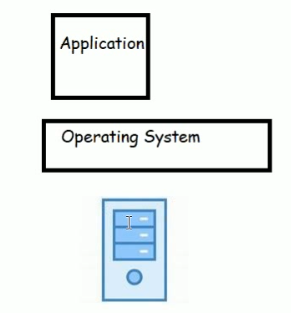
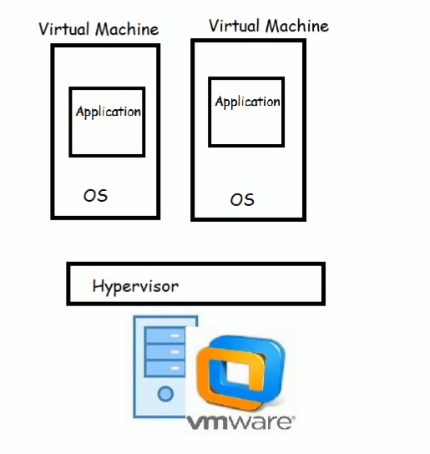
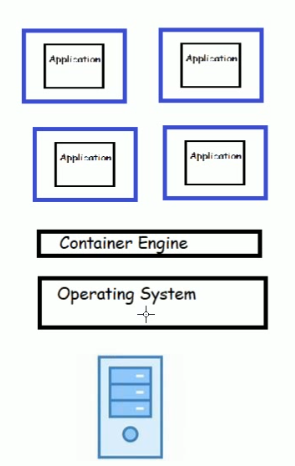
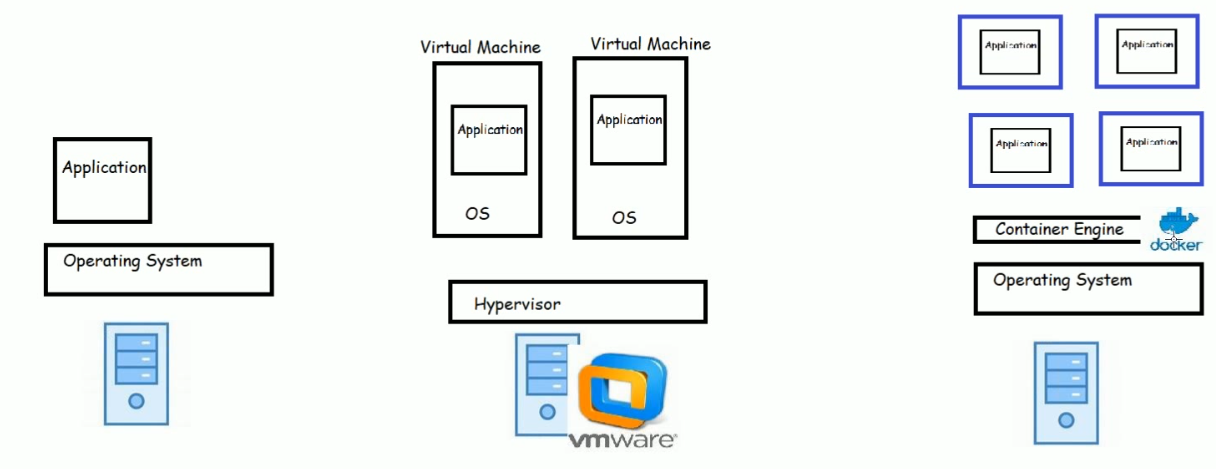
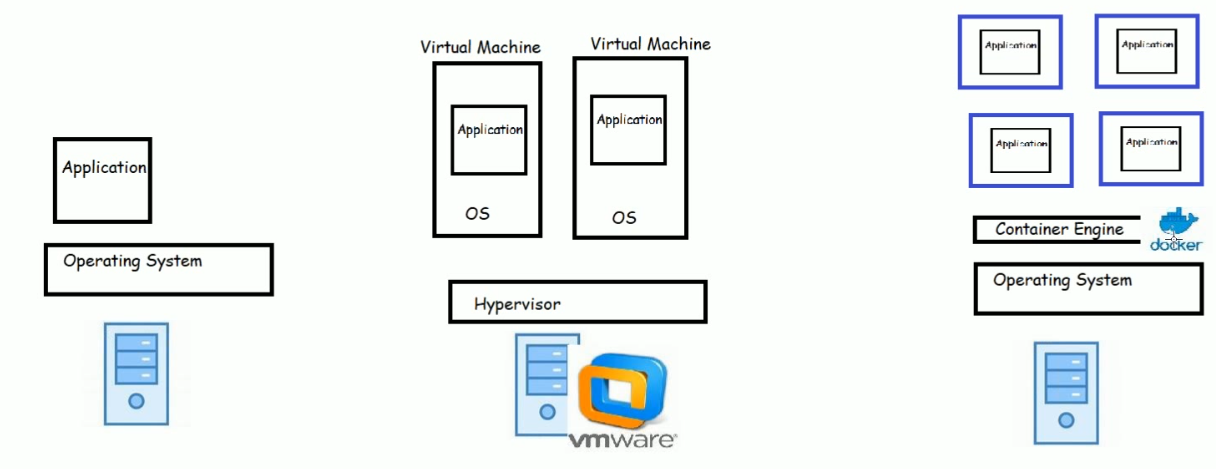
Docker-1 17/Aug/2021

