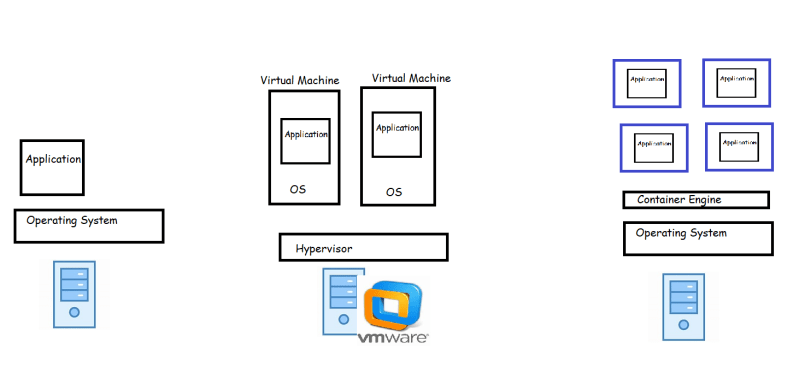
AUGUST 17, 2021

DevOps Classroom Series – 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 apple’s 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 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

Exercise:

1. Create an account in Docker Hub [Refer Here](https://hub.docker.com/signup)
2. Installation of software’s (Windows) [Refer Here](https://www.youtube.com/watch?v=mRILfUNbsIo&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=14)
3. Windows Terminal (For Windows 10 users ) [Refer Here](https://www.youtube.com/watch?v=qLVn2EvPsYc&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=11)

**Share this:**

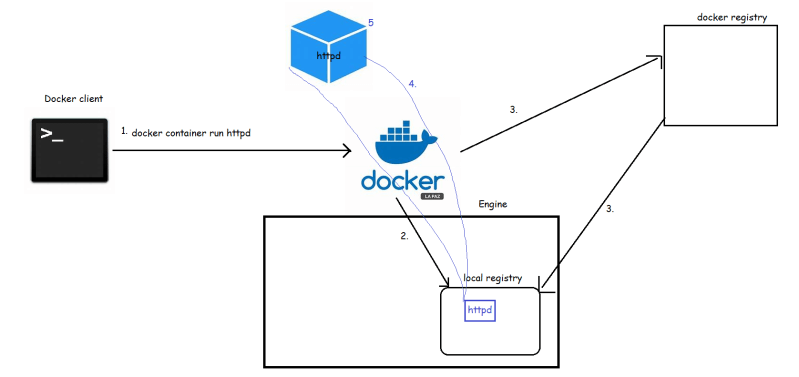
AUGUST 18, 2021

DevOps Classroom Series – 18/Aug/2021

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
* 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

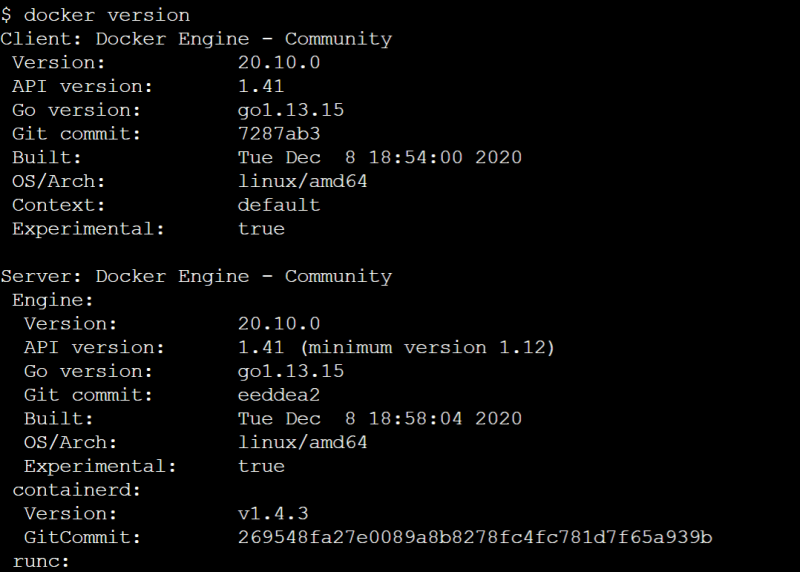
Docker Simplified Architecture

* 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, lets see the process in the following steps
* Steps when we try to create docker container:
  + request made from docker client to create http 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. Default docker registry is docker hub
  + 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. file system from docker image 

AUGUST 20, 2021

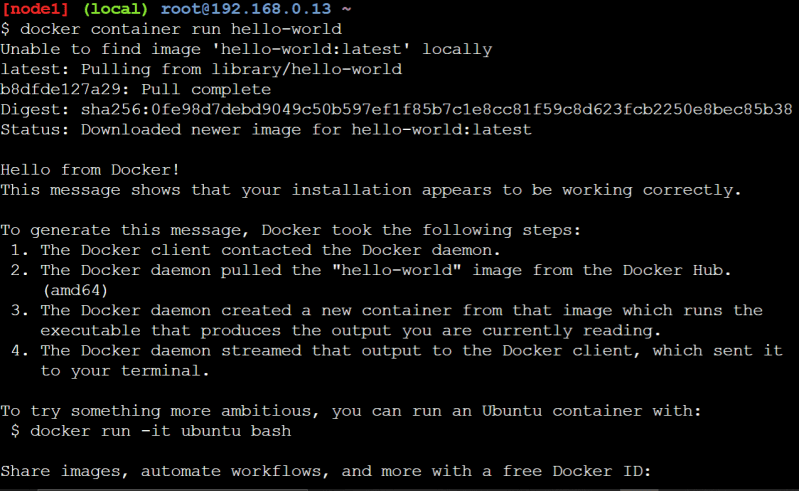
DevOps Classroom Series – 20/Aug/2021

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 command line use --help or cheat sheets
* Docker version command if everything works correctly will show the client and the server versions installed 
* Let’s try to run a first container hello-world

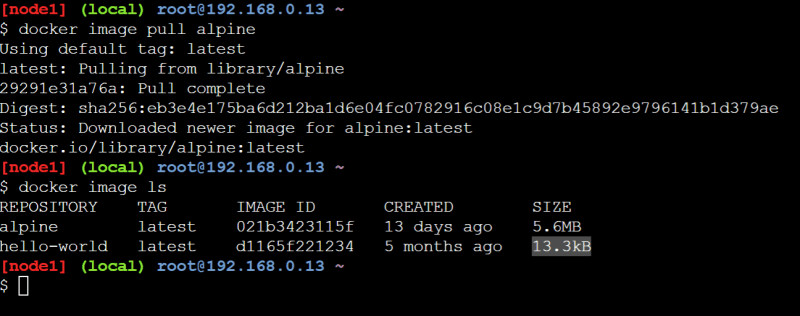
docker container run <docker-image>

docker container run hello-world



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

docker pull <image>



* Let’s try to create a container

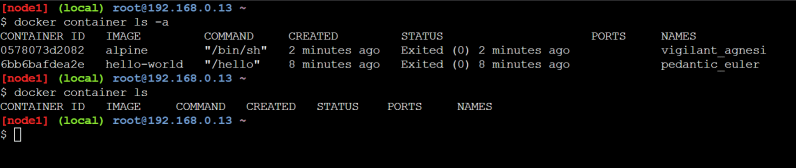
docker container run alpine

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

docker container ls

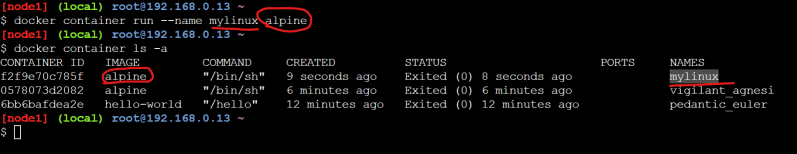
* Now let’s look all the container irrespective of their current state

