Lab 5 - Docker

# Week 4 Tasks: (10 points each)

# Installing Docker Engine on AWS EC2 instance

* 1. Screenshot of running docker hello-world

# Docker images and Dockerfiles

* 1. Screenshot of C Program successfully run inside container.

# Exposing ports,docker networks

* 1. Sample.html showing the web page on the browser. Screenshot should show public dns of the EC2 host of docker.
  2. Screenshot of docker container running nginx (terminal of EC2 host)
  3. Screenshot of python application successfully writing and reading from the MongoDB database
  4. Screenshot showing mongodb being run within network(docker command has to be clearly highlighted)
  5. Screenshot showing python file being run within network and successfully writing and reading from MongoDB(docker command has to be clearly highlighted)

# Docker compose

* 1. Screenshot of python-mongodb application running as a docker compose application(logs of the application)
  2. Screenshot of 3 python application writes and reads from MongoDB after scaling the python application.

# Docker volumes (This task is optional, No bonus marks will be given for submitting this screenshot)

* 1. Screenshot clearly showing command run to mount the host volume and manually running the python application inside the container

### Few key points to note:

1. All required files **except** Dockerfile(s) have been given in the drive folder(task wise) where this manual is uploaded.
2. It is very important to go through all reference material attached in this manual , as this will help you understand and debug the lab tasks.
3. Most of this lab focuses on 2 aspects building the right Dockerfile and running the right command.
4. Apart from the attached resources, you can always refer to the official docker documentation. The docker documentation is well maintained and should help you through all your tasks.<https://docs.docker.com/>
5. Additional resources have been given at the end of the manual.
6. When you run docker commands in the foreground, you cannot access the command prompt in which case press  **[Ctrl+C]** and continue with the next step or run docker in the background mode by using –d option as explained in the demo video. <https://www.tecmint.com/run-docker-container-in-background-detached-mode/>

# Understanding containers and Docker

Please watch the below youtube tutorial **mandatorily** before starting the lab, this will help kickstart your understanding of Docker and the commands.

[Docker Basic Commands | Docker Commands with Examples | Docker Commands Tutorial | Intellipaat](https://www.youtube.com/watch?v=nXV6qihj5uw)

Containers are the de-facto standard for creating,maintaining and deploying microservices. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Container engines are the technology/software that are used to run the containers. There are many [container engines](https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction/) available, but the most popular and easy to use CE is *Docker.*

The following links will help you get started on why we need containers, how they are different from VMs and why Docker:

[What is a Container? | App Containerization](https://www.docker.com/resources/what-container) - **Must Read!**

[What is Docker?](https://opensource.com/resources/what-docker)

# Task 1: Installing Docker Engine on EC2

**Sub-tasks are:**

1. Create an EC2 instance with the following specs:

|  |  |
| --- | --- |
| **AMI** | Ubuntu Server 20.04 LTS (HVM), SSD Volume Type |
| **Instance Type** | t2.medium |
| **Instance Details** | Default values |
| **Storage** | Default values |

1. Get keypair(pem) file and SSH into the instance.
2. Install docker engine on the instance , instructions are in the given link:
   1. [Install Docker Engine on Ubuntu](https://docs.docker.com/engine/install/ubuntu/)
3. Use the following link to make docker a “sudo-less” command:
   1. [Post-installation steps for Linux](https://docs.docker.com/engine/install/linux-postinstall/)
4. Verify that Docker Engine is installed correctly by running : docker run hello-world

# Task 2: Docker images and docker files

In AWS EC2 instances we saw *AMI(Amazon Machine Images)*, which was basically the operating system and in case of snapshots, the current *state* of the VM. Docker images are similar, but are not the same. Docker images are “ready to go” meaning everything that needs to be installed(requirements) has already been taken care of when building the image and they don't need to be installed/taken care of again.

Docker files are used to create Docker images. Dockerfile are a series of steps that specify which base image to use, which files/folders to copy into the container, run which commands while starting up the container and which will be the main process of the container. We need to create dockerfiles, then build it into an actual runnable docker image.