**Containers from High Level:**

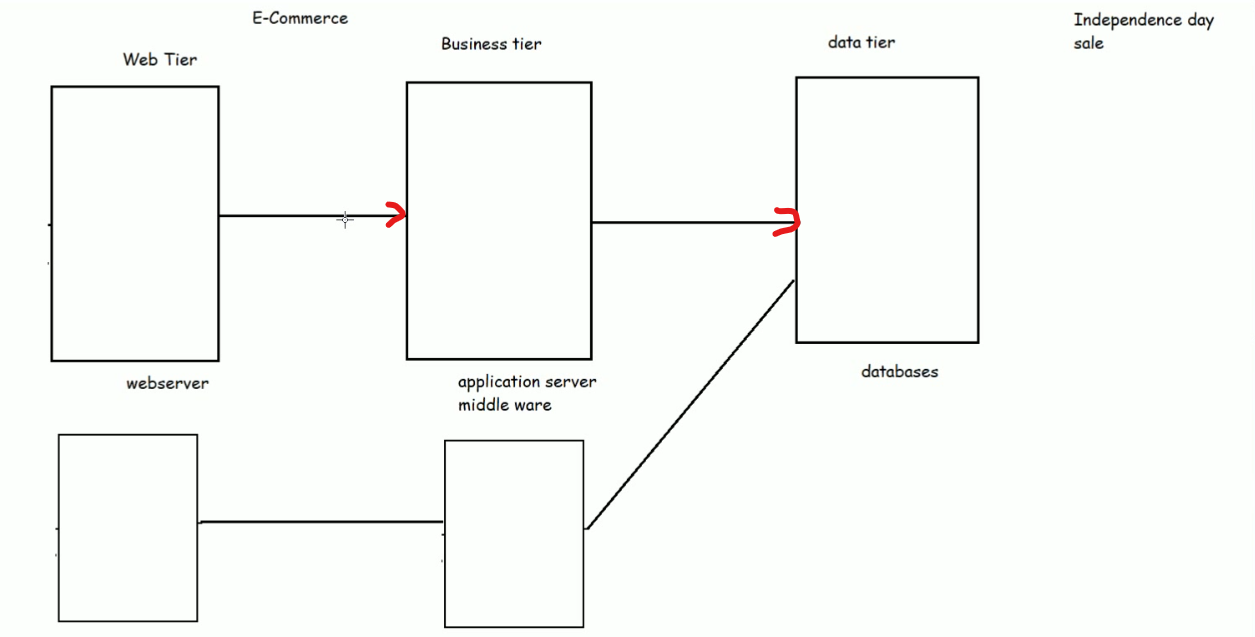
* Applications are heart of the business.
* Applications run on Servers.
* **The bad old days:**
  + Every time the business needed a new application, IT department would buy a new server
  + So IT bought big and this resulted in over powered server operations.
  + This approach is tragic waste of capital and environmental resources.
  + 
* **Hello VMWare!**
  + Amid all of this VMWare Inc. gave world the gift the virtual machine (VM)
  + This was a game changer, IT departments no longer needed to procure a brand new oversized server every time the business needed new application
  + They could now run the new applications on existing server with spare capacity
  + 
* Hello Containers!
  + For a long time, big web scale players, like Google, have been using container technologies to address the short comings of VM Model
  + The container model is roughly analogous to the VM, A major difference is that containers do not require a full blow OS.
  + All the containers on a single host (single server) share the host’s OS. This frees up huge amount of system resources like CPU, RAM and storage. It also reduces potential licensing costs and reduces the overhead of OS Patching and other maintenance.
  + Containers are lot fast to start and ultra-portable.



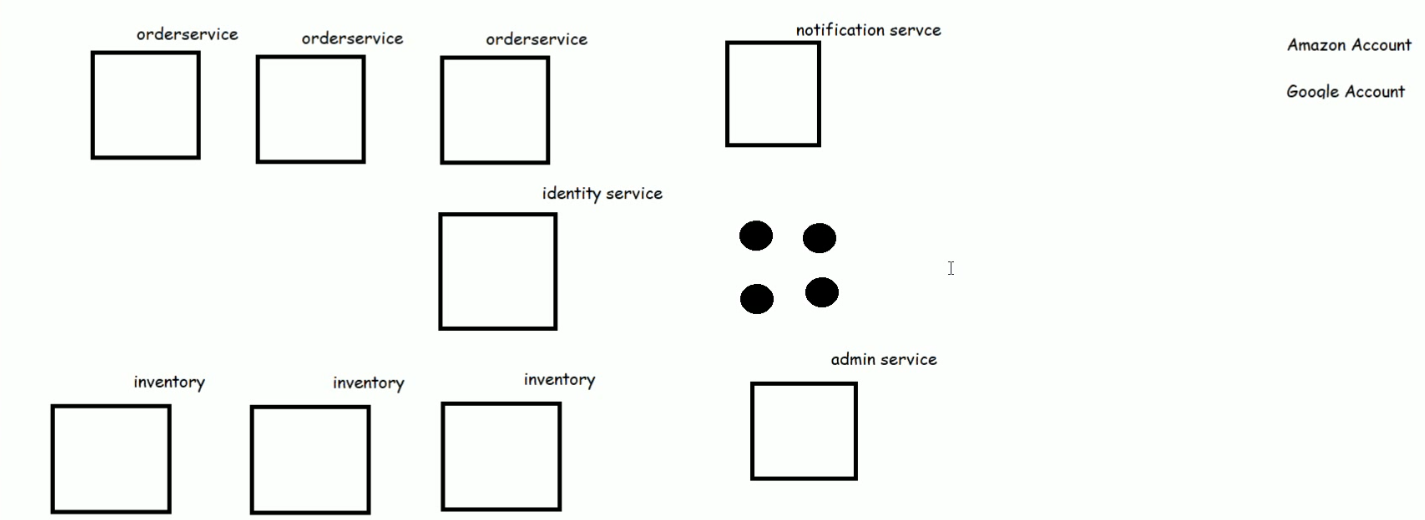
* **Hello Docker! :**
  + Docker has made it extremely simple to create containers which was very much difficult prior to docker to create and maintain containers.
  + 
* **Windows Containers:**
  + Over the past few years Microsoft Corp has worked extremely hard to bring docker and container technologies to the Windows Platform.
  + Microsoft has worked closely with Docker Inc. to achieve the container support.
  + As of now, The Containers are supported natively on the
    - Windows 10 and later
    - Windows Server 2016 and later



While the situation of Independence Day for e-commerce offers, the e-commerce site needs more webserver to auto scale as per load. But here we are scaling full server, those webserver not used full server right, so we have to avoid this process.



And we have to scale up particular services only.



Exercise:

1. Create an account in Docker Hub [Refer Here](https://hub.docker.com/signup) (<https://hub.docker.com/signup>)
2. Installation of softwares (Windows) [Refer Here](https://www.youtube.com/watch?v=mRILfUNbsIo&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=14) (<https://www.youtube.com/watch?v=mRILfUNbsIo&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=15>)
3. Windows Terminal (For Windows 10 users ) Refer Here (<https://www.youtube.com/watch?v=qLVn2EvPsYc&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=12>)

**Docker commands**

Ip config

Ip addr

Docker container run -d -P nginx

Docker container run -d -P httpd

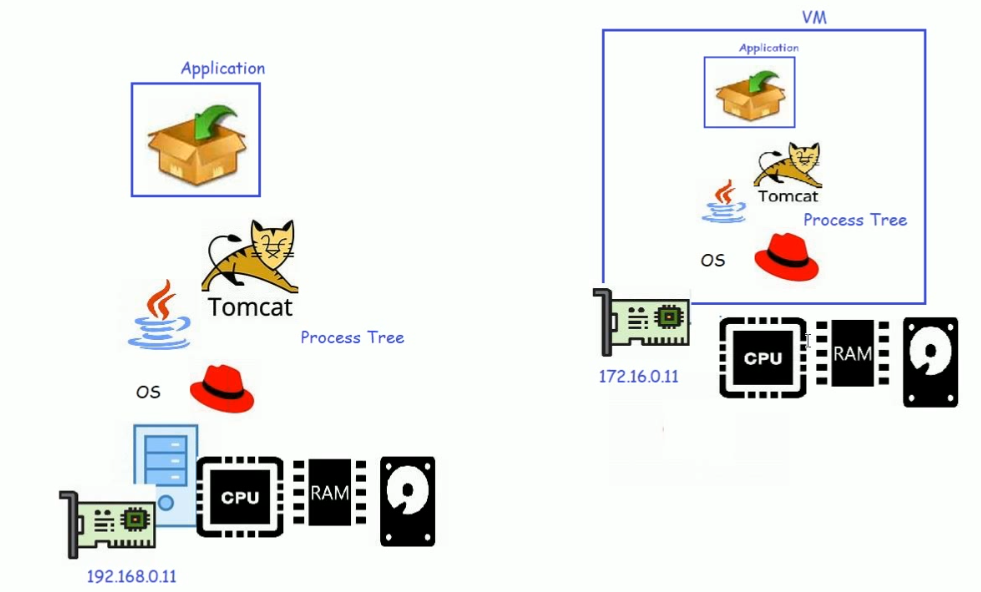
Docker stats (to see memory and cpu utilization)

