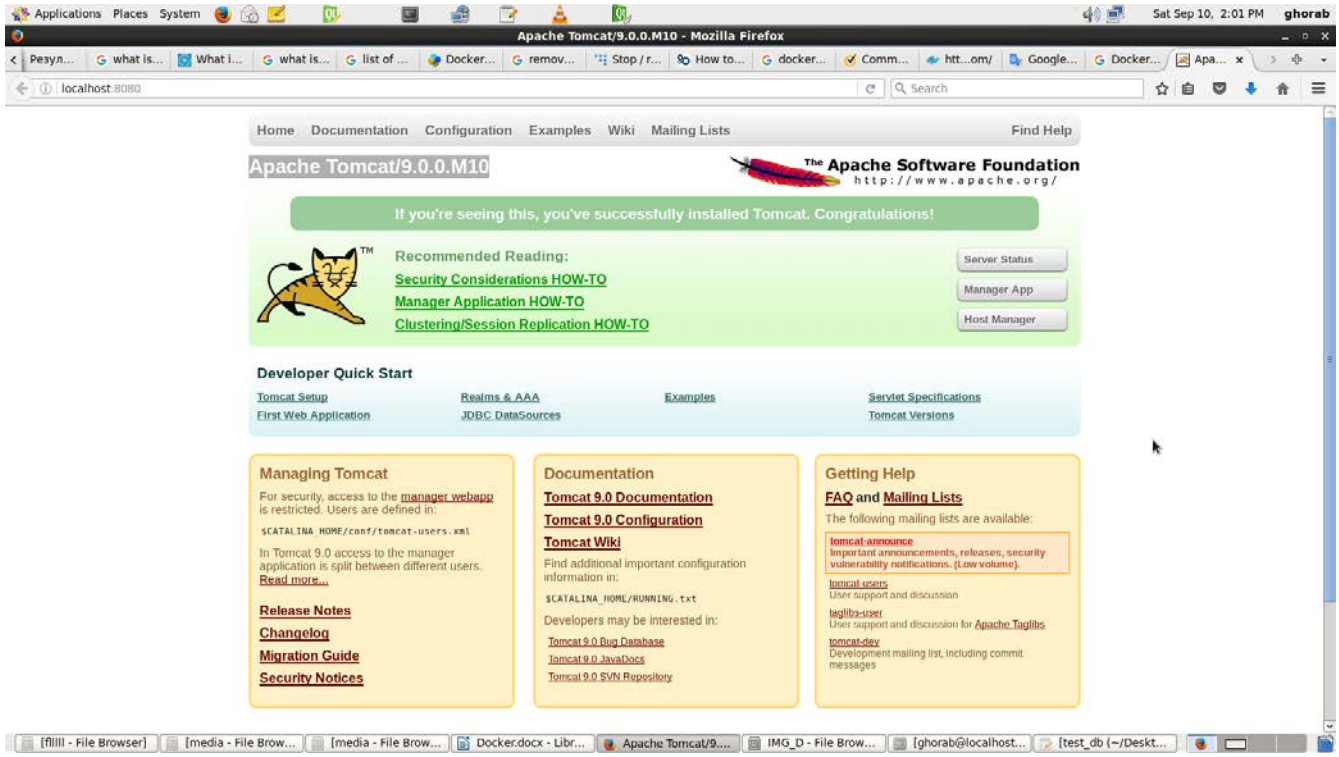
In this part of tutorial I want to deploy very simple java web application on Tomcat and run it on multiple containers. First we need Tomcat image with jdk 8 , I already pulled it.



Lets run a container from it :
$ sudo docker run --name tomcatcontainer -p 8080:8080 tomcat:9.0
(with -p flag we can map host_port : container_port)



and here we go :



Next step I want to put my java web application .WAR file to the Tomcat webapps folder, but how I can put them in running container ? Well there are many ways like using docker copy command in terminal or using volumes or … but Here I want to use a Dockerfile and create my own web application image.

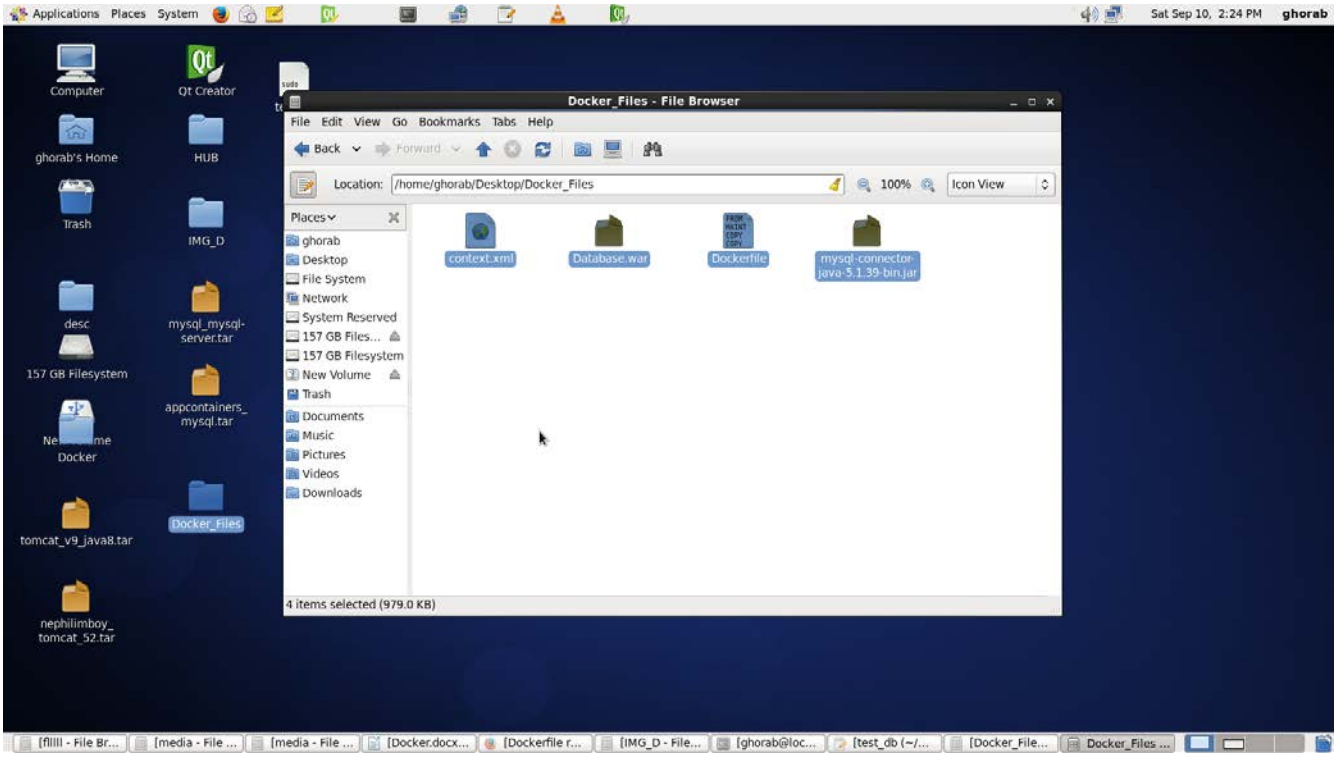
DockerFiles

Docker file (it called Dockerfile with capital “D”) 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. The command for building image is like this. (we are in Dockerfile path)
$sudo docker build -t “image name”:”tag” .

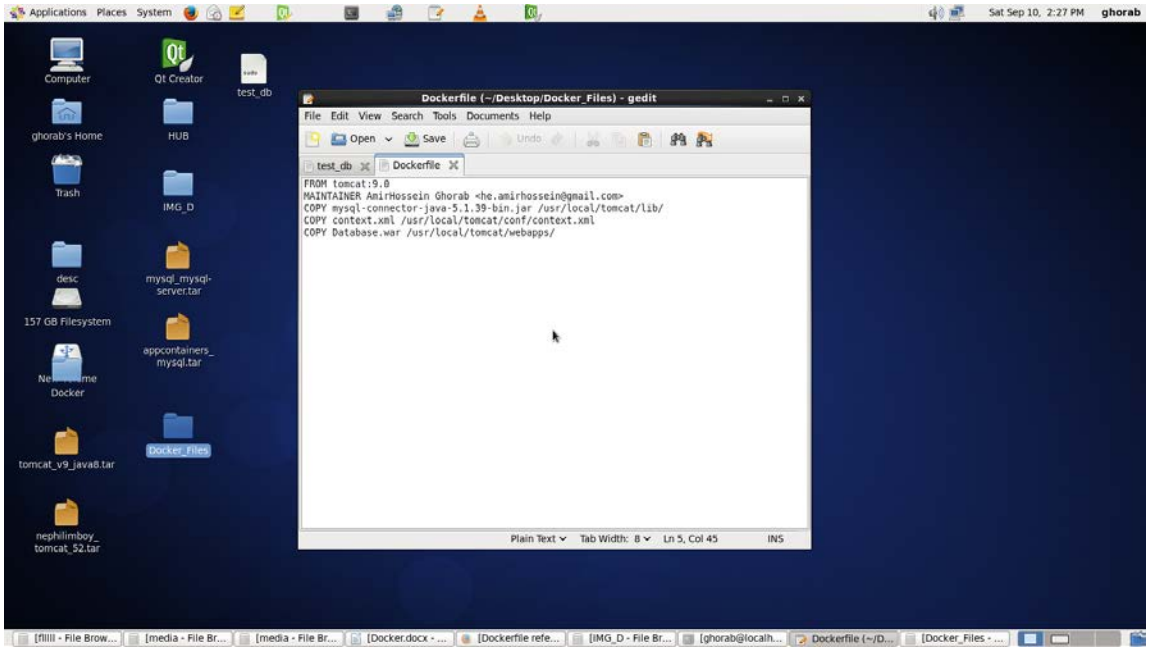
docker files is very important so you can read the references for more information :

