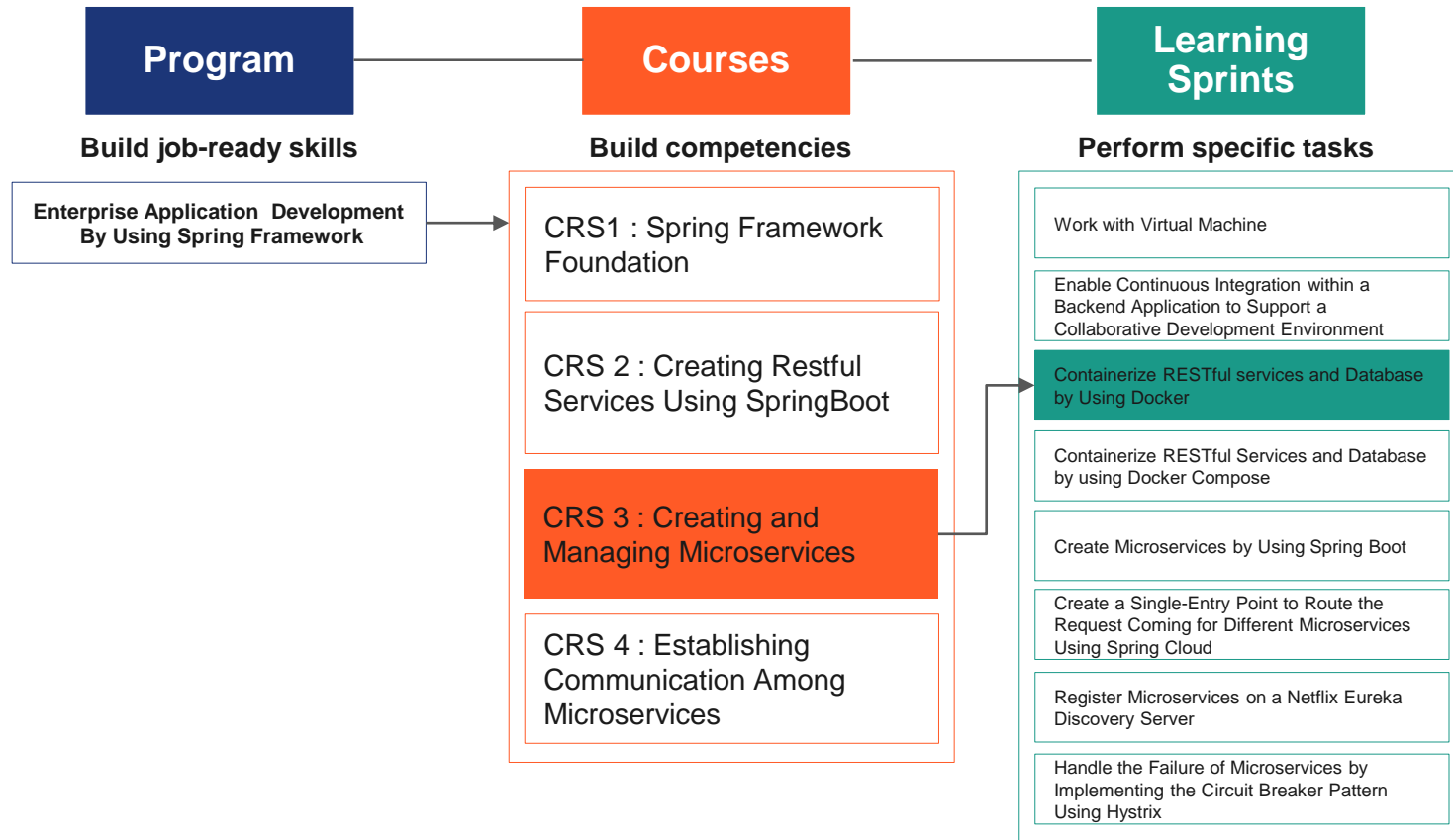
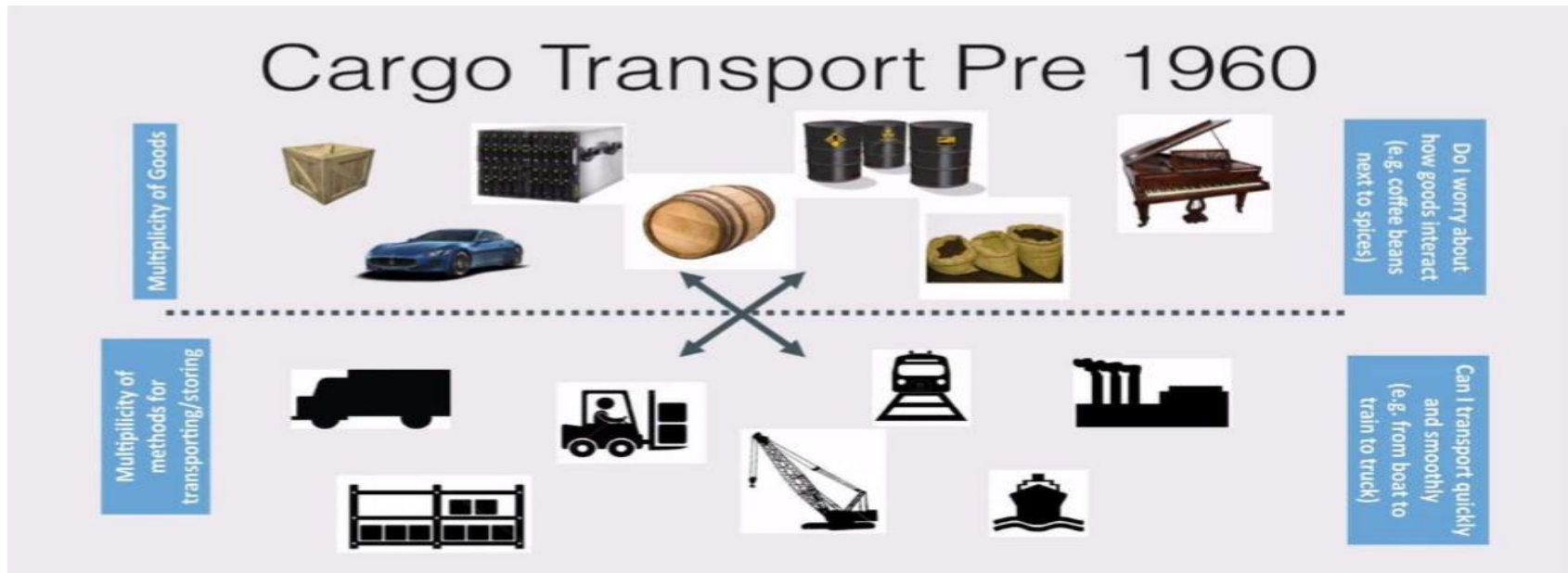


