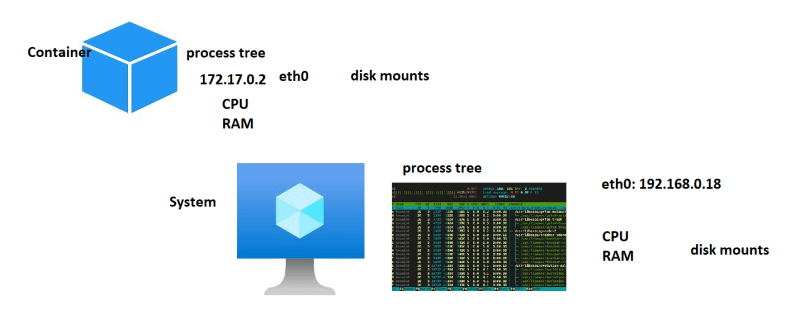
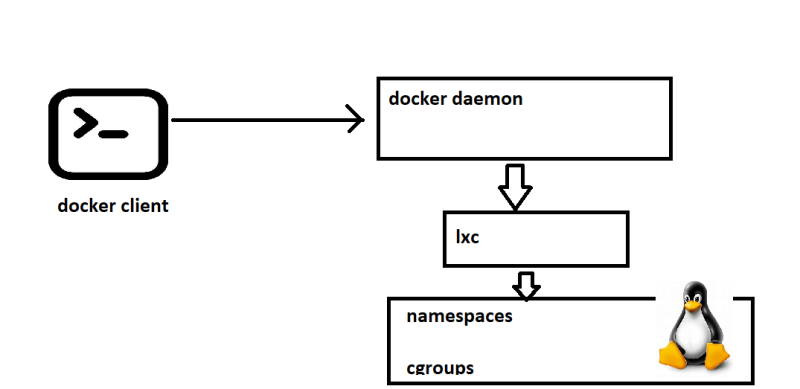
**How Isolations are created or How Containers Work**

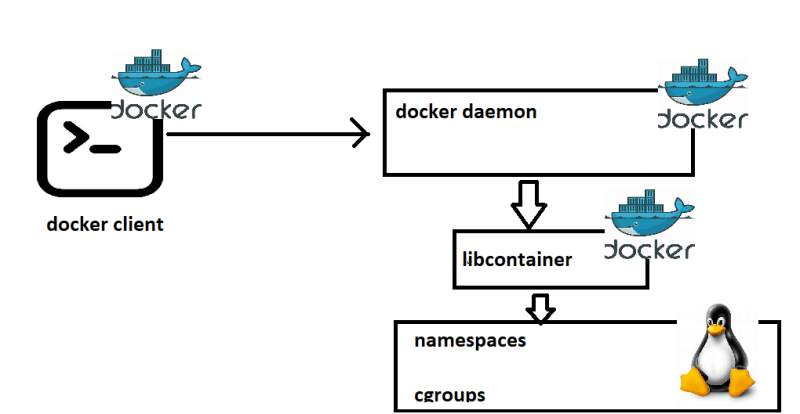
* Each container is getting a
  + new process tree
  + disk mounts
  + network (nic)
  + cpu/memory
  + users
* <https://directdevops.blog/2019/01/31/docker-internals/> for Docker Internals  
  

**Docker Architecture**

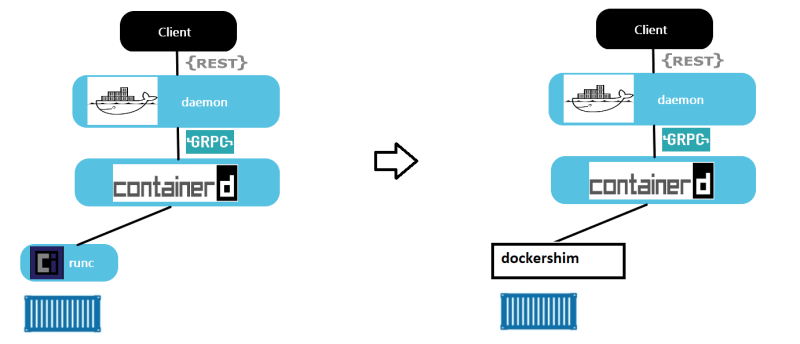
**Generation 1:**

* This was first gen, where docker daemon used lxc (a linux kernel feature) to create containers  
  