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

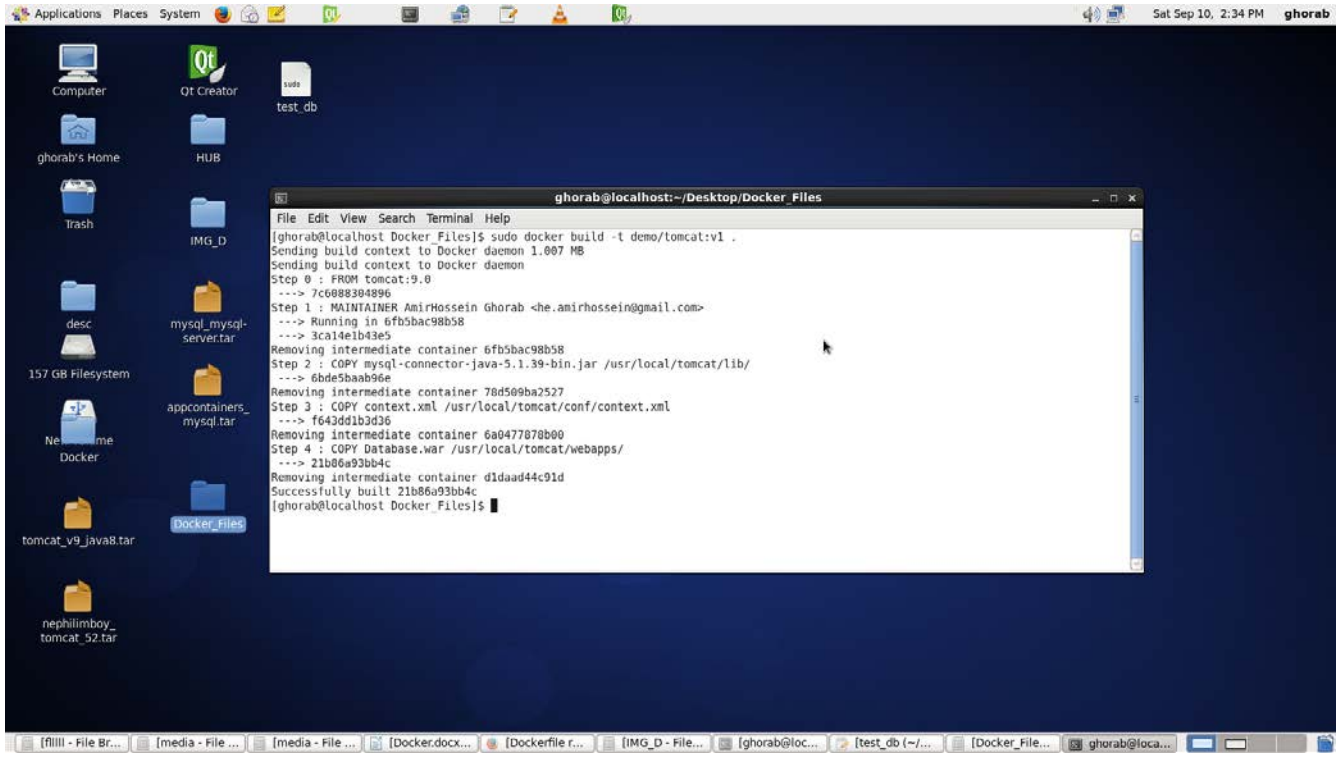
So in order to create an image we need the folder (let’s call it “Docker_Files”) and then populate it with Dockerfile and all files that we need to create the image.

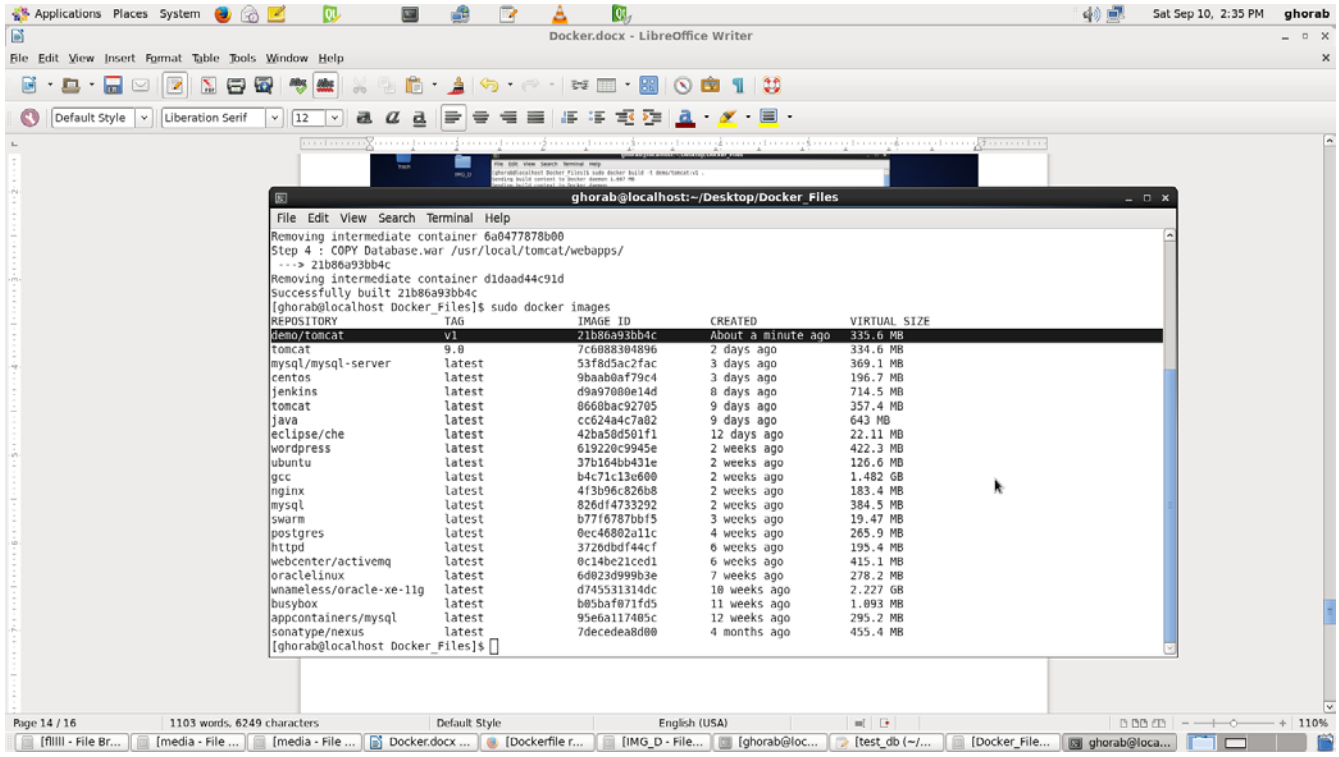


and my Dockerfile :

