Sanjeev Agrawal Global Educational University, Bhopal

Lab Manual
of
Containerization using Dockers
Course Code: CA21B403

Prepared by:
Mr. Bhavesh Shah
School of Computer Application
Session: Spring 2023-24

Index

Sr. No.	Title of Experiment	Scheduled date	Actual date	Remarks
1	Installation of Docker on your base OS using Docker toolbox.			
2	Create an account with docker hub and login.			
3	Install and Launch Docker Desktop (use Docker Desktop Installer) over guest OS (ex-Windows 10) running on a VM with the help of Vmware Workstation.			
4	After logging into Docker Hub, explore and customize settings of docker desktop in your virtual machine to facilitate communication between client(CLI) and server(Docker daemon) by enabling experimental features			
5	Using command line, test and validate the functioning of docker using some basic commands: (a) docker version (b) docker help (c) docker info			
6	As a client, pull and run default docker image 'hello-world' on CLI. Once done, check the following: a. status of container formed at runtime using 'docker info' command b. Get details of image id, size and when image was created.			
7	Log into Docker Hub account with your docker id and create a public repository			
8	Create a Custom Image from a Dockerfile.			
9	Explore kubernetes: a) GKE on google cloud platform b) EKS on AWS			
10	Enable kubernetes in docker desktop and check the working of kubernetes by validating with commands from https://docs.docker.com/get-started/orchestration/ .			
11	Log into play with kubernetes : https://labs.play-with-k8s.com/ with either docker hub or git hub account. Try the steps of kubernetes handson workshop : https://training.play-with-kubernetes.com/kubernetes-workshop/			

Experiment-1

Title: Installation of Docker on your base OS using Docker toolbox.
Theory:

Following steps are needs to follow:

1. **Check System Requirements:**
   Make sure your system meets the minimum requirements for running Docker Toolbox. You need a 64-bit version of Windows 10, at least 4 GB of RAM, and Virtualization Technology enabled in BIOS.

2. **Download Docker Toolbox:**
   Go to the Docker Toolbox GitHub releases page (https://github.com/docker/toolbox/releases) and download the latest version of Docker Toolbox for Windows.

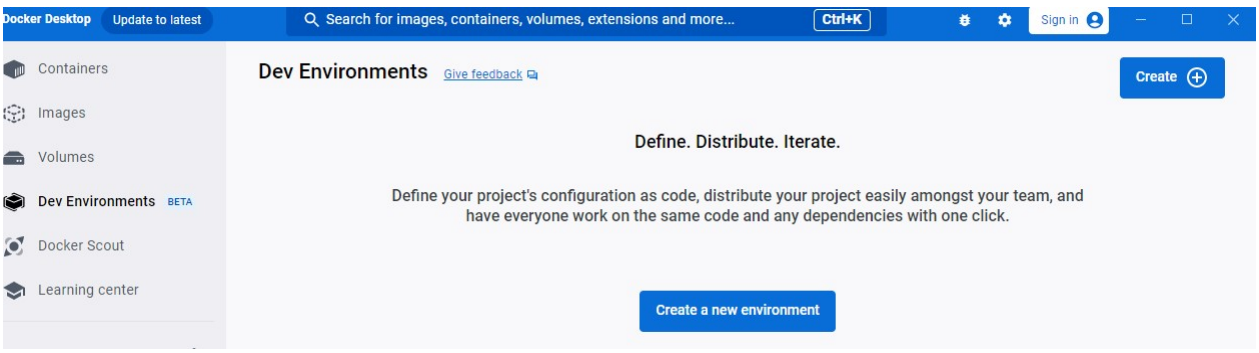
3. **Install Docker Toolbox:**
   Double-click the downloaded installer to start the installation process. Follow the on-screen instructions to complete the installation.

4. **Launch Docker Toolbox:**
   After the installation is complete, launch Docker Toolbox from the Start menu. It will set up a Docker environment for you, including creating a VirtualBox VM called "default".

5. **Initialize Docker Environment:**
   Once Docker Toolbox is launched, it will open a terminal window. Wait for it to finish setting up the environment. It will output some information once it's done.

