**What is Docker?**

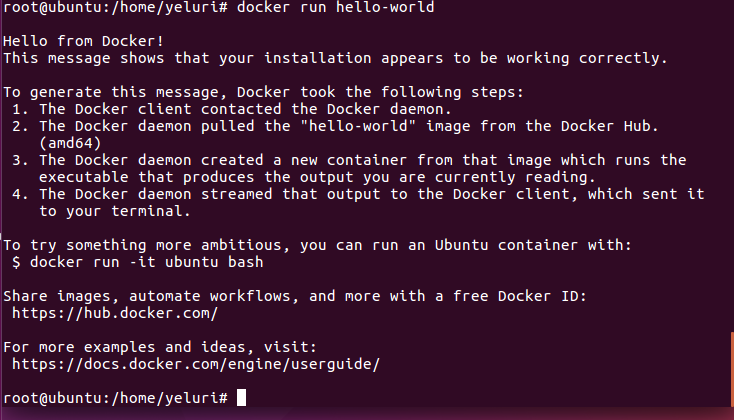
* Docker is a tool meant for using to create, deploy and run the applications using the containers. In the simple meaning you can consider Docker as a Virtual machine which is more convenient than VVM’s and it uses the same Linux Kernel for launching the Applications. It’s an open source. Docker is used by both system admins as well as Developers. So we can consider Docker is meant to a tool in DevOps section. When it comes to the section of DevOps – it’s flexible for Developers, means the developers need not to worry about the application launching and how it can run, as docker is meant to handle this kind of situations for Developers and when we talk about Operations team, it’s even more flexible to build and deploy the so created application.

What are containers?

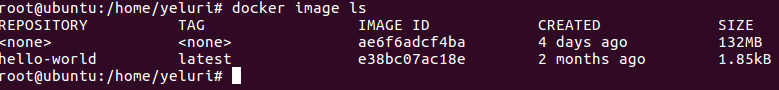
* Container is similar to VM but has got much more flexibility then VM, Container is an image which can accommodate the applications to get developed, executed and deployed once those are build and it can have different env running in one single container for executing the application. Every container is made to run on the Linux kernel and much different than to the VM technology.
* Using the Linux containers for deploying the applications is known as containerization

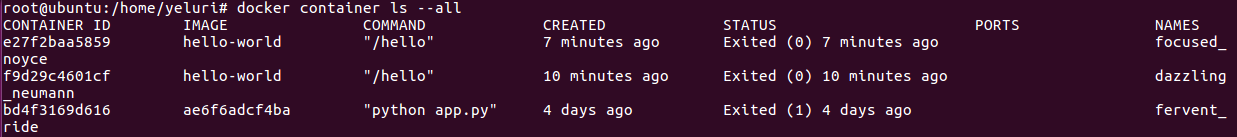
Let’s start the installation and configuration of Docker and its components,

* Check the Docker on the instance/machine –
  + docker –version
* to check for more details of the docker which was installed please,
  + docker info
* **Note: *to overcome permission errors please use SUDO while executing the commands.***
* To test the docker installation use the below mentioned commands,
  + with these command execution we will be creating a simple docker image “hello-world”
    - docker run hello-world



* once its executed we need to check the image and container whether its created are not, this can be tested using the following command
  + docker image ls – this command will showup all the containers and images that were created previously.
  + docker container ls –all – this will return all the created containers along with the imageID’s





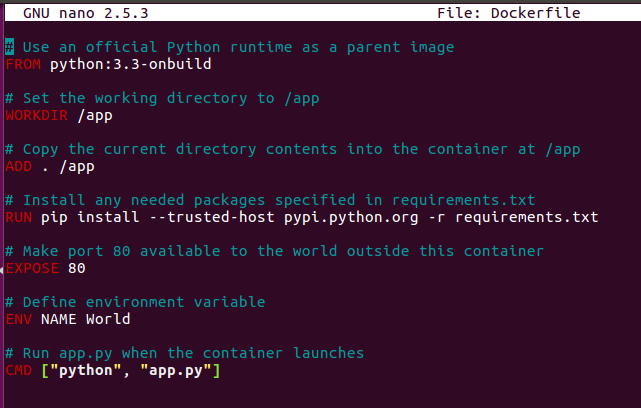
* let’s summarize the concept for this section,
  + Containerization will make the application will not have any system dependencies
  + Containerization will help the CI/CD much easier.
  + docker --help will give use the list of commands and its syntax as well
  + docker
  + docker container –help
  + docker --version
  + docker version
  + docker info
  + docker run hello-world ( executes the docker image )
  + docker image ls – lists the docker images
  + docker container ls – lists the containers
  + docker container ls --all lists all the containers created till now.
  + docker container ls –aq – lists all in quite mode (gives only the container ID’s).