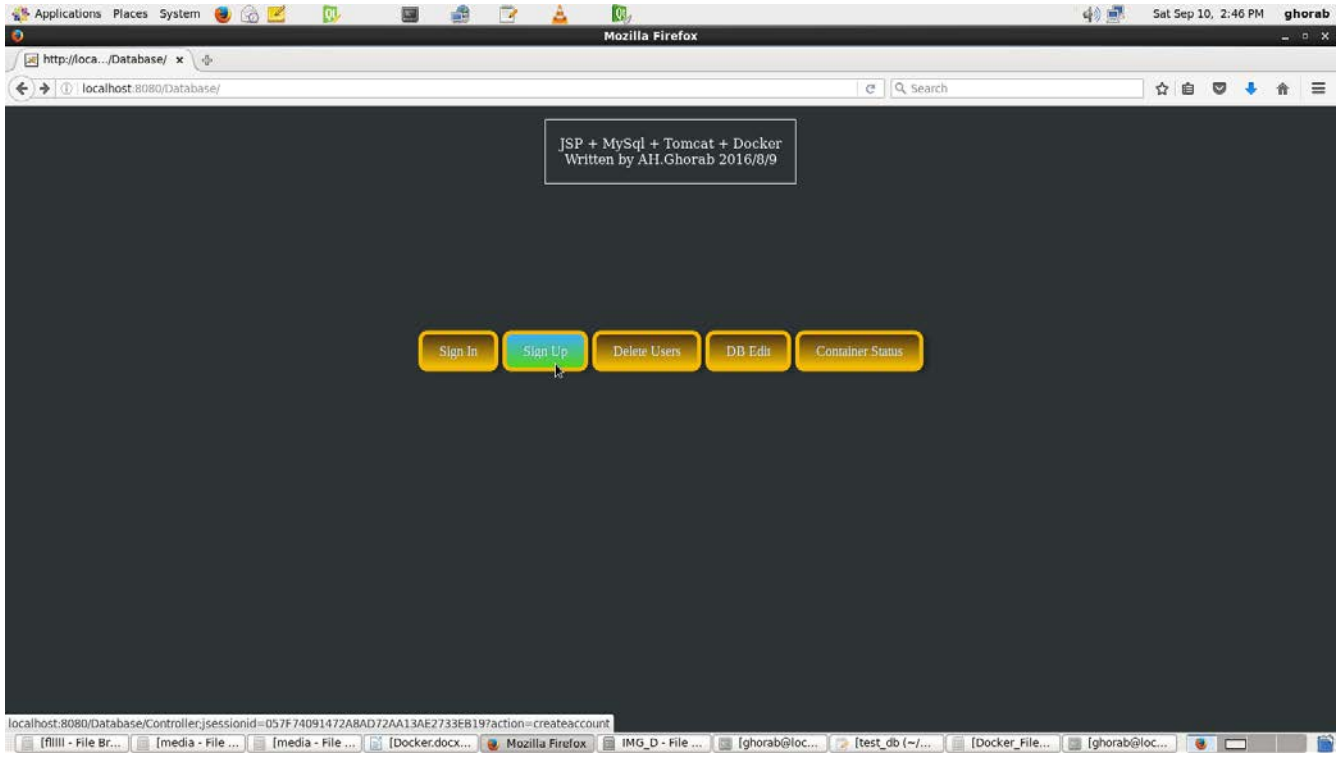


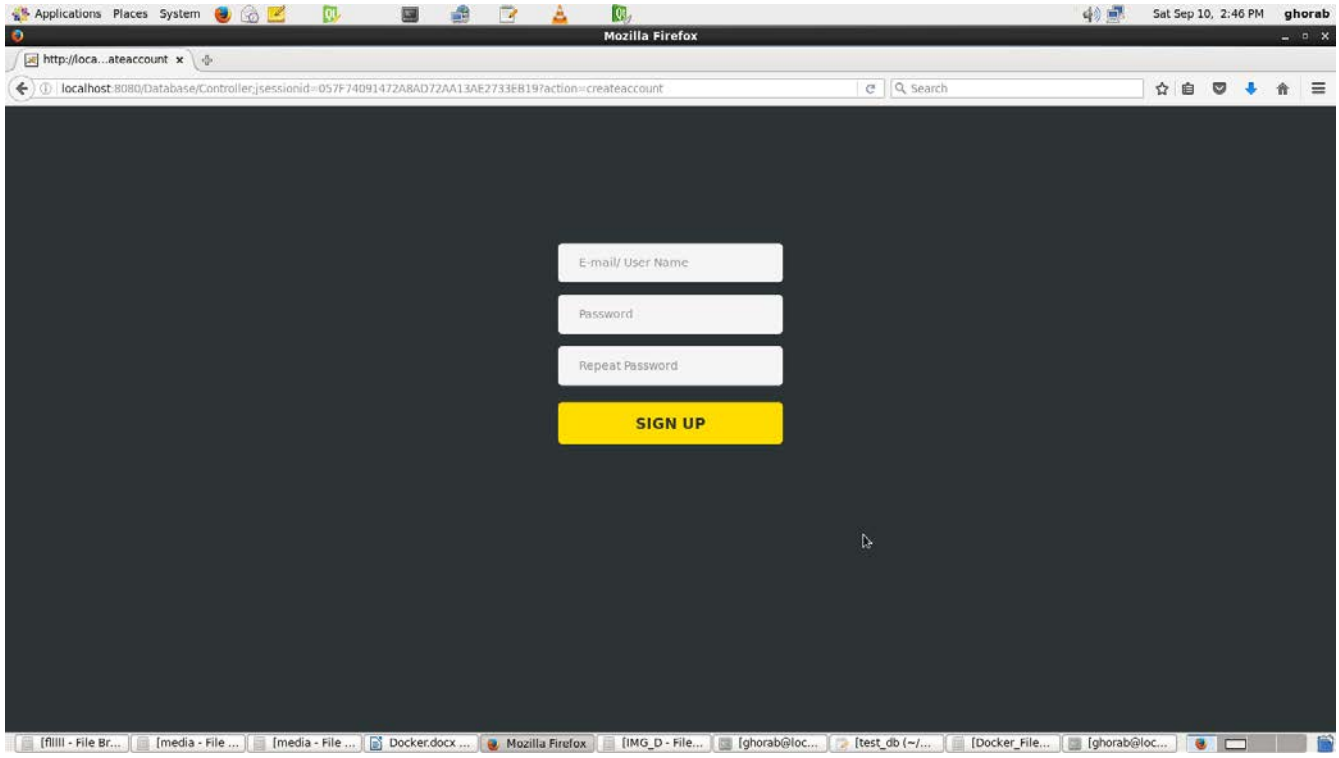
Let’s create the image :


Applications Places System
Sat Sep 10, 2:34 PM ghorab
Computer
Qt Creator
test_db
ghorab's Home
HUB
Trash
IMG_D
desc
mysql_mysql-server.tar
157 GB Filesystem
appcontainers_mysql.tar
Docker
Docker_Files
tomcat_v9_java8.tar
nephilimboy_tomcat_52.tar
ghorab@localhost:~/Desktop/Docker_Files
File Edit View Search Terminal Help
[ghorab@localhost Docker_Files]$ sudo docker build -t demo/tomcat:v1 .
Sending build context to Docker daemon 1.007 MB
Sending build context to Docker daemon
Step 0 : FROM tomcat:9.0
---> 7c6088304896
Step 1 : MAINTAINER AmirHossein Ghorab <he.amirhossein@gmail.com>
---> Running in 6fb5bac98b58
---> 3ca14e1b43e5
Removing intermediate container 6fb5bac98b58
Step 2 : COPY mysql-connector-java-5.1.39-bin.jar /usr/local/tomcat/lib/
---> 6bde5baab96e
Removing intermediate container 78d509ba2527
Step 3 : COPY context.xml /usr/local/tomcat/conf/context.xml
---> f643dd1b3d36
Removing intermediate container 6a0477878b00
Step 4 : COPY Database.war /usr/local/tomcat/webapps/
---> 21b86a93bb4c
Removing intermediate container d1daad44c91d
Successfully built 21b86a93bb4c
[ghorab@localhost Docker_Files]$