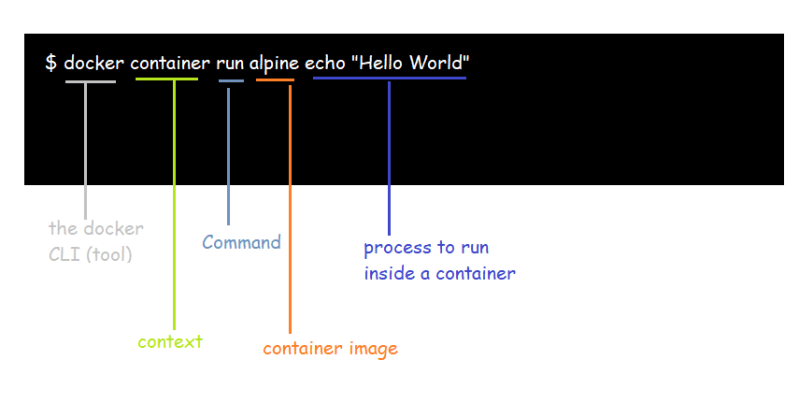
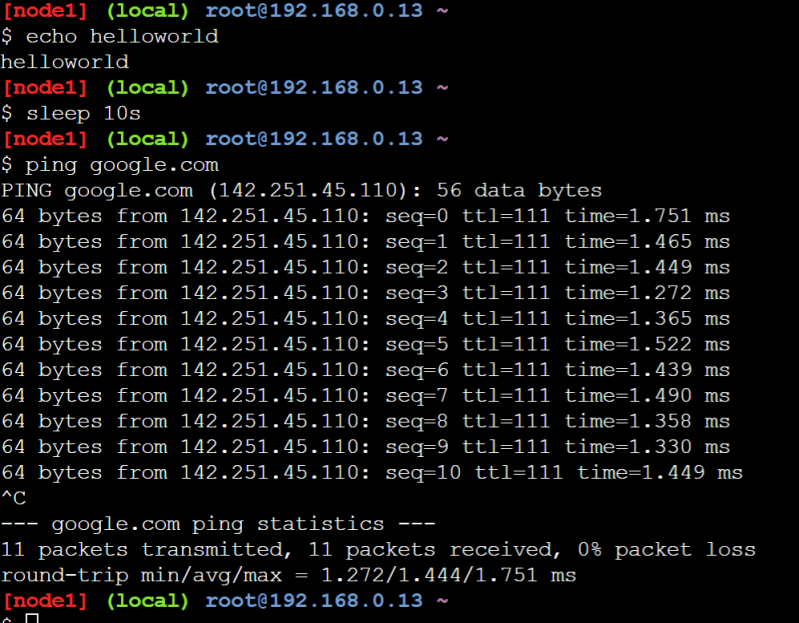
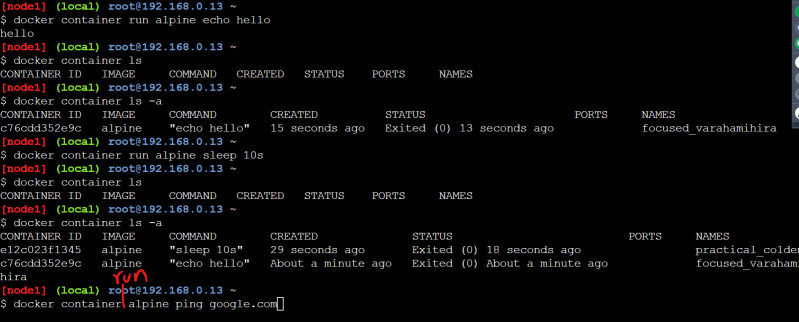
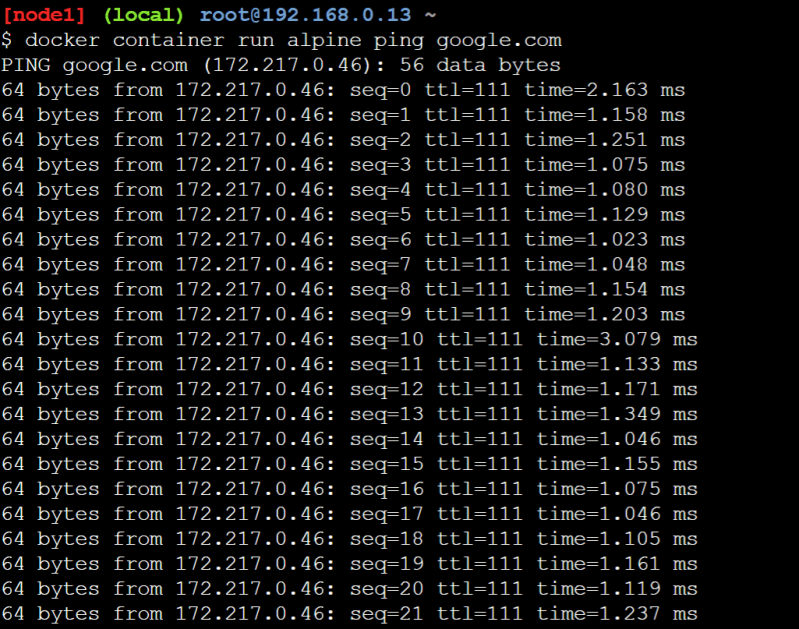
docker container ls -a



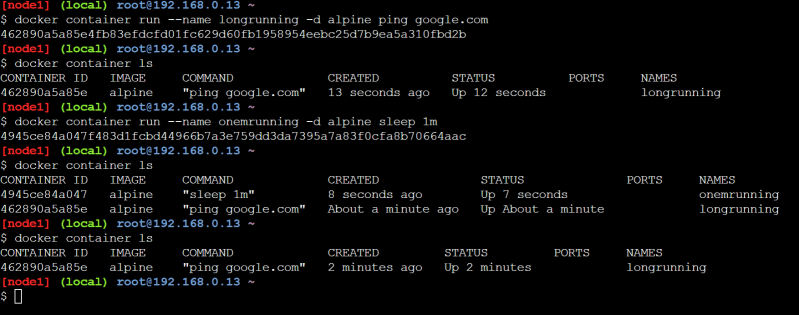
* For every container created docker will create
  + container id
  + container name if not passed
* So let’s to create a container from some image with a name