Backend Program: Course 3: Structure



Traditional Cargo Shipping



- Do I need to worry about how goods interact?
- Can I transport efficiently?

Modern Cargo Shipping

Intermodal Shipping Container

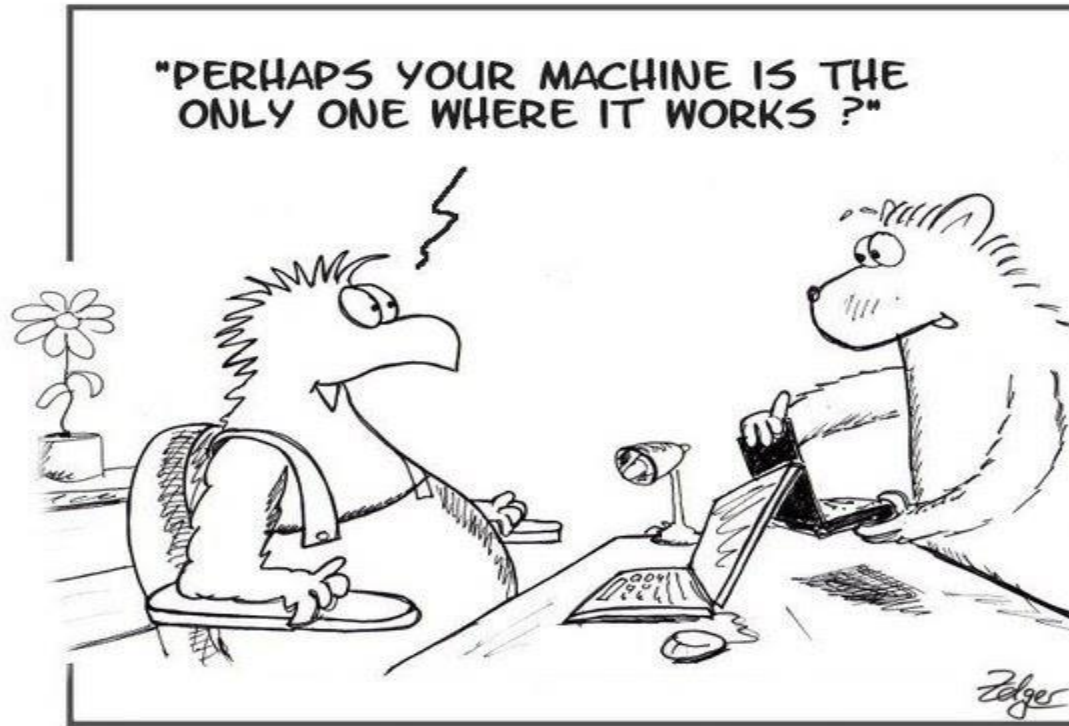


Time to Think



- Software is developed by a team as per requirements made by a customer. How does the team deploy all the components of a software application for the customer?
- How can we ship or shift this software to the customer system?
- How can we have the required versioning between development, test, and production beds?
- How can we be sure that the software runs across all the platforms?

Challenges of Shipping a Software



It works on my machine

Think and Tell

- Where is deployment involved in the lifecycle of a software ?
- If a new feature needs to be added to an application, how is the redeployment process performed?
- Can the new feature work across platforms?
- If new releases must be done by the development team periodically, more changes to a software will introduce more defects.
- What should be the strategy to reduce the defects?



Containerize RESTful Services and Database by Using Docker



Learning Objectives

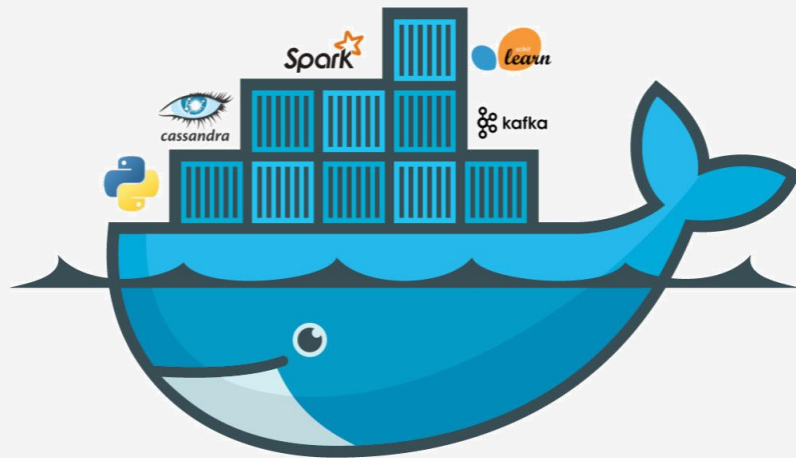
- Differentiate between containers and Virtual Machines
- Explain Docker
- Describe the relationship between a Docker image and container
- Explain the process of containerizing database MongoDB
- List steps to Dockerize the Spring Boot application



Docker for Developers

What is Docker?

- Docker is a container technology used to build, ship, and run any app, anywhere.
- Docker helps separate applications from the infrastructure so that a software can be delivered quickly.
- With Docker, we can manage our infrastructure in the same way we manage our applications.
- Using Docker methodologies for shipping, testing, and deploying a code can significantly help reduce the delay caused during writing a code and running it in production.

