INTRODUCTION TO MICROSERVICES WITH SPRING BOOT AND SPRING CLOUD

AGENDA

- Concept Study
- Build(Clone) our application
- Dockerize our application
- Deploy our application to Aws
- Conclusion

Concept Study

Docker is a container management service that eases building and deployment.

Many people use containers to wrap their Spring Boot applications, and building containers is not a simple thing to do.

In this presentation, we will cover how to dockerize a Spring boot application as well as how to deploy this dockerize application in Aws.

Concept Study

❑ Basic Concepts of Docker and Containers

Docker is an open-source containerization platform.

Containerization is simply a way of packaging software applications into containers.

Containers are executable units of software. They package everything your application needs including the code, libraries, and dependencies so that your application can run anywhere, in any environment.

Here are the benefits of using Containers:

- **Portable: you can run it in any environment whether it is on someone else's laptop or in the cloud without having to re-configure.**
- **Efficiency: Containers enable engineers to maximize the physical machine's CPU and memory usage.**
- **Modernize Applications: containers are suitable for cloud-native and microservice architectures.**

Concept Study

❑ Basic Concepts of Cloud and AWS

Cloud computing is the on-demand delivery of compute power, database, storage, applications, and other IT resources through a cloud services platform through the internet with pay-as-you-go pricing.

The AWS Cloud encompasses a broad set of global cloud-based products that includes compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing.

Here are the different service models which have emerged in cloud computing:

- **Infrastructure as a Service (IaaS)**
- **Platform as a Service (PaaS)**
- **Software as a Service (SaaS)**

Build(Clone) our application

❑ Introduction to the Project

We have implemented an user registration CRUD with just the Create and Read functionalities. We will proceed as follow

- **Create an entity class User with name, email, gender and so on. as attributes**
- **Create a dto UserRequestException**
- **Create an exception UserNotFound for custom exception handling**
- **Create an Interface UserRepository**
- **Create a class UserService for our business logic**
- **Create the controller UserController**
- **Create an exception advice ApplicationExceptionHandler for custom exception handling**

Build(Clone) our application

❑ Introduction to the Project

- **Create a package validation. This will contain our Validator as well as our New Annotation.**
- **Create a new @interface called ValidateGender to define our annotation. This contains the default message "Invalid Gender: It should be either Male or Female".**
- **Create a validator class called ValidatorGender that enforces the rules of our validation i.e test if the gender is either Male or Female.**
- **Finally, add the @ValidateGender annotation above the gender field into the UserRequest class into the dto package.**

Dockerize our application

❑ Install Docker Desktop on Windows

For this we will go to https://docs.docker.com/desktop/install/windows-install/ and follow all the steps

