Docker + MongoDB

A useful guide to Docker basics

ARTICLES

Docker + MongoDB

January 21, 2019 - 5 minutes read - 889 words

Docker Introduction

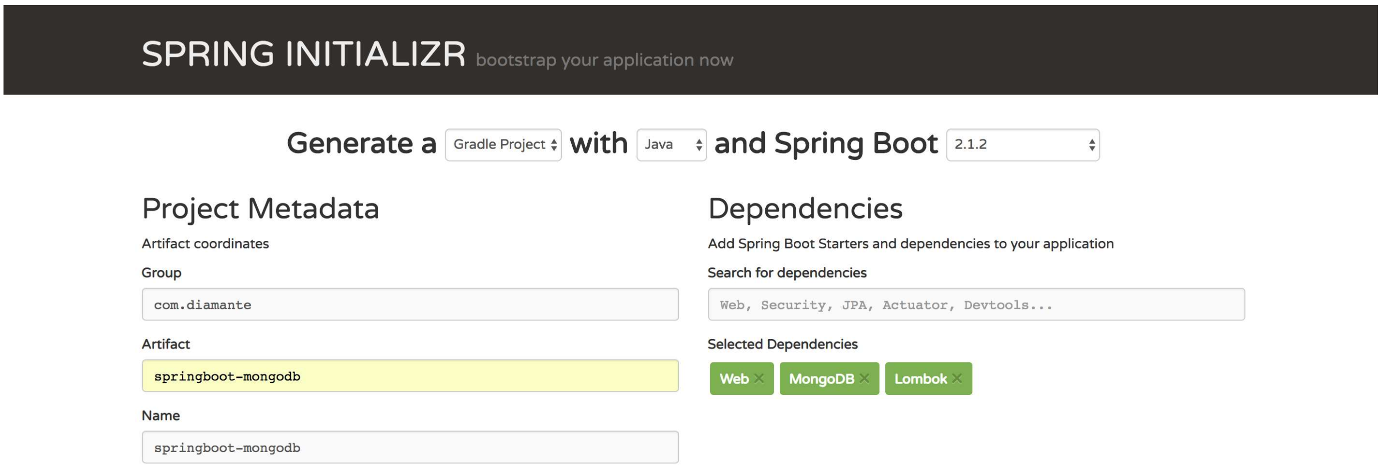
I guarantee you that any company you work for nowadays will be utilizing some kind of container for testing and deploying applications. The most popular of the container softwares is Docker. The main benefit of Docker is that if someone wants to run your application on their computer, they do not have to try to match your software versions just to run it. All they need to do is have Docker installed on their machine and they will be able to run your program in an isolated container that you have given the instructions on which version of software to run and how to launch the program.

Docker Terminology

A docker image is a mold that you can use to create containers out of. A docker container is the physical container that you run your program on. So a docker image can have many containers. A Dockerfile is a set of instructions that builds up your customized docker image. Tasks such as downloading specific files, running a linux package manager like apt-get or apk-get, and exposing ports for your application can all be used to build your custom image.

Let’s Dockerize a Spring Boot App

I am going to show you how to dockerize a Spring Boot application and link it to a MongoDB instance running in another container. Navigate to the Spring Initialzr and pick Web, MongoDB, and Lombok.



Generate a Gradle Project with Java and Spring Boot 2.1.2

Project Metadata
Artifact coordinates
Group: com.diamante
Artifact: springboot-mongodb
Name: springboot-mongodb

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies: Web, Security, JPA, Actuator, Devtools...
Selected Dependencies: Web, MongoDB, Lombok

Now download the zip and import it into IntelliJ or your preferred IDE.

Entity Class

This will be a simple Mongo JPA Data example as to focus more so on the dockerizing of the application.

```java
package com.diamante.springbootmongodb.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "employees")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Employee {
    @Id
    private Integer id;
    private String name;
    private String role;
    private double salary;
}
```

Employee.java hosted with ♡ by GitHub view raw

Repository Class

Now create a repository interface so we can interact with MongoDB. This will extend MongoRepository

```java
package com.diamante.springbootmongodb.repository;

import com.diamante.springbootmongodb.entity.Employee;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface EmployeeRepository extends MongoRepository<Employee, Integer> {
}
```

