**Introduction to Containers w/ Docker, Kubernetes & OpenShift**

Important Terminologies:-

1. Docker Image:-

**What it is**: A lightweight, standalone, and executable software package that includes everything needed to run a piece of software (code, runtime, libraries, dependencies, and configurations).

1. Docker Container:-

**What it is**: A runtime instance of a Docker image. It's a lightweight, isolated, and portable unit where applications run.

**Analogy**: If an image is a recipe, a container is the dish prepared from it.

1. Docker file:-

**What it is**: A text file containing a series of instructions for building a Docker image.

4. Docker Hub:-

**What it is**: A public registry (repository) where Docker images are stored and shared.

**Purpose**: Allows developers to pull prebuilt images or push their custom images.

### 5. Volume:-

**What it is**: A persistent storage mechanism used to store data generated and used by Docker containers.

### 6. Network:-

**What it is**: Docker's way of handling communication between containers and with the external world.

**Purpose**: Provides isolation and connectivity for containers.

**Types**:

* + **Bridge** (default): Allows containers to communicate on the same host.
  + **Host**: Removes isolation between the container and the host.
  + **Overlay**: Connects containers across multiple hosts.

### 7. Registry

* **What it is**: A storage and distribution system for Docker images.
* **Purpose**: Hosts Docker images privately or publicly. Docker Hub is a public registry, but you can set up your private registry too.

### 8. Docker Compose

* **What it is**: A tool for defining and running multi-container Docker applications using a YAML file.
* **Purpose**: Simplifies the management of multi-container environments.
* **Example docker-compose.yml**:

version: '3'

services:

  web:

    image: nginx

    ports:

      - "8080:80"

  db:

    image: mysql

    environment:

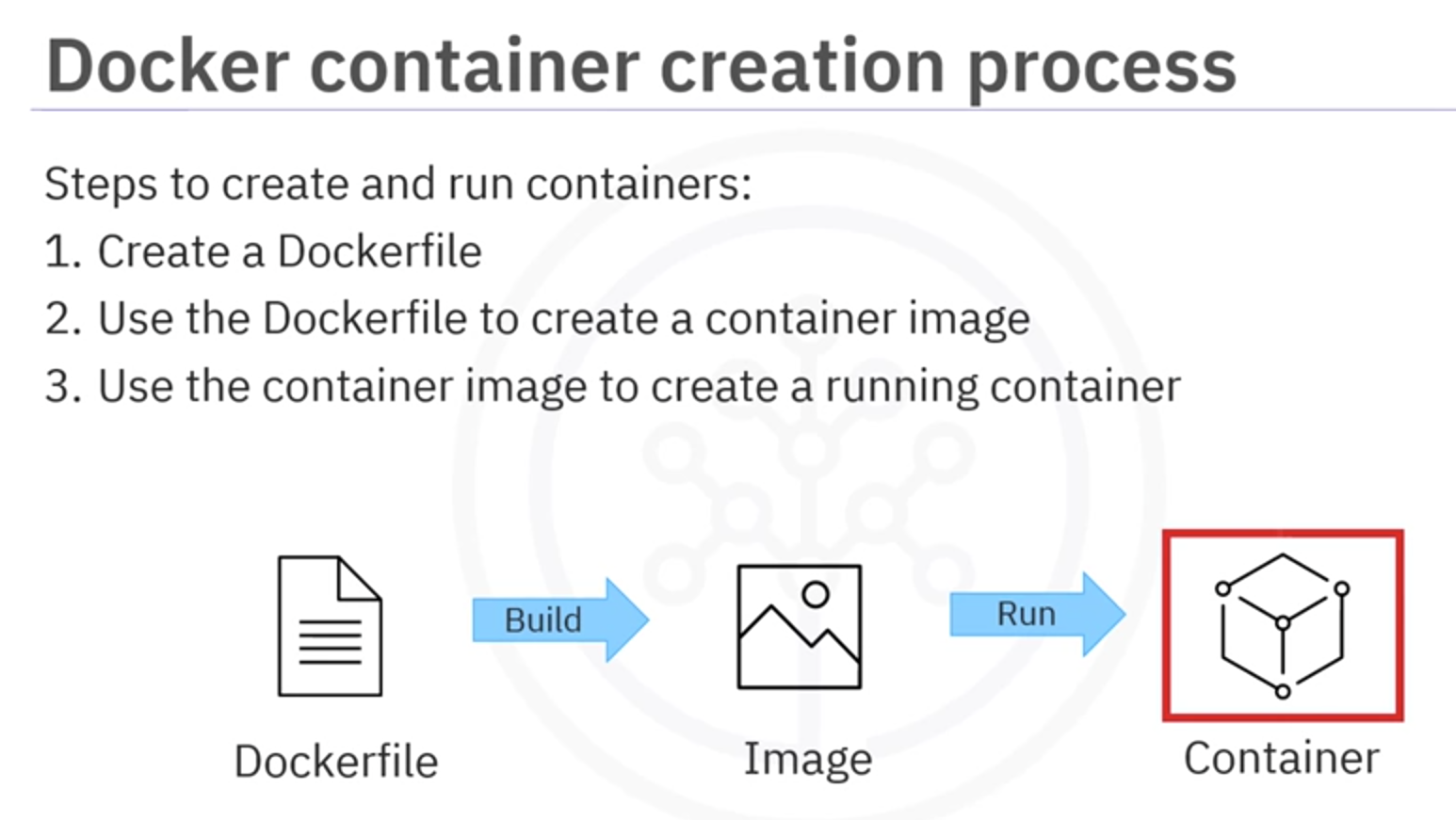
      MYSQL\_ROOT\_PASSWORD: root

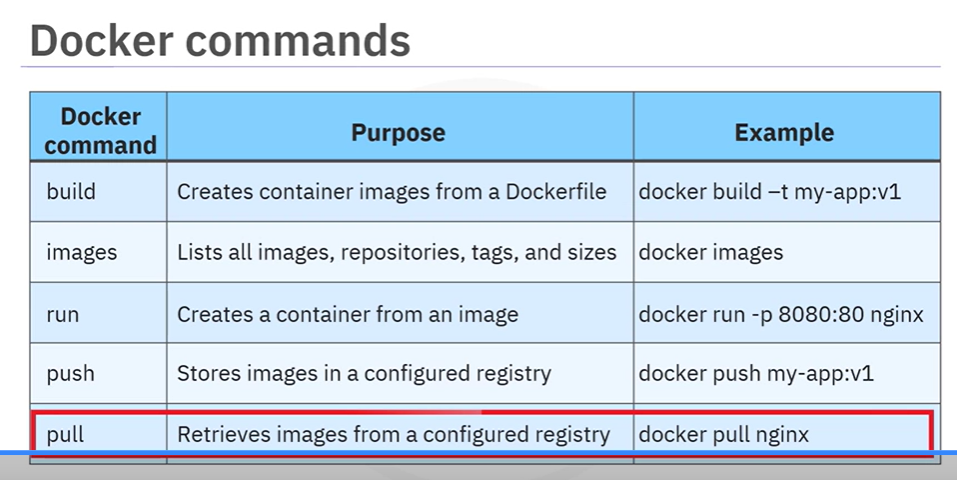
### 9. Kubernetes

* **What it is**: An open-source container orchestration tool, often used alongside Docker.
* **Purpose**: Automates deployment, scaling, and management of containerized applications.
* **Relation to Docker**: Docker containers are often managed by Kubernetes.

### 10. **Docker CLI**

* **What it is**: The command-line interface used to interact with the Docker Daemon.
* **Example Commands**:
  + docker run
  + docker build





/\*

Challenge:

1. Write a function that searches for an employee in 'organizationData'. The function should recursively traverse the nested objects and find all employees with a specified ID.