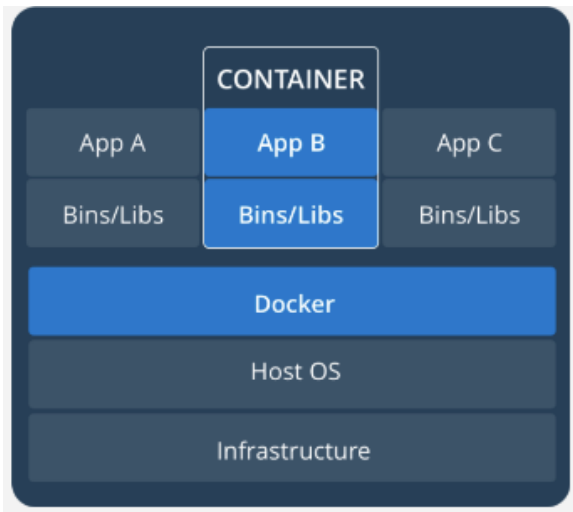


Docker– Software Shipping Container Technology

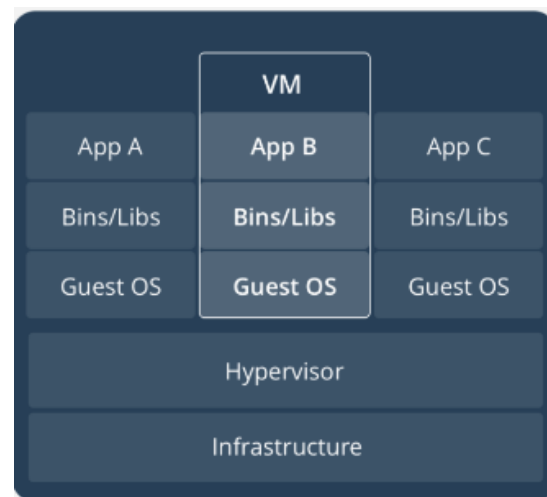


Container vs. Virtual Machine

Containers are app level construct.



VMs are an infrastructural level construct used to turn one machine into many servers.



Container vs. Virtual Machines

Virtual Machines(VM)	Containers
VM is a piece of software that allows installation of other software inside it, so that it can be controlled virtually as opposed to installing the software directly on the computer.	A container is a software that allows different functionalities of an application independently.
It virtualizes the computer system.	They virtualize only the operating system.
It is very large in size.	Containers are very light, probably a few megabytes.
VM takes minutes to run, due to its large size.	They take a few seconds to run.
These machines are useful when we require all the OS resources to run various applications.	Containers are useful when we are required to maximise the running applications using minimal servers.

Benefits of a Docker

- Consistent and isolated environment
- Cost effective with fast deployment
- Mobility – Ability to run anywhere
- Repeatability and automation
- Test, roll back and deploy
- Flexibility
- Collaboration, modularity and scaling

Components of a Docker

- Image
 - A read only template.
 - Includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container at runtime.
 - Usually derives from multiple base images that are layers stacked on top of each other to form the container's file system.
 - Is immutable once it has been created.
- Container
 - A runnable instance of a Docker image.
 - Can be created, started, stopped, and deleted.

Layers in Docker Images



I'll make the pizza

My Pizza Image



Layer 5 (Herbs Topping)

Layer 4 (Veg Topping)

Layer 3 (Ingredients)

Layer 2 (Sauce)

Layer 1 (Crust)