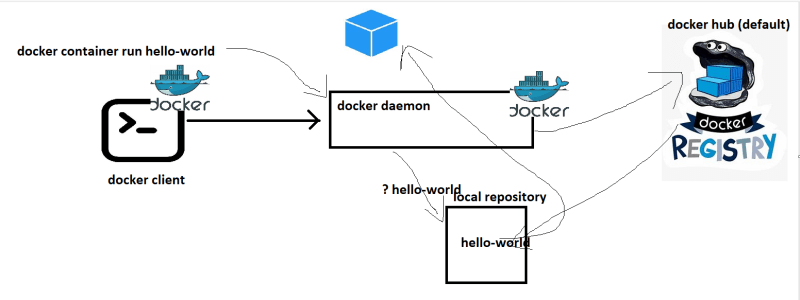
**Generation2:**

* Since docker was relying on lxc which was kernel feature, updates to kernel frequently used to break containers created by docker.
* So docker has created its own component called libcontainer (libc) to create containers.
* Docker wanted containers to be multi os and lxc was definetly not the way forward.  
  
* Adoption of docker was drastically increased as it was stable.

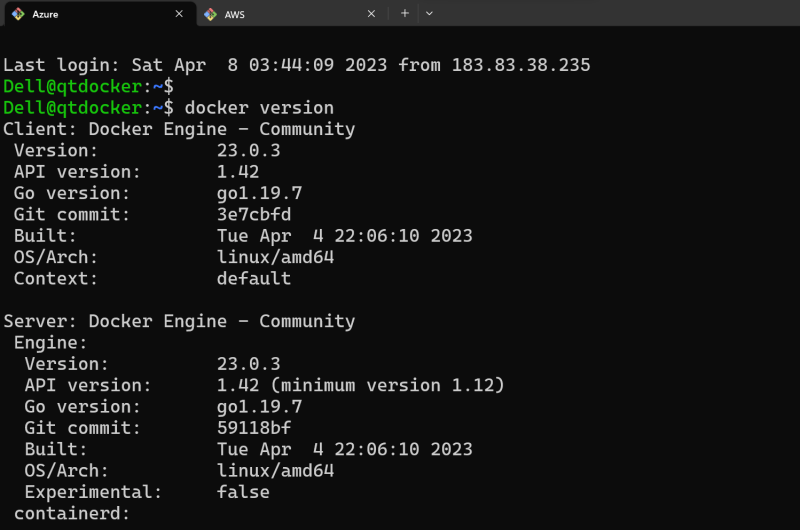
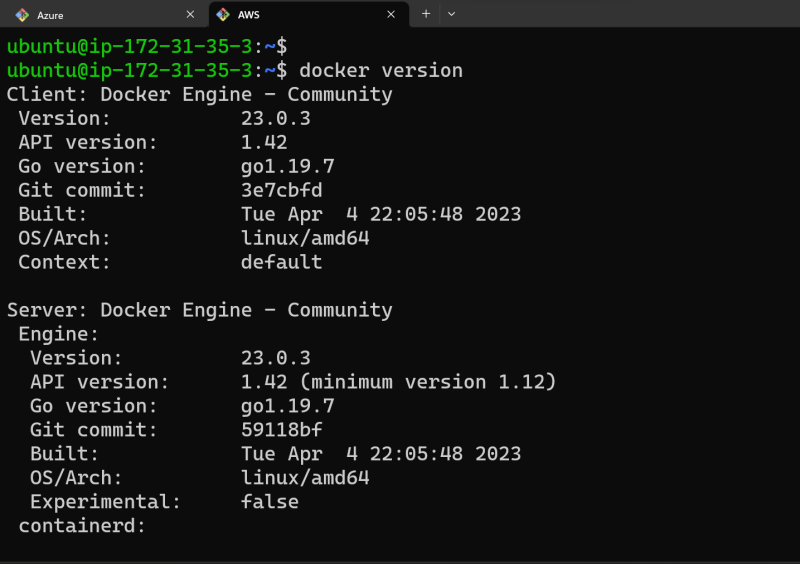
**Generation 3:**

* In this generation, docker engine was revamped from monolith to multi component architecture and the images and containers were according to OCI (open container initiative) image spec and runtime spec.
* In the latest architecture
* docker daemon exposes api’s to listen requests from docker client.
* Passes the requests to containerd. This manages the lifecycle of container
* containerd forks a runc process which creates container. once the container is created the parent of the container will be docker shim  
  