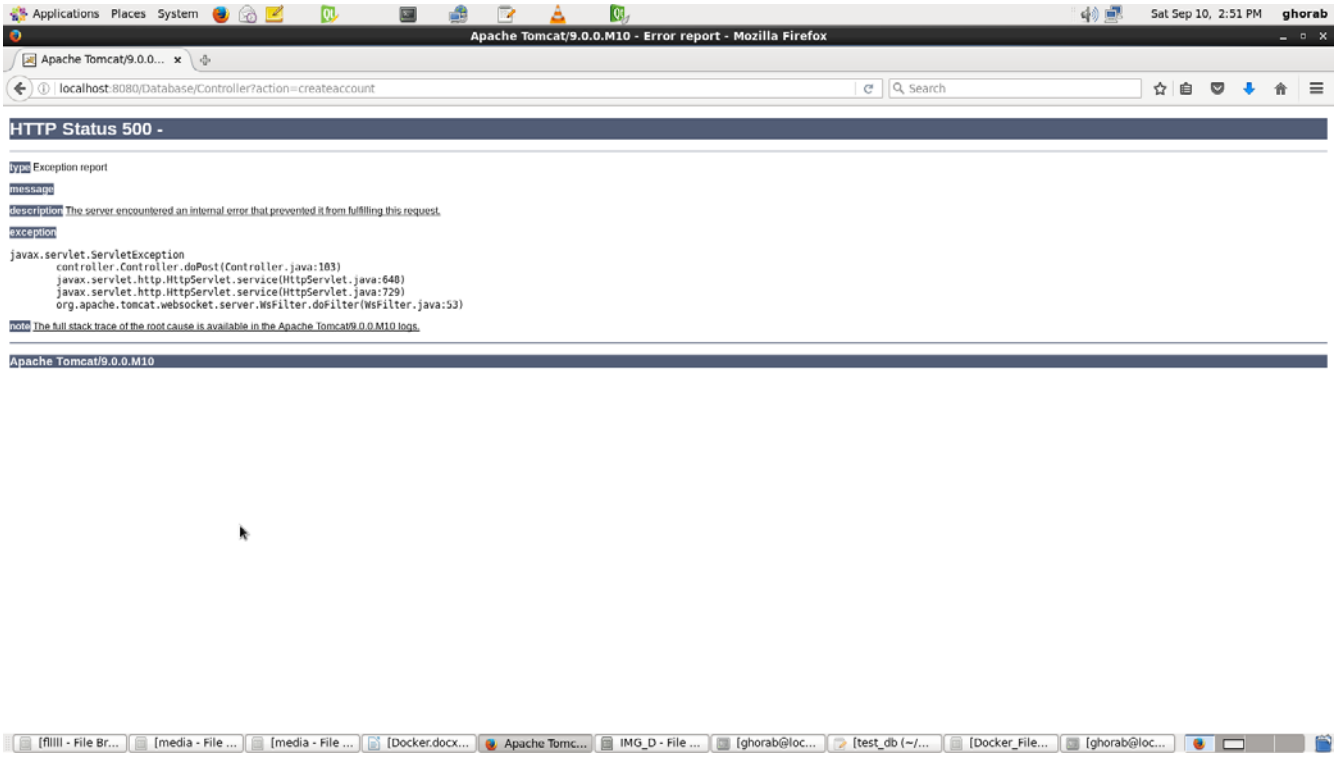

Applications Places System
Sat Sep 10, 2:35 PM ghorab
Docker.docx - LibreOffice Writer
File Edit View Insert Format Table Tools Window Help
Default Style
Liberation Serif
12
ghorab@localhost:~/Desktop/Docker_Files
File Edit View Search Terminal Help
Removing intermediate container 6a0477878b00
Step 4 : COPY Database.war /usr/local/tomcat/webapps/
---> 21b86a93bb4c
Removing intermediate container d1daad44c91d
Successfully built 21b86a93bb4c
[ghorab@localhost Docker_Files]$ sudo docker images
REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE
demo/tomcat v1 21b86a93bb4c About a minute ago 335.6 MB
tomcat 9.0 7c6088304896 2 days ago 334.6 MB
mysql/mysql-server latest 53f8d5ac2fac 3 days ago 369.1 MB
centos latest 9baab0af79c4 3 days ago 196.7 MB
jenkins latest d9a97080e14d 8 days ago 714.5 MB
tomcat latest 8668bac92705 9 days ago 357.4 MB
java latest cc624a4c7a82 9 days ago 643 MB
eclipse/che latest 42ba58d501f1 12 days ago 22.11 MB
wordpress latest 619220c9945e 2 weeks ago 422.3 MB
ubuntu latest 37b164bb431e 2 weeks ago 126.6 MB
gcc latest b4c71c13e600 2 weeks ago 1.482 GB
nginx latest 4f3b96c826b8 2 weeks ago 183.4 MB
mysql latest 826df4733292 2 weeks ago 384.5 MB
swarm latest b77f6787bbf5 3 weeks ago 19.47 MB
postgres latest 0ec46802a11c 4 weeks ago 265.9 MB
httpd latest 3726dbdf44cf 6 weeks ago 195.4 MB
webcenter/activemq latest 0c14be21ced1 6 weeks ago 415.1 MB
oraclelinux latest 6d023d999b3e 7 weeks ago 278.2 MB
wnameless/oracle-xe-11g latest d745531314dc 10 weeks ago 2.227 GB
busybox latest b05baf071fd5 11 weeks ago 1.093 MB
appcontainers/mysql latest 95e6a117485c 12 weeks ago 295.2 MB
sonatype/nexus latest 7decedea8d00 4 months ago 455.4 MB
[ghorab@localhost Docker_Files]$
Page 14 / 16
1103 words, 6249 characters
Default Style
English (USA)
110%

Let’s run it....


Applications Places System
Sat Sep 10, 2:46 PM ghorab
Mozilla Firefox
http://loca.../Database/
localhost:8080/Database/
JSP + MySql + Tomcat + Docker
Written by AH.Ghorab 2016/8/9
Sign In
Sign Up
Delete Users
DB Edit
Container Status
localhost:8080/Database/Controller;jsessionid=057F74091472A8AD72AA13AE2733EB19?action=createaccount


Applications Places System
Sat Sep 10, 2:46 PM ghorab
Mozilla Firefox
http://loca...ateaccount
localhost:8080/Database/Controller;jsessionid=057F74091472A8AD72AA13AE2733EB19?action=createaccount
E-mail/ User Name
Password
Repeat Password
SIGN UP

We can also run it on multiple container on different ports. This web application cant access to any database so if we click on SIGN UP we get ERROR...



Docker Volumes and Docker Links

Docker Volumes

A *data volume* is a specially-designated directory within one or more containers that bypasses the Union File Systems. Data volumes provide several useful features for persistent or shared data:

- Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization. (Note that this does not apply when mounting a host directory.)
- Data volumes can be shared and reused among containers.
- Changes to a data volume are made directly.
- Changes to a data volume will not be included when you update an image.
- Data volumes persist even if the container itself is deleted.

Data volumes are designed to persist data, independent of the container's life cycle. Docker therefore *never* automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

Mount a host directory as a data volume

In addition to creating a volume using the `-v` flag you can also mount a directory from your Docker engine's host into a container.

$ docker run -d -P --name web -v /src/webapp:/opt/webapp demo/tomcat:v1

This command mounts the host directory, `/src/webapp`, into the container at `/opt/webapp`. If the path `/opt/webapp` already exists inside the container's image, the `/src/webapp` mount overlays but does not remove the pre-existing content. Once the mount is removed, the content is accessible again. This is consistent with the expected behavior of the `mount` command. More information is on :

https://docs.docker.com/engine/tutorials/dockervolumes/

Docker Links

linking system allows you to link multiple containers together and send connection information from one to another. When containers are linked, information about a source container can be sent to a recipient container. This allows the recipient to see selected data describing aspects of the source container.

To establish links, Docker relies on the names of your containers. You've already seen that each container you create has an automatically created name or you can specify a name for it . This naming provides two useful functions:

1. It can be useful to name containers that do specific functions in a way that makes it easier for you to remember them, for example naming a container containing a web application `web`.
2. It provides Docker with a reference point that allows it to refer to other containers, for example, you can specify to link the container `web` to container `db`.

Links allow containers to discover each other and securely transfer information about one container to another container. When you set up a link, you create a conduit between a source container and a recipient container. The recipient can then access select data about the source. To create a link, you use the `--link` flag. First, create a new container, this time one containing a database.

$ docker run -d --name db demo/mysql

his creates a new container called `db` from the "demo/mysql" image, which contains a MySQL database.

Now, create a new `web` container and link it with your `db` container.

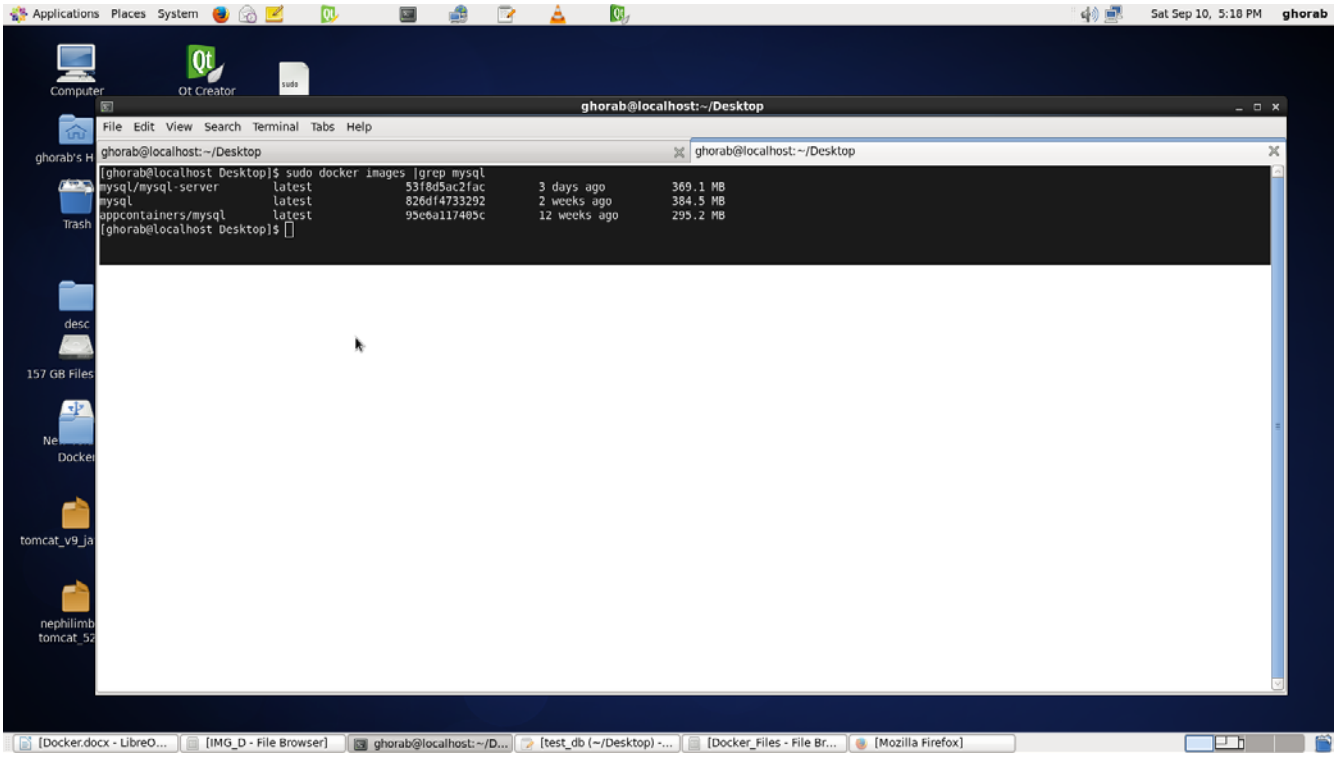
$ docker run -d -P --name web --link db:db demo/tomcat:v1