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



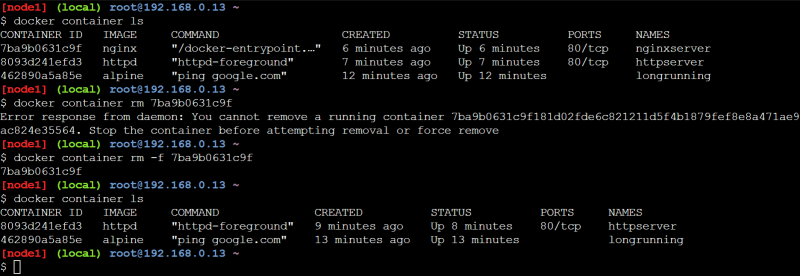
* Let’s dissect the command which we are passing 
* Three basic linux commands which finish immediately, after 10 seconds, till user quits (ctrl+c) 
* Now lets try to run these commands as process inside the alpine container 
* By default when we run docker container, it attaches the standard output 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 container is running, so as long as app in container is running container will also be in running state.
* How to remove containers

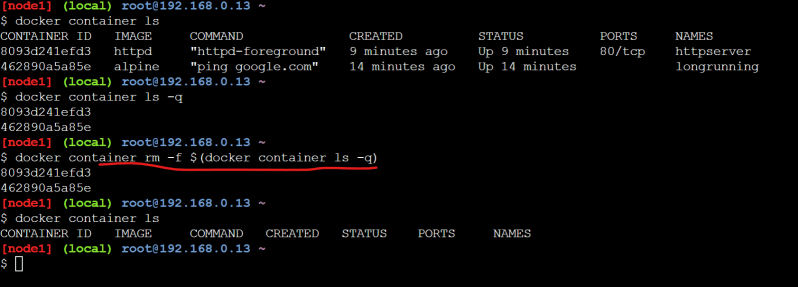
docker container rm <container id or name> # this is for the stopped or exited

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



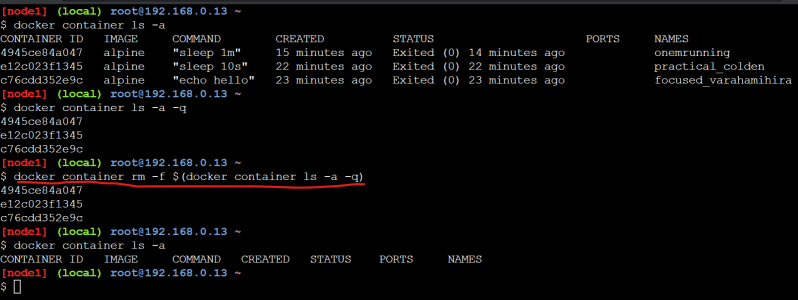
* if you want to remove all the running containers

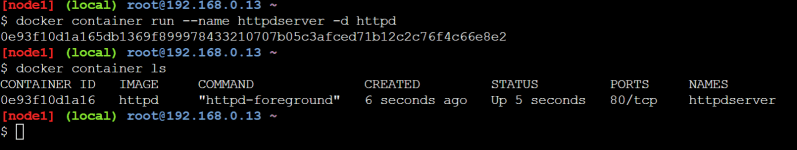
docker container rm -f $(docker container ls -q)



* IF you want to remove all the containers

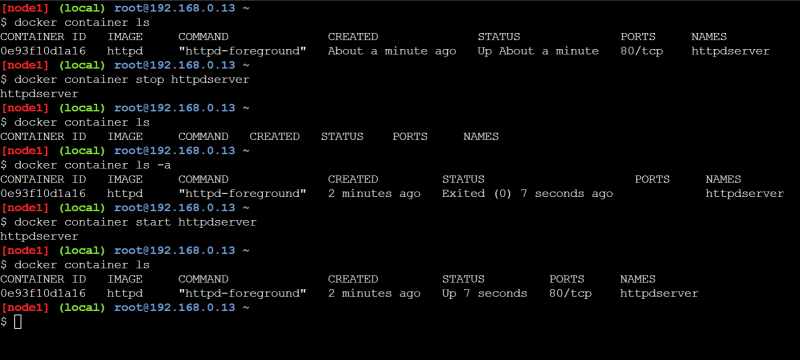
docker container rm -f $(docker container ls -a -q)



* 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>

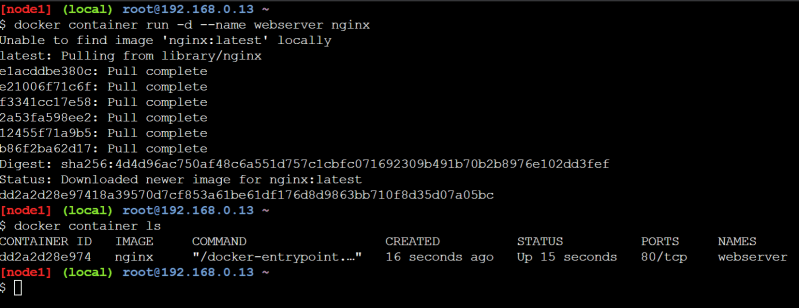
 \*

**Share this:**

AUGUST 22, 2021

# DevOps Classroom Series – 21/Aug/2021

## Containers Continued

* Let’s create a new nginx container 
* 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 let’s try to attach to the container

docker container attach <container-name/id

* 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

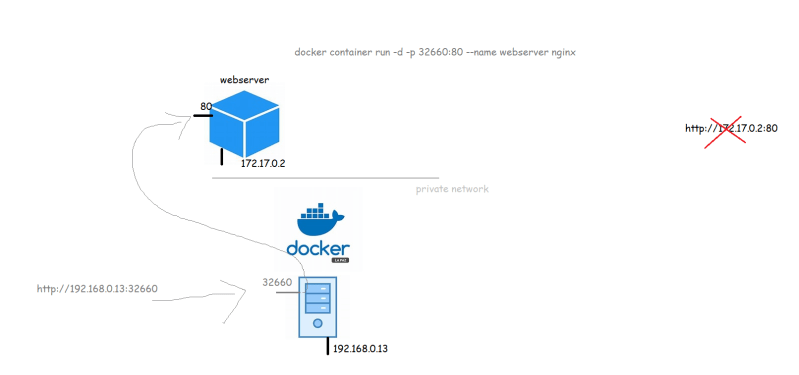
docker container logs <container-name>

* 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>

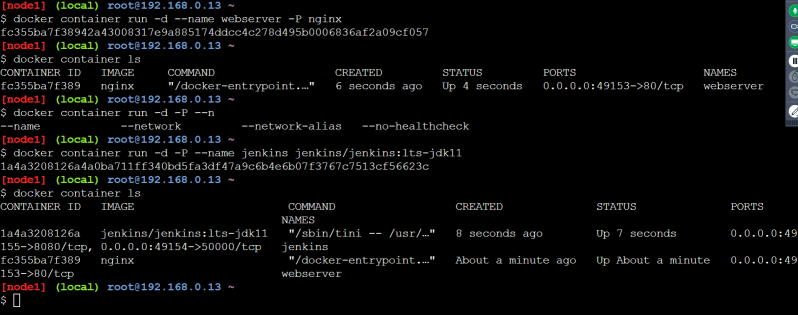
* When we create containers, container run on the internal network, which is not accessible from outside world, but we would still want to access the applications running inside containers.
* So while creating containers we can do port forwarding. This port-forwarding forwards the request received on 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