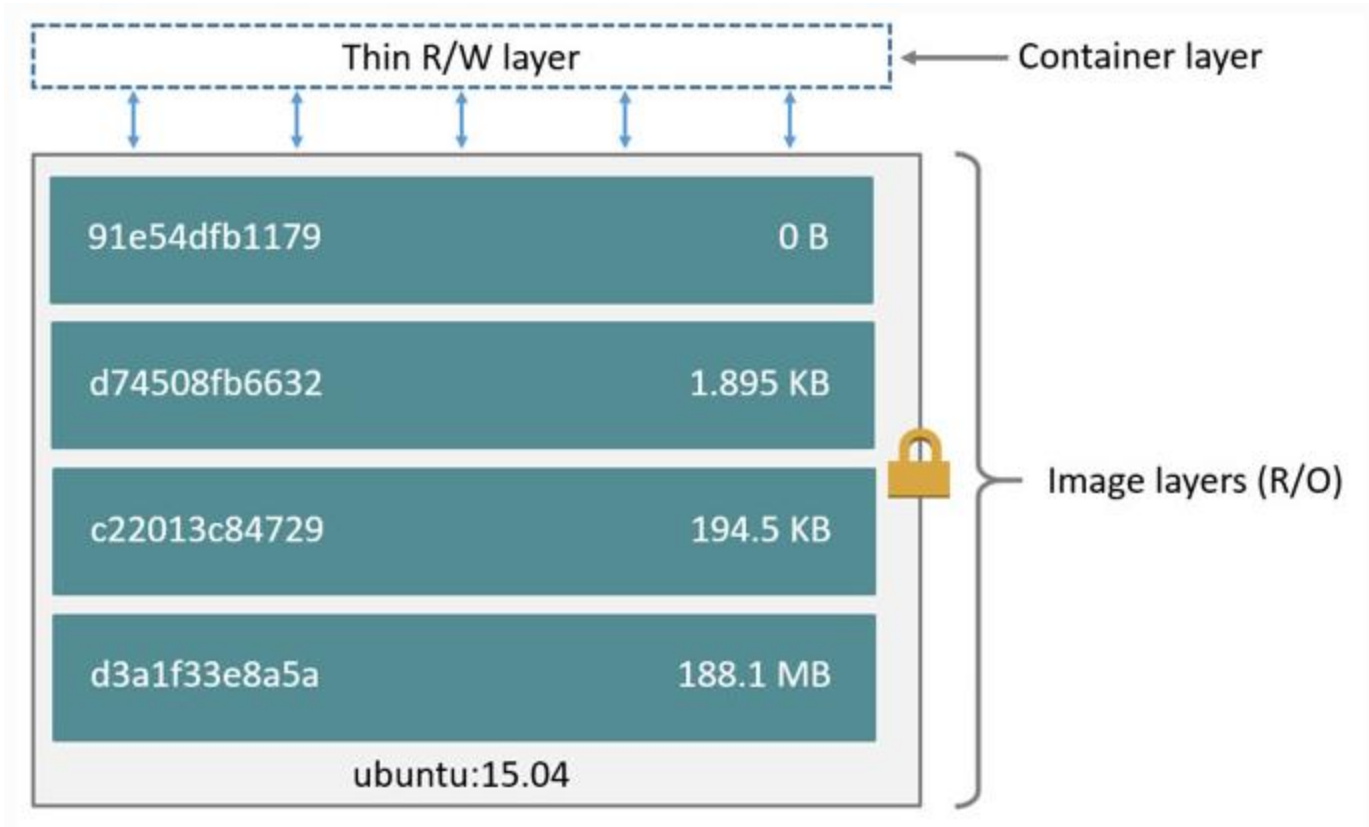
Docker Image

- Docker images are templates to create Docker containers.
- They contain a set of instructions.
- An image is comparable to a snapshot in a virtual machine (VM) environment.
- A Docker image contains application code, libraries, tools, dependencies and other files needed to run an application.
- These images have multiple layers, each one originates from the previous layer but is different from it.
- They are a reusable and can be deployed on any host.
- Developers can take the static image layers from one project and use them in another.
- Images are immutable, once built the files making up an image do not change.

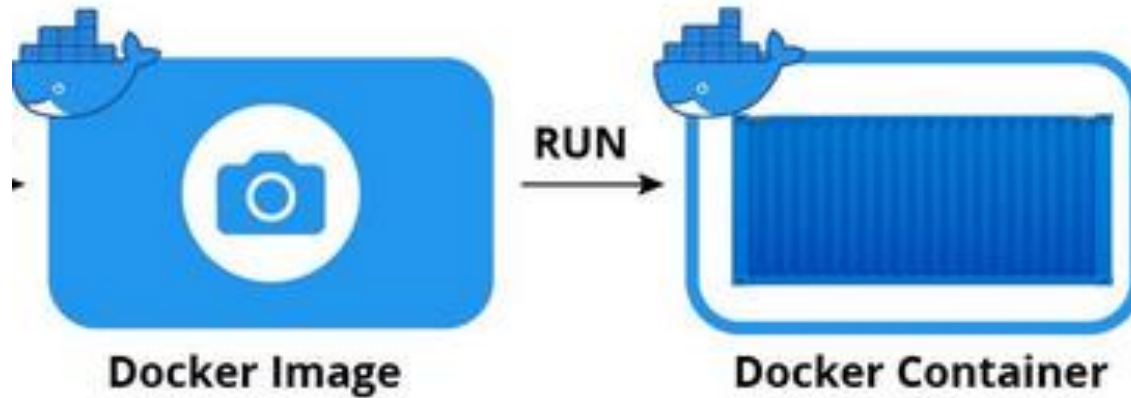
Docker Container

- A Docker Container is a standardized unit which can be created to **deploy a particular application or environment.**
- Docker Containers are created from the Docker Images.
- They are **running instances of the images** and they **hold the entire package needed to run the application.**

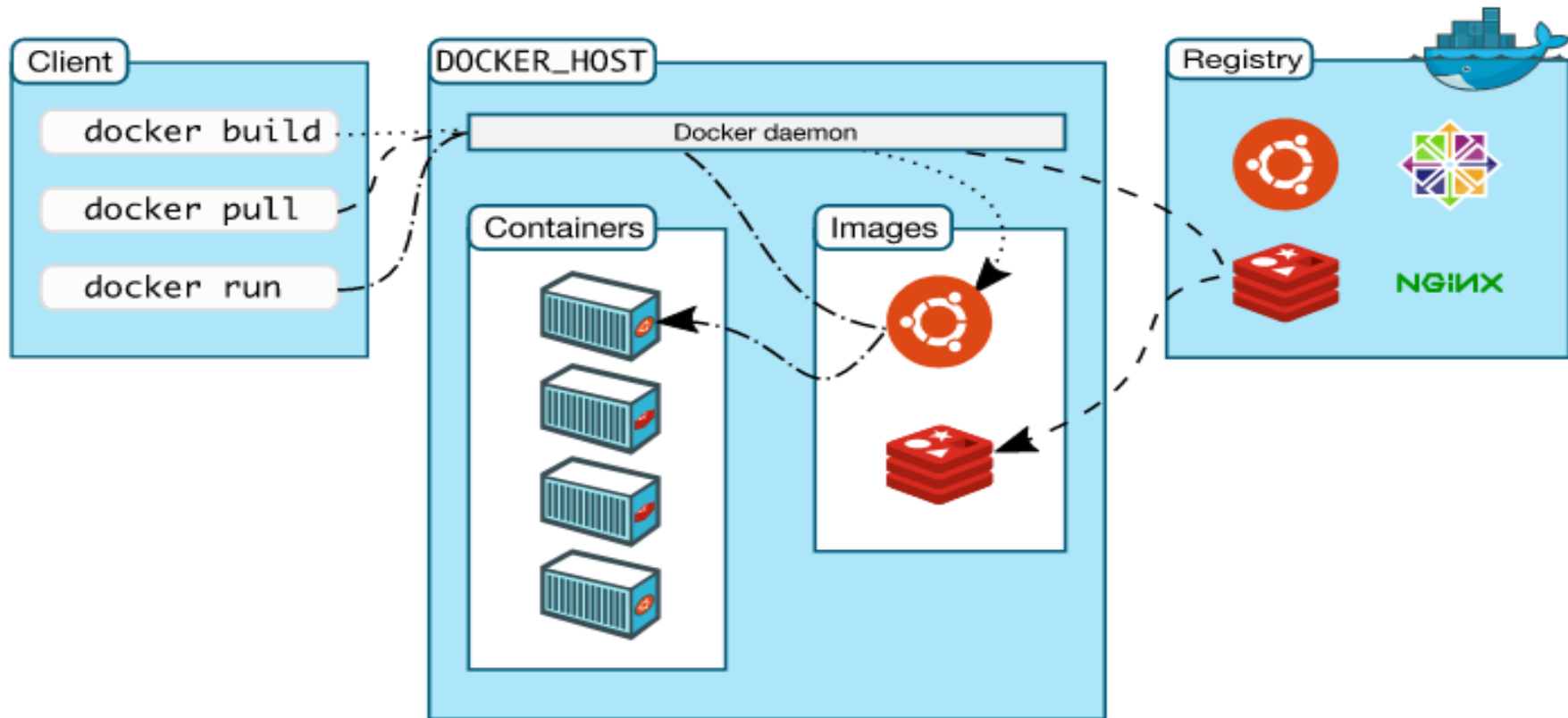
Docker Container (contd.)