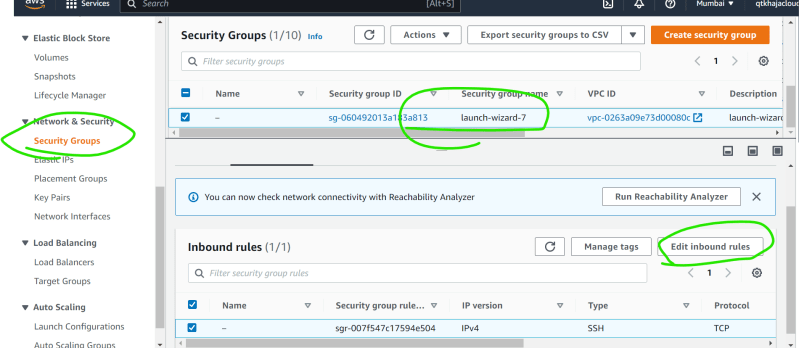
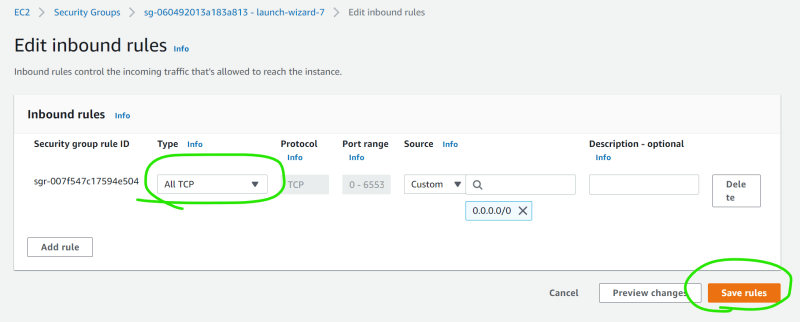
**Creating our first docker container**

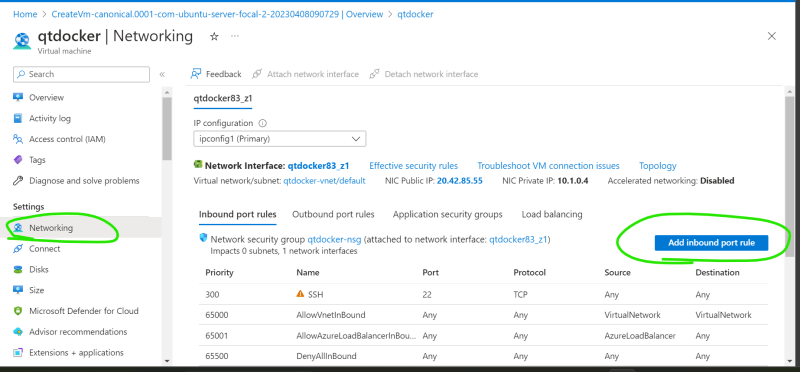
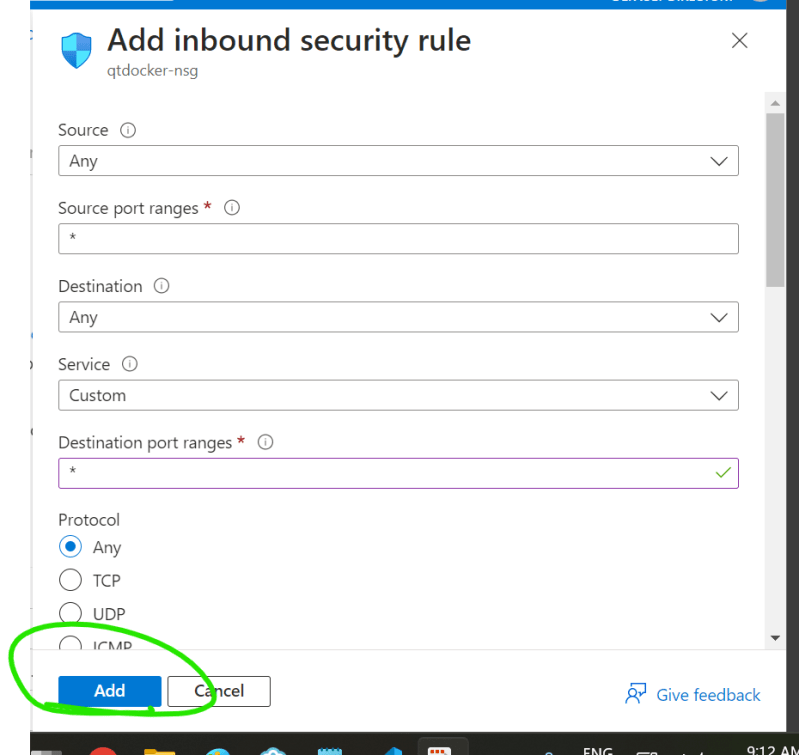
* docker container creation:
* To create container, we need some image in this case let’s take hello-world
* The command docker container run hello-world executed
* What happens
  + docker client will forward the request to docker daemon
  + docker daemon will check if the image exists locally. if yes creates the container by using image
  + if the image doesnot exist, then docker daemon tries to download the image from docker registry connected. The default docker registry is docker hub.
  + Downloading image into local repo from registy is called as pull.
  + Once the image is pulled the container is created.  
    