EmployeeRepository.java hosted with ♡ by GitHub view raw

Service Class

Now let's create a service class as to further extract the repository.

```java
package com.diamante.springbootmongodb.service;

import com.diamante.springbootmongodb.entity.Employee;
import com.diamante.springbootmongodb.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    public void save(Employee employee) {
        employeeRepository.insert(employee);
    }

    public List<Employee> getAll() {
       return employeeRepository.findAll();
    }

    public Employee getOne(Integer id) {
        return employeeRepository.findById(id).get();
    }

    public void update(Employee updatedEmployee) {
        Employee originalEmployee = employeeRepository.findById(updatedEmployee.getId()).get();
        if(originalEmployee != null) {
            originalEmployee.setName(updatedEmployee.getName());
            originalEmployee.setRole(updatedEmployee.getRole());
            originalEmployee.setSalary(updatedEmployee.getSalary());
            employeeRepository.save(originalEmployee);
        }
    }

    public void delete(Integer id) {
        employeeRepository.deleteById(id);
    }
}
```

EmployeeService.java hosted with ♡ by GitHub view raw

Controller Class

```java
package com.diamante.springbootmongodb.controller;

import com.diamante.springbootmongodb.entity.Employee;
import com.diamante.springbootmongodb.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.net.URISyntaxException;
import java.util.List;

@RestController
@RequestMapping(path = "/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @PostMapping
    public ResponseEntity<String> saveEmployee(@RequestBody Employee employee) throws URISyntaxEx
            employeeService.save(employee);
            return ResponseEntity.created(new URI("/employee/"+employee.getId())).build();
    }

    @GetMapping
    public ResponseEntity<List<Employee>> getAllEmployees() {
        List<Employee> allEmployees = employeeService.getAll();
        return ResponseEntity.ok(allEmployees);
    }

    @GetMapping("/{employeeId}")
    public ResponseEntity<Employee> getEmployee(@PathVariable Integer employeeId) {
        Employee employee = employeeService.getOne(employeeId);
        return ResponseEntity.ok(employee);
    }

    @PutMapping("/{employeeId}")
    public ResponseEntity<String> updateEmployee(@RequestBody Employee updatedEmployee) {
        employeeService.update(updatedEmployee);
        return ResponseEntity.accepted().body("Successfully updated!");
    }

    @DeleteMapping("/{employeeId}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Integer employeeId) {
        employeeService.delete(employeeId);
        return ResponseEntity.noContent().build();
    }
}
```

EmployeeController.java hosted with ♡ by GitHub view raw

MongoDb Initial Loader Class

```java
package com.diamante.springbootmongodb.config;

import com.diamante.springbootmongodb.entity.Employee;
import com.diamante.springbootmongodb.repository.EmployeeRepository;
import com.diamante.springbootmongodb.service.EmployeeService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@EnableMongoRepositories(basePackageClasses = EmployeeRepository.class)
@Configuration
public class MongoDBConfig {
    @Bean
    CommandLineRunner commandLineRunner(EmployeeService employeeService) {
        return strings -> {
          employeeService.save(new Employee(1, "Bobby", "Java Developer", 80000));
          employeeService.save(new Employee(2, "Sam", "Dev Ops", 120000));
          employeeService.save(new Employee(3, "Hank", "Human Resources", 65000));
        };
    }
}
```

MongoDBConfig.java hosted with ♡ by GitHub view raw

Change the port if you wish

In application.properties we can change the port (optional) by adding this line:

```
server.port=8085
```

Download and run mongoDb container

Instead of downloading MongoDB on our laptop, we can run an instance of MongoDB in a docker container and link our Spring Boot project to it.

The image we will be using is named mongo and is hosted on Docker Hub which is Docker’s version of GitHub.

```
docker run -d -p 27017:27017 --name mongo mongo
```

This command will download the mongo image from Docker Hub and run a container based off of that image. By default, Docker names all container with random names, but using –name we can give the container a fixed name. Doing this makes it easier to interact with later and it is easier to remember than focused_lamarr.

By providing the flag -d this tells the container to be run in the background. The flag -p exposes and binds the port of the container to the port of the local machine.

Basic Commands

List all images

```
docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
springboot-mongo    latest              52855cc58fb2        6 days ago          124M
mongo               latest              7177e01e8c01        4 weeks ago         393M
openjdk             8-jdk-alpine        97bc1352afde        3 months ago        103M
```

List all containers

```
docker ps -a

CONTAINER ID        IMAGE               COMMAND                  CREATED
6898625112f3        mongo               "docker-entrypoint.s…"   4 minutes ago
```

Stop container

```
docker stop mongo
```

Start container

```
docker start mongo
```

Remove container

```
docker rm mongo
```

Remove image

```
docker rmi {IMAGE ID}
```

Dockerfile

There are many images that are available for you to use on Docker Hub, however most times we need to design a custom image instead. That is where the Dockerfile comes into play.