**Sub tasks:**

1. Docker hub is a central public repository containing Docker images and documentation on how to use these base images. Explore the different images at : <https://hub.docker.com/>
2. The images in docker hub, need to be *pulled* into our EC2 instance in order to use them(Pulling is the equivalent of downloading the images onto you docker host).
   1. [docker pull](https://docs.docker.com/engine/reference/commandline/pull/)

Pull the following images onto your docker instance:

* Ubuntu 18.04
* Nginx
* Python
* MongoDB

Stick to the latest images. To find the images that exist on your instance you can run the [docker images](https://docs.docker.com/engine/reference/commandline/images/) command. This shows the image name, when you pulled the image, the tags(versions) of the images.

1. Now that we have the images we need to actually run the containers,
   1. We can run a container using the [docker run](https://docs.docker.com/engine/reference/commandline/run/) command.
   2. To list the *currently running* container we use the [docker ps](https://docs.docker.com/engine/reference/commandline/ps/) command
   3. To stop a running container safely we use [docker stop](https://docs.docker.com/engine/reference/commandline/stop/)
   4. Usually after a container is stopped it goes into a “exited state”, this can block some I/O operations needed by future containers and you will not be able to use the name of the exited container. [What are the possible states for a docker container?](https://stackoverflow.com/questions/32427684/what-are-the-possible-states-for-a-docker-container)

To safely and cleanly remove a container, use the [docker rm](https://docs.docker.com/engine/reference/commandline/rm/) command.

1. *Containers are light weight VMs*, this means we should be able to somehow interact with a running container, like a terminal.
   1. Start an interactive bash(shell) session with an ubuntu:18.04 as a base image. Explore around the container using the shell. **What is different and what is same as an ubuntu VM?**
      1. [Docker 'run' command to start an interactive BaSH session](https://gist.github.com/mitchwongho/11266726)
   2. [Get a shell into an already running container](https://docs.docker.com/engine/reference/commandline/exec/#run-docker-exec-on-a-running-container)
2. Let us now create our own Docker image using Ubuntu:18.04 as a base image. The image you will create will run a simple C program, after installing the GCC compiler. The Dockerfile must:
   1. Specify the base image as **Ubuntu:18.04**
   2. Copy the program.c file from your instance to the docker image.
   3. Update the “apt” repository and install the GCC compiler
   4. Compiler the C program.
   5. Run the ./a.out *command*

*Hint: What should the name of the dockerfile you are creating?*

Use the following C program as a base, only modify your SRN:

#include<stdio.h>

int main()

{

printf("Running this inside a container !\n");

printf("My SRN is <YOUR SRN HERE>\n");

}

1. After you have your Dockerfile, build the image using [docker build](https://docs.docker.com/engine/reference/commandline/build/) and run the container.

References to help you get started:

* [How do you write a Dockerfile?](https://www.educative.io/edpresso/how-do-you-write-a-dockerfile)
* [Docker Build: A Beginner's Guide to Building Docker Images](https://stackify.com/docker-build-a-beginners-guide-to-building-docker-images/)
* [How to Create a Docker Image From a Container](https://www.scalyr.com/blog/create-docker-image/)

# 

# Task 3: Exposing ports,docker networks

Containers are also used to run web applications , they can be web servers such as Apache or Nginx, or web applications using REST APIs or any other web application. Similar to *Security Groups* for EC2 instances, we need to *expose* the right ports to access the container’s web apps.

1. Download the sample HTML file from the Lab 5 drive folder , and modify your SRN.
2. Create a Dockerfile and then a docker image having an **nginx:latest** base image, and copying the html file into the default folder in the container. Build the docker image and **tag/name it your SRN**.
3. Run the docker container using the previously created docker image. Expose the HTTP port and check connectivity.
   1. [Publishing/Exporting ports in a docker container](https://docs.docker.com/engine/reference/commandline/run/#publish-or-expose-port--p---expose)
   2. [Docker Container Tutorial #8 Exposing container ports](https://www.youtube.com/watch?v=G36I1iqDZig)

*Hint: You are running a container on an EC2 instance. After exposing the port on the container , you might be able to access the nginx web server from within the EC2 instance, but not from the internet(your system). What needs to be configured for the AWS EC2 instance running your container?*

In lab 3 , we saw VPC networks and how we can create a virtual network connectivity between virtual machines(EC2 instances). Similar networks need to be created for containers. We will explore connectivity without docker networks and see how docker networks make it much easier to connect within containers. To demonstrate this we will create a simple application using a python client and a MongoDB NoSQL server.