* Registry is collection of docker images hosted for reuse.
* Docker hub <https://hub.docker.com/search?q=>

**Playing with containers**

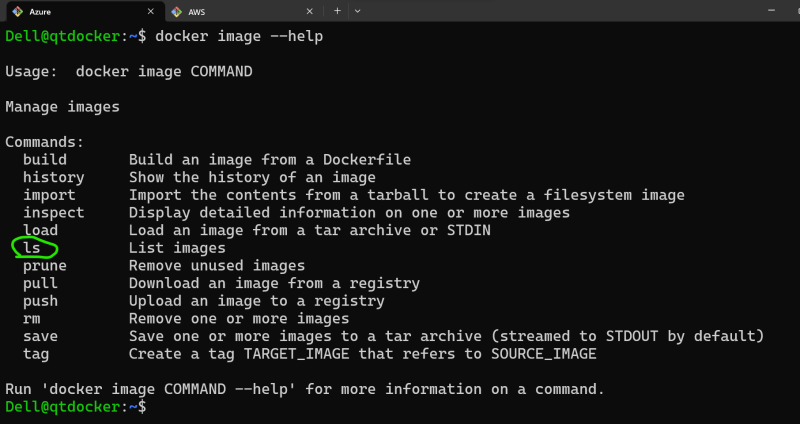
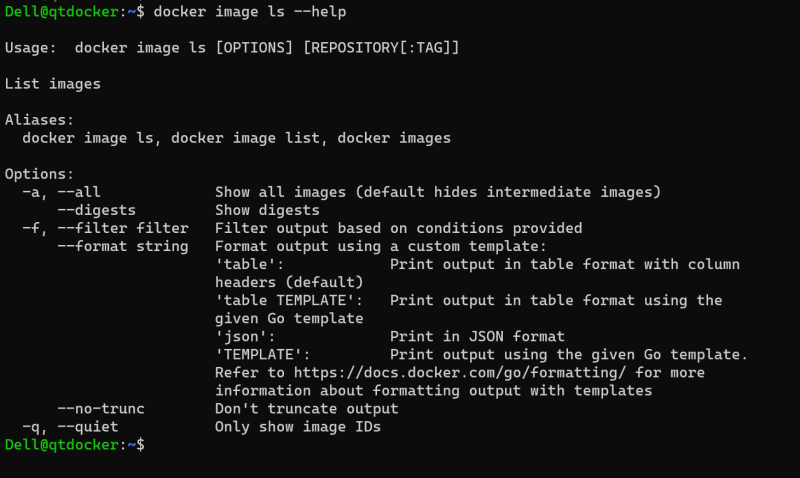
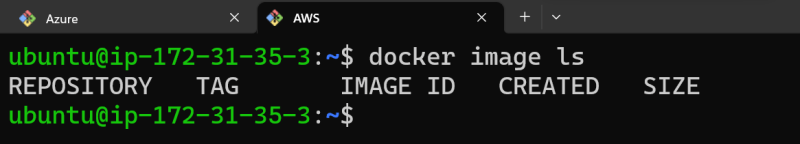
* Create a new linux vm and install docker in it  
    
  