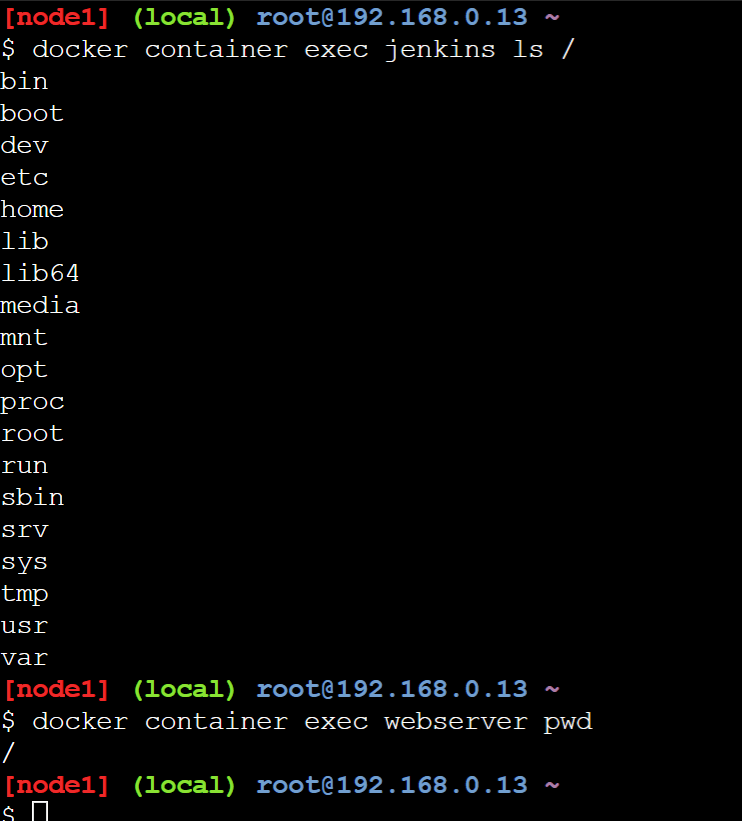
* There is also an option to dynamically create port forwarding

docker container run -P <image>



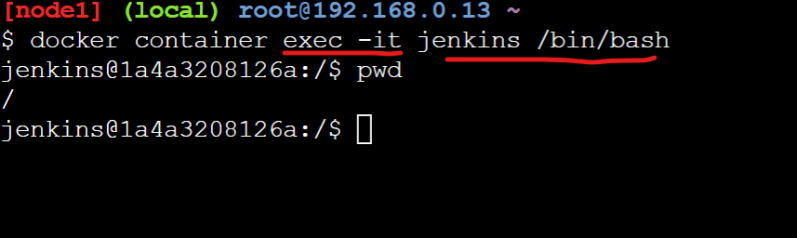
* Executing commands in the docker container

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

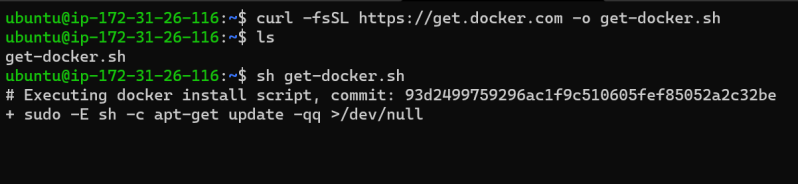
docker container run -it <image-name> <terminal>

## Installing docker

* Let’s 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)
  + Azure VM: [Refer Here](https://www.youtube.com/watch?v=P9X-4Z-NeGg&list=PLuVH8Jaq3mLuqXuGs6aeqxhuvCYSzB1kT&index=2)
* 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/)
  + Mac [Refer Here](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>

AUGUST 23, 2021

# DevOps Classroom Series – 22/Aug/2021

## Docker Internals

* Questions to understand
  + How are container isolations created?
    - Network
    - Process tree
    - File System
  + 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
  + [Refer Here](https://directdevops.blog/2019/01/31/docker-internals/) for 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 c groups (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 application
  + 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

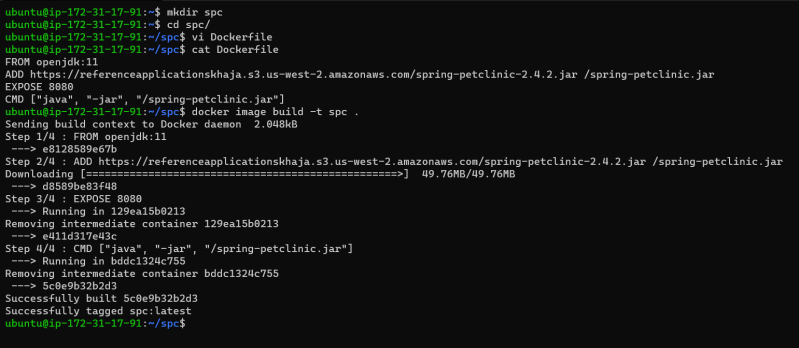
* Now access the application from http using http://<public-ip&gt;:8080
* Let’s see if we can do the same in containers
* Now access the machine where Docker is installed.
* Option 1:
  + Try to search for a docker container with ubuntu image
* docker container run --name exp1 -p 8000:8080 -it ubuntu /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 container are we are able to access the application using http://<host-ip&gt;:8000
* Option 2:
  + Searching for open jdk 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 container 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 image’s choosing a base container with necessary software is installed (option 2) will be sensible.
* In Docker if you want to create images we create Docker file, 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 Docker file 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

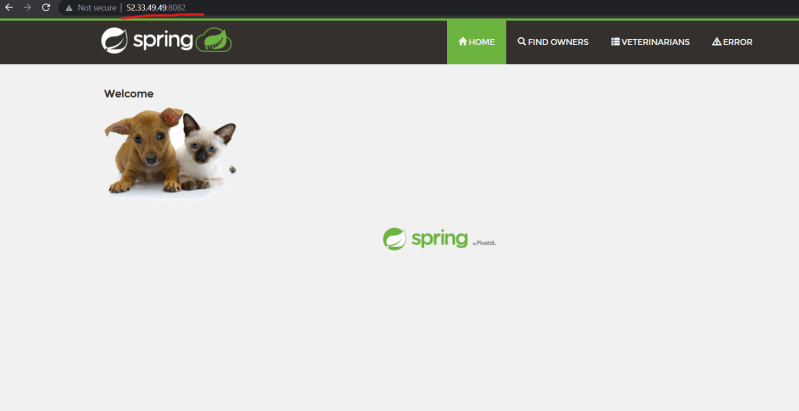
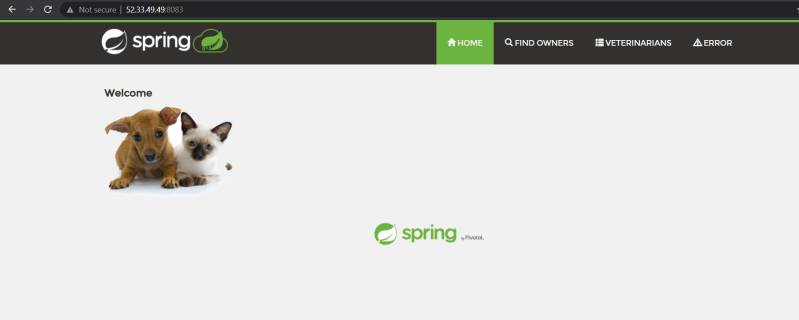
EXPOSE 8080