❑ Dockerize our application

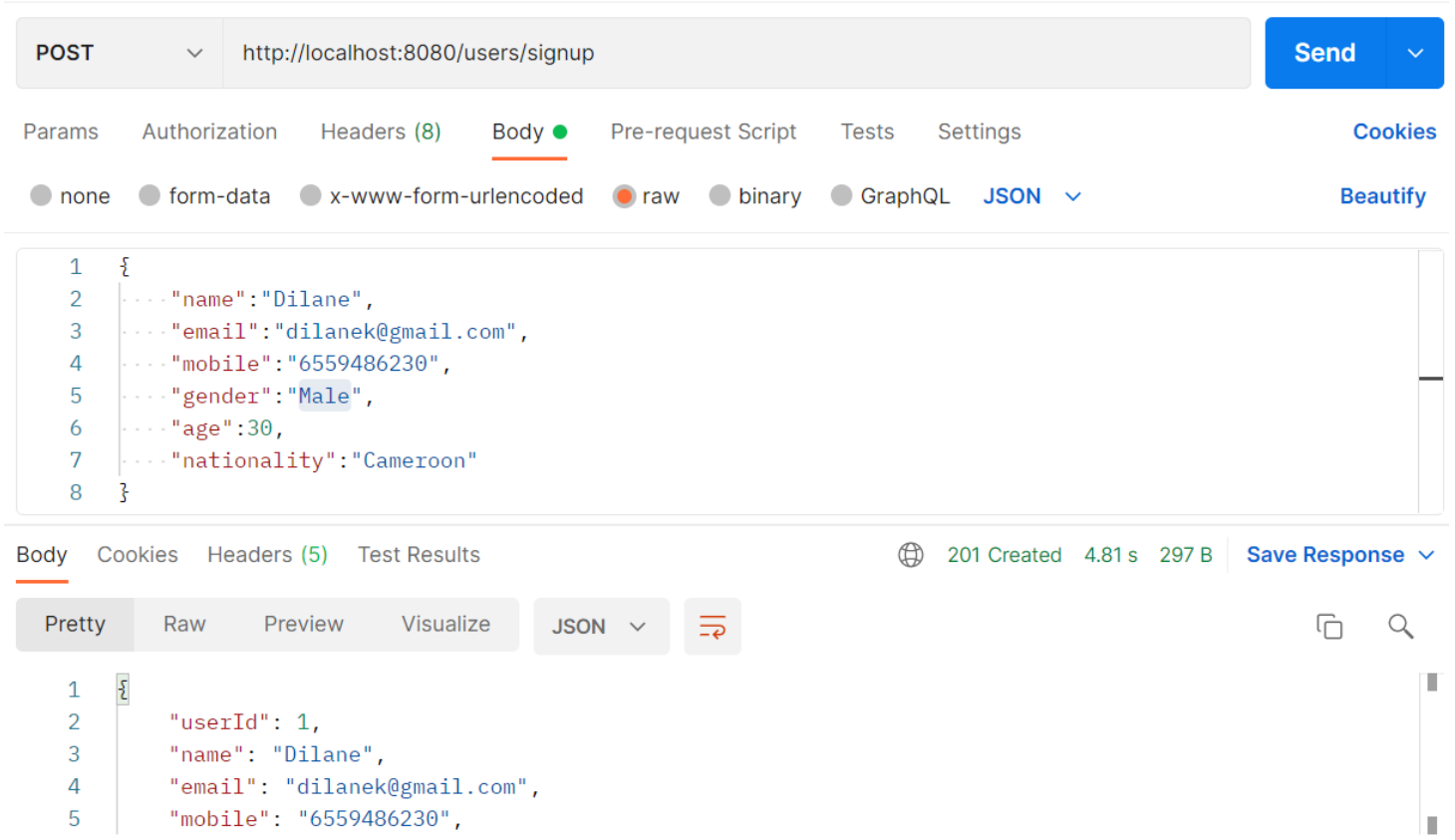
Previously, We have implemented an user registration CRUD. We will proceed as follow in order to dockerize our application.

- **Add a file name into a pom.xml at the end of build**
- **Create a Dockerfile at the root of our project with FROM, EXPOSE, ADD and ENTRYPOINT**
- **Run Docker Desktop**
- **Open our terminal, move to the project directory**
- **Create our docker image by typing docker build –t filename.jar .**
- **Type docker image ls to check the docker image**
- **Run the docker image by typing docker run –p 8080:8080 dockerimagename**

Dockerize our application

❑ Test the docker image

Let’s test our docker image using Postman



Dockerize our application

❑ Create a Docker Hub account

For this we will go to https://hub.docker.com/ and follow all the steps

❑ Push our image from docker desktop to docker hub

Previously, We have created our image. We will proceed as follow in order to push our image to docker hub.

- **Run Docker Desktop**
- **Open our terminal**
- **Create a tag by typing docker tag imagename username/imagename**
- **Type docker image ls to check the docker image**
- **Push the docker image to docker hub by typing docker push imagename**

Deploy our application to Aws

❑ Create a Aws account

For this we will go to https://aws.amazon.com/getting-started/ and follow all the steps

❑ Create a Aws EC2 instance

Previously, We have created our aws account. We will proceed as follow in order to create our aws ec2 instance.

- **Login into our aws account**
- **Type EC2 into the search bar**
- **Click on Launch instance and follow all the steps**
- **At the end create a new key pair, download it and save it into a folder**
- **Click on Launch instances**
- **Change inbound rules to allow sources from anywhere**

Deploy our application to Aws

❑ Download and install putty

For this we will go to https://www.puttygen.com/download-putty#PuTTY_for_windows and follow all the steps