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
EXPOSE 8085
ADD build/libs/springboot-mongodb-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-Dspring.data.mongodb.uri=mongodb://mongo/test", "-jar", "app.jar"]
```

Dockerfile hosted with ♡ by GitHub view raw

This is a specific set of instructions on how to build the image and what commands to run once a container is built from this image.

Our Dockerfile is simple. The container will run the jar and give it a mongodb related property.

Build the image

Before building our image, let's build an application jar using gradle:

```
./gradlew clean build -x test
```

Now build the image:

```
docker build -t springboot-mongo .
```

This command is building an image off of the Dockerfile in the current directory. The image will be tagged with the name springboot-mongo.

Run a container and link it to the already running mongo container

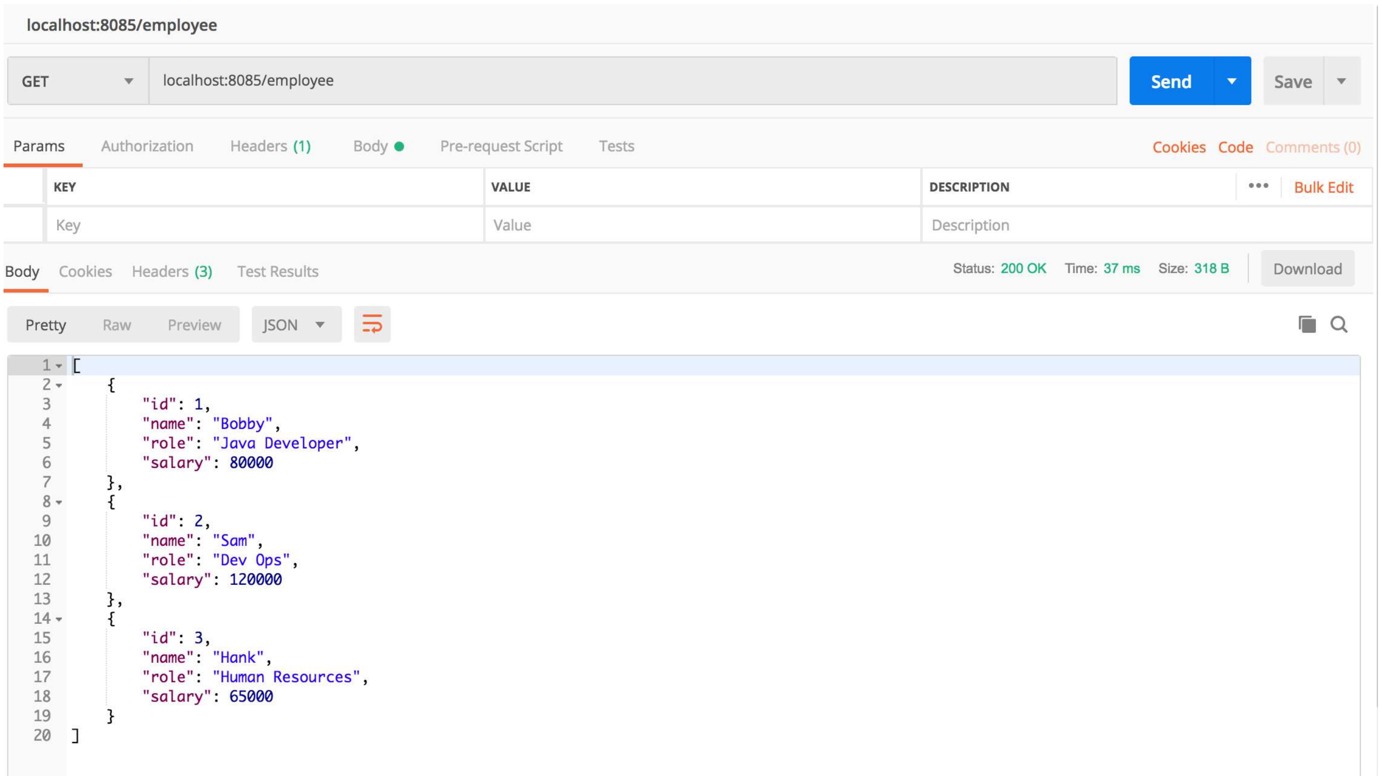
```
docker run -d -p 8085:8085 --name springboot-mongo --link=mongo  springboot-mongo
```

Test out the Rest API

Open up Chrome or an API development environment like Postman or Insomnia and try it out!

I'll be using Postman to show that the Rest Controller is operating as expected.

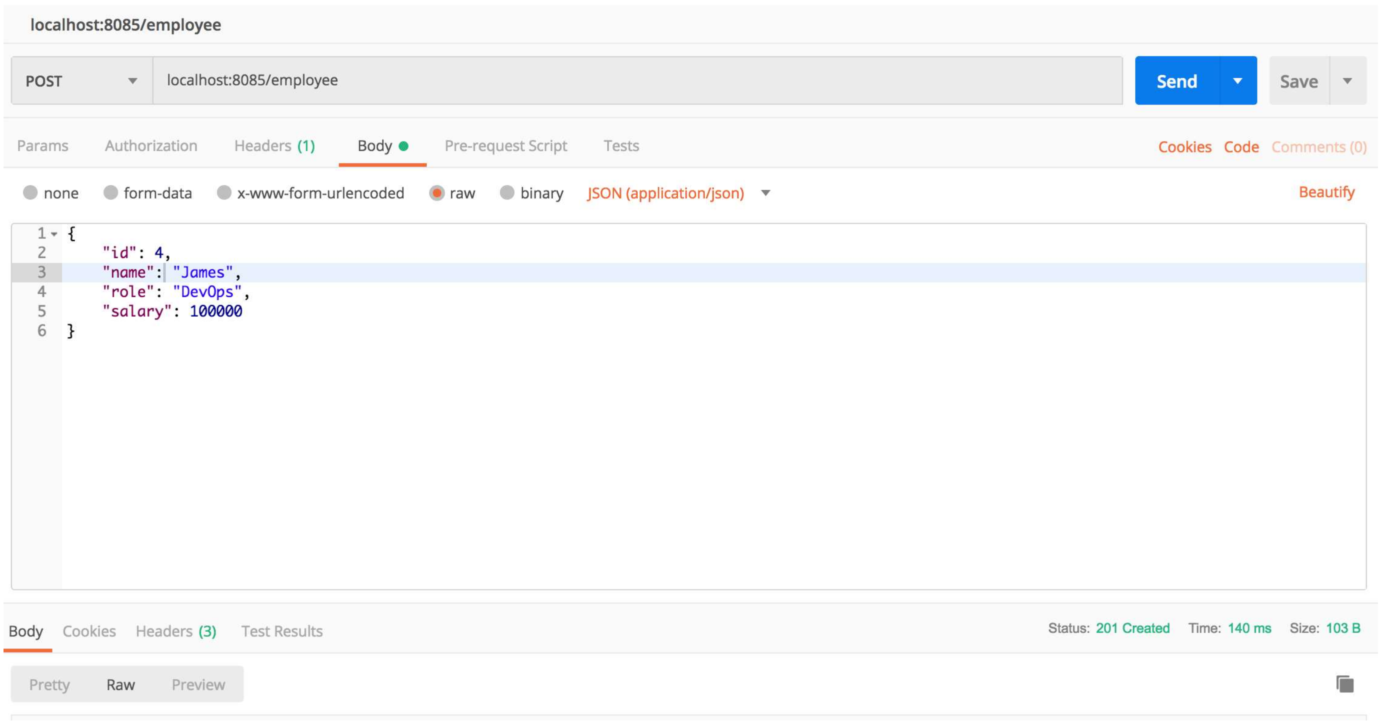
Get all employees



Status: 200 OK Time: 37 ms Size: 318 B