With this we can conclude that we have learnt some basics on docker configuration.

Now let’s begin to develop an app on Docker, so we need to create a container to accommodate the app and on top of this container we need to have a **service** so that the app can be executed along the container. And on top of these all we have the Stack which defines the interaction of the services. We will be discussing in a step by step process in the coming sections.

You can see the structure of Docker Execution mentioned below and we will be discussing all of these in the coming posts.

* + - Stack
    - Service
    - ***Container –*** *now we will discuss this.*
* Now lets implement an app that runs with python as base for coding and this will be implemented on the container. for this we will be using the dockerfile which is meant to define the behavior of the container and its execution process.
  + Containers we can consider as the images were we will be hosting the app for execution and these containers use the Linux Kernel.
  + Dockerfile has got the execution behavior of the container and this also defines the networking interfaces and drives used in the above image on container.
* First lets create a Dockerfile under a directory,
  + mkdir DockerProj – with this a directory named “DockerProj” will be created.
  + cd DockerProj – go into this directory
  + nano Dockerfile – this will create a Dockerfile which will help us to define the execution behavior of the container so copy the below mentioned code into the so called “***Dockerfile”***
* ***Refer Pic1.jpg for code reference.***

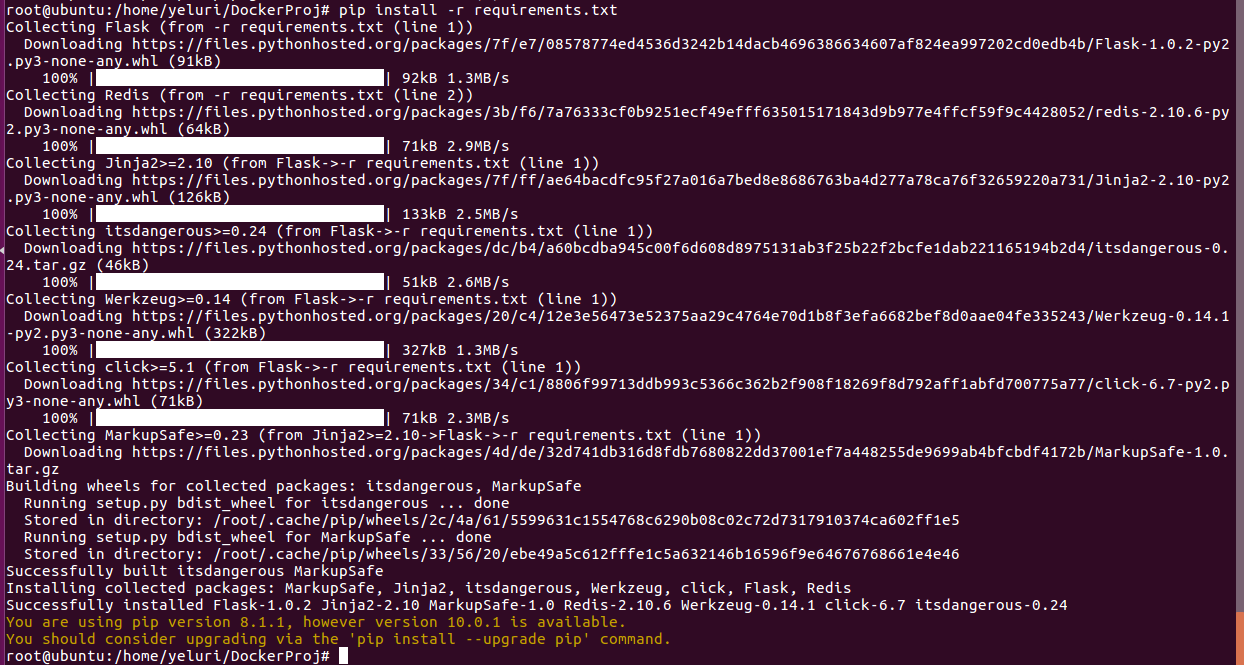


* Here in the above code we are telling the dockerfile to create some files once it gets executed and the names of the files are **“app.py and requirements.txt"**
* Now update the new files with the following code mentioned below
* ***Refer Pic2.jpg & Pic3.jpg for code reference.***