Docker-2 18/Aug/2021

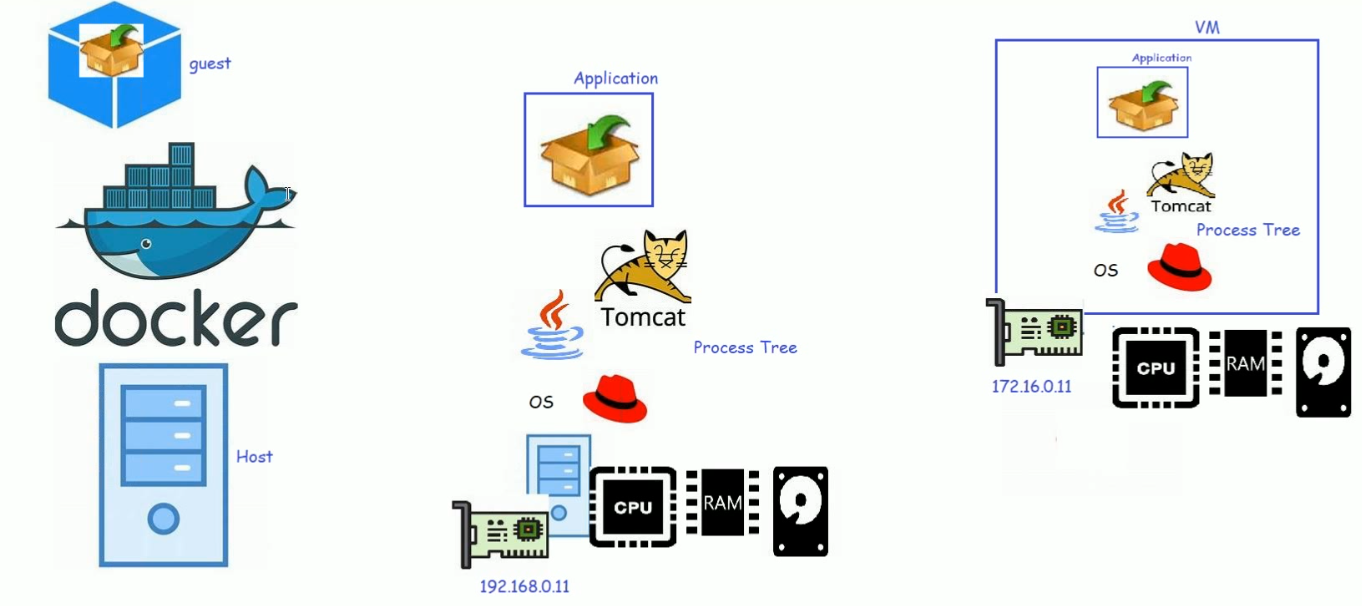
**Introduction Docker, Docker Simplifed Architecture, Build multiple container in docker using images http, nginx and jenkins.**

Docker Continued

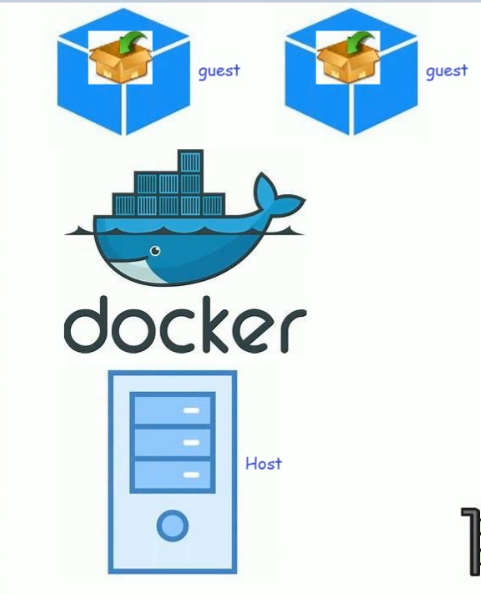
* Inside the container, applications get
  + cpu cycles
  + RAM
  + Disk space
  + process tree
  + network
* So applications have all of the things necessary to be executed, so they work in containers as they work in vm’s or physical system
* **Container can be defined as an isolate area created on the system with**
  + **cpu**
  + **RAM**
  + **Disk /file system**
  + **process tree**
  + **network (ip address)**
  + **thin layer of os**



|  |
| --- |
| * docker container run –it alpine /bin/sh (to turn as linux terminal) * ps (to list running tasks) * top(to see cpu and memory status) * apk search python (to search python is available or not) * apk add pyhton3 (to add python3 to container) * apk update (to update container) * apk search java * apk add openjdk11 * exit (to turn into docker terminal) * docker container run -P httpd (to create nginx container in docker) * docker container run –P jenkins/Jenkins:lts-jdk11 (to install java & create Jenkins server container in docker) * docker container run –d –P nginx( to create nginx container in detached mode. image avilble in docker hub) * To create docker containers we need to use images |



We can run multiple same servers and versions of server on same docker



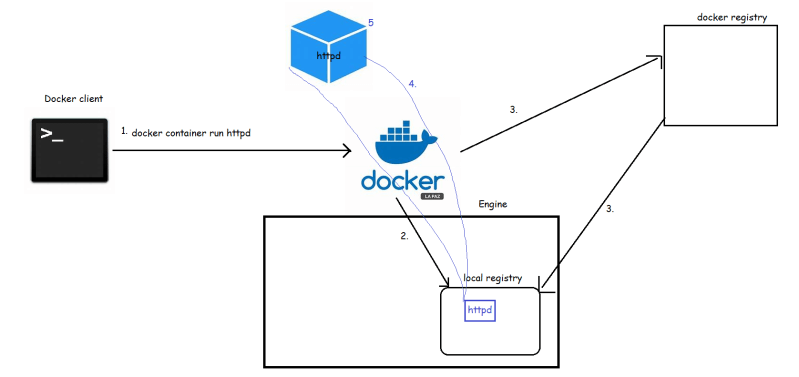
It doesn’t matter, whether we run same application multiple times or multiple different application in multiple times

* To create a container docker, needs an Image.
* Initial Responsibilities of DevOps Engineer
  + Creating Docker Images for the applications developed
  + Sharing Docker Images with the servers where the application has to be deployed

**Docker Simplified Architecture**

* **When we install Docker on any system, we get two major components**
  + **Docker client**
  + **Docker daemon (Docker Engine)**
* When we want to run an application, we need to create a container, let’s see the process in the following steps