6. **Test Docker Installation:**
   Open a command prompt or PowerShell window and type docker version to verify that Docker is installed correctly. You should see the version information for both the client and the server.

Output:



Experiment-2

Title: Create an account with docker hub and login.

Theory:

1. Go to Docker Hub: Open your web browser and go to the Docker Hub website: hub.docker.com.

2. Sign Up: On the Docker Hub homepage, you should see a "Sign Up" button at the top right corner. Click on it to begin the registration process.

3. Fill in Details: You'll be asked to provide details such as your username, email address, and password. Fill in the required fields and click on "Sign Up for Docker Hub".

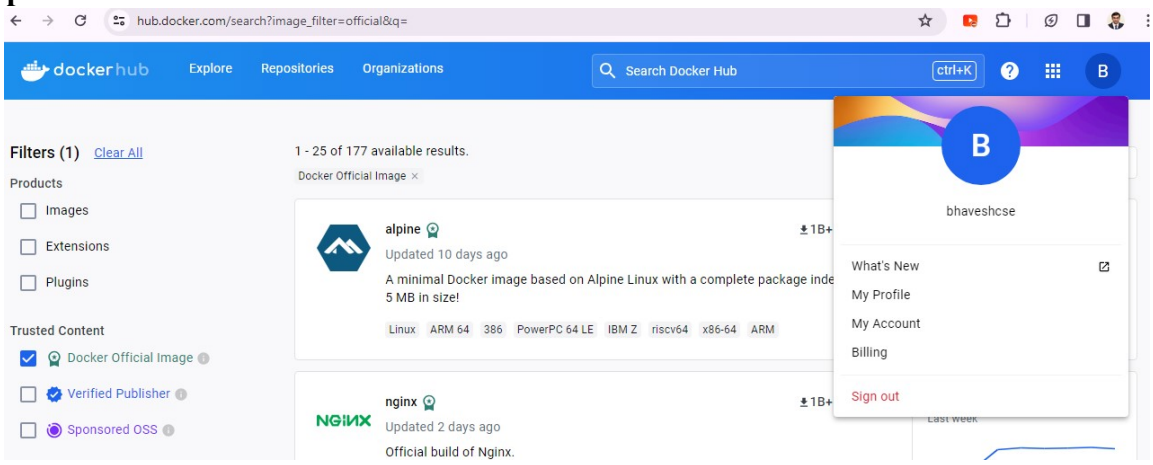
4. Verification: After submitting the sign-up form, you might need to verify your email address. Check your email inbox for a verification email from Docker Hub and follow the instructions provided to verify your account.

5. Login: Once your account is verified, go back to the Docker Hub website. Click on the "Log In" button at the top right corner.

6. Enter Credentials: Enter your username and password that you used during the sign-up process.

7. Login: Click on the "Log In" button, and you should now be logged in to your Docker Hub account.

Output:



Experiment-3

Title: Install and Launch Docker Desktop (use Docker Desktop Installer) over guest OS (ex-Windows 10) running on a VM with the help of Vmware Workstation.

Theory:

1. Download Docker Desktop Installer:
   Go to the Docker website and download the Docker Desktop Installer for Windows.

2. Install VMware Tools (Optional):
   Before proceeding, it's recommended to install VMware Tools on your guest OS (Windows 10). VMware Tools enhance the performance of the virtual machine and allow seamless interaction between the host and guest OS.

3. Start VMware Workstation:
   Launch VMware Workstation and start your Windows 10 virtual machine.

4. Mount Docker Desktop Installer:
   If you downloaded Docker Desktop Installer on your host machine, you need to make it accessible within the virtual machine. You can do this by either copying the installer to the virtual machine's filesystem or by mounting a shared folder that contains the installer.

5. Run Docker Desktop Installer:
   Navigate to the location where you saved the Docker Desktop Installer within your Windows 10 guest OS. Double-click on the installer to start the installation process.

6. Follow Installation Steps:
   Follow the installation wizard steps provided by the Docker Desktop Installer. You may need to approve system prompts and agree to the terms and conditions.