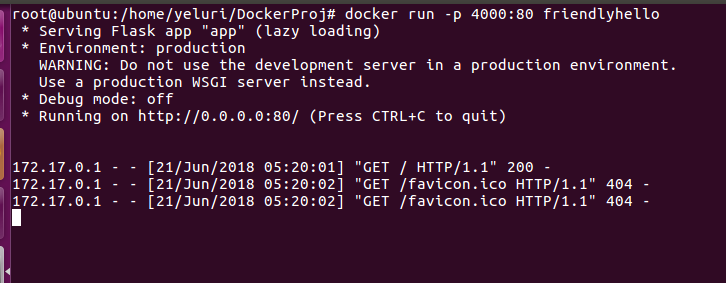


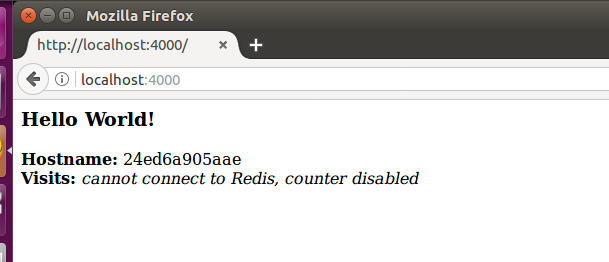


* Try to install the PIP using the command mentioned below,
  + apt-get install python-pip
    - then execute the mentioned code
    - pip install -r requirements.txt -- this will install the “***Flask and Redis”***
* ***Refer Pic4.jpg for code reference.***

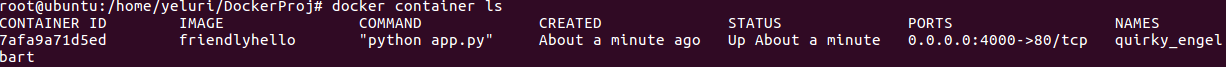


* now we need to build the app so the image is created on the container, so execute the following command to build the app
  + - docker build -t friendlyhello .
  + Once the above command is executed successfully you need to run the below command
    - docker image ls – this will help you to check whether the container image is created or not.
    - now run the app using the piece of code mentioned below
      * docker run -p 4000:80 friendlyhello
      * here we are saying the docker to use 4000 port to run the image with the app named ***“friendlyhello”***
      * try hitting the localhost:4000 to see the result.
* ***Refer Pic5.jpg for code reference.***





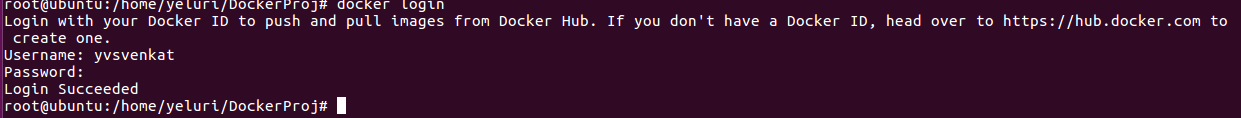
* Once everything got executed successfully, you can stop the container running using the mentioned command
  + docker container ls



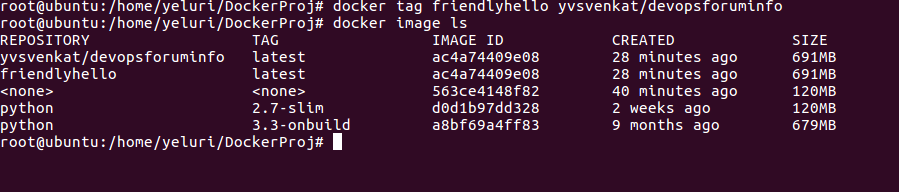
* docker container stop <CONTAINER ID > - this command will stop the container running further.

