[Microservices](javascript:void(0);)

**Objectives**

* Explain the need and benefit of Microservices
  + Challenges of enterprise applications, drawbacks of monolithic services, benefits of micro services, challenges of microservices
    - Reference - https://medium.com/@madhupathy/simplified-microservices-building-with-spring-cloud-netflix-oss-eureka-zuul-hystrix-ribbon-2faa9046d054
* Create microservices using Spring Web
  + Spring RESTful Web Services, server.port definition
* Create Eureka Discovery Services using Spring Cloud
  + Eureka Server, @EnableEurekaServer, Eureka Discovery Client, @EnableDiscoveryClient
    - Reference - https://spring.io/guides/gs/service-registration-and-discovery/
* Create Zuul Gateway to route multiple microservices
  + Purpose of Zuul Gateway, routing microservices, Zuul Gateway Server, @EnableZuulProxy, pre filter
    - Reference - https://spring.io/guides/gs/routing-and-filtering/

*IMPORTANT NOTE:* The above objectives needs to be covered in a half day session. After which the following two learning sessions needs to be implemented by self-learning and without trainer intervention.  
**Session 1 (4 Hours)**

* Segregate the spring-learn application into two microservices
* One microservice for user authentication
* Second microservice for employee service

**Session 2 (4 Hours)**

* Add the two microservices to eureka discovery
* Add the two microservices to zuul gateway
* Integrate angular application with the end points of zuul gateway

**Enterprise Application**   
  
Have you ever thought of various products and services offered by a Bank. Have a look at the diagram below. 

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/banking-services.PNG |

Still there are few aspects missing like Phone Banking, SMS Banking, Micro Finance, etc.  
  
Let us take one service "Savings Account" from the above diagram. Few aspects associated with Savings Account is defined below and this is not an exhaustive list. 

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/savings-account.PNG |

Systems and applications that manages the service levels of these kind of organizations are called **Enterprise Application**.  
  
**Activity**

* Go to <https://www.airtel.in/> and find out all products and services that are offered.
* SME to write the services offered by airtel on the board based on inputs from the learners.

**Monolithic Service**   
  
Let us take the frequently used operation in a Bank, get account balance of an account. We already understand that this operation can be initiated from various roles and using various devices. Few example scenarios where this service is used: 

* A customer viewing his account details in a mobile app
* A bank teller viewing account details of a customer
* A customer viewing account details over net banking application
* An IVR system reading out an account balance to a customer
* A customer service representative views the account details of a customer whom had called customer services
* A batch job that deducts EMI on your account should know the account balance before performing the transaction
* A customer viewing his account details in an ATM
* A SMS system that needs to send transaction details to customer's mobile should know the account balance, so that it can be included in the SMS text message.
* so on and so forth ......

Similar to the one above, various operations of a bank can get initiated from various devices, roles, internal systems, etc.  
  
A XYZ Software Services company develops an application for a Bank that implements all operations of the bank as RESTful Web Services. You name any service offered by the Bank, a service is available that can be consumed by the respective application. This project had been completed and had been recently launched live.  
  
After few days of launch, on one fine day at 4PM ...

* A loan agent was not able to submit a loan application, he might miss his monthly target
* An insurance agent was not able to process closure of an insurance and hand over the sum assured payment cheque. The customer is waiting for more than one hour to receive this cheque.
* A customer is waiting in customer service queue for the 25 minutes to report a stolen credit card

The primary reason for the above situation is that the RESTful Web Service application recently launched has become very slow in responding. Due to festival season shopping there were huge volume of transactions for getting account balance, since there was a memory leak in the code and there is not memory left, because of which new request coming to the server were either rejected or timed out.  
  
To overcome this situation the entire server had to be restarted. After restart, the situation becomes normal after 2 to 3 hours. The support team keeps their fingers crossed not sure when this issue crops us again.  
  
**Activity**  
Discuss with learners and come up with ideas to handle this situation

**Monolithic Services vs. Microservices**   
  
**Monolithic Services** - A large number of critical enterprise applications hosted as a single web application is called as monolithic service.  
  
The following are the drawbacks of this Monolithic Services:

* As everything is packaged in one EAR/WAR. If any one service has a performance or memory leak issue, it brings down all the services.
* There is no possiblity to try a different technology stack for building the services

**Microservices**- Instead of having a single monolithic service, the services can be split into multiple services. Taking the example from the bank, insurance related operations can be handled by a separate service. Each microservice will be running   
  
**Advantages of Microservices**

* Decentralized
* Independent (Let us consider the bank application, the get balance service failure resulted in bringing down other services related to insurance and loan processing. This single point failure can be avoided when implementing as microservices)
* Doing one this well
* Agility in development of a service
* Scalable - add multiple instances of a service with new hardware included without affecting the existing production environment
* Easier to identify which service is in fault
* Makes it easier for a new developer to understand the functionality of a service, enables continuous delivery.