This will link the new `web` container with the `db` container you created earlier. More information is on :

https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks

Deploy simple web application with database (mysql)

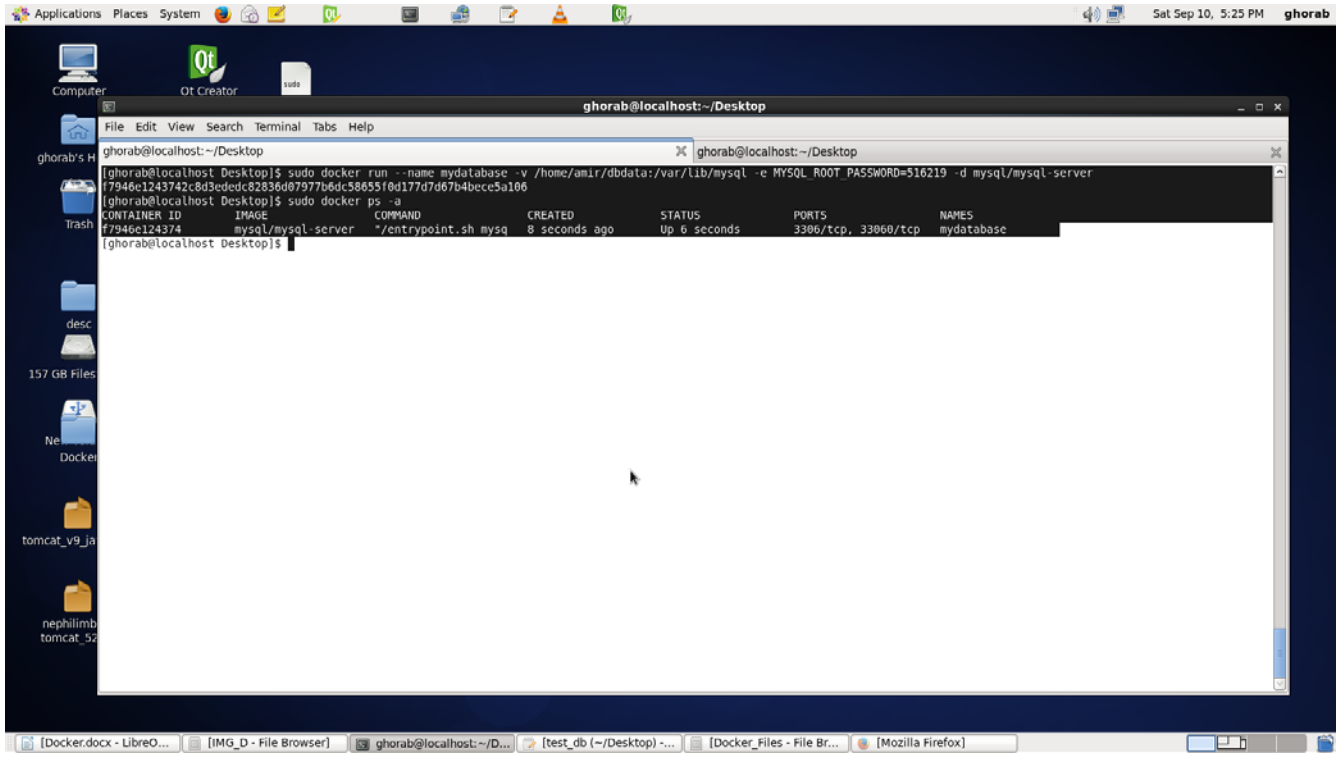
In this part I want to deploy my previous web application with MySQL database fist we need image of Mysql, I already pulled it from Docker HUB...



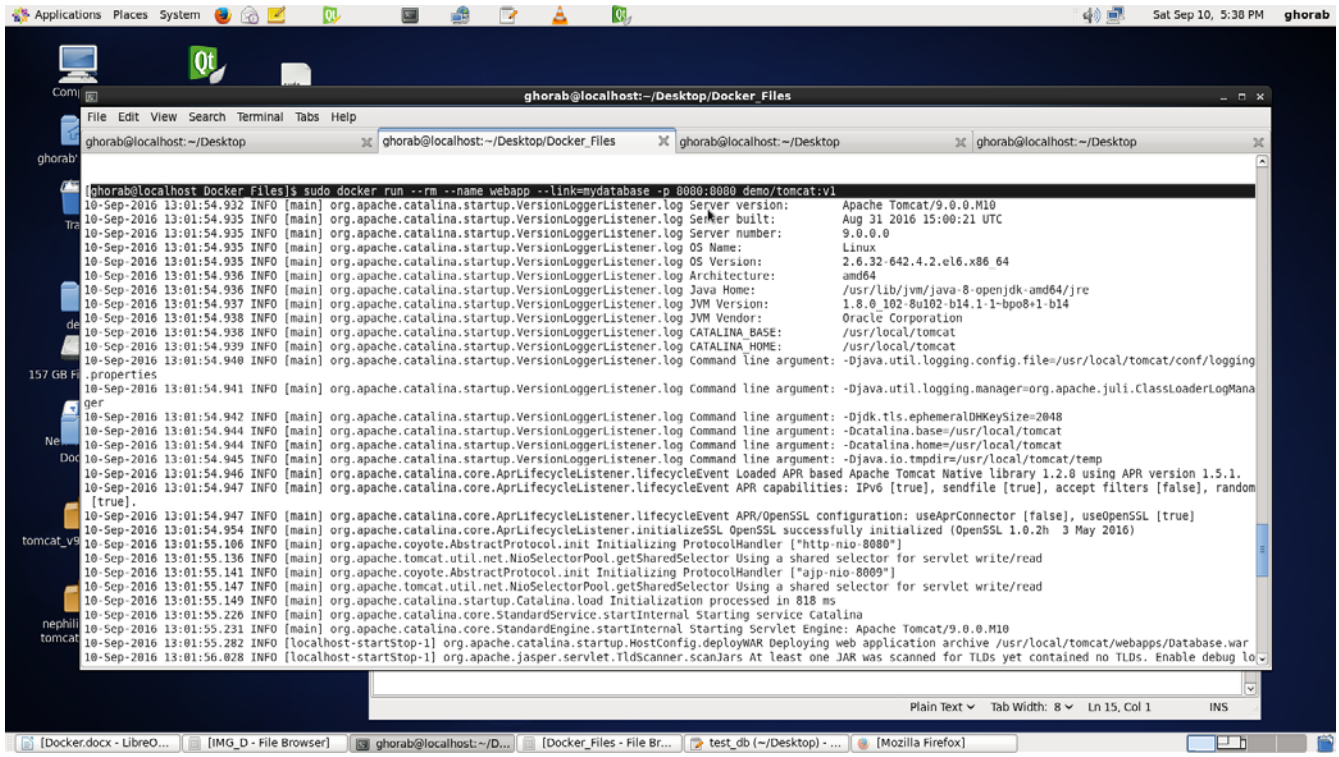
Let’s run it :

$ sudo docker run --name mydatabase -v /home/amir/dbdata:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=516219 -d mysql/mysql-server