Now we need to push this image/container into the Docker registry for this we need to login to the Docker Cloud using the username and password which you need to signup in advance so use the below mentioned URLS to do the signup so that you can proceed with the below mentioned **TASK**

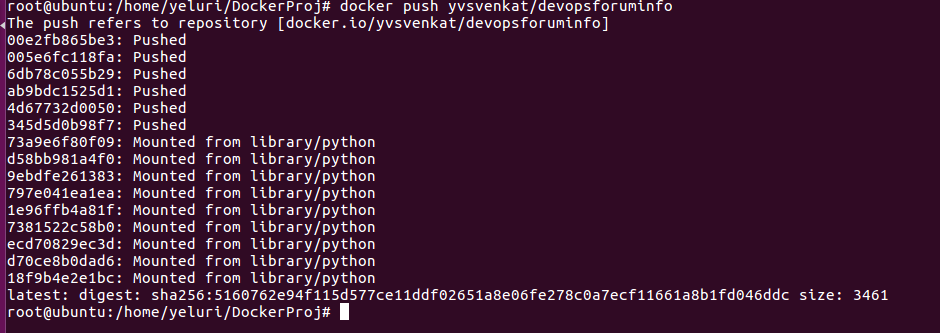
* <https://cloud.docker.com/>
* using the above URL do the signup and make sure you remember/note-down the username and password so that those can be used in our next task that you need to follow the steps mentioned below
* docker login – this will ask for username so give the username which you created at the time of signup and also proceed with password.
* ***Refer Pic6.jpg for code reference.***



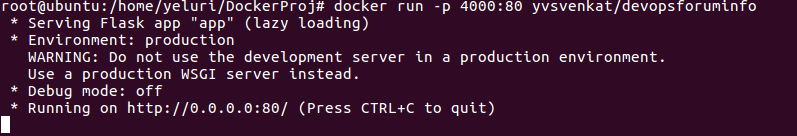
* now execute this command
  + docker tag image username/repository:tag
* where you need to give the
  + image name – friendlyhello
  + username – the one you created at the time of signup
  + repository:tag – devopsforum:info
  + looks like this manner -- docker tag friendlyhello yvsvenkat/devopsforuminfo
* with the execution of the above command you can see the below mentioned screen with the new tag name showing up.
* ***Refer Pic7.jpg for code reference.***



* Now we need to publish the image which is been created above
  + docker push username/repository:tag
* change the details accordingly and execute the command.
* ***Refer Pic8.jpg for code reference.***



* final step to finish the creation of the app and pushing into the docker registry, we need to execute this command so that the image is available locally as well.
* docker run -p 4000:80 username/repository:tag -- frame this command with the appropriate names.



If you run the localhost:4000 it should return the results. And with this we are done with task.

A Quick walk through of this section which we have done till now,

* We created the Dockerfile and the required files for execution of the app on the image.
* we have executed the image
* We have pushed the image to the docker registry for further usage.

List commands which we followed in this above task

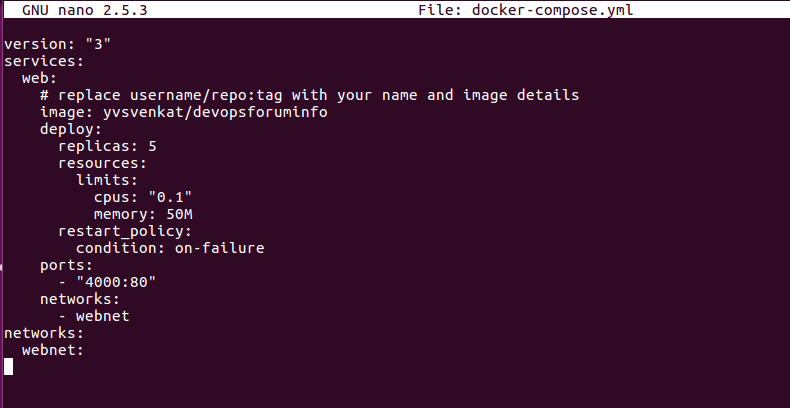
* + docker build -t friendlyhello . # Create image using this directory's Dockerfile
  + docker run -p 4000:80 friendlyhello # Run "friendlyname" mapping port 4000 to 80
  + docker run -d -p 4000:80 friendlyhello # Same thing, but in detached mode
  + docker container ls # List all running containers
  + docker container ls -a # List all containers, even those not running
  + docker container stop <hash> # Gracefully stop the specified container
  + docker container kill <hash> # Force shutdown of the specified container
  + docker container rm <hash> # Remove specified container from this machine
  + docker container rm $(docker container ls -a -q) # Remove all containers
  + docker image ls -a # List all images on this machine
  + docker image rm <image id> # Remove specified image from this machine
  + docker image rm $(docker image ls -a -q) # Remove all images from this machine
  + docker login # Log in this CLI session using your Docker credentials
  + docker tag <image> username/repository:tag # Tag <image> for upload to registry
  + docker push username/repository:tag # Upload tagged image to registry
  + docker run username/repository:tag

