#### [Docker need and benefits](javascript:void(0);)

**Objectives**

* Demonstrate the steps involved to deploy an application in a remote server
  + identify software required for remote, identify steps to deploy an application

* Explain the need and benefit of Docker
  + open source; create, deploy and run applications in containers; package application; virtual machine; image; container
    - Reference: https://opensource.com/resources/what-docker

[Deploying end to end application using compose](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate hosting a web application in nginx using command line and Dockerfile
  + nginx, pull, run, listing images, container name, detaching the process, port number, volumes, listing containers, listing non running containers, starting and stoping a container, Dockerfile, FROM, COPY, ENTRYPOINT, build, remove images and containers
    - Nginx Reference - https://hub.docker.com/\_/nginx
    - Docker CLI - https://docs.docker.com/engine/reference/commandline/cli/

* Demonstrate hosting a MySQL server with schema creation using docker
  + docker-compose.yml, docker-compose up command, mapping MySQL data file to local folder, schema creation script execution definition, defining port, password definition, docker compose up, executing mysql client on the mysql server container
    - Reference - https://hub.docker.com/\_/mysql

* Demonstrate hosting a REST API Microservice using docker
  + Defining Dockerfile for REST API, building Dockerfile from docker-compose.yml, using depends\_on in docker compose to define dependencies, using links to establish connectivity between REST API service and MySQL server, modify connection properties in REST API to connect to the docker instance
    - Reference - https://hub.docker.com/\_/openjdk
    - docker compose link - https://docs.docker.com/compose/compose-file/#links

* Demonstrate linking angular, zuul gateway, microservices and database using docker compose
  + Including configuration for components like angular, zuul gateway, microservices and database in docker-compose.yml and make them work together end to end

**What is Docker?**   
  
Let us do a simple exercise before trying to understand Docker and it's benefits.

* Execute the following commands. The explanation for these statements will be provided later:

docker image ls

docker container ls -a

* Open command prompt and execute the following command.

docker run -it debian

* This command downloads linux debian operating system from docker hub (<https://hub.docker.com/>) and runs linux operating system within the local docker server.
* A linux operating system command prompt opens up and should look something similar to this:

root@86bdf44d1bd7:/#

* Let us execute few linux commands to find out how it works.
  + Print current directory

pwd

* List all files and folders in the current directory:

ls

* List system information

uname -a

* Exit from debian linux

exit

**Key take away points**

* One can understand how quickly we have launched an operating system
* The "docker run" command performs the following tasks:
  + Downloads debian ***image*** from https://hub.docker.com
  + Runs the debian image in a ***container***
  + The -it option provides an interactive command prompt
* An image in docker contains the binary version of a software application
* A container is created for creating an working environment based on the image
* To view the list of images and containers execute the following commands, which will display the image and container details for debian
* In a similar fashion we can roll out JDK, MySQL, etc. on top of this linux platform and run them in containers.

**What Docker is?**

* Docker is open source
* Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.
* Docker allows a developer to package an application
* By testing in Docker, a developer can be rest assured that the code will work as expected in a linux platform without having a need to have an actual platform.
* Docker is like a virtual machine
* Benefits both developers and system administrators
* This forms part of DevOps toolchains.

**Run a web server in Docker**   
  
Nginx is a server technology that can be used to host a web application with static html pages.  
  
Create an Hello World html page and host it in the nginx docker container.  
  
Follow steps below to incorporate the same:

* Create folder d:\docker-learn\html
* Create a file named home.html in the new folder created above. In home.html, include html script that displays Hello World message
* Execute the following docker command that gets the nginx image from https://hub.docker.com

docker pull nginx:1.17.5

* Verify if image is available with this command:

docker image ls

* Execute the following command to run the nginx container from the nginx image. [NOTE: This command can be directly executed without executing the pull command. If the image is not available, the run command itself will download it and then run the container]

docker run --name my-nginx -d -p 8085:80 -v d:\docker-learn\html:/usr/share/nginx/html nginx:1.17.5

