**Docker Assignment 4**

1. What are features possible only under Docker Enterprise Edition compared to the Docker community edition?

* One of the main difference is the operating systems on which we can run docker.
* Docker CE can run on Windows 10 and Mac, on Azure and AWS, as well as CentOS, Debian, Fedora, and Ubuntu
* Docker EE can run on **Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), Oracle Linux**, Ubuntu, Windows Server 2016, as well as Azure and AWS.
* one more differences are that Docker CE is a free to use open source and, while to use **Docker EE we have pay or subscribe to plans** which they offer with some dollars/Rs
* Docker Enterprise 3.0 consists of two separately purchased and licensed products:
* Universal Control Plane (UCP) and Docker Trusted Registry (DTR) with Docker Engine – Enterprise: Installed on servers and licensed based on the size of the environment.
* Docker Desktop Enterprise: Installed on developer workstations and separately licensed addition to the platform.
* **Docker EE comes with additional features that can help enterprises launch, manage, and secure their containers more efficiently. Docker Enterprise platform features below**
* Docker Kubernetes Service
* Docker Swarm Service
* Centralized cluster management
* Deploy, manage, and monitor
* Built-in security and access control
* Below are the featured that user/companies can do when using Docker Enterprise Edition:
* Gain access to certified Docker images and plugins
* View your container clusters in a single pane view
* Access controls for cluster and image management
* Receive official same-day support from Docker
* Run vulnerability scans on your Docker images
* Run Docker EE engine with FIPS 140-2 certification
* Advanced image and container management, LDAP/AD user integration, and role-based access control which is now part of the Docker EE plan
* Continuous vulnerability monitoring and Docker Security Scanning which is now part of the Docker EE plan

1. How to create a user-defined bridge network?

* A bridge network is a Link Layer device which forwards traffic between network segments.
* A bridge can be a hardware device or a software device running within a host machine kernel
* Containers on the default bridge network can only communicate each other by IP addresses, unless you use the --link option
* While on a user-defined bridge network, containers can resolve each other by name or alias. you can image like DNS
* To create a user defined bridge network below are the steps
* As a pre-req if containers are already currently connected to the network then please disconnect them first using command **docker network rm network\_name**
* **command to create a user-defined bridge network** is   
  **syntax - docker network create *network\_name***

**command - docker network create *user\_network***

* Now create a container let's say test\_nginx(you can give any)and mention the network name for eg.
* **docker create --name *test\_nginx* --network *user\_network* --publish 8080:80 *nginx:latest***
* To connect to a running container to an existing user-defined bridge, use the docker network connect command. The following command connects an already running *test-nginx* container to an already-existing *test-net* network:
* **docker network connect *user\_network test\_nginx***
* To disconnect from user defined bridge run below command
* **docker network disconnect *test-network test-nginx***
* To remove from network - **docker network rm network\_name**

1. Does Docker support IPv6?

* yes it supports but for that we need to enable IPv6 support in the Docker daemon
* IPv6 networking is only supported on Docker daemons running on Linux hosts.

1. How to deploy your images with azure container instances?

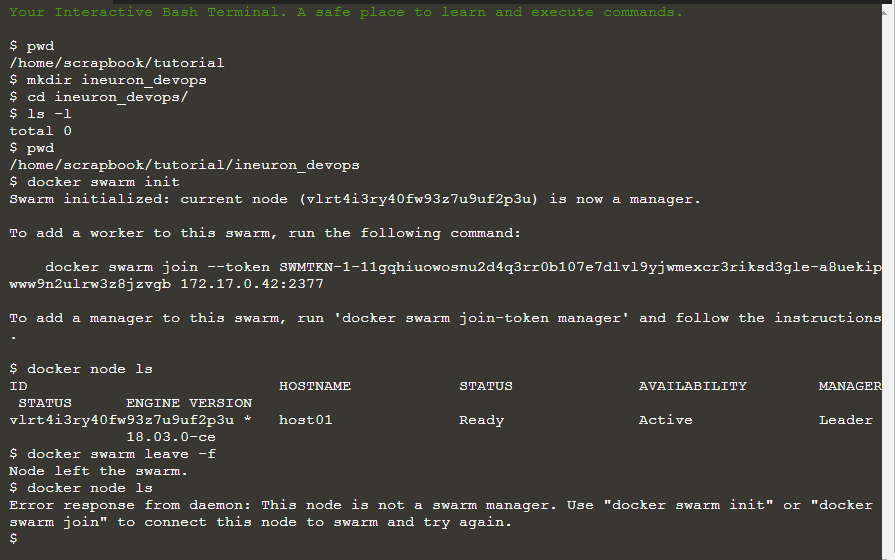
* Build the container image
* let us use eg. as node.js application
* **step 1 create a dockerfile as** :