Stretch Goal:

💪 Complete the challenge without declaring any variable in the global scope.

\*/</body>

------------------------------------------------------------

To generate this message, Docker took the following steps:

 1. The Docker client contacted the Docker daemon.

 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

    (amd64)

 3. The Docker daemon created a new container from that image which runs the

    executable that produces the output you are currently reading.

 4. The Docker daemon streamed that output to the Docker client, which sent it

    to your terminal.

------------------------------------------------------------

Docker FILE

FROM node:9.4.0-alpine

COPY app.js .

COPY package.json .

RUN npm install &&\

    apk update &&\

    apk upgrade

EXPOSE  8080

CMD node app.js

------------------------------------------------------------

app.js

var express = require('express')

var os = require("os");

var hostname = os.hostname();

var app = express()

app.get('/', function(req, res) {

  res.send('Hello world from ' + hostname + '! Your app is up and running!\n')

})

app.listen(8080, function() {

  console.log('Sample app is listening on port 8080.')

})

------------------------------------------------------------

package.json

{

  "name": "hello-world-demo",

  "private": false,

  "version": "0.0.1",

  "description": "Basic hello world application for Node.js",

  "dependencies": {

    "express": "4.x"

  }

}

------------------------------------------------------------

//List Image

docker images

//Pull your first image from Docker Hub.</body>

docker pull hello-world

//List images again.

docker images

//Run the hello-world image as a container.

docker run hello-world

//List the containers to see that your container ran and exited successfully.

docker ps -a

//Check the container ID

docker container rm <container\_id>

//List the containers to see that your container ran and exited successfully.

docker ps -a

//Build command along with the Tag. Build above image

docker build . -t myimage:v1

//Below is the output. Also, note that the imgae hello-word is pulled from the docker HUB

s/1\_ContainersAndDocker$ docker images

REPOSITORY    TAG       IMAGE ID       CREATED         SIZE

myimage       v1        1a8e126826c0   4 seconds ago   77.7MB

hello-world   latest    d2c94e258dcb   20 months ago   13.3kB

theia@theiadocker-kolsukarneha:/home/project/CC201/lab

//Now that your image is built, run it as a container with the following command:

docker ps

//Run the curl command to ping the application as given below.

curl localhost:8080

//Now to stop the container we use docker stop followed by the container id. The following command uses docker ps -q to pass in the list of all running containers:

docker stop $(docker ps -q)

//Check if the container has stopped by running the following command.

docker ps

**Docker:-**

|  |  |
| --- | --- |
| **Command** | **Description** |
| **curl localhost** | Pings the application. |
| **docker build** | Builds an image from a Dockerfile. |
| **docker build . -t** | Builds the image and tags the image id. |
| **docker CLI** | Start the Docker command line interface. |
| **docker container rm** | Removes a container. |
| **docker images** | Lists the images. |
| **docker ps** | Lists the containers. |
| **docker ps -a** | Lists the containers that ran and exited successfully. |
| **docker pull** | Pulls the latest image or repository from a registry. |
| **docker push** | Pushes an image or a repository to a registry. |
| **docker run** | Runs a command in a new container. |
| **docker run -p** | Runs the container by publishing the ports. |
| **docker stop** | Stops one or more running containers. |
| **docker stop $(docker ps -q)** | Stops all running containers. |
| **docker tag** | Creates a tag for a target image that refers to a source image. |
| **docker –version** | Displays the version of the Docker CLI. |
| **exit** | Closes the terminal session. |
| **export MY\_NAMESPACE** | Exports a namespace as an environment variable. |
| **git clone** | Clones the git repository that contains the artifacts needed. |
| **ibmcloud cr images** | Lists images in the IBM Cloud Container Registry. |
| **ibmcloud cr login** | Logs your local Docker daemon into IBM Cloud Container Registry. |
| **ibmcloud cr namespaces** | Views the namespaces you have access to. |
| **ibmcloud cr region-set** | Ensures that you are targeting the region appropriate to your cloud account. |
| **ibmcloud target** | Provides information about the account you’re targeting. |
| **ibmcloud version** | Displays the version of the IBM Cloud CLI. |
| **ls** | Lists the contents of this directory to see the artifacts. |