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

### Share this:

AUGUST 24, 2021

# DevOps Classroom Series – 24/Aug/2021

## Docker file Instructions and Image Creations

* To create a docker container we need an image.
* Applications will have different versions, to accommodate this in docker container images, Docker images have tags.
* The syntactic name of docker image <image-name>:<tag-name>.
* If we do not 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 “Docker file“ 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/) for the Docker file instructions and reference
* Let’s 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 without any specific base image FROM scratch
    - [Refer Here](https://docs.docker.com/engine/reference/builder/#from)
  + LABEL: Adds metadata to the image [Refer Here](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 in for Docker that the container is listening to the specific network 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)
* The Docker file for spring pet clinic 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

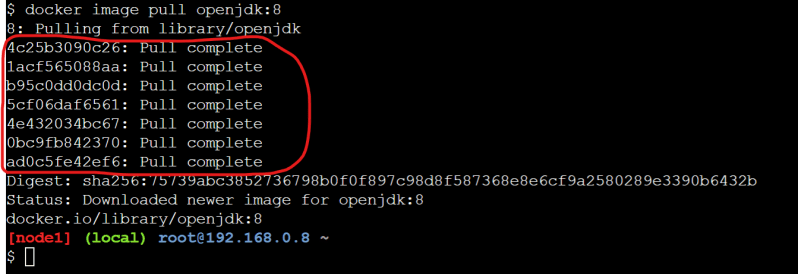
ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

### Share this:

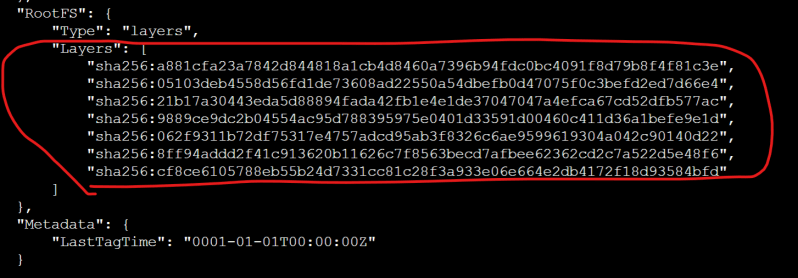
AUGUST 30, 2021

# DevOps Classroom Series – 30/Aug/2021

## Docker Image Layers and Containers

* When we pull the docker image we see some strange id’s 
* Now execute the command docker image inspect <image-name>

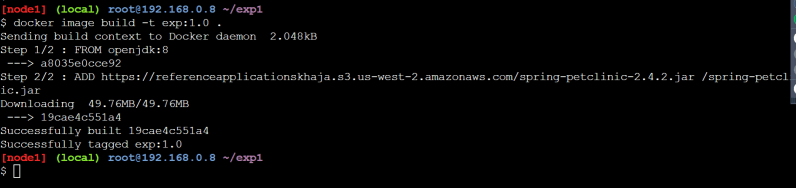
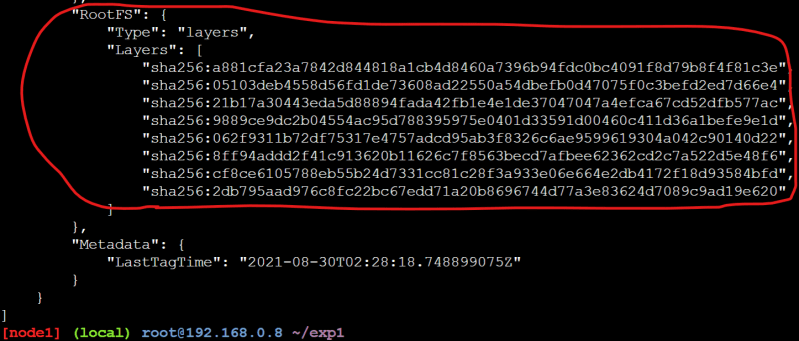
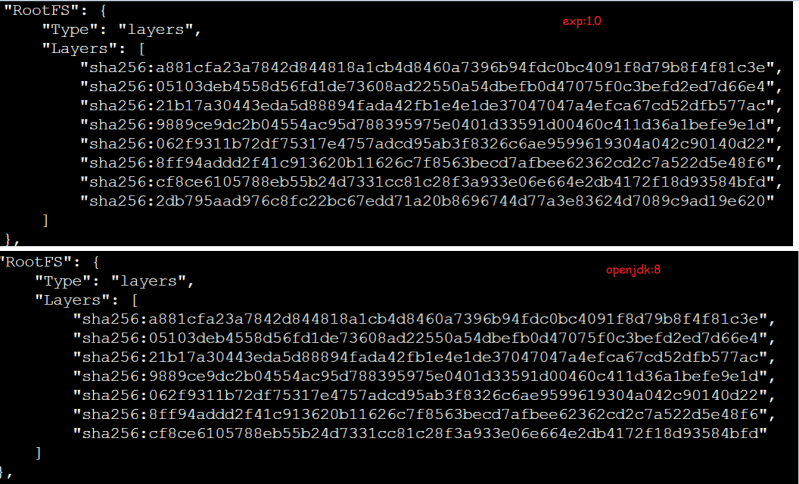
docker image inspect openjdk:8



* Docker image is collection of image layers.
* Now let’s try to build our own image with openjdk:8 as base image

FROM openjdk:8

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

* Now lets build the docker image with name exp:1.0 
* Now let’s inspect exp:1.0 
* Now let’s compare layers of open jkd and exp:1.0 
* Image layers are reused across Docker images.
* Any instruction in Docker file, that adds/changes existing content will lead to a new layer creation i.e. EVERY ADD/RUN/COPY instruction will create one extra layer (in most of the cases)
* Each layer consists of the changes which are done and Now Docker will mount all of the image layers as one/more mounts to the container
* Docker layers are mounted on top of each other and a writable layer is created with the help of Storage Drivers
* [Refer Here](https://directdevops.blog/2019/09/27/impact-of-image-layers-on-docker-containers-storage-drivers/) for the article on layers and storage driver
* Docker has a concept of docker volumes where we can preserve the data that has been created/modified by container.