**Steps when we try to create docker container:**

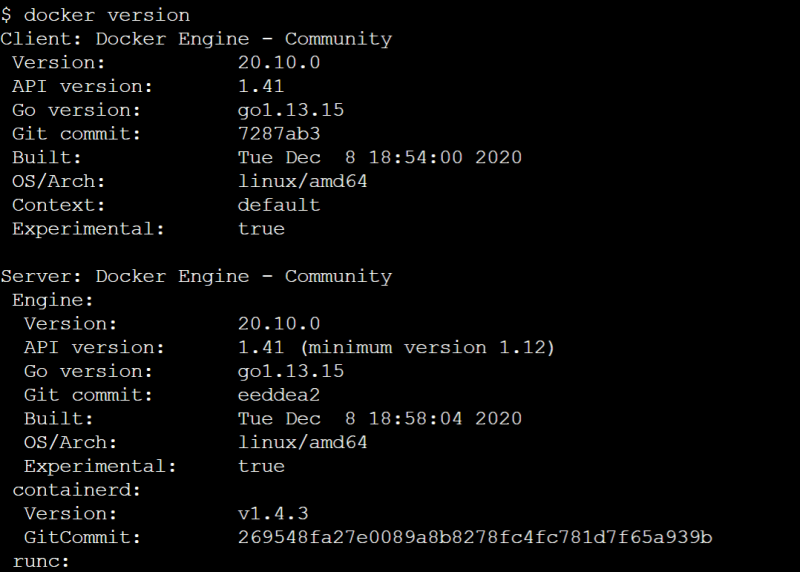
* + request made from docker client to create httpd container reaches the docker engine
  + docker engine searches the httpd image locally
  + Since the image is not present, docker engine tries to download the image (pull) from docker registry (registry is nothing but collection of images). Default docker registry is dockerhub
  + Once the image is pulled, docker engine creates the container from the local image pulled/downloaded
  + The container will get a
    1. container id
    2. container name
    3. some cpu/memory
    4. an ip address
    5. filesytem from docker image
    6. 

|  |
| --- |
| docker image ls(to see docker image in list)  docker container ls(to see what are the containers are running) |

Docker-3 20/Aug/2021

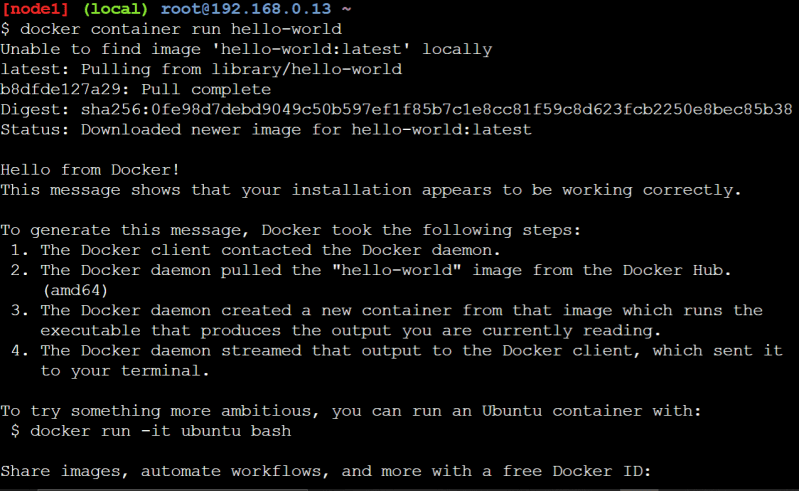
**Docker Container**: Running, starting, stopping & removing containers.

Containers

* We will try to cover the following
  + Running the first container
  + Starting, Stopping and Removing Containers
  + Inspecting Containers
  + Exec into running container
  + Attaching to a running container
  + Retrieving container logs
  + Port Forwarding
* To Get used to commandline use --help or cheatsheets
* “**docker** **version**” command if everything works correctly will show the client and the server versions installed
* 
* Let’s try to run a first container take image from docker hub hello-world
* docker container run this command always creates a new container

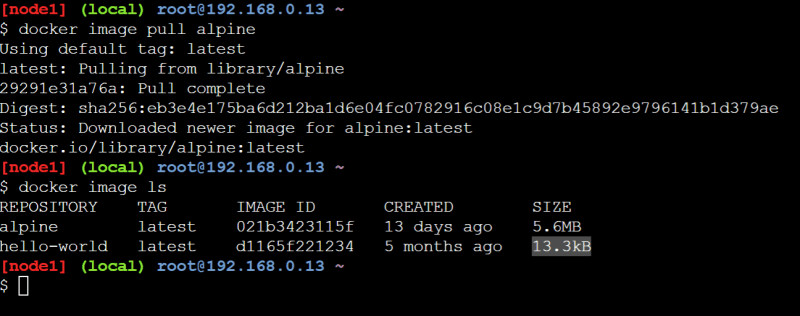
docker container run <docker-image>

docker container run hello-world



* Let’s try to pull the image and view the sizes of the images

docker pull <image>

do

* Let’s try to create a container

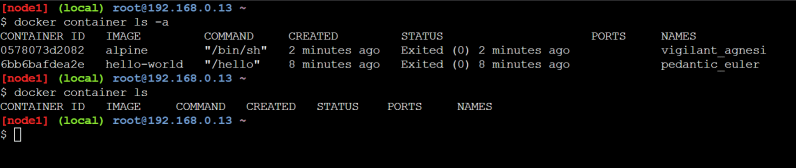
docker container run alpine

* Now let’s look at the containers running on the system

docker container ls

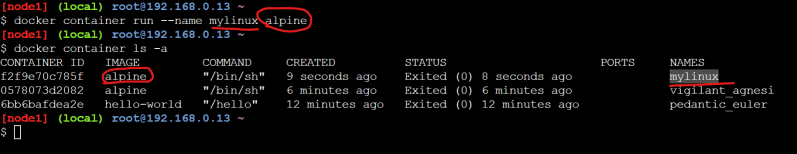
* Now let’s look all the container irrespective of their current state