|  |
| --- |
| FROM node:8.9.3-alpine  RUN mkdir -p /usr/src/app  COPY ./app/\* /usr/src/app/  WORKDIR /usr/src/app  RUN npm install  CMD node /usr/src/app/index.js |

* **step 2: build with a tag as aci-tutorial-app:**
* docker build ./aci-helloworld -t aci-tutorial-app
* **step 3 check the docker images with command**
* docker images
* **step 4 Run the container locally and also map to a port number**
* docker run -d -p 8080:80 aci-tutorial-app
* **step 5 In browser check the app is running or not locally with url - http://localhost:8080**
* **Step 6 Next push the image to azure container registry**
* Before you create your container registry, we need a resource group to deploy it to.
* A resource group is a logical collection into which all Azure resources are deployed and managed.
* az group create --name myResourceGroup --location eastus
* Replace <acrName> with a unique name for your registry:
* az acr create --resource-group myResourceGroup --name <acrName> --sku Basic
* **step 7 Logint to container registry**
* az acr login --name <acrName>
* **step 8 Tag container image:**
* az acr show --name <acrName> --query loginServer --output table
* **step 9: check with - docker images**
* **step 10** Tag the aci-tutorial-app image with the login server of your container registry. Also, add the :v1 tag to the end of the image name to indicate the image version number.
* docker tag aci-tutorial-app <acrLoginServer>/aci-tutorial-app:v1
* **step** 11 : Push image to Azure Container Registry
* docker push <acrLoginServer>/aci-tutorial-app:v1
* **step 12 List images in Azure Container Registry**
* az acr repository list --name <acrName> --output table
* **step 13 - Login to container registry login server**
* az acr show --name <acrName> --query loginServer
* **step 14 - to deploy container**
* az container create --resource-group myResourceGroup --name aci-tutorial-app --image <acrLoginServer>/aci-tutorial-app:v1 --cpu 1 --memory 1 --registry-login-server <acrLoginServer> --registry-username <service-principal-ID> --registry-password <service-principal-password> --dns-name-label <aciDnsLabel> --ports 80

1. What is the docker swarm?

* Let us see first see container orchestration this will basically give us an idea of what docker swarm is
* orchestration tool is to help us deploy our application on a large number of containers
* orchestration tool will help us to launch all of those containers in one go all we have to define is what do we want in those containers what kind of image we need to be creating
* Docker swarm is a container orchestration tool part of the docker engine.
* Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host.
* With it developers and IT administrators can deploy and manage cluster of docker nodes as a single virtual system
* docker swarm you can use it to deploy and manage a cluster of docker nodes now these docker nodes will basically be docker containers these nodes will be the servers of their own and you can manage all of these things together as a single virtual System
* Architecture would be like something like it will have a **single manager node** and you have different **multiple worker nodes** which are connected to the manager node and each of these will have a particular task to perform. worker nodes will have tasks like you know implementing in new features or something and the manager node will have the special task of managing all of these worker nodes and this manager will be handled by services which in turn is managed by a docker engine
* Few of the Feature are as below
* Decentralized design
* Declarative service model
* Scaling
* Desired state reconciliation
* Multi-host networking
* Service discovery
* Load balancing
* Secure by default
* Rolling updates

A Small poc as in below snap  
  


1. What is a memory-swap flag?

* Memory-swap is a modifier flag that only has meaning if- memory is also set. Swap allows the container to write express memory requirements to disk when the container has exhausted all the RAM which is available to it.