AUGUST 31, 2021

# DevOps Classroom Series – 31/Aug/2021

## Docker file Instructions

* We have looked into
  + FROM
  + ADD/COPY
  + CMD
  + LABEL
  + EXPOSE
* RUN instruction: This instruction executes any valide command

FROM ubuntu:xenial

RUN apt update && apt install apache2

* Lets try to create a docker image with lamp stack [Refer Here](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04)

FROM ubuntu:bionic

LABEL author='khaja'

FROM ubuntu:bionic

LABEL author='khaja'

RUN export DEBIAN\_FRONTEND=noninteractive \

&& apt update && apt install apache2 -y \

&& apt install php libapache2-mod-php php-mysql -y \

&& apt install php-cli -y \

&& echo "<?php phpinfo();>" > /var/www/html/info.php

* Using local file and copy would be better

FROM ubuntu:bionic

LABEL author='khaja'

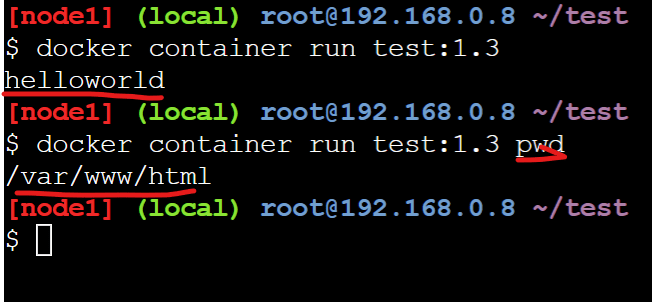
RUN export DEBIAN\_FRONTEND=noninteractive \

&& apt update && apt install apache2 -y \

&& apt install php libapache2-mod-php php-mysql -y \

&& apt install php-cli -y

COPY info.php /var/www/html/info.php

* WORKDIR: This instruction defines the working directory. If not specified / is the default workdir
* CMD:
  + This instruction is executed when the container is created and can be overritten by passing arguments when creating container 
* ENTRYPOINT:
  + This instruction is executed when the container is created and cannot be over written by passing arguments when creating container
  + Generally in entry point we give executable path and in CMD we give arguments for the containers where the command to be executed when starting containers is fixed.

java -jar spc.jar

Give flexibility to run anything during creating container

CMD ["java", "-jar", "spc.jar"]

Dont give flexibility

ENTRYPOINT ["java", "-jar", "spc.jar"]

Give option on arguments to be passed

ENTRYPOINT ["java"]

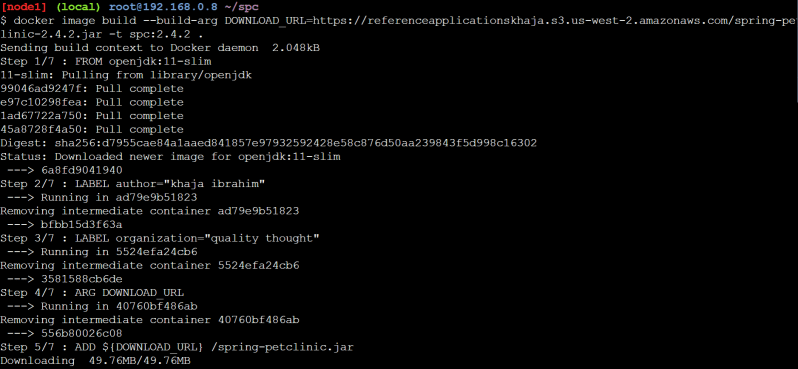
CMD ["-jar", "spc.jar"]

* We should be using ENTRYPOINT/CMD instructions to wait till the application is running
* When we run applications we have two types of startups
  + Command’s which will start the application will not return till the application is killed. For these kind of applications directly add the command to CMD
    - Examples: java -jar <package>.jar, python main.py
  + Command’s which will start the application and will return immediately. For these type of command’s we need to look for alternative so that command will continue running as long as application is running.
    - Examples: service nginx start

SEPTEMBER 1, 2021

# DevOps Classroom Series – 01/Sept/2021

## Docker file contd..

* ARG :
  + This instruction can be helpful in defining the variable and pass argument during build time
  + The arguments can be passed from docker command line “`docker image build –build-arg <arg>=value
  + [Refer Here](https://github.com/asquarezone/DockerZone/commit/4f65ccdb4ba8036e66d5bc6c11ed053dbf23683d) for the changes 
* ENV: This instruction sets the environmental variable and is available while running the container and this can be changed while running your container
* Refer examples done in the class
* Exercise: Try to fix the container not starting issue with the following docker file

FROM openjdk:11-slim

LABEL author="khaja ibrahim"

LABEL organization="quality thought"

ARG DOWNLOAD\_URL="https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar"

ENV APPLICATION\_PATH="/spring-petclinic.jar"

ENV TEST\_ENV="hello"

ADD ${DOWNLOAD\_URL} ${APPLICATION\_PATH}

EXPOSE 8080

CMD [ "java", "-jar", ${APPLICATION\_PATH} ]

* ESCAPE: The escape directive sets the escape characters in the Dockerfile. If not specified the default escape is \

#escape=`

* Sample dockerfile

#escape=`

FROM ubuntu:bionic

LABEL author='khaja'

RUN export DEBIAN\_FRONTEND=noninteractive `

&& apt update && apt install apache2 -y `

&& apt install php libapache2-mod-php php-mysql -y `

&& apt install php-cli -y

COPY info.php /var/www/html/info.php

WORKDIR /var/www/html

EXPOSE 80

ENTRYPOINT [ "echo" ]

CMD [ "helloworld" ]

* Exercise: There is a sample python code [Refer Here](https://github.com/khajasampleapps/basicflask)
* Manual Steps:
  + Ensure python is installed
  + Ensure pip is installed
  + clone the code from git hub git clone <https://github.com/khajasampleapps/basicflask.git>
  + Execute pip install -r requirements.txt
  + To run your application python app.py
* Try to create a Dockerf ile with all the necessary instructions to create an image for basic flask
* Few sessions earlier we have created a Docker file for gameoflife, which needed tomcat to work.

FROM tomcat:jdk8

LABEL author="khaja"