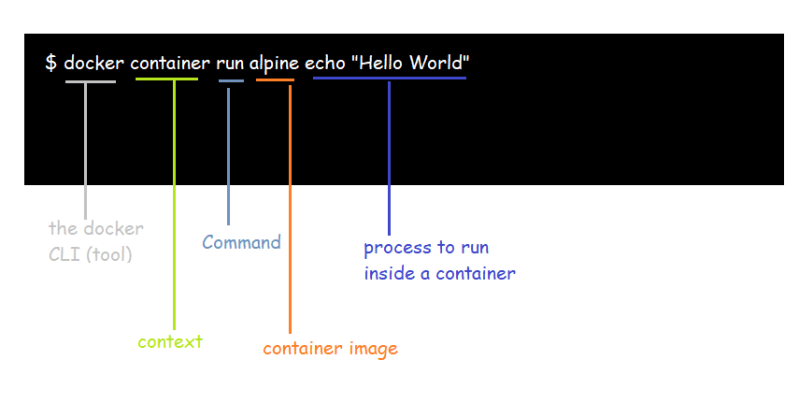
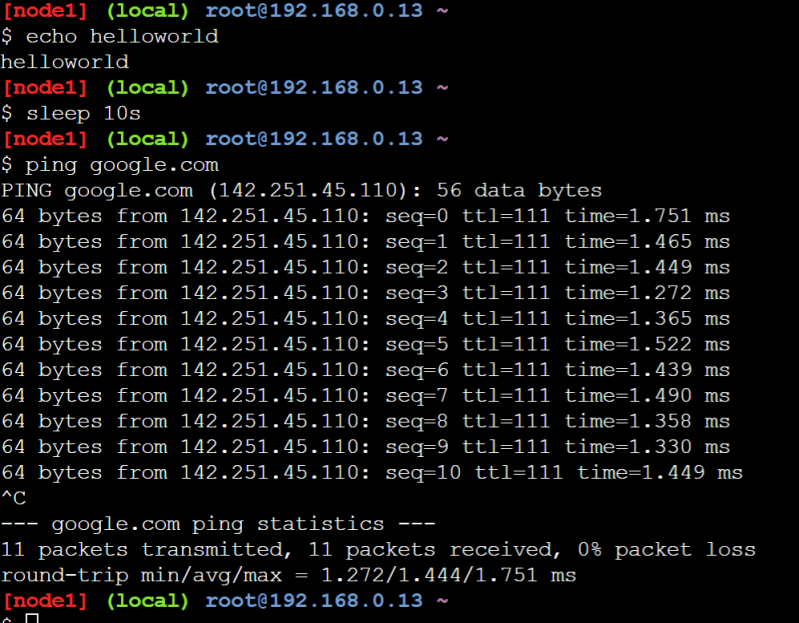
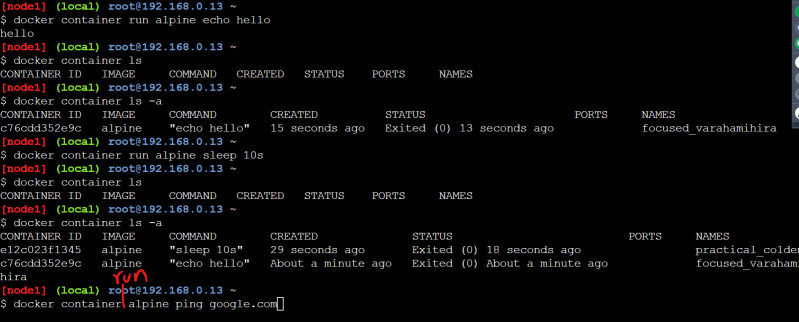
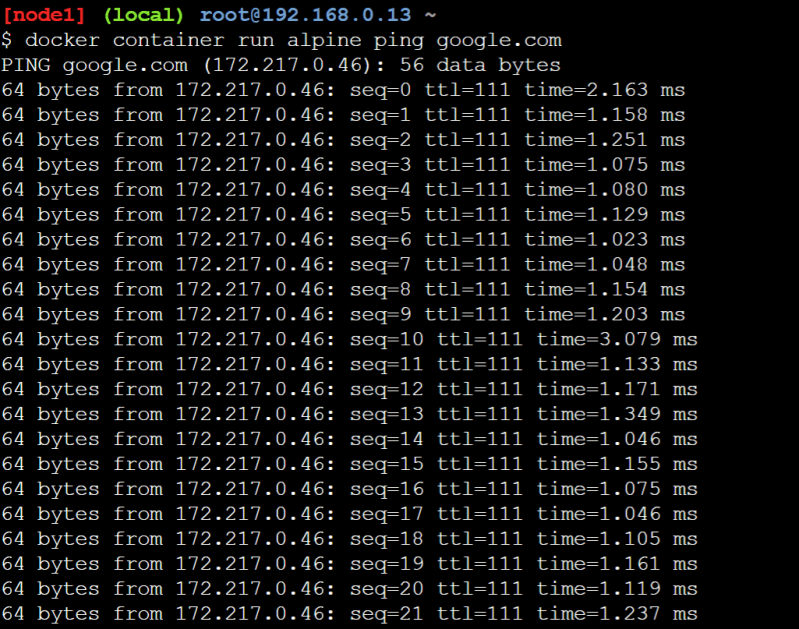
docker container ls -a



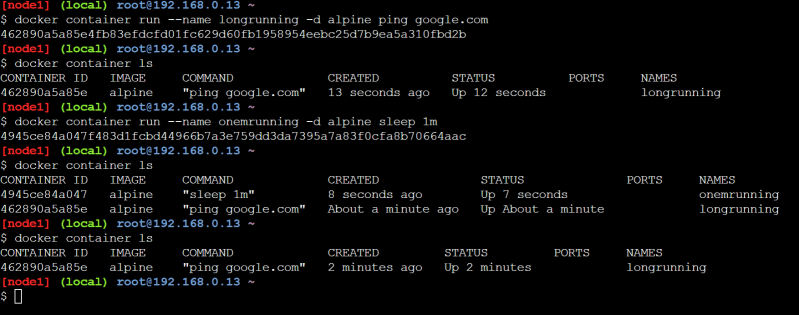
* For every container created docker will create
  + container id
  + container name if not passed
* So let’s to create a container from some image with a name

docker container run --name <name> <docker-image>



* Let’s disect the command which we are passing
* 
* Three basic linux commands which finish immediately, after 10 seconds, till user quits (ctrl+c)
* 
* Now let’s try to run these commands as process inside the alpine container
* 
* By default when we run docker container, it attaches the standard output to the terminal where we execute commands
* 
* We can specify the docker to run in background mode(detached mode)

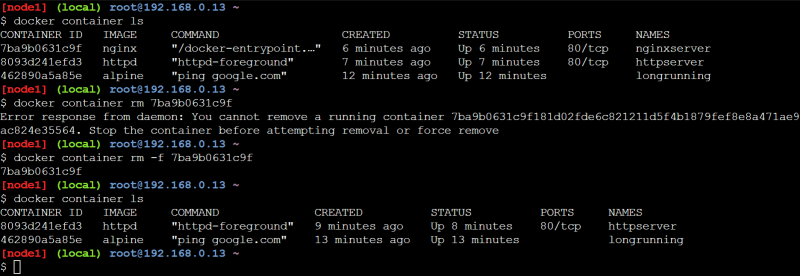
docker container run -d <image-name>

* The containers lifetime depends on how long the command or process that gets started when container is created is running
* 
* All of the docker images have some command that gets executed when the container is created & in many cases (httpd, nginx etc) these are good enough as the they start the application and the command execution doesn’t stop as long as application in contianer is running, so as long as app in container is running container will also be in running state.