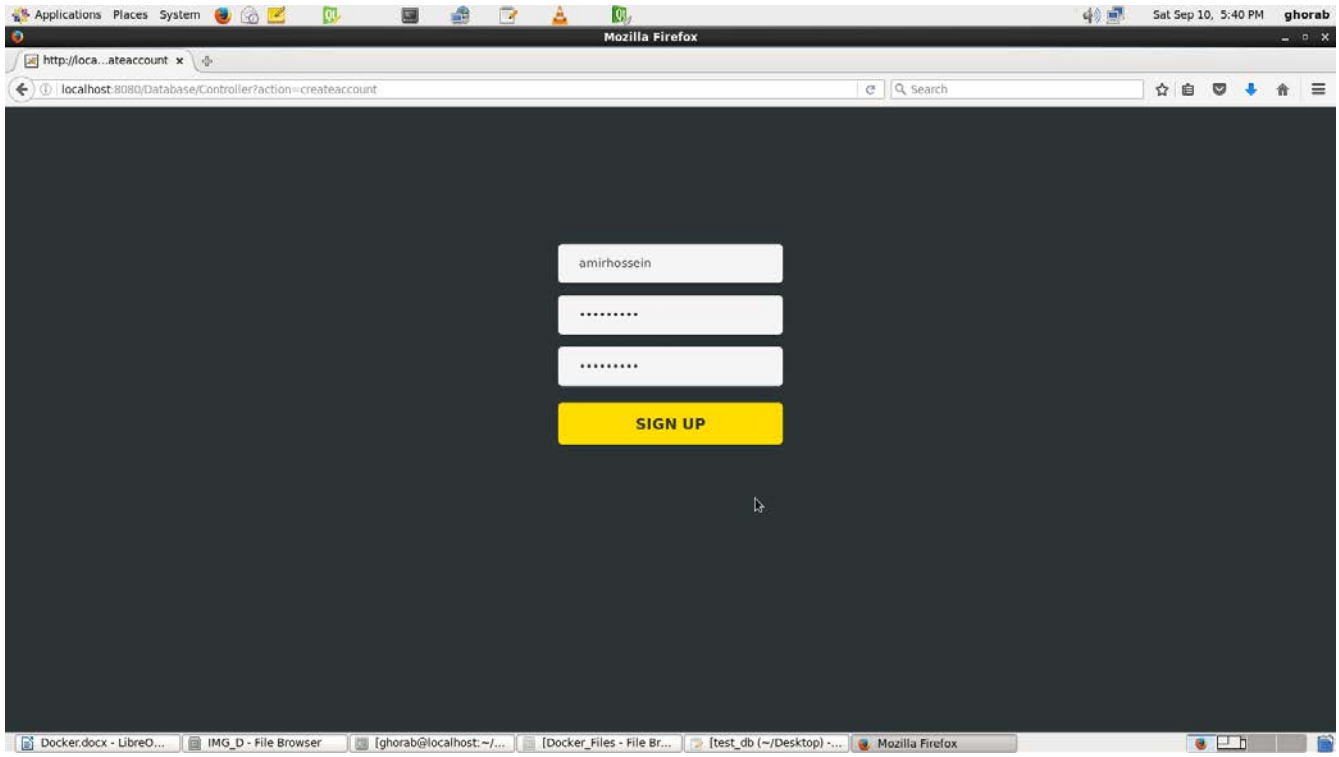
you can read https://hub.docker.com/r/mysql/mysql-server/ for more details. As you can see in the run command I used volumes and map it to my database container.



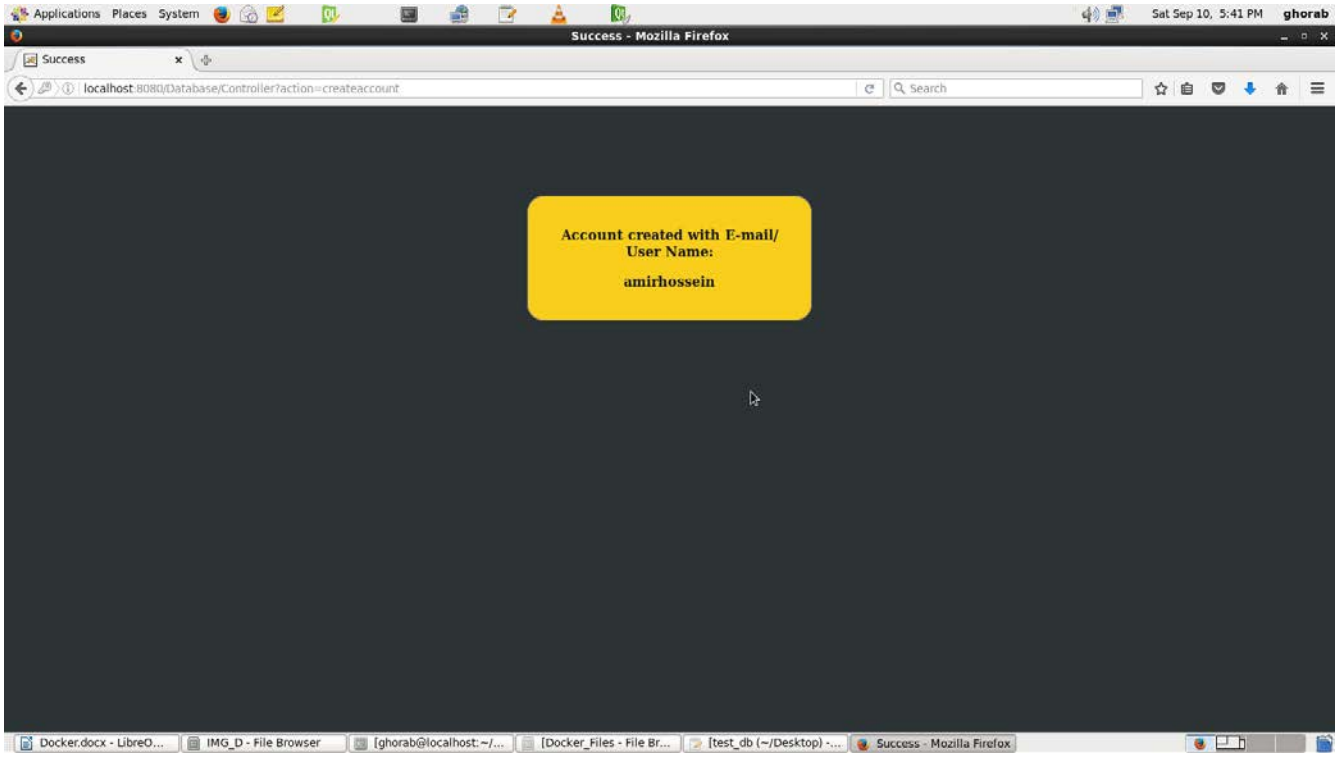
And also let’s run our web applicatio, this time with “–-link” flag...
$sudo docker run --name webapp --link=mydatabase -p 8080:8080 demo/tomcat:v1