❑ Pull our image from Docker Hub to Aws EC2 instance

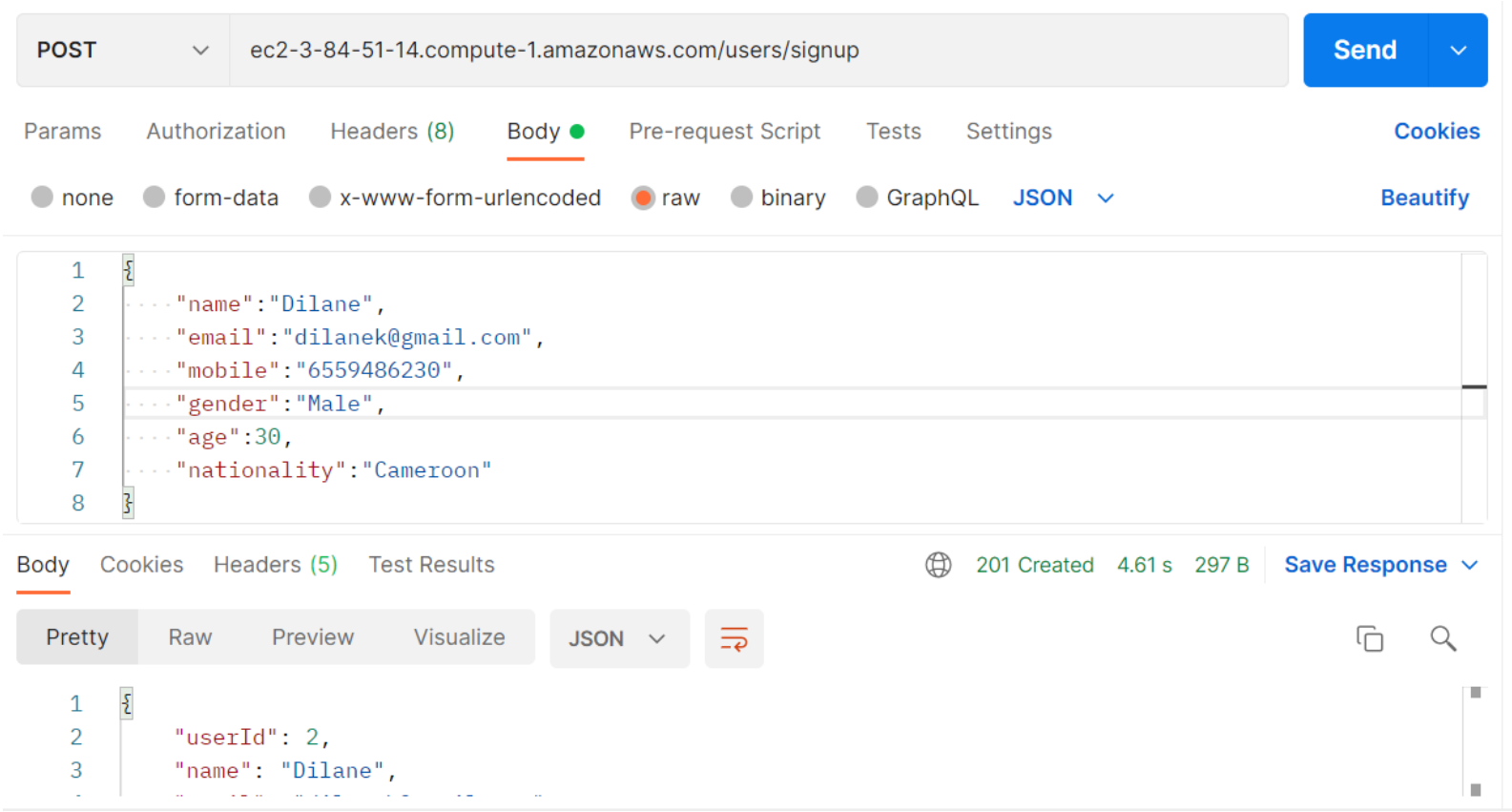
Previously, We have created our aws account as well as installed putty. We will proceed as follow in order to pull our image to aws ec2 instance.

- **Connect to our instance via putty**
- **Change to super user by typing sudo -i**
- **Update the system in order to install docker by typing sudo yum update -y**
- **Install docker by typing sudo yum install docker**
- **Click docker by typing sudo service docker start**
- **Pull the image from docker hub by typing docker pull imagename:tagname**
- **Run the image on aws ec2 by typing docker run –p 80:8080 imagename:tagname**

Custom Validation Implementation

- Test our image on aws ec2

We have created a new user, so our application works fine on AWS



Conclusion

We have learned how to create custom validators to verify a field or class, and then wire them into Spring MVC. The links of our resources are attached below

https://medium.com/geekculture/docker-basics-and-easy-steps-to-dockerize-spring-boot-application-17608a65f657

https://www.youtube.com/watch?v=e3YERpG2rMs&t=54s

https://www.youtube.com/watch?v=oGxkLH_OAlc&t=11s

https://www.youtube.com/watch?v=vmbhFZJUTR0&t=515s

https://towardsdatascience.com/deploying-a-docker-container-with-ecs-and-fargate-7b0cbc9cd608

Here is the GitHub repository

https://github.com/Dilane-Kamga/Springboot-Docker-Aws.git

MERCI

Pour votre attention