**How to remove containers**

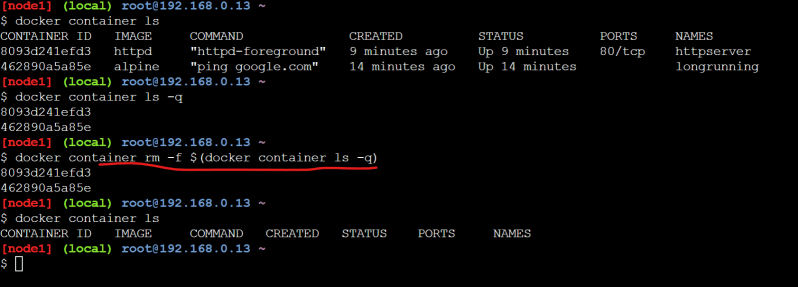
docker container rm <container id or name> # this is for the stopped or exited

docker container rm -f <container id or name> # this works for container in running as well as stopped



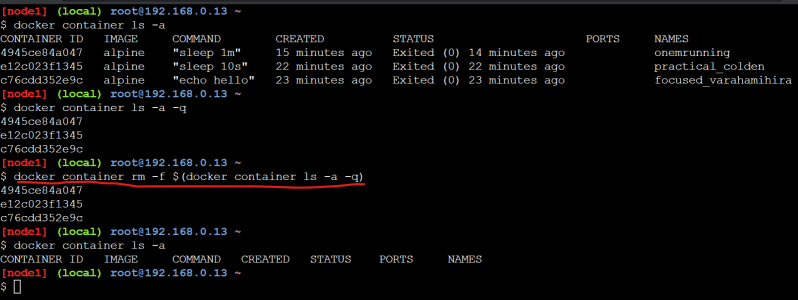
* if you want to remove all the running containers

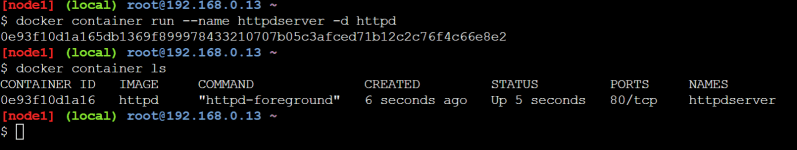
docker container rm -f $(docker container ls -q)



* IF you want to remove all the containers

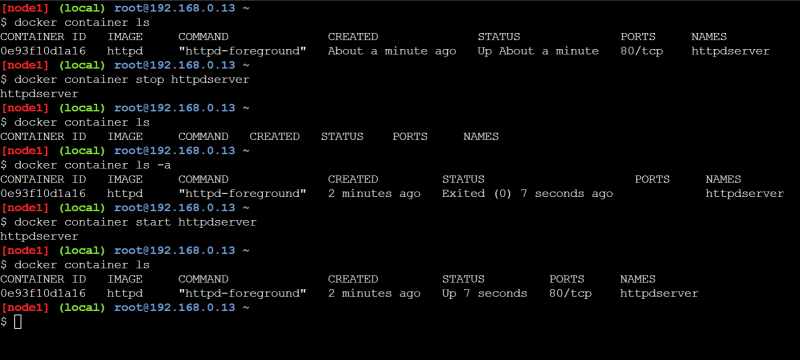
docker container rm -f $(docker container ls -a -q)



* Now lets create a httpd container
* 
* We can start the container and stop the container

docker container start <container id or name>

docker container stop <container id or name>



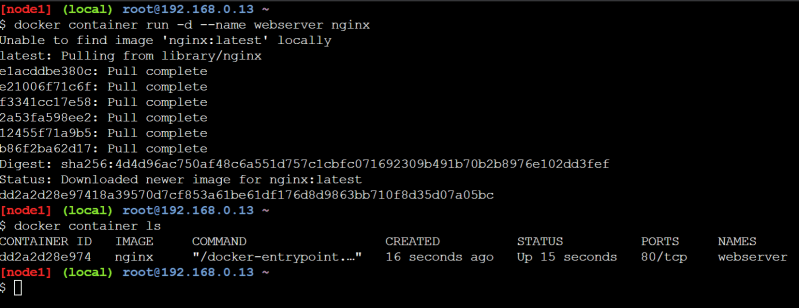
=========================================

Docker-4 21/Aug/2021

**Docker commands: Inspect, logs, Attach, Dynamic port forwarding, exec and Install Docker on Ubuntu**

## Containers Continued

* Let’s create a new nginx container

(To get full screen of docker playgroung “alt+enter”)

* Inspecting containers
  + Command: docker container inspect <container-name/id>
  + Containers have lot of associated data that characterizes its behavior, To get the information we can use inspect command
  + The response from inspect is a json object with full of details
  + 
* We have started the container in detached mode, now lets try to attach to the container

docker container attach <container-name/id

docker container attach webserver

* Once we attach to the container and use Ctrl+c then the container will be stopped.
* To quit from the attached mode without stopping container (Ctrl+ pq)
* When the application in a container writes the logs to stdout and stderr, we can view those logs from logs command

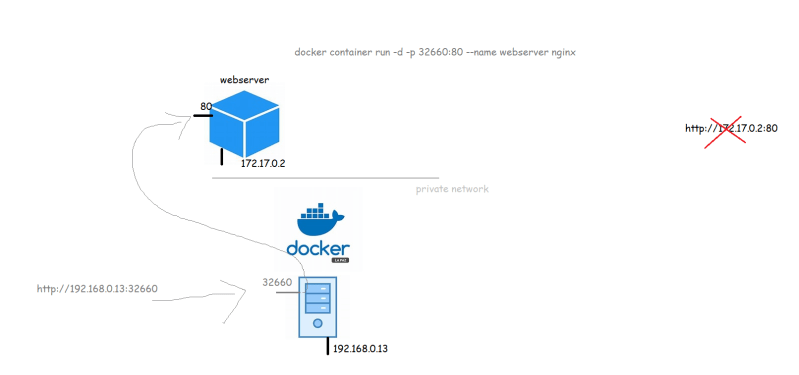
docker container logs <container-name>

* If you want to get only last few entries

docker container logs --tail 5 <container-name>

* If you want to follow the logs

docker container logs --tail 5 --follow <container-name>

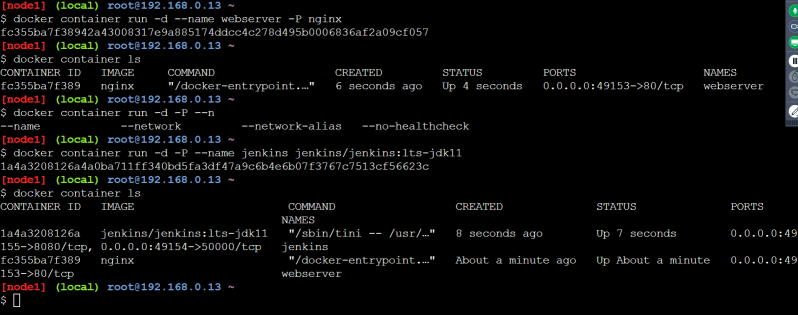
* When we create containers, container run on the internal network which is not accesible from outside world, but we would still want to access the applications running inside containers.
* So while creating containers we can do port forwarding. This port-forwarding forwards the request recieved on the host (machine where docker is installed) on some specific port to the port in the container
* 
* If we want to specify the manual ports to be mapped then