=====================================================================================

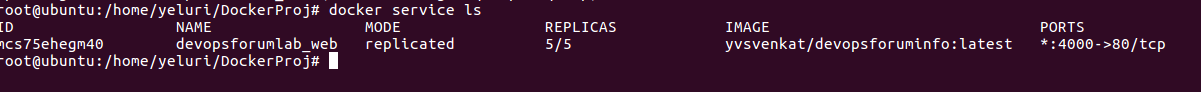
Let’s See the details on the Services in Docker,

Services, in docker we can explain the services as the pieces of the app which are called services. let’s explain this in a simple understandable manner. Take a website which allows users to upload the files and make the end users view the updaloaded one’s, here you can consider video uploader as one service, database uploading as another service and the front end visuals of the uploaded video is another service. These all services execution will make the application.

* To run, define and scale the services with the docker platform we need to define these execution steps is ***“docker-compose.yml”*** file.
  + This YAML file helps to define the docker containers how these needs to behave in production.
* Just update the docker-compose.yml file with below mentioned code,



* The above docker-compose.yml tells the Docker to do the below mentioned:
  + It will pull the image which we pushed into the registry previously.
  + It will allow to run 5 instances with 10% of the CPU usage of each instance and also restrict the instance with 50MB of RAM usage.
  + Also it will map the 4000 on the host with port 80.
* Now let’s start the load-balanced app for this let’s follow the below mentioned steps and commands,
* docker swarm init – this swarm explanation we will see in the coming sections.
* Lets execute the below step so that it will deploy the compose file and lets give the app a name and its named as **”*devopsforumlab”***
  + docker stack deploy -c docker-compose.yml devopsforumlab
* once everything goes fine check whether the 5 instances are created or not using the following command,



* Lets start the tasks for each instance and for this creation let’s use the below command
  + **docker service ps devopsforumlab\_web** -- this command will showup the running instances
* to check on the browser you can execute this command
  + curl -4 <http://localhost> -- will give the html result.
* Execute the commands and observe the changes from the previous
  + docker stack deploy -c docker-compose.yml deveopsforumlab
  + docker container ls –q
* to take down the app and the swarm you can execute these steps mentioned below, these will remove the app and stop the swarm.
  + docker stack rm devopsforumlab
  + docker swarm leave –force

A glossary of the commands which we executed in this section are drafted below

* docker stack ls # List stacks or apps
* docker stack deploy -c <composefile> <appname> # Run the specified Compose file
* docker service ls # List running services associated with an app
* docker service ps <service> # List tasks associated with an app
* docker inspect <task or container> # Inspect task or container
* docker container ls -q # List container IDs
* docker stack rm <appname> # Tear down an application
* docker swarm leave --force # Take down a single node swarm from the manager

Let’s continue the task and now we will discuss about ***swarm*** – you can consider swarm as a group of machines which are responsible to run Docker and these are joined in a cluster. These machines are executed on a cluster using **Swarm Manager.** Oncetheseswarms are joined then these are called as nodes. Now we will see how to switch into swarm mode and this will convert the current machine as Swarm Manager. A swarm has got multiple nodes, which we can consider as physical or VM’s.

Let’s begin the task,

* to initialize the swarm we need to use the command,
  + docker swarm init – this enable the swarm mode.
  + docker swarm join – this allows the other machine to join the swarm as workers.

Docker commands

* to remove the images
  + docker rmi --force $(docker images -a -q)
* to view the images list
  + docker image ls