**Opening ports**

AWS  
  


Azure  
  


**Check docker images in the host**

**pull the images from docker hub**

* image naming convention

[username]/[repository]:[<tag>]

shaikkhajaibrahim/myspc:1.0.1

username => shaikkhajaibrahi

repository => what image => myspc

tag => version => 1.0.1

* default tag is latest

nginx

nginx:latest

official images dont have username

nginx

ubuntu

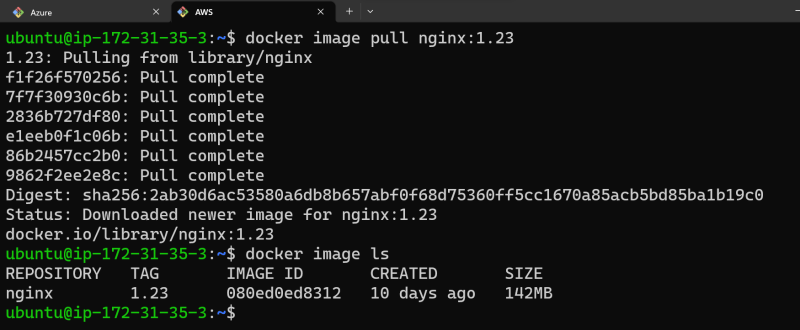
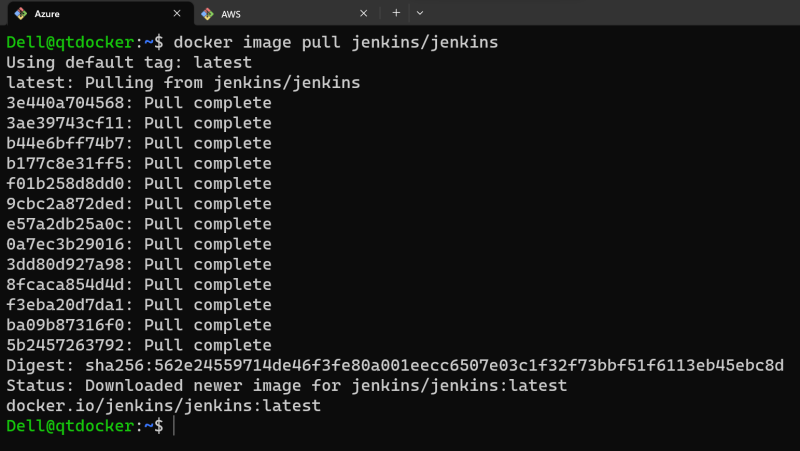
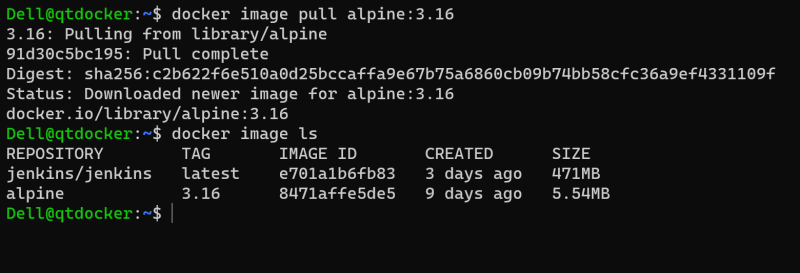
alpine

shaikkhajaibrahim/myspc

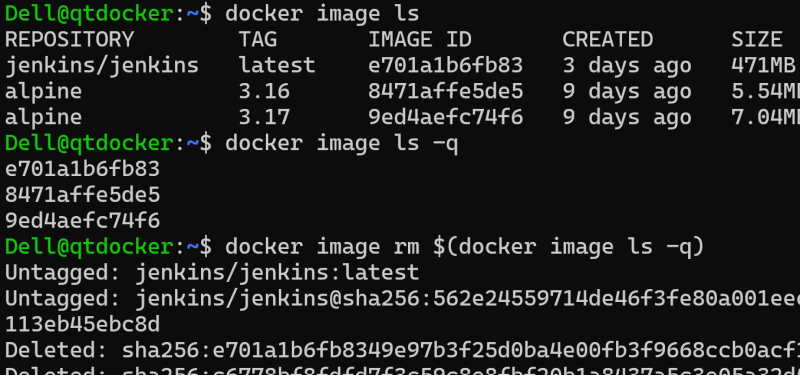
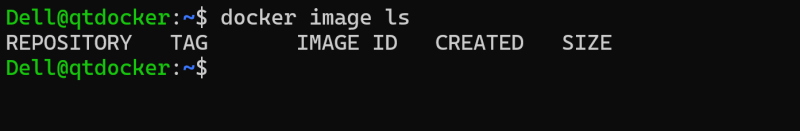
* Lets pull the image nginx with tag 1.23