docker container run -p <port on host>:<container port> <image>

docker container run -d -p 32660:80 --name webserver nginx

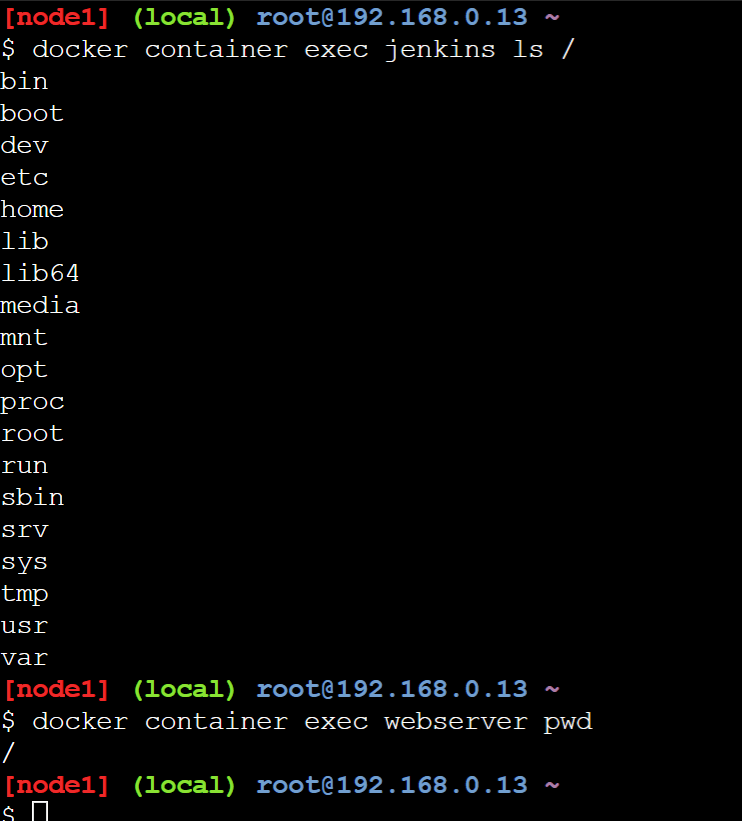
* There is also an option to dynamically create port forwarding

docker container run -P <image>



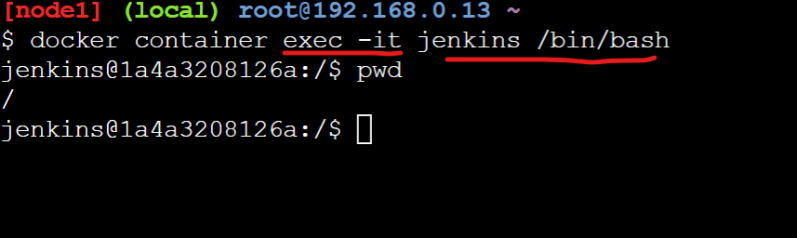
* Executing commands in the docker container

docker container exec <container-name> <command>



* Now if you want to login in into docker container terminal for already created container

docker container exec -it <container> <terminal>



* Note: When we try to work in containers in many cases you might not find all the linux utilities inside the container & this is for greater good. The container should not have any additional tools/utilities apart from what are required to run your application.
* If you want to run the contianer (create a new container) in the interactive mode

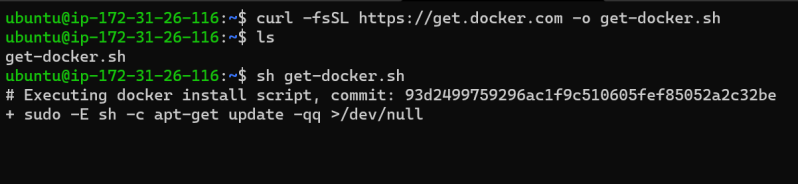
docker container run -it <image-name> <terminal>

## Installing docker

* Lets try to install docker on a Linux Machine
  + Create a ubuntu VM in AWS:
  + Create a ubuntu VM in Azure
* note:
  + AWS EC2 instance: [Refer Here](https://www.youtube.com/watch?v=me2s3mTNwGo&list=PLuVH8Jaq3mLszrC7lv68a0VcrDripW-HK&index=2) (<https://www.youtube.com/watch?v=me2s3mTNwGo&list=PLuVH8Jaq3mLszrC7lv68a0VcrDripW-HK&index=3>)
  + Azure VM: [Refer Here](https://www.youtube.com/watch?v=P9X-4Z-NeGg&list=PLuVH8Jaq3mLuqXuGs6aeqxhuvCYSzB1kT&index=2) (<https://www.youtube.com/watch?v=P9X-4Z-NeGg&list=PLuVH8Jaq3mLuqXuGs6aeqxhuvCYSzB1kT&index=3>)
* To install docker on a linux machine

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh



* Now to give the access of docker to non-root users, we need to add them to docker group

whoami

sudo usermod -aG docker $(whoami)

# Now logout and login into the terminal

docker version

* Docker can also be installed on Desktops
  + Windows 10 [Refer Here](https://docs.docker.com/desktop/windows/install/) (<https://docs.docker.com/desktop/windows/install/>)
  + Mac [Refer Here](https://docs.docker.com/desktop/mac/install/) (<https://docs.docker.com/desktop/mac/install/>)
* Note: Docker Commands
  + Docker commands in the earlier versions were docker <command> but docker now recommends using docker <context> <command>

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

Docker-5 22/Aug/2021

## Docker Internals

* Questions to understand
  + How are container isolations created?
    - Network
    - Process tree
    - FileSystem
  + How are cpu and memory resources allocated to this isolation
* For more content on containers
  + [Refer Here](https://directdevops.blog/devops-blog-imported-from-qt-blog/) for the list of the blog items (<https://directdevops.blog/devops-blog-imported-from-qt-blog/>)
  + [Refer Here](https://directdevops.blog/2019/01/31/docker-internals/) for docker internals (<https://directdevops.blog/2019/01/31/docker-internals/>)
* In linux we have a kernel feature called as Namespaces that can be used to create isolations.
* To apply resource limits to this isolation we have one more kernel feature which is called cgroups (Control groups)