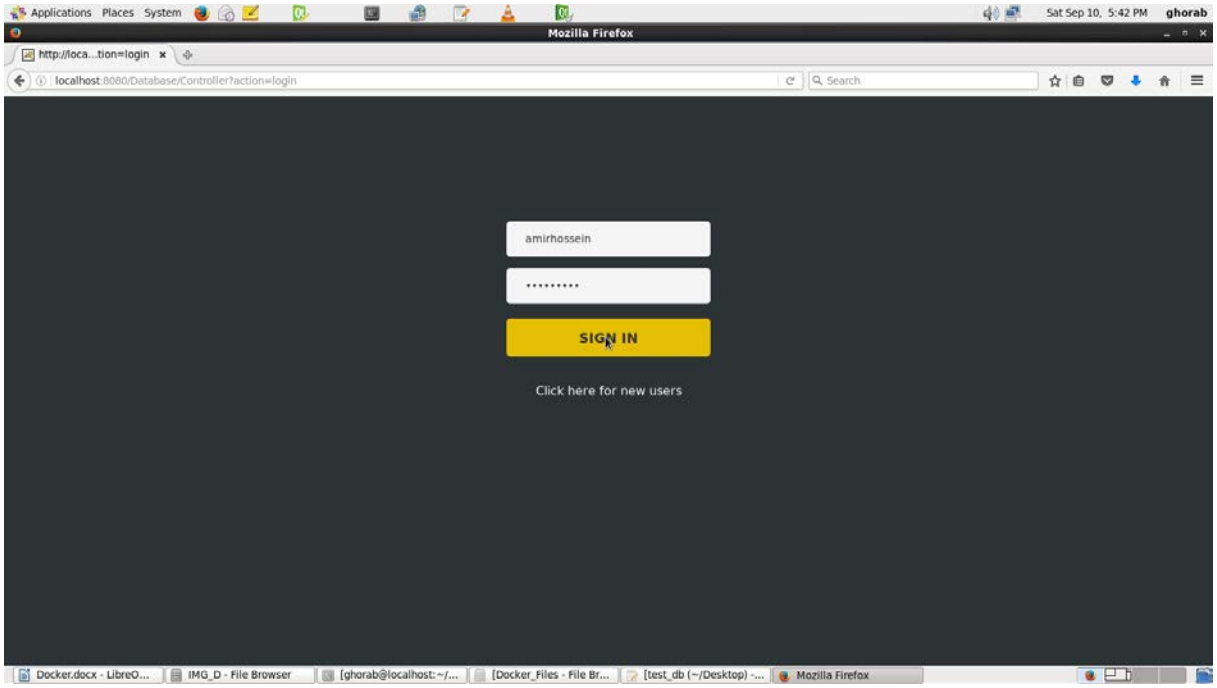


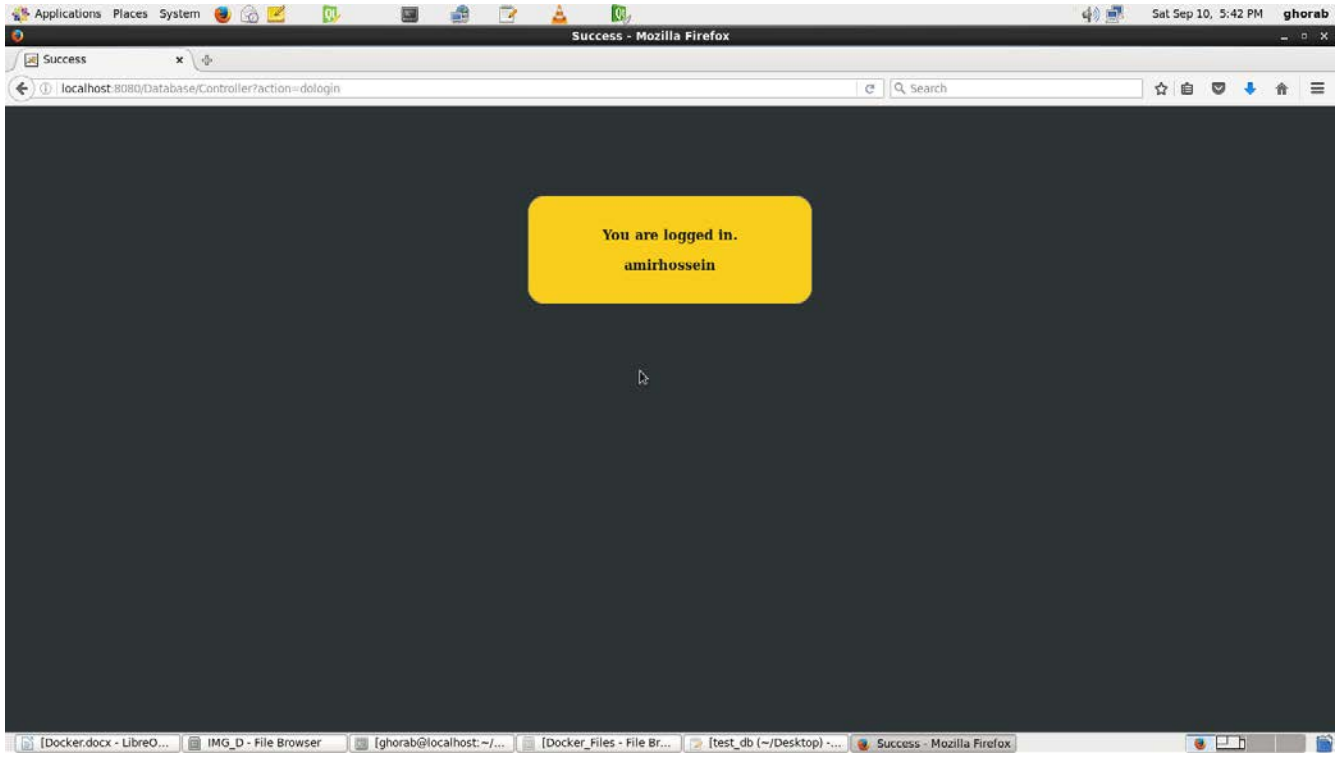
OK lets run our web application...



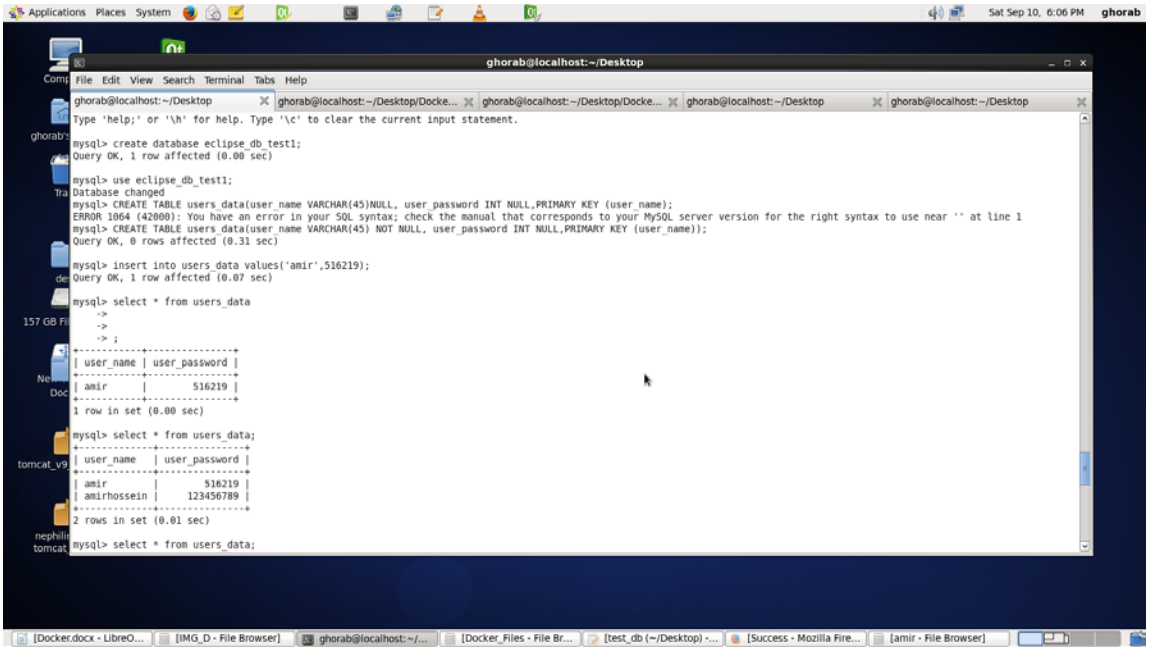
Everything work fine, Let try to login ….