```
[
    {
        "id": 1,
        "name": "Bobby",
        "role": "Java Developer",
        "salary": 80000
    },
    {
        "id": 2,
        "name": "Sam",
        "role": "Dev Ops",
        "salary": 120000
    },
    {
        "id": 3,
        "name": "Hank",
        "role": "Human Resources",
        "salary": 65000
    }
]
```

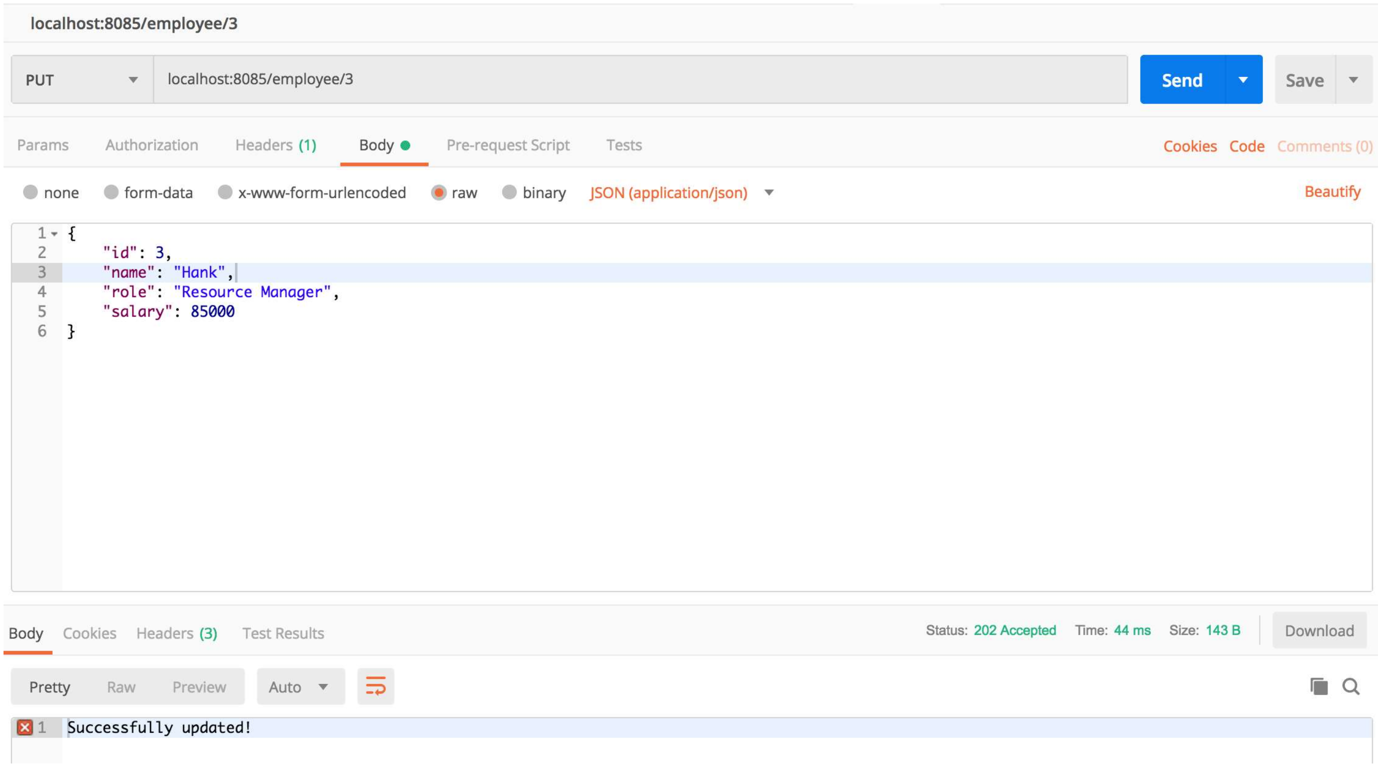
Create new employee