7. Complete Installation:
   Once the installation is complete, Docker Desktop should be installed on your Windows 10 guest OS.

8. Launch Docker Desktop:
   After installation, Docker Desktop should automatically launch. If not, you can find and launch it from the Start menu or desktop shortcut.

9. Verify Installation:
   To verify that Docker Desktop is installed and running correctly, you can open a command prompt or PowerShell window and run the command docker --version to check the Docker version.

10. Start Using Docker:
    Docker Desktop should now be ready to use within your Windows 10 virtual machine. You can start pulling, building, and running Docker containers as needed.

Experiment-4

Title: After logging into Docker Hub, explore and customize settings of docker desktop in your virtual machine to facilitate communication between client(CLI) and server(Docker daemon) by enabling experimental features Using command line, test and validate the functioning of docker using some basic commands: (a) docker version (b) docker help (c) docker info

Theory:

C:\Users\bhavesh>docker -v

Docker version 24.0.6, build ed223bc

C:\Users\bhavesh>docker --help

```
Usage:  docker [OPTIONS] COMMAND
A self-sufficient runtime for containers
Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
```

C:\Users\bhavesh>docker info

```
Client:
 Version:    24.0.6
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.11.2-desktop.5
    Path:     C:\Program Files\Docker\cli-plugins\docker-buildx.exe
  compose: Docker Compose (Docker Inc.)
    Version:  v2.23.0-desktop.1
    Path:     C:\Program Files\Docker\cli-plugins\docker-compose.exe
  dev: Docker Dev Environments (Docker Inc.)
    Version:  v0.1.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-dev.exe
  extension: Manages Docker extensions (Docker Inc.)
    Version:  v0.2.20
    Path:     C:\Program Files\Docker\cli-plugins\docker-extension.exe
  init: Creates Docker-related starter files for your project (Docker Inc.)
    Version:  v0.1.0-beta.9
    Path:     C:\Program Files\Docker\cli-plugins\docker-init.exe
```

Experiment-5

Title: As a client, pull and run default docker image ‘hello-world’ on CLI. Once done, check the following:

a. status of container formed at runtime using ‘docker info’ command

b. Get details of image id, size and when image was created.

Theory:

C:\Users\bhavesh>docker info

```
Client:
 Version:    24.0.6
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.11.2-desktop.5
    Path:     C:\Program Files\Docker\cli-plugins\docker-buildx.exe
```

C:\Users\bhavesh>docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	20.04	18ca3f4297e7	13 days ago	72.8MB

Experiment-6

Title: Log into Docker Hub account with your docker id and create a public repository.

Theory:

To log into Docker Hub and create a public repository, follow these steps:

1. Install Docker Desktop (if not already installed):
   Ensure that Docker Desktop is installed and running on your system. If you haven't installed it yet, you can follow the steps provided earlier to install Docker Desktop.

2. Launch Docker Desktop:
   Open Docker Desktop on your system. You should see the Docker icon in the system tray or menu bar.

3. Log into Docker Hub:
   Click on the Docker icon in the system tray or menu bar to open the Docker Desktop menu. Then, click on "Sign in / Create Docker ID" and enter your Docker ID credentials to log into Docker Hub.

4. Create a Public Repository:
   Once you're logged in, you can create a new repository by clicking on your Docker ID in the Docker Desktop menu and selecting "Create Repository" or by visiting the Docker Hub website and navigating to the "Repositories" section. Choose to create a new repository and provide a name for your repository. Make sure to set the visibility to "Public" if you want it to be accessible to everyone.

5. Configure Repository Settings (Optional):
   You can optionally configure additional settings for your repository, such as description, tags, and automated builds, depending on your requirements.

6. Create the Repository:
   Once you've configured the settings for your repository, click on the "Create" or "Save" button to create the repository.

7. Verify Repository Creation:
   After creating the repository, you should see it listed in your Docker Hub account or in the Docker Desktop menu, depending on where you created it. You can now start pushing Docker images to this repository or share the repository URL with others.