**kubernetes**

|  |  |  |
| --- | --- | --- |
| **Category** | **Command** | **Description** |
| **Cluster Information** | kubectl cluster-info | Displays information about the cluster's master and services. |
|  | kubectl get nodes | Lists all nodes in the cluster. |
|  | kubectl describe node NODE\_NAME | Shows detailed information about a specific node. |
| **Namespace Management** | kubectl get namespaces | Lists all namespaces in the cluster. |
|  | kubectl create namespace NAMESPACE\_NAME | Creates a new namespace. |
|  | kubectl config set-context --current --namespace=NAMESPACE\_NAME | Switches the current context to the specified namespace. |
| **Pod Management** | kubectl get pods | Lists all Pods in the current namespace. |
|  | kubectl describe pod POD\_NAME | Shows detailed information about a specific Pod. |
|  | kubectl logs POD\_NAME | Displays logs for a specific Pod. |
|  | kubectl exec -it POD\_NAME -- /bin/bash | Opens an interactive terminal session inside a container. |
|  | kubectl delete pod POD\_NAME | Deletes a specific Pod. |
| **Deployment Management** | kubectl get deployments | Lists all Deployments in the current namespace. |
|  | kubectl create deployment DEPLOYMENT\_NAME --image=IMAGE\_NAME | Creates a new Deployment with the specified container image. |
|  | kubectl scale deployment DEPLOYMENT\_NAME --replicas=COUNT | Scales a Deployment to the desired number of replicas. |
|  | kubectl set image deployment/DEPLOYMENT\_NAME CONTAINER\_NAME=NEW\_IMAGE | Updates the container image for a Deployment. |
|  | kubectl rollout status deployment DEPLOYMENT\_NAME | Checks the rollout status of a Deployment. |
| **Service Management** | kubectl get services | Lists all Services in the current namespace. |
|  | kubectl describe service SERVICE\_NAME | Displays detailed information about a Service. |
|  | kubectl expose deployment DEPLOYMENT\_NAME --type=LoadBalancer --port=PORT | Exposes a Deployment as a Service with the specified type and port. |
| **Volume Management** | kubectl get pv | Lists all Persistent Volumes (PVs). |
|  | kubectl get pvc | Lists all Persistent Volume Claims (PVCs). |
| **Resource Management** | kubectl apply -f FILE.yaml | Creates or updates resources defined in a YAML/JSON file. |
|  | kubectl delete -f FILE.yaml | Deletes resources defined in a YAML/JSON file. |
|  | kubectl get all | Lists all resources in the current namespace (Pods, Services, Deployments, etc.). |
| **Debugging and Logs** | kubectl logs POD\_NAME | Fetches logs from a Pod. |
|  | kubectl logs POD\_NAME -c CONTAINER\_NAME | Fetches logs from a specific container in a Pod. |
|  | kubectl describe RESOURCE\_NAME | Provides detailed information about a resource. |
|  | kubectl exec -it POD\_NAME -- COMMAND | Executes a command in a container inside a Pod (e.g., bash, ls). |
| **Scaling and Updates** | kubectl scale deployment DEPLOYMENT\_NAME --replicas=COUNT | Scales the number of replicas for a Deployment. |
|  | kubectl rollout undo deployment DEPLOYMENT\_NAME | Rolls back to the previous version of a Deployment. |
| **System and Utilities** | kubectl version | Shows the Kubernetes client and server versions. |
|  | kubectl config view | Displays the current Kubernetes context and cluster configuration. |
|  | kubectl get events | Lists recent events in the cluster. |