```
{
    "id": 4,
    "name": "James",
    "role": "DevOps",
    "salary": 100000
}
```

Status: 201 Created Time: 140 ms Size: 103 B

Update existing employee



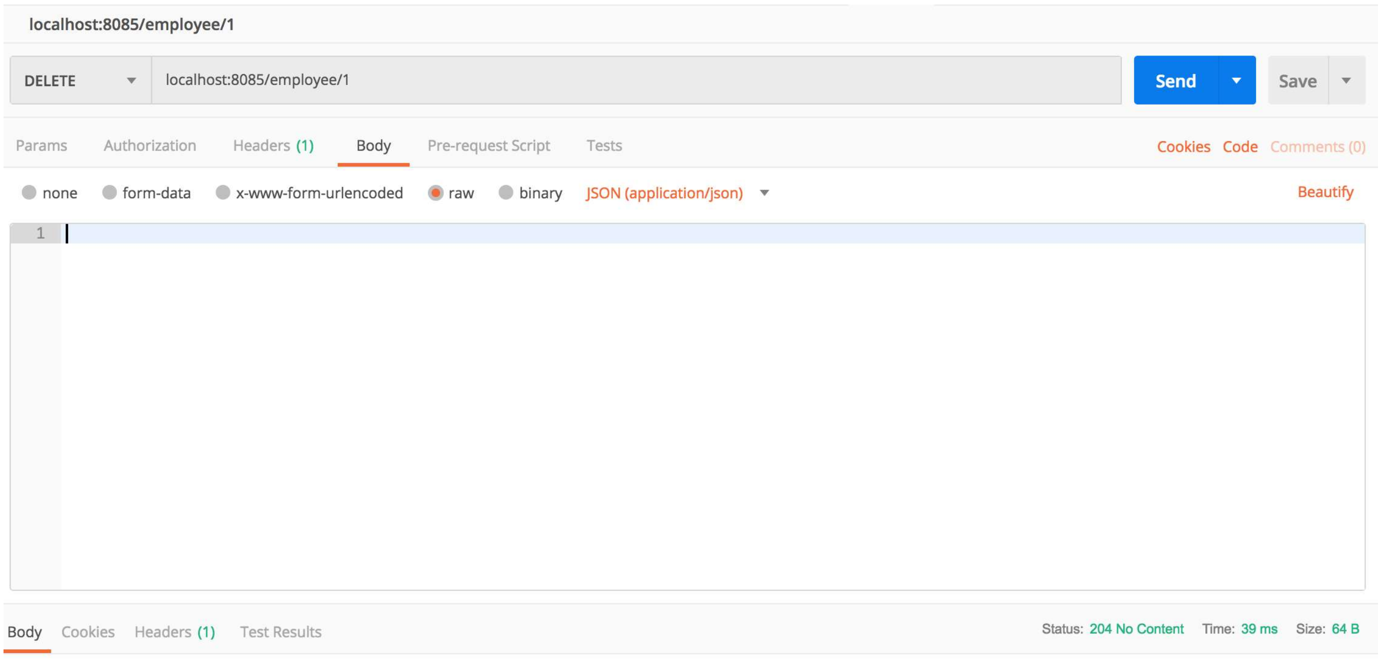
localhost:8085/employee/3

```
{
    "id": 3,
    "name": "Hank",
    "role": "Resource Manager",
    "salary": 85000
}
```

Status: 202 Accepted Time: 44 ms Size: 143 B

```
Successfully updated!
```

Delete an employee



localhost:8085/employee/1

Status: 204 No Content Time: 39 ms Size: 64 B

Now Bobby is gone

```
[
    {
        "id": 2,
        "name": "Sam",
        "role": "Dev Ops",
        "salary": 120000
    },
    {
        "id": 3,
        "name": "Hank",
        "role": "Resource Manager",
        "salary": 85000
    },
    {
        "id": 4,
        "name": "James",
        "role": "DevOps",
        "salary": 100000
    }
]
```

Conclusion

Docker is a fantastic tool that is utilized in all environments. If you are just testing an application or you are deploying it through a CI/CD pipeline you'll want to get the help of containers.

When I rotated onto the Android team at my job I utilized Docker right away. We were using a sandbox application to mock our server. We would have IntelliJ running the sandbox application and hit it with the emulator in Android Studio. We spent so much time waiting because two huge IDEs were running at the same time.

```bash
#!/bin/bash

echo "Restarting Sandbox Docker Container!"
echo "Which container would you like to build and run?"
while true; do
echo "1) Android Team Sandbox
2) IOS Team Sandbox"
read team
case $team in
  1) echo "Welcome Android"
        ./gradlew clean build -x test
        docker stop sandbox
        docker rm sandbox
        docker build -f Android_Docker/Dockerfile -t mobileandroid/sandbox-0.0.1 .
        docker run -p 8082:8082 -t --name sandbox mobileandroid/sandbox-0.0.1
        break
        ;;
  2) echo "Welcome IOS"
```

```
            ./gradlew clean build -x test
            docker stop sandbox
            docker rm sandbox
            docker build -f IOS_Docker/Dockerfile -t mobileios/sandbox-0.0.1 .
            docker run -p 8080:8080 -t --name sandbox mobileios/sandbox-0.0.1
            break
          ;;
      *) echo "This option does not exist, please try again.";;
    esac

    done
```

restart_sandbox.sh hosted with ♡ by GitHub view raw

I took the initiative and built the sandbox a docker container. I also wrote a bash script to quickly run all the commands necessary and restart the sandbox whenever necessary. The team loved it and continues to see the time saved every day.

docker

springboot

rest

mongodb

"Every great developer you know got there by solving problems they were unqualified to solve until they actually did it."
- Patrick McKenzie