Note: You don't need to know how to use mongodb, code to use mongodb has been provided to you.

1. Run the mongodb container in a detached mode, exposing the default port(27017) of mongodb.
   1. Use the **mongo:latest** image.
   2. Name the container as **mongodb** while running the container. [Naming a container](https://docs.docker.com/engine/reference/run/#container-identification)
   3. [Docker detached mode](https://www.freecodecamp.org/news/docker-detached-mode-explained/#:~:text=Detached%20mode%2C%20shown%20by%20the,receive%20input%20or%20display%20output.&text=If%20you%20run%20containers%20in,to%20its%20input%20and%20output)
2. Download sample.py from the drive folder. Modify the SRN wherever mentioned.
3. The MongoDB container is running, but we need to find out the *IP address* of the mongodb container. Find this IP using the [docker inspect](https://docs.docker.com/engine/reference/commandline/inspect/) command. Modify this IP address in the sample.py file.
   1. [Get a containers IP address](https://docs.docker.com/engine/reference/commandline/inspect/#get-an-instances-ip-address)
4. Create a Dockerfile, which:
   1. Uses **python** base image
   2. Updates the apt-repository
   3. Installs **pymongo** using pip ([pymongo 3.11.2](https://pypi.org/project/pymongo/) ).
   4. Copies the sample.py from the instance to the container.
   5. Runs the python *command* , to run the python file.
5. Build the docker image using the above Dockerfile and run the container.
6. You should see that the data was correctly inserted and fetched from the database container.

The above tasks showed the connectivity between 2 containers *without* a network. This can cause problems as every time a container is created it *could possibly* have a different IP address. This is the issue [*docker networks*](https://docs.docker.com/network/)tries to solve.

[Docker Networking Options](https://www.youtube.com/watch?v=Yr6-2ddhLVo)

[Docker Networking | Docker bridge network deep dive | Container bridge drive](https://www.youtube.com/watch?v=Tx12haz-4VA)

1. Create a docker bridge network , called **my-bridge-network.**
2. Stop the mongodb container you had created before and delete(rm) it. Run a mongodb container again, but now with the following parameters;
   1. network : **my-bridge-network**
   2. name: **mongodb**
   3. Exposed ports: **27017**
   4. Image: **mongo:latest**
3. Go back to sample.py , comment line 3 and uncomment line 4. We are now going to use the name of the database containers as the host name, leaving the ip address resolution to docker.
4. Build the python app docker image again as you did previously and run the container using the built image. You should see that insertion and retrieval have been done successfully.

# Task 4: Docker compose

Till now, we have been using docker commands to start and stop containers,create docker networks etc. But this becomes challenging in a real environment when a single application may have 100s of containers(microservices) that need to talk to each other within a single network. While docker networks solve this problem partially, it still doesn't solve the issue of having to run multi-container applications and to automatically bootstrap these containers into the same application, this is where *Docker Compose* comes into the picture.

Docker compose helps bootstrap complex multi container applications, helps maintain the dependencies between them , create a network among these containers and even help scale the entire application entirely or particular containers within the application.

[Overview of Docker Compose](https://docs.docker.com/compose/)

[“What, Why, How” Docker Compose?](https://medium.com/faun/what-why-how-docker-compose-482e8aa021b)

We will use the same python-mongodb application to use docker compose and even scale containers within the application automatically. Docker compose uses [YAML Files](https://blog.stackpath.com/yaml/#:~:text=Definition,used%20to%20write%20configuration%20files) to specify the different *pieces(containers)* of the application, along with the required configurations. YAML files are EXTREMELY popular when it comes to bootstrapping complex applications not only in Docker!