**Challenges of Microservices**

* Developing distributed systems can be complex
* Initial implementation of microservice is difficult

**Creating Microservices for account and loan**   
  
In this hands on exercises, we will create two microservices for a bank. One microservice for handing accounts and one for handling loans.  
  
Each microservice will be a specific independent Spring RESTful Webservice maven project having it's own pom.xml. The only difference is that, instead of having both account and loan as a single application, it is split into two different applications. These webservices will be a simple service without any backend connectivity.  
  
Follow steps below to implement the two microservices:  
  
**Account Microservice**

* Create folder with employee id in D: drive
* Create folder named 'microservices' in the new folder created in previous step. This folder will contain all the sample projects that we will create for learning microservices.
* Open <https://start.spring.io/> in browser
* Enter form field values as specified below:
  + **Group:** com.fis
  + **Artifact:** account
* Select the following modules
  + Developer Tools > Spring Boot DevTools
  + Web > Spring Web
* Click generate and download the zip file
* Extract 'account' folder from the zip and place this folder in the 'microservices' folder created earlier
* Open command prompt in account folder and build using mvn clean package command
* Import this project in Eclipse and implement a controller method for getting account details based on account number. Refer specification below:
  + Method: GET
  + Endpoint: /accounts/{number}
  + Sample Response. Just a dummy response without any backend connectivity.

{ number: "00987987973432", type: "savings", balance: 234343 }

* Launch by running the application class and test the service in browser

**Loan Microservice**

* Follow similar steps specified for Account Microservice and implement a service API to get loan account details
  + Method: GET
  + Endpoint: /loans/{number}
  + Sample Response. Just a dummy response without any backend connectivity.

{ number: "H00987987972342", type: "car", loan: 400000, emi: 3258, tenure: 18 }

* Launching this application by having account service already running
* This launch will fail with error that the bind address is already in use
* The reason is that each one of the service is launched with default port number as 8080. Account service is already using this port and it is not available for loan service.
* Include "server.port" property with value 8081 and try launching the application
* Test the service with 8081 port

Now we have two microservices running on different ports.  
  
**NOTE:** The console window of Eclipse will have both the service console running. To switch between different consoles use the monitor icon within the console view.

**Create Eureka Discovery Server and register microservices**   
  
Eureka Discovery Server holds a registry of all the services that are available for immediate consumption. Anybody whom wants to consume a RESTful Web Service can come to the discovery server and find out what is available and ready for consumption. Eureka Discovery Server is part of spring cloud module.  
  
Follow steps below to implement:  
  
**Create and Launch Eureka Discovery Server**

* Using https://start.spring.io generate a project with following configuration:
  + Group: com.fis
  + Artifact: eureka-discovery-server
  + Module: Spring Cloud Discovery > Eureka Server
* Download the project, build it using maven in command line
* Import the project in Eclipse
* Include @EnableEurekaServer in class EurekaDiscoveryServerApplication
* Include the following configurations in application.properties:

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF

logging.level.com.netflix.discovery=OFF

* The above configuration runs the discovery service in port 8761
* The eureka properties prohibits direct registration of services, instead discovery server will find available services and register them.
* Launch the service by running the application class
* The discovery service can be view by launching http://locahost:8761 in the browser.
* This will display the discover server details
* Look into the section "Instances currently registered with Eureka", which will have an empty list
* Follow steps below to add account and loan service to this discovery server.

**Register Account REST API to eureka discovery**

* Go to https://start.spring.io and provide the following configuration:
  + Group: com.fis
  + Artifact: account
  + Modules:
    - Spring Boot DevTools
    - Eureka Discovery Client
    - Spring Web
* Click "Explore", which will open pom.xml
* Use copy option in the opened window to copy the pom.xml and overwrite the pom.xml in account project
* Build the project using maven in console
* Include @EnableDiscoveryClient annotation to  application class of account project
* Include application name for account application as specified below in application.properties. This is the name that will be displayed in the eureka discovery registry.

spring.application.name=account-service

* Stop all services (account, loan, eureka-discovery-server) using the console window of Eclipse. Use the monitor icon in console view to switch between applications and use the Terminate button to stop the server.
* First start eureka-discovery-server and wait till the application starts completely. Then open http://locahost:8761 in browser. The service list should be empty.
* Then start account application and wait till the application starts.
* Refresh the eureka-discovery-server web page in browser, the account-service will be listed in the registry
* Perform similar steps for loan application and have it registered with eureka-discovery-server.

Reference Code : FSE-MCR-006Complexity : Level1

**Create Zuul Gateway**   
  
In the microservices created in previous exercises, one could clearly see that the URL for each microservice is different.  
  
This severly impacts the angular code as we will have different end points for each service.  
  
To handle this issue, we are going to create a gateway that will act as a single point of entry for all REST APIs.  
  
The Zuul Gateway can also be configured as a load balancer for each REST API, which is not in the scope of this learning.  
  