## Lets try to run some java application inside container

* Manual Steps:
  + Creating a ubuntu vm
  + installing java
  + downloading the application(jar file)
  + running the command to start the applciation
  + Access the application by using http url
* Commands

sudo apt update

sudo apt install openjdk-11-jdk -y

wget https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar

java -jar spring-petclinic-2.4.2.jar

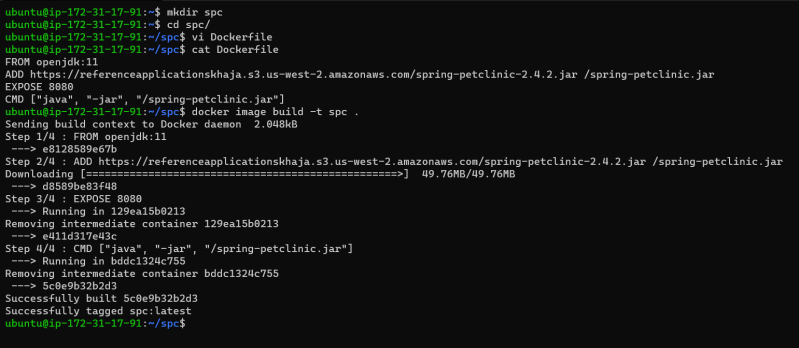
* Now access the application from http usign http://<public-ip&gt;:8080
* Lets see if we can do the same in containers
* Now access the machine where docker is installed.
* Option 1:
  + Try to search for a docker container with ubuntu image
* docker container run --name exp1 -p 8000:8080 -it ubuntu /bin/bash
* apt update
* apt install openjdk-11-jdk wget -y
* wget https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar
* java -jar spring-petclinic-2.4.2.jar
  + When we the run the above commands the application runs inside the contaiener are we are able to access the application using http://<host-ip&gt;:8000
* Option 2:
  + Searching for openjdk image has resulted in openjdk:11
* docker container run --name exp2 -p 8001:8080 -it openjdk:11 /bin/bash
* curl https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar --output spring-petclinic.jar
* java -jar spring-petclinic.jar
  + When we the run the above commands the application runs inside the contaiener are we are able to access the application using http://<host-ip&gt;:8001
* Observations: When we want to run our application inside the container
  + manually executing commands as shown above is not a sensible idea
  + When we are creating images choosing a base container with necessary softwares installed (option 2) will be sensible.
* In Docker if you want to create images we create Dockerfile, which has instructions on how to build the container image and command to be executed when container is execute
* Create a file in a new folder called as Dockerfile and add the below contents

FROM openjdk:11

ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar /spring-petclinic.jar

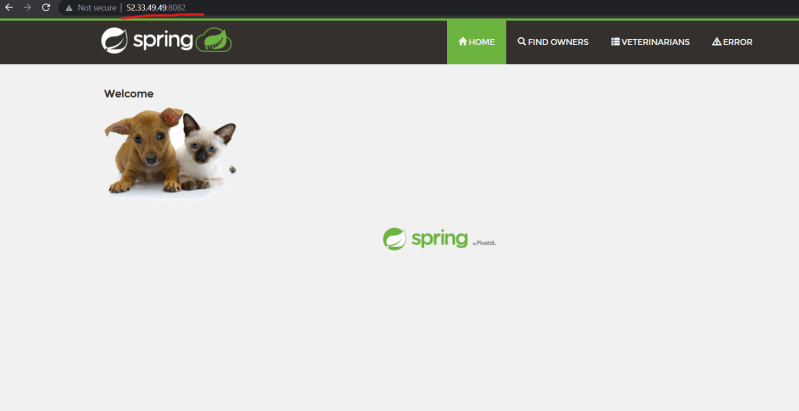
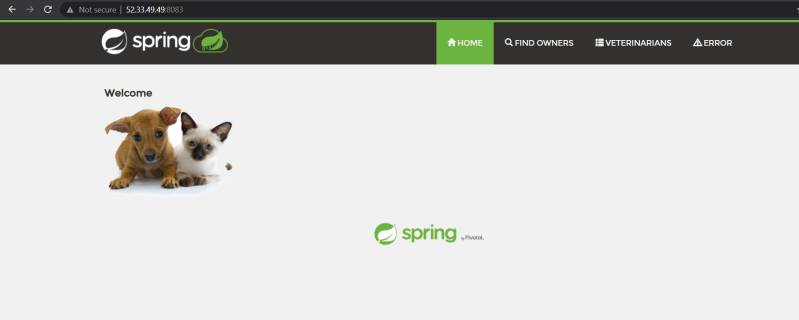
EXPOSE 8080

CMD ["java", "-jar", "/spring-petclinic.jar"]

* Now lets try to build the image using docker image build -t spc .
* 
* Now create the container

docker container run -d --name exp3 -p 8082:8080 spc

docker container run -d --name exp4 -p 8083:8080 spc

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

Docker-6 24/Aug/2021

## Dockerfile Instructions and Image Creations

* To create a docker container we need an image.
* Applications will have different versions, To accomodate this in docker container images, Docker images have tags.
* The syntactic name of docker image <image-name>:<tag-name>.
* If we don’t pass the tag-name the default tag latest will be applied.
* To create any Docker image, we need to know the manual steps for configuring and Running the application.
* To make the process of creating docker images simple, Docker has `Dockerfile“ which is text file and contains instructions on how to build a container image.
* This is declarative way of building images
* [Refer Here](https://docs.docker.com/engine/reference/builder/) (<https://docs.docker.com/engine/reference/builder/>) for the Dockerfile instructions and reference
* Lets start with simple instructions
  + FROM:
    - Using this we define what is the base image for building our custom application image
  + FROM <base-image>:<tag>
    - It is recommended not to use the latest tag prefer specific tags over here
    - If you really want to start to build a custom image with out any specific base image FROM scratch
    - [Refer Here](https://docs.docker.com/engine/reference/builder/#from)(<https://docs.docker.com/engine/reference/builder/#from>)
  + LABEL: Adds metadata to the image [Refer Here](https://docs.docker.com/engine/reference/builder/#label) (<https://docs.docker.com/engine/reference/builder/#label>)
  + ADD: This instruction can be used to copy the files from http or docker host into docker image
* ADD <source> <destination>
  + COPY: This instruction can be used to copy the files from docker host into docker image
* COPY <source> <destination>
  + RUN
  + EXPOSE: Expose instruction infor Docker that the container is listenting to the specific netowrk ports
* EXPOSE <port> [<port>/<protocol>]
  + CMD: This instruction executes the command when the container is created [Refer Here](https://docs.docker.com/engine/reference/builder/#cmd) (<https://docs.docker.com/engine/reference/builder/#cmd>)
* The Dockerfile for springpetclinic which we have created is

FROM openjdk:11

LABEL author="khaja ibrahim"

LABEL organization="quality thought"

ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar /spring-petclinic.jar

EXPOSE 8080

CMD [ "java", "-jar", "/spring-petclinic.jar" ]

* Now lets try to build a docker image

docker image build -t <image-name>:<tag> <path to folder where Dockerfile>

docker image build -t <image-name>:<tag> -f <path to Dockerfile>

* We can alos use the slim versions from base image if available

FROM openjdk:11-slim

LABEL author="khaja ibrahim"

LABEL organization="quality thought"

ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar /spring-petclinic.jar

EXPOSE 8080

CMD [ "java", "-jar", "/spring-petclinic.jar" ]

* Exercise: Try create an docker container image for gameoflife
  + Package file: <https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/gameoflife.war>
  + This file will run on tomcat 9 version with java version 8 and this war file has to be copied into the webapps folder of tomcat
  + This application will use tomcat’s port which is 8080

FROM tomcat:jdk8

LABEL author="khaja"

EXPOSE 8080

ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

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

Docker-7 18/Aug/2021

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

Docker-8 18/Aug/2021

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

Docker-9 18/Aug/2021

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

Docker-10 18/Aug/2021

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

Docker-11 18/Aug/2021

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