Docker Image and Container



Docker Architecture

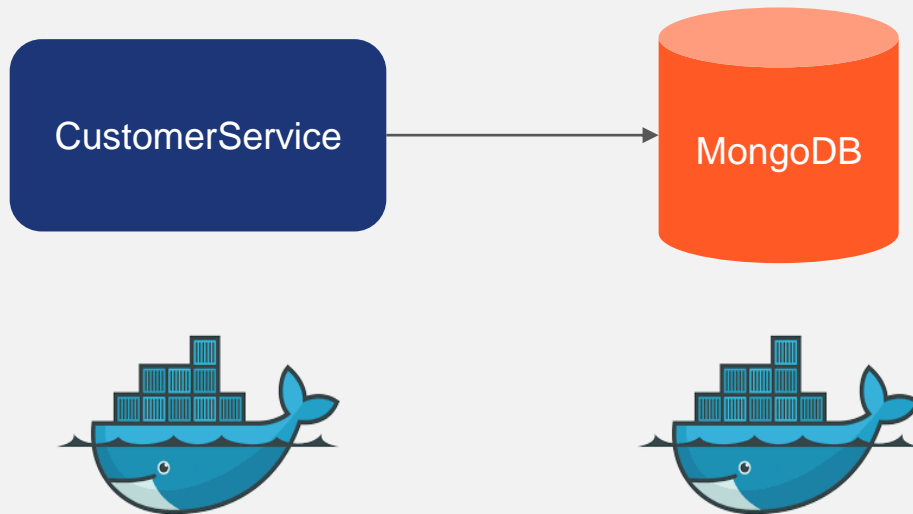


Dockerizing a Customer Service Application

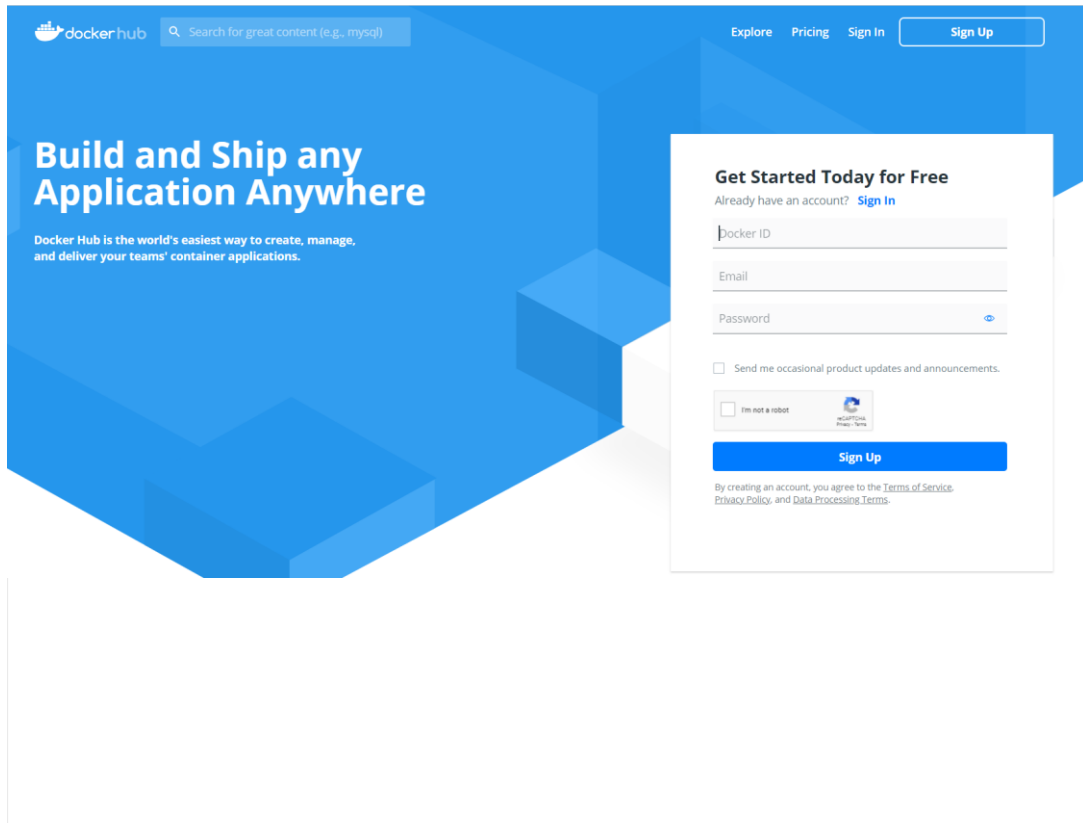
Spring Boot + MongoDB

We have created a Spring Boot application that connects to MongoDB.

- How many images should we create for the application?
- Are **some standard images available from a repository?**
- **How many containers** can we run for an application?



Docker Hub



The screenshot shows the Docker Hub website's sign-up interface. The background is blue with a geometric pattern. The main heading is "Build and Ship any Application Anywhere". Below it, a subheading states: "Docker Hub is the world's easiest way to create, manage, and deliver your teams' container applications." The navigation bar includes "Explore", "Pricing", "Sign In", and a "Sign Up" button. A search bar is present with the placeholder text "Search for great content (e.g., mysql)". A central white box contains the "Get Started Today for Free" section, which includes a link for existing users ("Sign In"), input fields for "Docker ID", "Email", and "Password", a checkbox for product updates, a "I'm not a robot" checkbox with a CAPTCHA image, and a "Sign Up" button. At the bottom of this box, there is a link to the Terms of Service, Privacy Policy, and Data Processing Terms.