docker image pull nginx:1.23

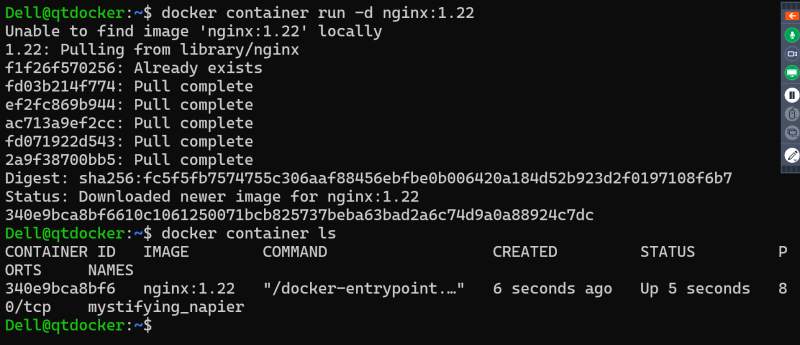
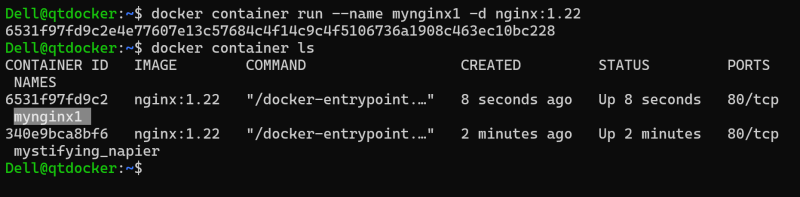
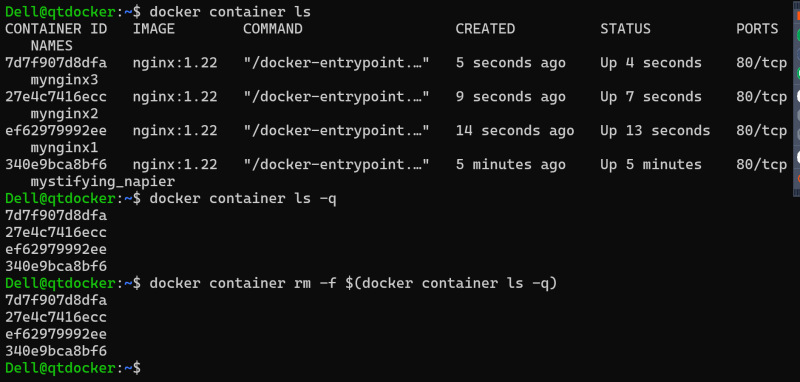
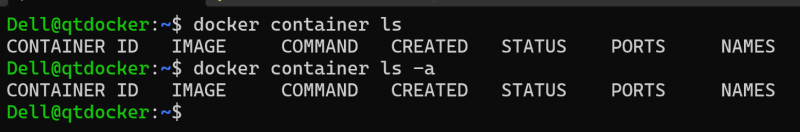
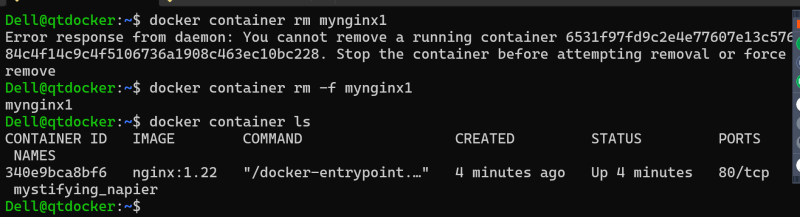
docker image ls

  
\* Lets pull the jenkins image with latest version  
  
\* Lets find the alpine and pull the image  


**Remove images from local**

* Every image will have unique image id and image name
* We can delete individually docker image rm alpine:3.17
* if i have to delete all the images `docker image rm $(docker image ls -q)  
    
  

**Create a container with nginx**

* To create and start the container we use run command  
  
* note: i will be using -d for some time and we will discuss importance of this in next session
* every container gets an id and a name. name can be passed while creating container, if not docker will give random name  
  
* Remove all the running containers docker container rm -f $(docker container ls -q )  
    
  
* Remove specific container  
  
* Remove all containers docker container rm -f $(docker container ls -a -q )  
  