EXPOSE 8080

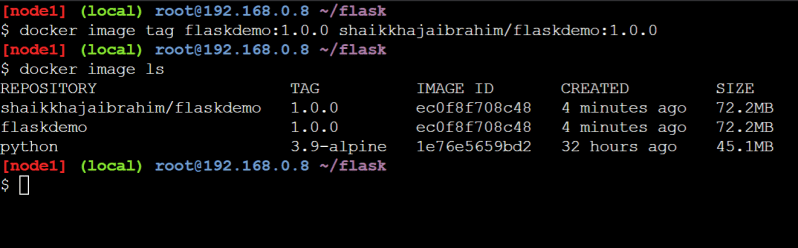
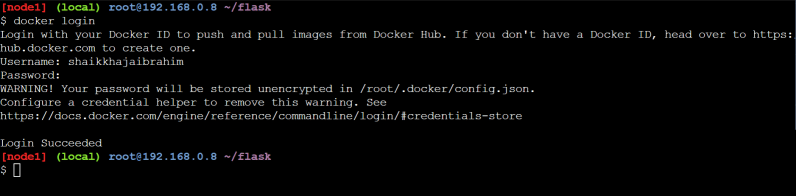
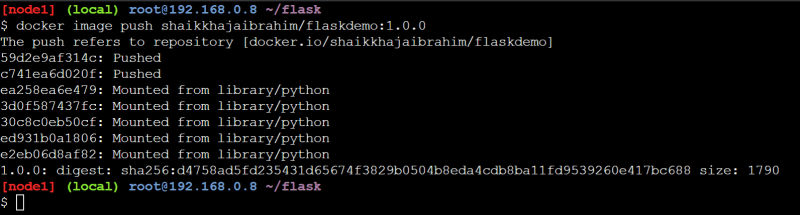
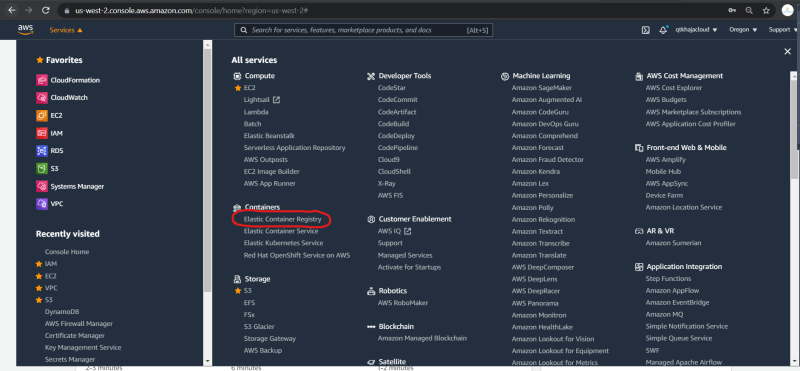
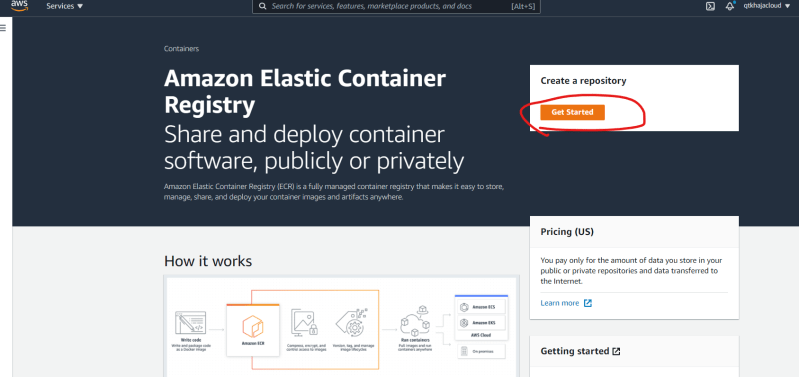
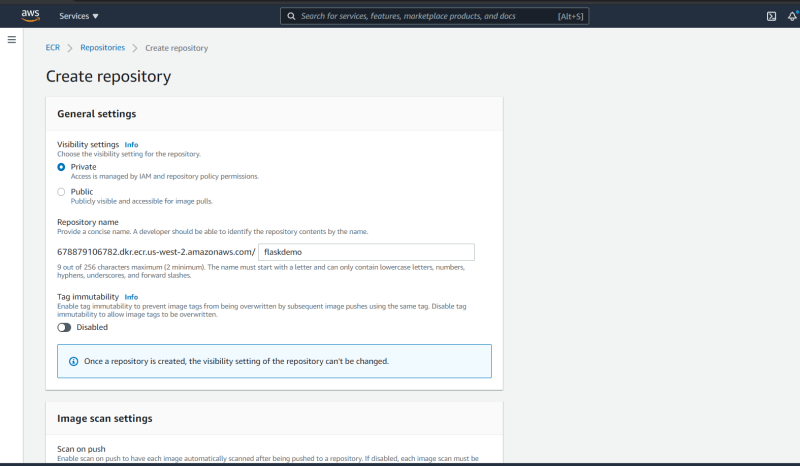
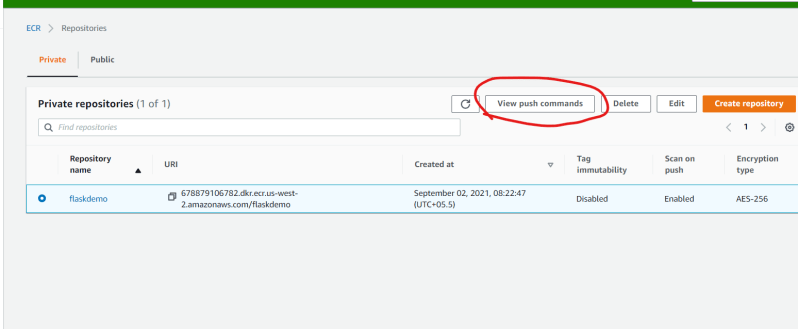
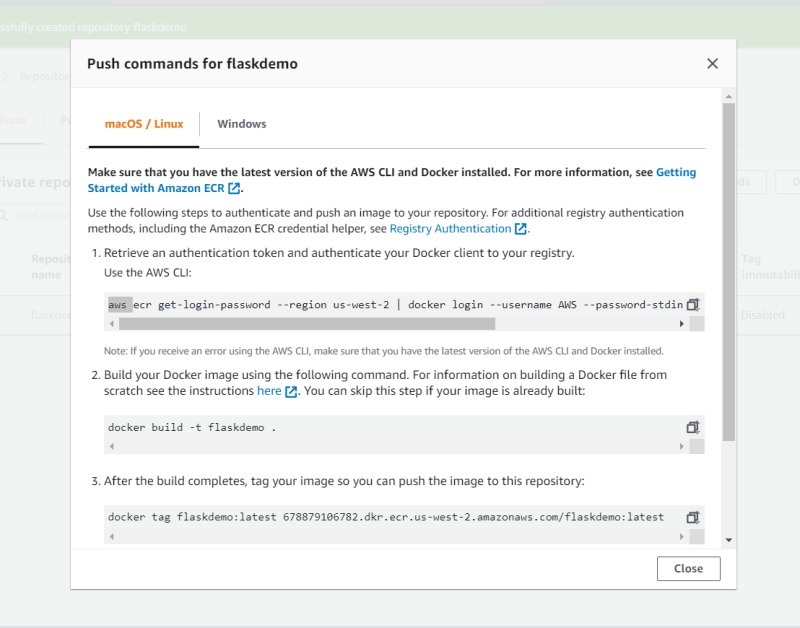
ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

* In this Docker file, we have not written ENTRYPOINT/CMD because we want to use base images CMD.