- Docker Hub is the place where open Docker images are stored.
- A Docker image can be pulled from Docker Hub.
- A Docker image can be pushed to Docker Hub.
- When we pulling an Image from the DockerHub docker first checks whether that image is available on the local computer if not available locally the image is downloaded from Docker Hub.

Containerize Database - MongoDB

MongoDB Image From the Docker Hub

- The images for mongo exist in the Docker Hub.
- Search and pull the mongo image from there.



mongo ☆

Docker Official Images

MongoDB document databases provide high availability and easy scalability.

↓ 1B+

Container

Linux

Windows

x86-64

ARM 64

IBM Z

Databases

Official Image

Copy and paste to pull this image

```
docker pull mongo
```



[View Available Tags](#)

Pull MongoDB Image

- Pull the image from the Docker hub
 - `docker pull mongo`
- Show the images
 - `docker images` – it will show all the images
 - Here `Tag - latest` means the latest version of a MongoDB image is pulled so, if you do not mention any version than by default the latest will get pulled.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mongo	latest	196db2355239	25 hours ago	671MB

Create MongoDB Container

- Create the container
 - `docker run -d mongo:latest`
 - Here `-d` means in `detach mode` also it is not compulsory to write.
 - `mongo: latest` is the image which was created earlier

```
docker run -d mongo:latest
d71e0b45a79ba00e896ce91ce7602b660a7e
```

Check Status of the Container

- Check container is created or not

```
$ docker ps
```

§ The docker ps command only shows **running containers by default.**

```
$ docker ps -a
```

§ This will **show all the containers running and even stopped container.**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
587ecfbfc5c3	mongo:latest	"docker-entrypoint.s..."	15 minutes ago	Up 15 minutes	27017/tcp	competent_cohen

Unique Id for container

Image – this was pull from Docker hub

Status up means that the container is running

This mongo container is listening on 27017 port which is default port of mongo

Connect to MongoDB Container

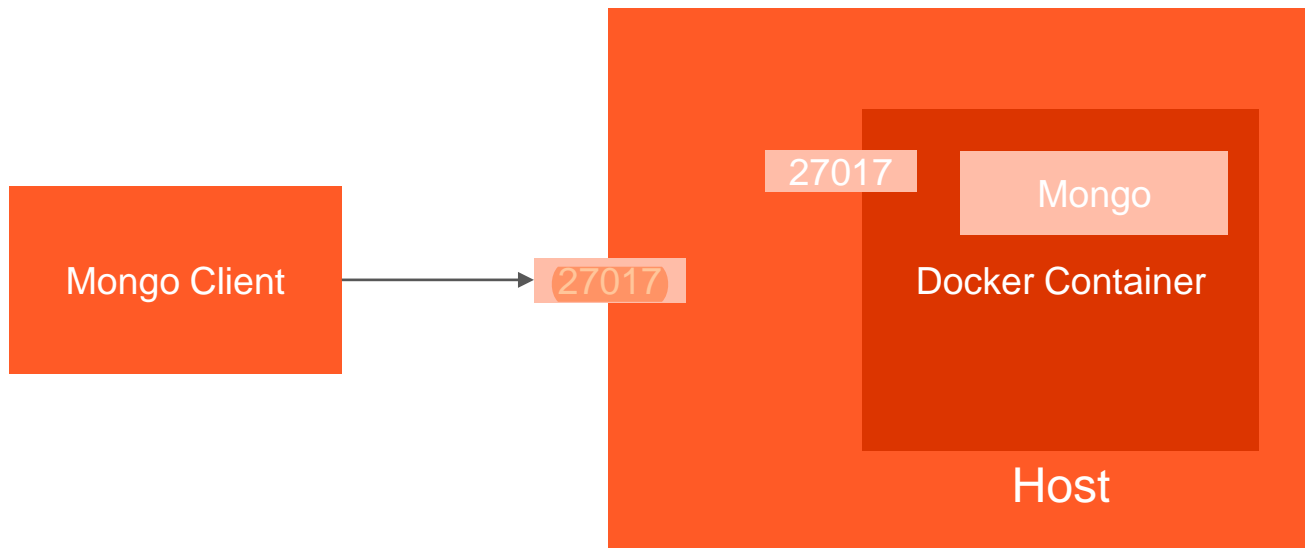
- Type mongo on the terminal
 - `mongo`

```
pc@pc-ThinkPad-E470:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
2021-07-15T08:26:51.146+0530 W NETWORK [thread1] Failed to connect to 127.0.0.1:27017, in(checking socket for error after poll),
action refused
2021-07-15T08:26:51.180+0530 E QUERY [thread1] Error: couldn't connect to server 127.0.0.1:27017, connection attempt failed :
connect@src/mongo/shell/mongo.js:251:13
@(connect):1:6
exception: connect failed
```

There is an error, stating that it couldn't connect to the server – 27017.