* Explanation for the above command:
  + run - starts the container
  + --name - provides an user defined name for the container
  + -d - runs nginx in the background and get back the control to the prompt
  + -p - specifies that port 80 of nginx needs to be mapped to 8085 port of local desktop
  + -v - creates a volume so that the html file in the desktop can be copied to the folder where nginx container looks for html files, so that it copies home.html to /usr/share/nginx/html folder of the container.
  + nginx - denotes the image
  + 1.17.5 is known as the tag that points to the respective software version
* The output of the above command does not result in any significant output, but it would have started the nginx server.
* Issue the below docker command to check if the container is running:

docker container ls

* Test the execution of the nginx server by opening http://locahost:8085 in the browser
* Command to stop the server

docker stop my-nginx

* Now issue the below command will not display the container. As this command does not list stopped container

docker container ls

* Issue the following command to view stopped containers

docker container ls -a

* Command to start the server

docker start my-nginx

**Start nginx using Dockerfile**

* Create file named 'Dockerfile' in d:\docker-learn\html folder
* Include following content in 'Dockerfile':

FROM nginx:1.17.5

COPY home.html /usr/share/nginx/html

* FROM command pulls the image if it is not locally available
* COPY command transfers file from desktop to a folder in container
* If nginx container is already running, stop it
* In command prompt go to d:\docker-learn\html folder
* Execute the following command to run the container:

docker build .

* The above command uses the options specified in Dockerfile and runs the container
* Check if the application runs using browser

**Deleting container and image**

* Use 'docker image ls', 'docker container ls' and 'docker container ls -a' to view the list of existing images and containers
* Use 'docker image rm [IMAGE\_ID]' and 'docker container rm [CONTAINER\_ID]' to remove an image. IMAGE\_ID And CONTAINER\_ID can be obtained using the ls command

**Preparation for dockerizing angular-learn and spring-learn microservice**   
  
Following are the high level activities that deploys angular-learn and spring-learn application:

* Start mysql in a container with necessary schema, table and data available
* Deploy authentication microservice REST API and link this service with mysql container.
* Deploy application microservice REST API in a container that addresses the business logic of the application
* Deploy eureka discovery service in a container
* Deploy zuul gateway service that acts as a gateway for authentication and application microservices
* Deploy angular application in nginx
* Test if the application works end to end

**Organize Folder**

* Create a new folder named *docker-build* in C:\Users\[EMP\_ID] folder. This will be the root folder for all the components that we are going to dockerize.
* Copy all project folders in this new folder. The list of projects are provided below for quick reference:
  + dbscript - This folder should contain the schema creation script for 'ormlearn' database
  + authentication-service - This is the microservice project split from spring-learn that authenticates and generates JWT
  + employee-service - This is the microservice project split from spring-learn that handles services related to employee
  + eureka-discovery-service - Service Registry
  + zuul-gateway-service - Gateway Service for authentication-service and employee-service
  + angular-learn - This folder has to contain the angular project

Refer steps specified in the subsequent exercises to incorporate the above deployment.

**Containerize MySQL**   
  
Steps to setup MySQL container

* Create a file named docker-compose.yml file in docker-build folder
* Ensure that the schema creation SQL file is placed in the dbscripts folder
* Include the following configuration in the docker-compose.yml:

version: '3'

services:

  payroll-mysql:

    image: mysql:8.0.18

    ports:

      - "33061:3306"

    environment:

      - MYSQL\_ROOT\_PASSWORD=root

    volumes:

      - d:/payrolldb:/var/lib/mysql

      - ./dbscripts:/docker-entrypoint-initdb.d

* Explanation for the above configuration
  + version - denotes the docker-compose file syntax version
  + services - section denotes various services that can be part of this docker compose. Currently in this configuration we have only one service which is named as 'payroll-mysql'.
  + image - denotes the mysql server image that needs to be used
  + ports
    - port number in left hand side (33061) denotes the port that will be exposed in desktop
    - port number in the right hand side (3306) denotes the port that will be exposed inside the container
  + environment - this defines the root password
  + volumes
    - First line denotes the location in the desktop PC where MySQL data files will be stored. This ensures that every time the mysql container is stopped the changes are saved locally, else any changes made to data in the server will be lost.
    - The second line denotes that the scripts available in dbscripts folder need to be executed when starting the container for the first time
* Execute the following command in docker-build folder:

docker-compose up

* This will start the MySQL server in container
* Open another command prompt and execute the below command to execute mysql client and check if the database is created and tables are populated with necessary data.