Experiment-7

Title: Create a Custom Image from a Dockerfile.

Theory:

Dockerfile

```
# Use an official Node.js runtime as the base image
FROM node:14

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port the app runs on
EXPOSE 3000

# Define environment variable
ENV NODE_ENV=production

# Command to run the application
CMD ["npm", "start"]
```

This Dockerfile assumes that your Todo App is a Node.js application. Here's a breakdown of what each line does:

FROM node:14: This line tells Docker to use the official Node.js Docker image version 14 as the base image.

WORKDIR /usr/src/app: Sets the working directory within the container to /usr/src/app.

COPY package*.json ./: Copies package.json and package-lock.json from the current directory (where your Dockerfile is located) to the working directory in the container.

RUN npm install: Installs the dependencies specified in package.json.

COPY . .: Copies the rest of the application code from the current directory to the working directory in the container.

EXPOSE 3000: Exposes port 3000 to allow connections to the Todo App from outside the container.

ENV NODE_ENV=production: Sets the NODE_ENV environment variable to production.

CMD ["npm", "start"]: Specifies the command to run the application when the container starts.

In this case, it runs npm start, assuming that you have a script named "start" defined in your package.json.

Experiment-8

Title: Explore kubernetes: a) GKE on google cloud platform b) EKS on AWS.

Theory:

Google Kubernetes Engine (GKE):
Google Kubernetes Engine (GKE) is a managed Kubernetes service provided by Google Cloud Platform (GCP). It allows users to deploy, manage, and scale containerized applications using Kubernetes without the need to manage the underlying infrastructure. Some key features of GKE include:

1. Managed Kubernetes Control Plane: GKE manages the Kubernetes control plane, including master nodes, etcd storage, and control plane components, ensuring high availability and reliability.

2. Node Management: GKE allows users to create and manage clusters of virtual machine instances (nodes) to run their containerized workloads. GKE automatically provisions, upgrades, scales, and repairs these nodes.

3. Integration with Google Cloud Services: GKE integrates seamlessly with other Google Cloud services such as Cloud Logging, Cloud Monitoring, Cloud IAM, and more, making it easier to build, deploy, and manage applications on GCP.

4. Security and Compliance: GKE provides built-in security features such as identity and access management (IAM), network policies, encryption at rest and in transit, and compliance certifications to ensure the security and compliance of containerized workloads.

5. Auto-scaling: GKE supports horizontal pod autoscaling (HPA) and cluster autoscaling, allowing users to automatically scale their applications based on CPU utilization, memory usage, or custom metrics.

Amazon Elastic Kubernetes Service (EKS):
Amazon Elastic Kubernetes Service (EKS) is a managed Kubernetes service provided by Amazon Web Services (AWS). It enables users to deploy, manage, and scale containerized applications using Kubernetes on AWS infrastructure. Some key features of EKS include:

1. Managed Kubernetes Control Plane: EKS manages the Kubernetes control plane, including master nodes, etcd storage, and control plane components, ensuring high availability and reliability.

2. Node Management: EKS allows users to create and manage clusters of Amazon Elastic Compute Cloud (EC2) instances (nodes) to run their containerized workloads. Users have full control over the EC2 instances and can customize them as needed.

3. Integration with AWS Services: EKS integrates with other AWS services such as Amazon Elastic Block Store (EBS), Amazon Virtual Private Cloud (VPC), AWS Identity and Access Management (IAM), and more, making it easier to build, deploy, and manage applications on AWS.

4. Security and Compliance: EKS provides built-in security features such as IAM integration, network policies, encryption at rest and in transit, and compliance certifications to ensure the security and compliance of containerized workloads.

5. Auto-scaling: EKS supports horizontal pod autoscaling (HPA) and cluster autoscaling, allowing users to automatically scale their applications based on CPU utilization, memory usage, or custom metrics.

Both GKE and EKS offer similar capabilities for running Kubernetes workloads on their respective cloud platforms, and the choice between them often depends on factors such as familiarity with the platform, existing infrastructure, pricing, and specific requirements of the application.