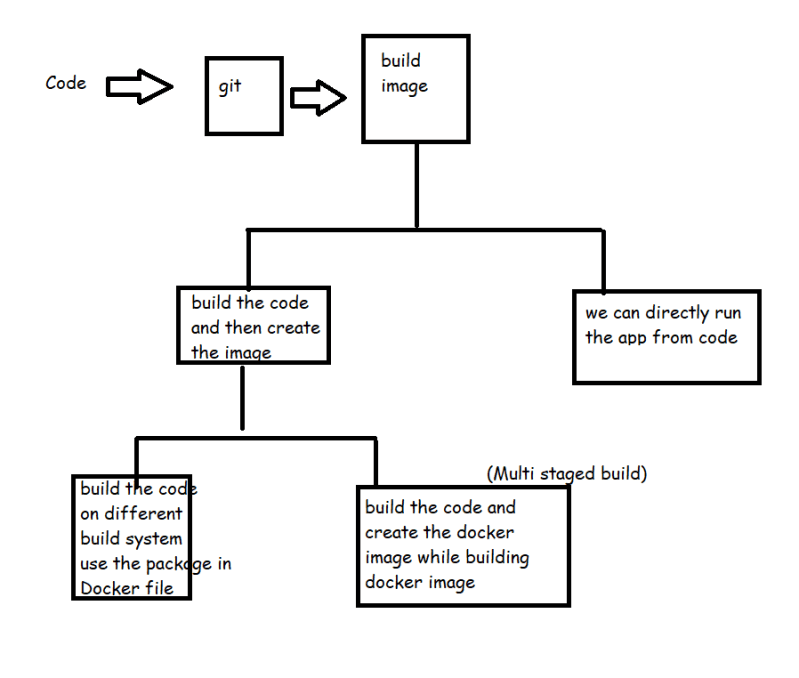
SEPTEMBER 2, 2021

# DevOps Classroom Series – 02/Sept/2021

## Docker image building

* Manual Steps:
  + Ensure python is installed
  + Ensure pip is installed
  + clone the code from github git clone <https://github.com/khajasampleapps/basicflask.git>
  + Execute pip install -r requirements.txt
  + To run your application python app.py
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/6b2f21b2bac6e166912682a9e6e3373c5499d36e) for the Dockerfile
* Now if we want to make this image available to others, we need to push the image to the registry.
* Generally the format for registry is <username>/<image>:<tag>
* DockerHub:
  + This is a public registry and also it has private registry option 
  + Execute the command docker login 
  + Now push the tagged image 
  + Now any one can run this flask application
* docker container run -d -P shaikkhajaibrahim/flaskdemo:1.0.0
* Other Registries
  + AWS ECR: This supports public and private registries hosted on AWS.
    - To work with this install AWS CLI [Refer Here](https://www.youtube.com/watch?v=HSScHRcO0CA&list=PLuVH8Jaq3mLszrC7lv68a0VcrDripW-HK&index=4)     
  + Azure Container Registry: This is registry hosted on azure [Refer Here](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-portal)
    - Install Azure Cli [Refer Here](https://www.youtube.com/watch?v=Aws6h-Fjmfw&list=PLuVH8Jaq3mLuqXuGs6aeqxhuvCYSzB1kT&index=4)
  + Artifactory/jfrog: [Refer Here](https://www.jfrog.com/confluence/display/JFROG/Docker+Registry)

## Multi Staged Builds

* Overview: 
* Lets try to build the springpetclinic
* Manual steps:
  + this needs jdk 11
  + We need maven
* git clone https://github.com/spring-projects/spring-petclinic.git
* cd spring-petclinic
* mvn package
* # this generates the jar file
* Dockerfile with multistaged build

FROM maven:3-openjdk-11 AS builder

RUN git clone https://github.com/spring-projects/spring-petclinic.git \

&& cd spring-petclinic \

&& mvn package

FROM openjdk:11

LABEL author="khaja"

EXPOSE 8080

COPY --from=builder /spring-petclinic/target/spring-petclinic-2.5.0-SNAPSHOT.jar /spring-petclinic.jar

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

* Lets try to build the gameoflife
* Manual steps:
  + This requires jdk 8
* git clone https://github.com/wakaleo/game-of-life.git
* cd game-of-life
* mvn package
* Exercise: Create a mutli stage build for game of life and spring pet clinic and push the image to docker hub and any private registry.

### Share this:

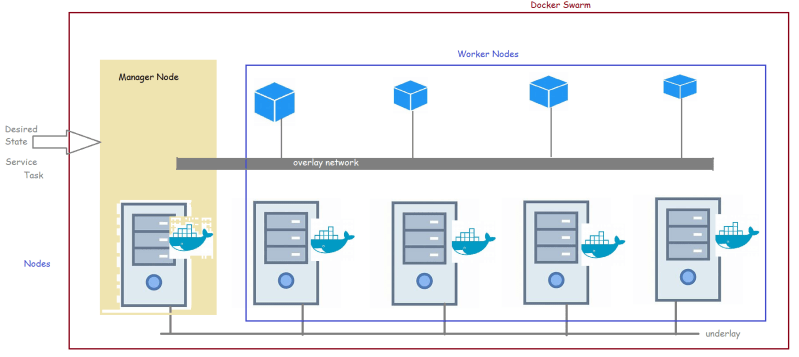
SEPTEMBER 5, 2021

# DevOps Classroom Series -04/Sept/2021

## Docker Volumes

* [Refer Here](https://directdevops.blog/2019/10/03/docker-volumes/) for the article on docker volumes
* The problem in containers is with the write layer of the container. This layer gets deleted when the container is deleted.
* If we are running any application on the container & the data generated by contianer is important or should be available even after the container is deleted, we need to use Docker Volumes
* Refer to the class room video for the commands/examples done

## Docker Networking

* refer the following articles
  + Networking 1 [Refer Here](https://directdevops.blog/2019/10/05/docker-networking-series-i/)
  + Swarm [Refer Here](https://directdevops.blog/2019/10/07/docker-swarm-mode/)
  + networking 2 [Refer Here](https://directdevops.blog/2019/10/07/docker-networking-series-ii-overlay-networks/)
* Docker swarm 

### Share this:

SEPTEMBER 6, 2021

# DevOps Classroom Series – 05/Sept/2021

## Viewing Docker Logs

* docker container logs can be viewed by the command docker container logs <container name>
* All the logs and errors redirected to STDERR and STDOUT stream
* Docker supports different logging drivers [Refer Here](https://docs.docker.com/config/containers/logging/configure/#supported-logging-drivers)

## Constraints on Docker containers

* By default docker container has no resource restrictions and it can use as much of resources given by the host kernel
* We can restrict memory and cpu while starting a container

docker container run --name memjenkins -P -d --memory 1024m jenkins/jenkins

* now execute docker stats Preview
* similarly
  + –memory-swap
  + –memory-reservation
  + –memory-swappiness
* We can also restrict cpu’s as well

docker container run --name memcpujenkins -P -d --memory 1024m --cpus="2" jenkins/jenkins

## Game of life multi stage build

* Docker file

FROM maven:3-openjdk-8 as builder

RUN git clone https://github.com/wakaleo/game-of-life.git && cd game-of-life && mvn package

FROM tomcat:9.0-jdk8

COPY --from=builder /game-of-life/gameoflife-web/target/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

EXPOSE 8080

## Using ENV in CMD

* Dockerfile

FROM ubuntu

ENV DURATION=1d

CMD ["/bin/bash", "-c", "sleep ${DURATION}"]

## Exercise

* Learn JSON and YAML [Refer Here](https://www.youtube.com/watch?v=ggOmHlnhPaM&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=15)