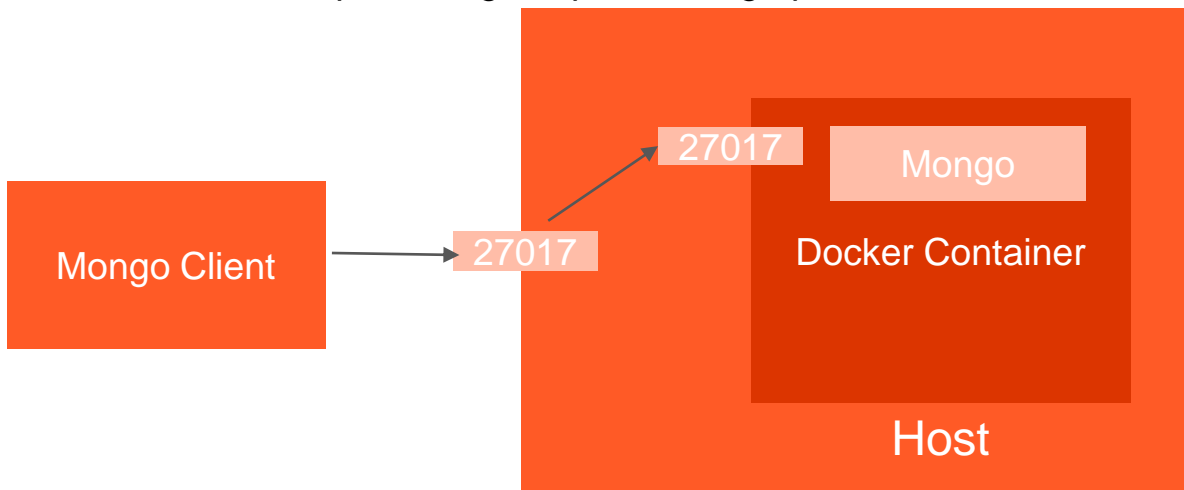
Connection Error Explained

- We now have a **Mongo instance running inside the docker container**. However, this port is only accessible within the Docker network.
- **The mongo client tries to access the port within the docker network, but the port 27017 is not exposed.**



Port Mapping

- In Docker, the containers themselves can have applications running on ports.
- When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host.
- If a local mongo client wants to access the port 27017 inside docker then the container must expose the port. We expose the Mongo container to the outside world by mapping the container's Mongo port to the host machine port using the publish flag `-p`.



Running the Container With Port Mapping

- `docker run -d -p 27107:27017 --name mongocontainer mongo:latest`
 - Here – **name** and **-d** are optional, **name** will give a name to the docker container, if **name** is not specified, docker will generate an arbitrary name.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fc4990621e71	mongo:latest	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	0.0.0.0:27107->27017/tcp, :::27107->27017/tcp	mongocon

- In the '**STATUS**' column
 - '**Up**' means the container is running,
 - '**Exited (1)**' the container ended while returning a non-zero value, it means that the program was terminated with an error.
- -p Port mapping is done.
- 27017:27017 – this implies <exposed port>:<host port>

Restart a MongoDB Container

- Start mongo

```
pc@pc-ThinkPad-E470:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2021-07-15T10:26:28.232+0530 I STORAGE [initandlisten]
2021-07-15T10:26:28.232+0530 I STORAGE [initandlisten] ** WARNING: Using the XFS
ge engine
2021-07-15T10:26:28.232+0530 I STORAGE [initandlisten] ** See http://doc
2021-07-15T10:26:29.810+0530 I CONTROL [initandlisten]
2021-07-15T10:26:29.810+0530 I CONTROL [initandlisten] ** WARNING: Access control
2021-07-15T10:26:29.810+0530 I CONTROL [initandlisten] ** Read and write
2021-07-15T10:26:29.810+0530 I CONTROL [initandlisten]
> show databases;
admin          0.000GB
config         0.000GB
customerdb     0.000GB
local          0.000GB
```

Summary

- Pull the mongo image from the Docker hub.
 - `docker pull mongo`
- To check all images
 - `docker images`
- Create the mongo container.
 - `docker run -d -p 27107:27017 --name mongocontainer mongo:latest`
 - `-d` – detached mode (Optional)
 - `-p` – Port mapping (Compulsory)
 - `--name` – name for the container (Optional)
- Check the status of the running container.
 - `docker ps`

Summary

- Check the status of all the containers (running and stopped)
 - `docker ps -a`
- Check the logs of the container.
 - `docker logs mongocontainer <containername>`
- Start the containerized Mongo Database
 - `mongo`

Containerize the Spring Boot - CustomerService

Steps to Dockerize a Spring Boot Application

- For dockerizing a Spring Boot application, we first need the docker image of the application.
- Then we need to run the docker image to create the container.
- The CustomerService application is created by us, we cannot get its image in the docker hub hence we need to create it.
- Steps to create the application container
 - Create a Dockerfile.
 - Build the Dockerfile to get the image.
 - Run the image to get the container.



Dockerfile

build



Docker Image

run



Docker Container

GEEK FLARE

What Is a Dockerfile?

- A **Dockerfile** is a text document that contains all the commands a user can call on the command line to assemble an image.
- A Docker can build images automatically by reading instructions from a Dockerfile.
- Dockerfile describes step-by-step instructions of all the commands required to assemble a Docker Image.

```
#here openjdk is docker image for Java 11, as in pom.xml Java version is 11
FROM openjdk

#creating a working directory inside the image
WORKDIR usr/lib

#setting environment variable same name that are there in application.properties file
ENV MONGO_DATABASE=custdb
ENV MONGO_URL=mongodb://localhost:27017/custdb

#Copy executable jar file getting created inside target and add it in usr/lib working director
ADD ./target/customer-0.0.1.jar /usr/lib/customer-0.0.1.jar

#Run the jar file
ENTRYPOINT ["java", "-jar", "customer-0.0.1.jar"]
```

Build the Image

```
Status: Downloaded newer image for openjdk:latest
--> f695f4f55ff0
Step 2/6 : WORKDIR usr/lib
--> Running in a746065062f4
Removing intermediate container a746065062f4
--> 87b19d6ba2e4
Step 3/6 : ENV MONGO_DATABASE=muzixdb
--> Running in 053c7700bb27
Removing intermediate container 053c7700bb27
--> 46c2a6e20c5b
Step 4/6 : ENV MONGO_URL=mongodb://localhost:27017/muzixdb
--> Running in 87c7f82b334c
Removing intermediate container 87c7f82b334c
--> e90586ecbbf3
Step 5/6 : ADD ./target/customer-0.0.1.jar /usr/lib/customer.
--> 304c41b516d3
Step 6/6 : ENTRYPOINT ["java","-jar","customer-0.0.1.jar"]
--> Running in 38d81b1b7fba
Removing intermediate container 38d81b1b7fba
--> cd3ae2cd591f
Successfully built cd3ae2cd591f
Successfully tagged spring-containerimage:latest
```

- Command to build a docker image from the Dockerfile.