[Docker Compose in 12 Minutes](https://www.youtube.com/watch?v=Qw9zlE3t8Ko)

1. For the purposes of this lab make sure all the following files are in the same directory
   1. docker-compose.yml
   2. Dockerfile
   3. sample.py
2. Install docker compose using the following link:
   1. [Install compose on linux systems](https://docs.docker.com/compose/install/#install-compose-on-linux-systems)
3. Go to the docker-compose.yml file and try to understand the syntax and what each line does.
4. Within the same directory, run: **docker-compose up**.
   1. [Docker Compose command-line reference](https://docs.docker.com/compose/reference/)

What you see is that Docker compose has built your python application, started the mongodb server , created links internally between the containers (network) and started both the containers together as a *unified application*.

The python container exits, since its done with its utility of writing and reading to the database.

1. Now we will scale the python application, so that we have 3 containers of the python application, but keep only one container of the mongodb.
   1. [docker-compose scale](https://docs.docker.com/compose/reference/scale/) - Use this link as a reference and scale **only the python application. Use the correct “service” to scale**

# Task 5: Docker volumes This task is for you to practice, No bonus marks will be given for submitting this screenshot

Docker volumes are an important and crucial concept. Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. Volumes are either created dynamically by Docker or you can mount the host directory. In this task,we will focus on mounting a host volume.

[Docker bind mount | Sharing data between host and Container](https://www.youtube.com/watch?v=1lsDvQfZYDs)

1. Create a new directory on the EC2 host , with the name as your SRN.
2. Add the following folders inside it:
   1. sample.py ( A simple python application which reads from a text file called details.txt and prints them line by line)
   2. A text file called details.txt containing your name,srn,section,semester.
3. We will be using **Ubuntu:18.04** as a base image.
4. For the ubuntu container, mount the host volume you have created and create a bash shell into the container.
   1. [Start a container with a bind mount](https://docs.docker.com/storage/bind-mounts/#start-a-container-with-a-bind-mount)
5. Inside the container run the python program to display contents of the list.

### Additional Resources/Common bugs you might encounter:

1. [What is Docker?](https://www.youtube.com/watch?v=u-YWtdbpEhQ)
2. [docker command not found even though installed with apt-get](https://stackoverflow.com/questions/30379381/docker-command-not-found-even-though-installed-with-apt-get)
3. [docker CLI & Dockerfile Cheat Sheet](https://design.jboss.org/redhatdeveloper/marketing/docker_cheatsheet/cheatsheet/images/docker_cheatsheet_r3v2.pdf)
4. [docker cheat sheet](https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf)
5. Not able to reach the container(for web apps)? Think about EC2 security groups.
6. [9 Common Dockerfile Mistakes - Runnablog](https://runnable.com/blog/9-common-dockerfile-mistakes)
7. [How to debug and fix common docker issues](https://www.digitalocean.com/community/tutorials/how-to-debug-and-fix-common-docker-issues#:~:text=Syntax%20errors%20and%20caching%20problems,running%20containers%20from%20those%20images).
8. Not able to build/run docker containers due to insufficient space on EC2: [How To Remove Docker Images, Containers, and Volumes](https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes)
9. [Docker - Container is not running](https://stackoverflow.com/questions/29599632/docker-container-is-not-running)
10. [exited with code 0 docker](https://stackoverflow.com/questions/44884719/exited-with-code-0-docker)
11. Docker container has a name conflict? [Remove all stopped containers](https://docs.docker.com/engine/reference/commandline/rm/#remove-all-stopped-containers)
12. Curious about MongoDB and what it is ?
    1. [What is NoSQL? NoSQL Databases Explained](https://www.mongodb.com/nosql-explained)
    2. [What Is MongoDB?](https://www.mongodb.com/what-is-mongodb)
    3. [Tutorial — PyMongo 3.11.2 documentation](https://pymongo.readthedocs.io/en/stable/tutorial.html)