1. Can you explain the different volume mount types available in Docker?

* **Bind mounts**: These can be stored anywhere on the host system
* **Volume mount:** they are managed by Docker and are stored in a part of the host filesystem.
* **tmpfs mount:** they are stored in the host system's memory. These mounts can never be written to the host's filesystem.
* suppose assume a docker container is destroyed then you will lose all your data, to avoid losing data we can use **volume** and **bind mounts**
* **For bind mounts the file or directory does not need to be present, if that does not exist it will create on demand and it can save the data on actual machine/OS**
* Volumes are used for adding a data persisting layer in your Docker containers within the docker itself for a situation where you need to persist data after shutting down your containers.
* It is independent of both your directory structure and the OS of the host machine.
* When you use a volume, a new directory is created within Docker’s storage directory on the host machine, and Docker manages that directory’s contents. Let see below eg.
* Let us take a real time example to understand and use this feature in docker
* Suppose you are using mysql as an application and want to use that in your docker container then the mysql data inside container would be store only in that containers FS
* eg. now you want to mount the disk or keep data external for eg. you have mysql as app inside container that will keep all data in containers path only like /var/lib/mysql so you want to keep them in external outside of container or host then can do that

**syntax:**

* **docker run -v <path of external host> :<path of/inside container> app\_name/image\_name**
* **docker run -v /home/ubuntu/data:/var/lib/mysql mysql**
* just fyi - tmpfs mounts are stored in the host system’s memory only, and are never written to the host system’s filesystem.

1. How to share data among Docker hosts?

* We can share the data among docker container and host by using bind mount let's take a below example with command
* The following command will create a directory called nginxlogs in your current user’s home directory and bindmount it to /var/log/nginx in the container:

**docker run --name=nginx -d -v ~/nginxlogs:/var/log/nginx -p 5000:80 nginx**

* We now have a copy of Nginx running inside a Docker container on our machine, and our host machine’s port 5000 maps directly to that copy of Nginx’s port 80.
* this will allow to access the data outside of container to the host on a different port.
* In you OS system try to open browser and see **http://your\_server\_ip:5000**

1. How to backup, restore, or Migrate data volumes under docker containers?

* In docker Volumes are useful for backups, restores, and migrations.

let us see first few flags like --volumes-from flag to create a new container that mounts that volume.

* Step 1 # Let us create a new container named raw\_data

**docker run -v /dbdata --name raw\_data ubuntu /bin/bash**

* Step 2 # take a backup

Now Launch a new container and mount the volume from the raw\_data container

Now next Mount a local host directory let say as /backup

Pass a command that tars the contents of the dbdata volume to a backup.tar file inside our /backup directory.

**docker run --rm --volumes-from raw\_data -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata**

* This is the point where we will have a backup of our dbdata volume.

What next jsut simply restore them :)

* Step 3 # Restore volume from backup

Now the backup which was created can be easily restored either into another container running on same machine or may be a new machine where container runs on it.

For simplicity let's create a new container named raw\_data2:

**docker run -v /dbdata --name raw\_data2 ubuntu /bin/bash**

Then un-tar the backup file in the new container`s data volume:

**docker run --rm --volumes-from raw\_data2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"**

1. Why does the docker service take 10 seconds to recreate or stop?

* whenever user issue a Compose stop then it attempts to stop a container by sending a SIGTERM.
* It then waits for a default timeout of 10 seconds.
* After the timeout, a SIGKILL is sent to the container to forcefully kill it.
* If you are waiting for this timeout, it means that your containers are not shutting down when they receive the SIGTERM signal.