`docker build -t spring-containerimage .`

- **build** means build the image
- **t** stands for tag
- **spring-containerimage** is the image name.
- **.** Signifies the current path of the Dockerfile

Run the Container

- Command to run the container
 - `docker run --name spring-app --network=host spring-containerimage:latest`
 - `--network=host` If you use the `host` network mode for a container, that container's network stack is not isolated from the Docker host (the container shares the host's networking namespace)
- Check the status of a container
 - `docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
fc4990621e71	mongo:latest	"docker-entrypoint.s..."	4 hours ago	Up 4 hours	0.0.0.0:27017->27017/tcp,
fc4990621e71	spring-app	"java -jar customer-..."	4 minutes ago	Up 4 minutes	



- Here we can see two container are up - one mongodb and second customerService.

Dockerized Application

Client (For example, frontend,
Postman, etc.)

Localhost

Docker Network

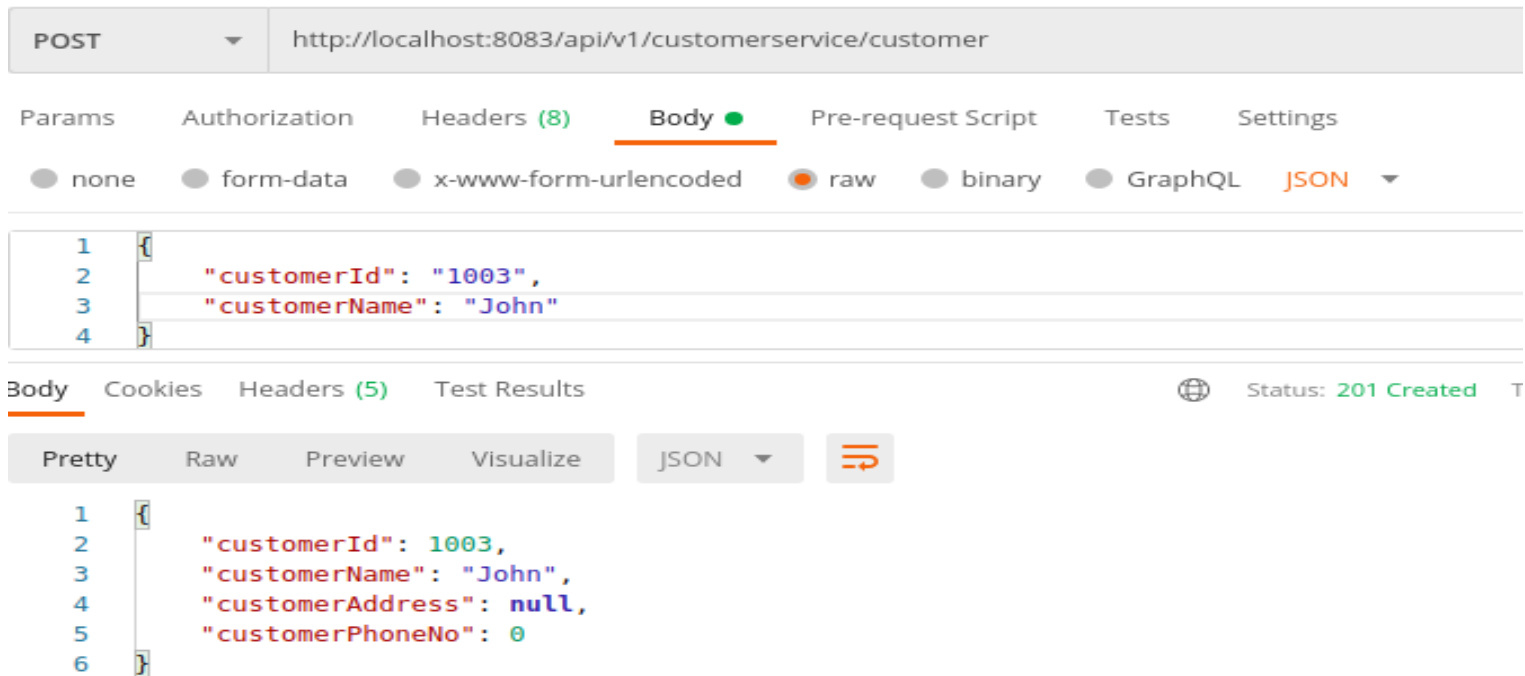
CustomerContainer

Application Active Profile:
Docker Port 8083

Mongo-Container

MongoDB Port 27017

Postman - Running the Dockerized Application



The image shows the Postman interface for a POST request. The URL is `http://localhost:8083/api/v1/customerservice/customer`. The request body is a JSON object with the following fields:

```

1 {
2   "customerId": "1003",
3   "customerName": "John"
4 }

```

The response status is 201 Created. The response body is displayed in the 'Body' tab, showing a JSON object with the following fields:

```

1 {
2   "customerId": 1003,
3   "customerName": "John",
4   "customerAddress": null,
5   "customerPhoneNo": 0
6 }

```

Docker Housekeeping

- Kill all running containers
 - `docker kill $(docker ps -q)`
- Delete all stopped containers (including data-only containers)
 - `docker rm $(docker ps -a -q)`
- Delete all exited containers
 - `docker rm $(docker ps -q -f status=exited)`
- Delete ALL images
 - `docker rmi $(docker images -q)`

Quick Check

How do you list all the running containers?

1. `docker rm <container id>`
2. `docker exec -it <container id> bash`
3. `docker ps`
4. All of the above



Quick Check: Solution

How do you list all the running containers?

1. `docker rm <container id>`
2. `docker exec -it <container id> bash`
3. **`docker ps`**
4. All of the above



Dockerize Customer Application

Dockerize the backend application created in the previous session. Dockerize the MongoDB and the CustomerService application.

DEMO



Key Takeaways

- Difference between Containers and Virtual Machines
- Docker – Components (Image and container)
- Dockerize Databases (MongoDB)
- Dockerize Spring Boot (CustomerService)



Thank you!