docker exec -it payroll-mysql bash

* This will open linux bash command prompt. Execute the following command to login into the mysql running in the container:

mysql -u root -p

* After login check if the schema is created and tables with data is present

**Linking Microservice with MySQL**   
  
To link authentication microservice with MySQL database follow steps below:

* Create a file named 'Dockerfile' in the authentication-service folder with the below specified content

FROM openjdk:8-jdk-alpine

COPY target/authentication-service-0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom -Djava.net.preferIPv4Stack=true","-jar","/app.jar"]

* Explanation for the Dockerfile configuration
  + FROM command pulls JDK 8 from docker hub
  + COPY command copies the jar created in target folder to the project root folder with name app.jar
  + ENTRYPOINT execute the java command and starts the REST API Service
* Modify docker-build\docker-compose.yml so that the entire file content looks like the one below. Ideally authentication-service had been added in the file:

version: '3'

services:

  payroll-mysql:

    image: mysql:8.0.18

    ports:

      - "33061:3306"

    environment:

      - MYSQL\_ROOT\_PASSWORD=root

    volumes:

      - d:/payrolldb:/var/lib/mysql

      - ./dbscripts:/docker-entrypoint-initdb.d

  authentication-service:

    image: authentication-app

    build: authentication-service/.

    ports:

      - 8091:8091

    depends\_on:

      - payroll-mysql

    links:

      - payroll-mysql

* Explanation for authentication-service configuration
  + image - defines the name for authentication-service
  + build - denotes that Docker file is present in authentication-service folder
  + ports
    - left hand side port number denotes the port that will be exposed
    - right hand side port number denotes the port number defined in the application.properties file of authentication-service
  + depends\_on - denotes that authentication-service requires start of mysql server
  + links - denotes that authentication-service is linked to payroll-mysql. This ensure database connectivity from authentication-service to mysql database
* Find below the database connection changes that needs to be done in application.properties file of authentication-service. Look out for the following changes
  + MySQL connection URL changes
    - localhost changed to payroll-mysql
    - port number changed to 33061
    - Configures public key retrieval and SSL
  + Change port of the microservice (server.port) to avoid port number conflicts
* The primary change is that, we are changing the localhost as payroll-mysql, port number. Change the server port aligned to the docker compose configuration.

server.port=8091

spring.datasource.url=jdbc:mysql://payroll-mysql:33061/ormlearn?allowPublicKeyRetrieval=true&useSSL=false

* Execute maven build in command line on authentication-service folder to create jar file with updated configuration
* Execute 'docker-compose up' command in docker-build folder. This will start mysql server and authentication-service
* Test the *http://localhost:8090/authenticate* REST API using Postman or curl command and verify if the service works end to end and returns back the token.

**Dockerizing other components**   
  
Based on the instructions provided in the previous four exercises implement the deployment of the following components:

* **employee-service**
  + Create a Dockerfile similar to the one done for authorisation-service
  + Modify port as 8092 and mysql connection URL in application.properties
  + Modify docker-compose.yml with inclusion of employee-service linking to payroll-mysql
  + Run docker compose and test a REST API service in employee-service to verify if the service works end to end
* **eureka-discovery-service**
  + Create a Dockerfile similar to the one done for authorisation-service
  + Modify port as 8093 and mysql connection URL in application.properties
  + Run docker compose and test if the registry service gets hosted
* **zuul-gateway-service**
  + Create a Dockerfile similar to the one done for authorisation-service
  + Modify port as 8094
  + Modify docker-compose.yml with inclusion of zuul-gateway-service linking to authentication-service and employee-service
  + Run docker compose and test a REST API service in authentication-service and employee-service to verify if the service works end to end
* **angular-learn**
  + Modify the REST API URL in the angular application refering to zuul-gateway-service
  + Execute the angular build which creates the dist folder
  + Create a Dockerfile that copies dist folder content to nginx container and starts the server (Reference: <https://hub.docker.com/_/nginx>)
  + Include docker-compose.yml with a new service named angular-service that runs the Dockerfile and links with zuul-gateway-service
  + Run docoker compose and test the angular application to see if it works end to end