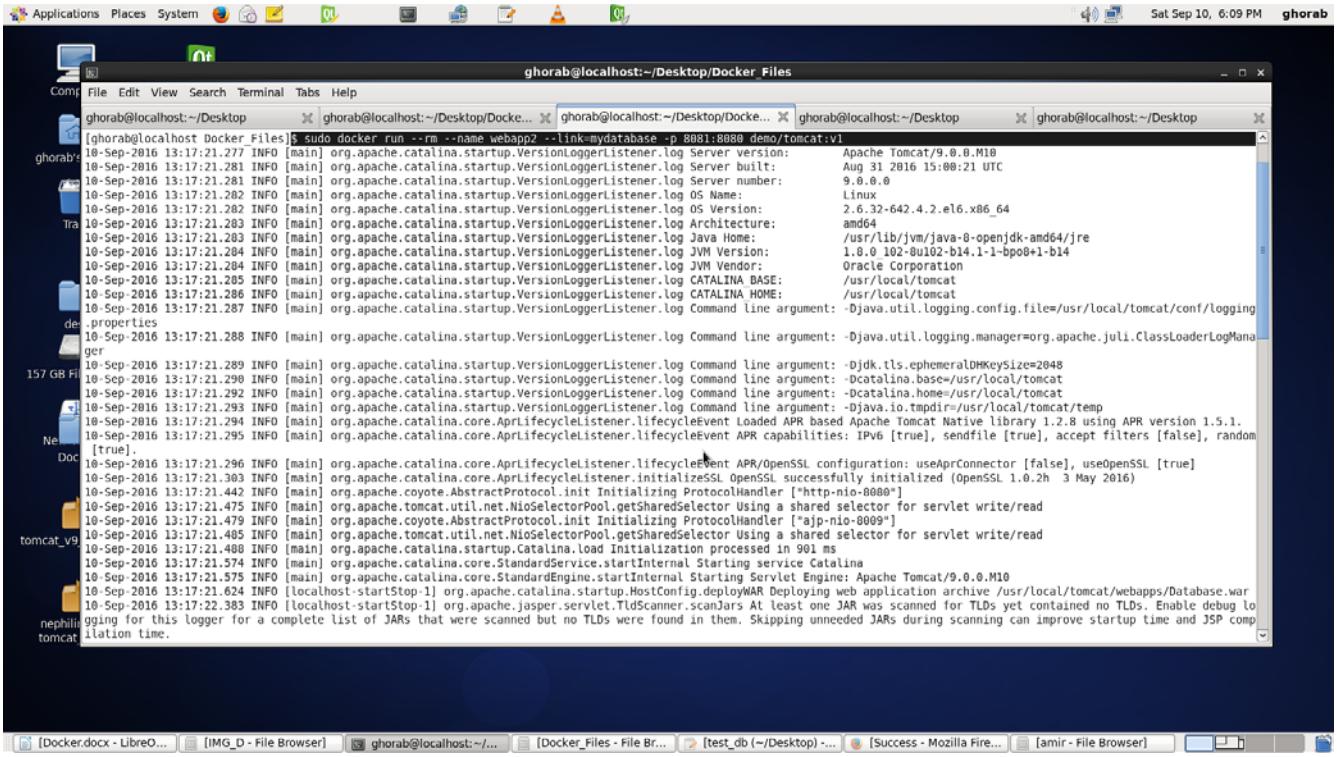




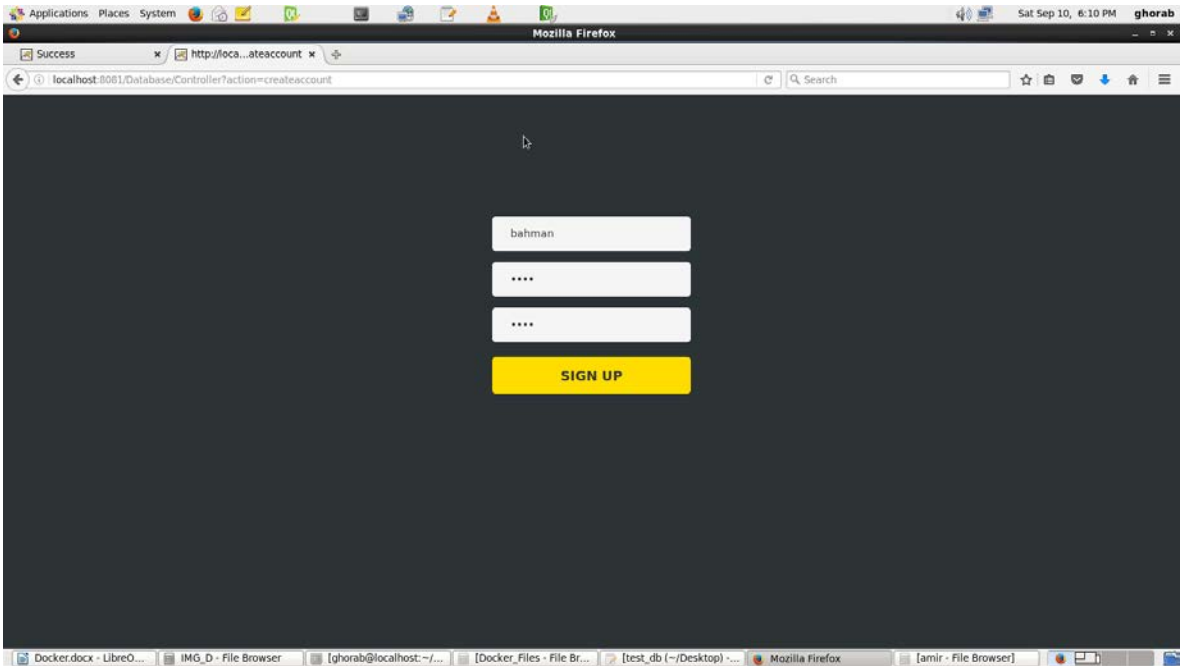
Now Let check database :



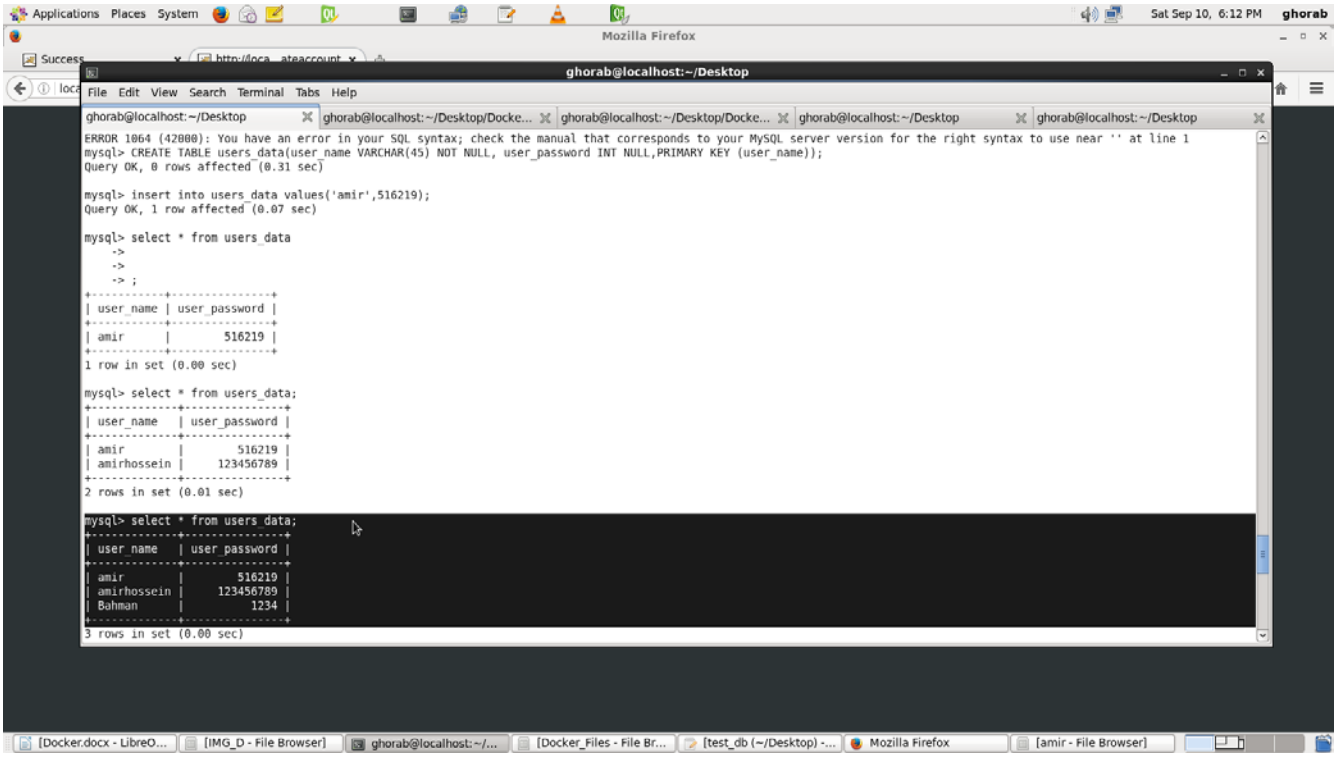
OK now let’s create another container from our web application , I call it webapp2....
(check out the port 8081)
sudo docker run --rm --name webapp2 --link=mydatabase -p 8081:8080 demo/tomcat:v1

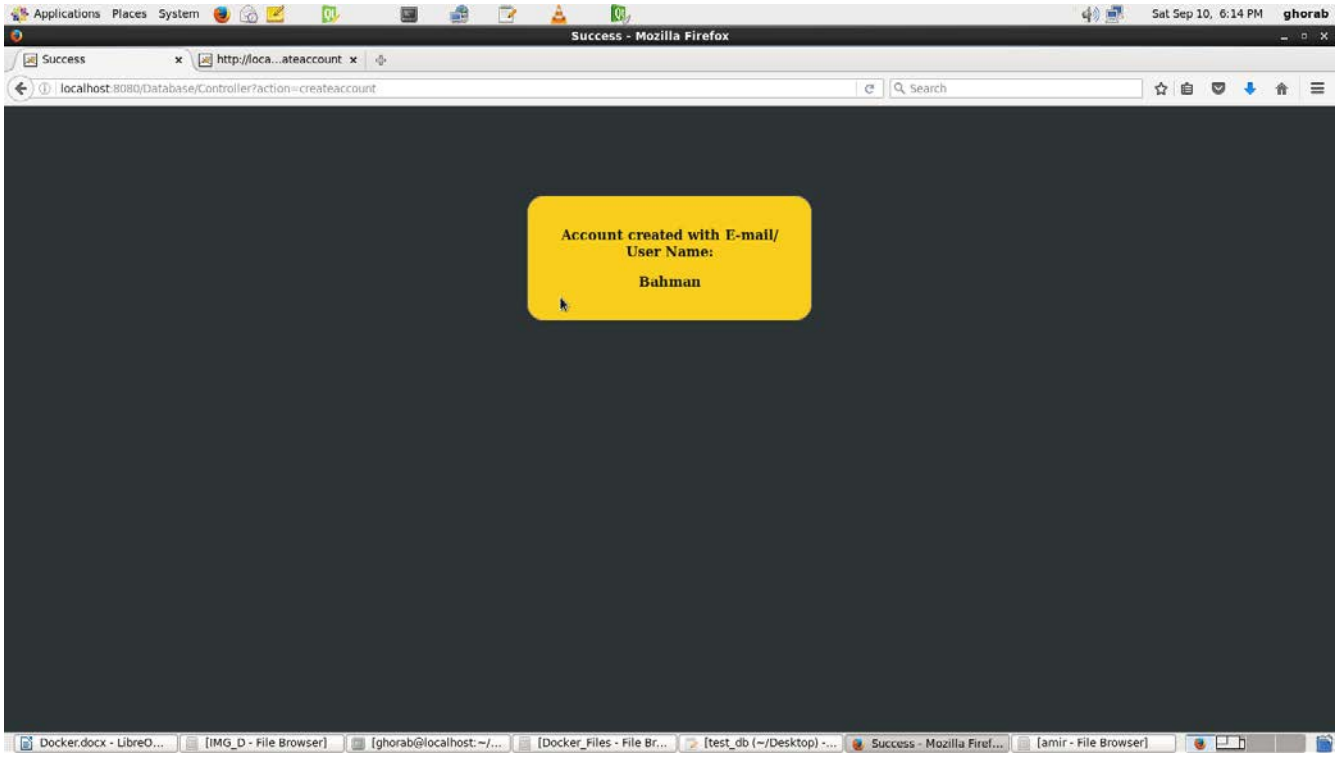


And create new account from it : (check out the URL with port 8081)



And now we have “bahman” user in our database …





and we can login with this user from first web application with port 8080 …..

As you can see all web application containers are connected to database but they are isolated from their selves. You can create more counter from web application image.

This is it for this part of tutorial...

Happy coding ….