Follow steps below to implement a gateway that acts as a single point of entry for account and loan REST APIs:

* Using https://start.spring.io create a new project named zuul-gateway with following dependencies
  + Zuul
  + Spring Boot DevTools
  + Spring Web
* Create maven build in command prompt
* Import zuul-gateway project in eclipse
* Include @EnableZuulProxy in ZuulGatewayApplication class
* Modify properties file to have the following properties

# routing for account service

zuul.routes.account-service.url=http://localhost:8080

# routing for loan service

zuul.routes.loan-service.url=http://localhost:8082

# disable load balancing

ribbon.eureka.enabled=false

# port for zuul proxy

server.port=8083

* Create a new class SimpleFilter that extends ZuulFilter class. Generate code for the methods to be implemented. Ensure following return value for each method:
  + run() - return null
  + shouldFilter() - return true
  + filterOrder() - return 1
  + filterType() - return "pre"
* Launch the application and try hitting the following URLs in the browser. This should display the results of account and loan application through zuul gateway.
  + <http://localhost:8083/account-service/accounts/324324>
  + <http://localhost:8083/loan-service/loans/324324>

​​​​​​​

#### [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

**Understand deployment of application in a remote host - PART 1**   
  
**Scenario**

* The applications angular-learn and spring-learn needs to be deployed in a remote windows server
* This servers can be accessed through Remote Desktop
* Except Windows Server operating system no other software is installed in this server

**Objective**

* Identify the list of software to be installed in the server for deploying spring-learn and angular-learn application [HINT: development tools should not be installed in the server (example: Eclipse)]

**Group Formation Guidelines for SME**  
Find the answer for the question in the above objective as a group activity.

* A group size should be four. If not possible to equally divide, few groups can have size of three.
* To randomize the group members, use the online tool available at  <https://www.aschool.us/random/random-pair.php>
* Enter “Max group size” as 4
* In the text area after “balance”, copy and paste learners list of the classroom. Each line should have one name
* Click “Submit” which will display the groups and the list of people in each group
* Project the screen and show the groups

**Expectation from each Group (Duration: 15 minutes)**

* Identify a leader in your group
* Identify a name for your group
* Each team member to share their point of view on the list of software required
* After discussion and consensus arrive at the final list of software required
* Write the final list of software in a paper under the below category:
  + For deploying angular-learn
  + For deploying spring-learn

**Finalizing the software list**

* The group leader has to convey the team name and the software list to the SME
* SME has to write the group name and the software list on the board under the respective team name
* Once all the teams had provided the details, the SME has to brainstorm and finalize the list of software required.

**Understand deployment of application in a remote server - PART 2**   
  
**Activity**

* List out the components for deploying angular-learn and spring-learn application with eureka discovery and zuul gateway
* For each component list out the steps required for deployment
* With the assumption that your PC has only the list of software identified in the previous activity
* The list of steps should only use the software identified and should not use any development tools
* Continue working as same groups formed in the previous activity
* The group lead of each team has to collect inputs from each team member, debate and arrive at the final list

**Expected Output in a white sheet**

* Component 1
  + Deployment step 1
  + Deployment step 2
* Component 2
  + Deployment step 1
  + Deployment step 2
  + Deployment step 3
* Component 3
  + Deployment step 1

**Activity Finalization**

* SME to collect inputs from each group
* The entire classroom has to collectively finalize the list of activities and write the list of activities on the board

**Deploy application in Desktop PC**   
  
**Scenario and Tasks**

* Assume your PC as remote server
* Refer activities from previous exercise and perform the deployment steps in your PC. (NOTE: Ensure that development tools are not used during deployment)
* Based on those steps ensure that angular-learn and spring-learn application works end to end
* Copy angular-learn and spring-learn projects to a new folder C:\Users\[EMP\_ID]\build. The perform the build and deployment steps for achieving the above objective

**Optimizing our deployment steps**   
  
Now we know the steps for deployment in a remote server, let us ponder over the below questions. 

* How much time is required to deploy our application in the remote server?

* Is there a possibility to have human error during deployment?

* Is it possible to automate deployment steps? (Find available tools by searching internet with keywords "deployment automation tools")

* In the remote server, JRE 11 is already installed and is used by an application. Our application is tested in JRE 8 and not tested in JRE 11. How do we address this issue? (Containerization addresses this issue, learn and find out various tools available by using the search keyword "container deployment tools")

**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.

#### [Deploying end to end application using compose](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

**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:

      - "3306:3306"

    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 (3306) denotes the port that will be exposed to 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

* If the start up of MySQL server fails with port conflict error, please follow the steps below to stop MySQL server running in the desktop PC
  + Open Task Manager
  + Click Services
  + Find item in the list that starts as “MySQL”
  + Right click on the item and select stop (later if MySQL needs to be started, come back to Task Manager and start it)
  + Run docker compose
* 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 above command 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

Reference Code : FSE-DKR-009Complexity : Level1

**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:

      - "3306:3306"

    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
    - 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:3306/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