# Docker

# Pre-requisites:

# Dockerhub account

# Visit <https://hub.docker.com/> and sign up for a new account.

# Github account

# https://github.com/

# AWS EC2 Instance

# Open an SSH client.

# Locate your private key file (singaporekeypair.pem)

# # chmod 400 singaporekeypair.pem

# Now you will be able to SSH using your Public DNS/Public IP

# ssh -i "singaporekeypair.pem" [ec2-user@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com](mailto:ec2-user@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com)

# NB:- If SSH connectivity does not work, see changing the chmod to 600 (chmod 600 singaporekeypair.pem)

# [ec2-user@ip-xx-xx-xx-xx ~]$ sudo su

# # yum install -y git

# Steps to Check out from git repository

# # git init

# # git config --global user.email "mymail@email.com"

# # git config --global user.name " git user name"

# # mkdir gitrepo

# # cd gitrepo

# # git clone https://github.com/nevin-cleetus/kubernetes.git

# Ensure

# 1. Internet is working

# 2. Nobody should be connected to vpn or any other proxy.

# 3. Disable firewall if enabled.

# Docker Installation

# login as: ec2-user

# [ec2-user@ip-xx-xx-xx-xx ~]$ sudo su

# $ sudo yum update -y

# $ sudo yum install -y docker

# Expected output

# Installed:

# docker.x86…

# Dependency Installed:

# Containerd libcgroup.x86\_64

# runc.x86\_64 pigz.x86\_64

# Complete!

# Start and Enable the docker service

# $ sudo service docker start

# $ sudo service docker status

# Make sure the service is Active: active (running)

# $ sudo usermod -a -G docker ec2-user

# $ sudo systemctl enable docker

# Run the following command to activate the changes to groups:

# $ newgrp docker

# $ docker run hello-world

# 

# Expected output: Hello world message from the application.

# Docker Commands

# #docker images List all local images

# #docker ps List all running containers

# #docker container ls List containers

# #docker ps -a List all running and stopped containers

# 

# #docker start <container id> To start a container

# #docker stop <container id> To stop a running container with the passed in container id

# #docker container rm <container id> To remove a container

# #docker container rm <container id> -f To remove a container forcefully

# #docker rmi <image id> To remove an image

# #docker rmi <image id,…> To remove images passed as “,” separated list

# #docker rmi <image id> -f To remove an image forcefully

# Run Nginx webserver as container

# #docker run nginx

# Try to access the below link using the external IP and port 80 and it won’t be accessible.

# http://<Public IP>:80

# Docker containers run as isolated sandboxed environment. In order to access the port from outside of the container, we need to map the port to a host port. This is done by passing port number as argument. Execute the run command by mapping the port as below

# #docker run -p 8000:80 nginx

# Access the link http://<Public IP>:8000 from browser and we should see the Nginx landing page.

# Search Images from docker

# #docker search ubuntu Search all image matching the key word ubuntu

# Pull an Image from docker hub

# #docker pull ubuntu Pull the latest version of ubuntu from docker hub

# Run “docker images” to confirm the ubuntu images is pulled and available locally.

# Run a linux container in interactive mode

# #docker run -it ubuntu bash

# You will connect to the bash shell of the running container. Run a “docker ps” command from another terminal now and you can see the container running.

# #exit

# Execute ‘docker ps’ once again and see the container is no longer running.

# Run a linux container in detached mode

# #docker run -it ubuntu bash

# #docker ps

# Attach to a running container

# #docker run -itd ubuntu top

# #docker ps

# #docker attach <containerid>

# Hit Ctrl-p then Ctrl-q to gracefully detach from the container. If we press Ctrl-C, the container will terminate.

# Connect to a running container using ‘exec’ command

# #docker run -it ubuntu bash

# Don’t exit from the current terminal. Open a new terminal and execute ‘docker ps’ and keep the container id. You will the shell of currently running Ubuntu container

# #docker exec -it <container id> /bin/sh

# #ls -ltr

# #exit

# Docker Commit - Create a new image from a container’s changes

# #docker run -itd ubuntu /bin/sh

# #docker ps

# Use the container Id and exec into the container and Install Open JDK

# #docker exec -it <container id> /bin/sh

# #su

# #apt-get update

# #apt-get install openjdk-8-jdk

# #exit

# #docker commit <container id> <docker hub username>/<image name>:1

# For Exp:- docker commit 6004e8e22493 nevincleetus/myopenjdk:1

# Execute ‘docker images’ and you will see a new image is created with the image name we passed.

# #docker images

# Create a new container from the newly created image and confirm jdk version is available.

# # docker run -it <image id of the newly created image> /bin/sh

# # java version

# openjdk version "1.8.0\_242"

# Push local image to Docker Hub

# #docker login -u <docker-registry-username>

# Input the password in the prompt

# Login Succeeded

# #docker tag <image id> <docker-registry-username>/<image-name>

# #docker push <docker-registry-username>/<image-name>

# For Exp:- docker tag 7f7f0a05a3b4 nevincleetus/myopenjdk:1

# docker push nevincleetus/myopenjdk:1

# Docker push will prompt for the docker hub username and password. Enter the credentials and once the image is pushed to docker repository, go to docker hub repository and confirm the image is available in the repository.

# Docker Image history

# #docker image history <image id>

# The above command shall return details on how the image was built.

# Inspect Docker Image

# #docker image inspect <image id>

# Save/load Images

# #docker image save <image id> > /tmp/myimage.tar

# #docker rmi <image id>

# #docker load -i /tmp/myimage.tar

# The “docker image save” command save one or more image to a tar file. The output tar file can be then used to load and re-create the image. Docker save will produce a tarball, but with all the parent layers and all tags + versions.

# Stop and kill all containers

# Stop ALL containers

# #docker stop $(docker ps -a -q)

# Remove ALL containers

# #docker rm -f $(docker ps -a -q)

# Clean the terminated containers

# #docker container prune -f

# Above command shall remove/clean all stopped containers

# Details of docker containers

# #docker container inspect <container id>

# Resource provisioning

# #docker container run -d --memory-reservation=750m --name mymem1 ubuntu:latest sleep 3600

# - memory-reservation= : The soft limit of memory the container can use.

# #docker container run -d --memory=20m --memory-swap=20m --name myPython python:3-alpine sleep 3600

# -m or --memory= : The maximum amount of memory the container can use.

# #docker container run -d –cpus=2 --name mycpus2 alpine:3.8 /bin/sh

# #docker container run -d --cpu-shares=100 --name mycpu100 alpine:3.8 /bin/sh

# Dockerfile

# Create your Dockerfile

# Open your editor and type the below content. Save it as ubuntu-jdk.dockerfile

# FROM ubuntu

# RUN apt update && apt-get install -y apache2

# RUN apt -y install openjdk-8-jdk

# # docker build -t ubuntu-jdk:1 -f ubuntu-jdk.dockerfile .

# # docker run -it <imageid> /bin/sh

# #java version

# Examples of Dockerfiles using the most commonly used instructions are available under “day\_01\_docker/dockerfiles/” folder for reference.

# Debugging a docker container

# #docker run -itd -p 8000:80 nginx

# #docker ps

# #docker stats <containerid>

# Prints the memory, CPU and IO details.

# #docker top <containerid>

# Prints the top .