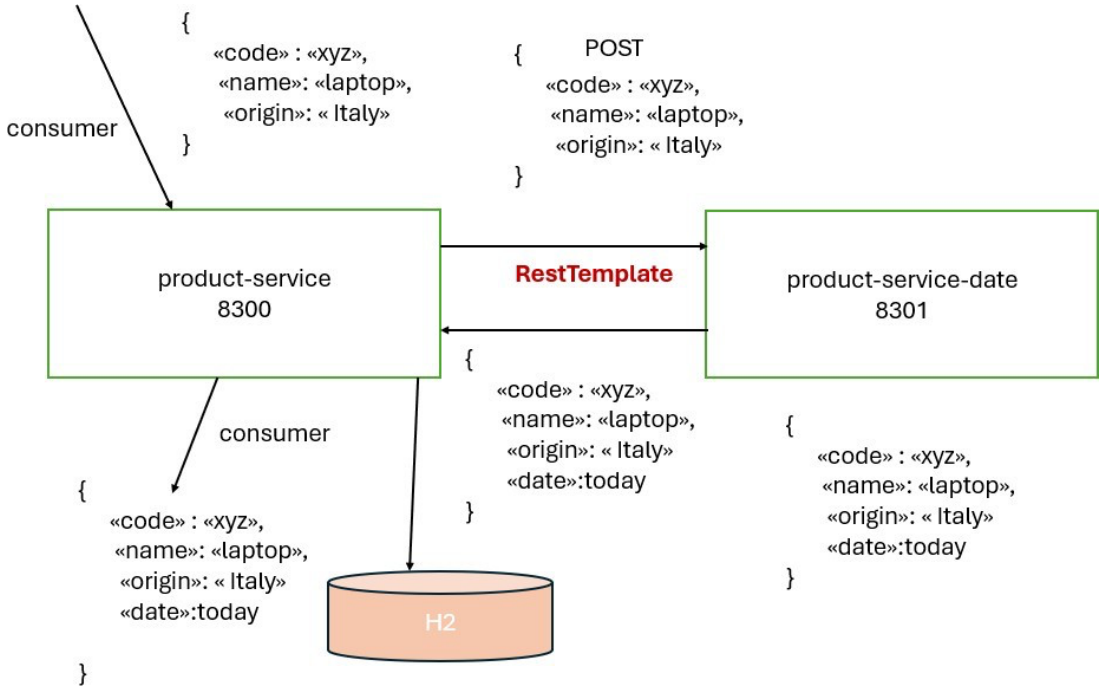
COMUNICAZIONE TRAMITE MICROSERVIZI VIA RestTemplate E DOCKER CONTAINERS

ARCHITETTURA APPLICATIVA REALIZZATA



REALIZZAZIONE PRIMO MICROSERVIZIO (product-service) - MAVEN COME package manager

1. Starters: web, spring data,h2

2. configurazione application.properties

```
spring.application.name=product-service
server.port=8300
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

3. Implementazione Entity Product

```
@Entity
public class Product implements Serializable {

  private static final long serialVersionUID = 1L;

  @Id
  private String code;

  private String name;

  private String origin;
```

```
    public String getCode() {

        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;

    }

    public void setName(String name) {
        this.name = name;
    }

    public String getOrigin() {
        return origin;
    }

    public void setOrigin(String origin) {
        this.origin = origin;
    }


    protected Product() {

    }

    public Product(String code, String name, String origin) {

        this.code = code;
        this.name = name;
        this.origin = origin;
    }
}
```

4. Implementazione Interfaccia Repository Spring Data JPA

```
public interface ProductRepository extends JpaRepository<Product, String> {}
```

5. Implementazione Service Layer

```
package com.sistemi.informativi.service;

import com.sistemi.informativi.entity.Product;

public interface ProductService {

    public Product saveProduct(Product product);

}
```

```
package  com.sistemi.informativi.service;

import org.springframework.stereotype.Service;

import com.sistemi.informativi.entity.Product;
import com.sistemi.informativi.repository.ProductRepository;

@Service
public class ProductServiceImpl implements ProductService{

	private ProductRepository productRepository;

	public ProductServiceImpl(ProductRepository productRepository) {

		this.productRepository = productRepository;
	}


	// BUSINESS LOGIC TO DO

	@Override
	public Product saveProduct(Product product) {

		return productRepository.save(product);
	}

}
```

6. Implementazione RestController

```
package com.sistemi.informativi.controller;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import com.sistemi.informativi.entity.Product;
import com.sistemi.informativi.service.ProductService;

@RestController
@RequestMapping("/rest/api/products")
public class ProductController {

	/*
	 * Una volta chiesto a Spring Boot di istanziare il RestTemplate possiamo
	 * chiedere la DI di RestTempate
	 */

	private ProductService productService;

	private RestTemplate restTemplate;
```

```
public ProductController(ProductService productService, RestTemplate restTemplate) {

this.productService = productService;
this.restTemplate = restTemplate;
}

/*
 * Questo Microservizio deve ricevere un JSON che rappresenta un prodotto
 * ESEMPIO: { "code":"xyz", "name:"laptop", "origin":"Italy" }
 *
 * Dopo aver ricevuto il JSON fa in modo che il suo contenuto venga salvato sul
 * database H2 Successivamente invia il JSON del prodotto ad un secondo
 * microservizio che chiameremo product-service-date e che starà in ascolto
 * sulla porta 3301 con url rest/api/products/date
 *
 *
 */

@PostMapping
public Object saveAndSendProduct(@RequestBody Product product) {

/* URL DEL MICROSERVZIO CON CUI COMUNICHIAMO AL QUALE INVIAMO IL JSON DEL PRODOTTO; FAREMO LA CHIAMATA SU UN DOCKER CONTAINER CHIAMATO microservice2*/
String apiURL = "http://microservice2:8301/rest/api/products/date";

// IL PRODOTTO VIENE INSERITO SU H2
productService.saveProduct(product);

/*
 * Tramite il metodo postForObject consumiamo una operazione esposta sotto forma
 * di post dal secondo microservizio
 */
return restTemplate.postForObject(apiURL, product, Object.class);

/*
 * VIENE RESTITUITO AL CONSUMER UN JSON ARRICCHITO CON DATA ODIERNA ESEMPIO {
 * "code":"xyz", "name:"laptop", "origin":"Italy", "date" :today }
 */

}

}
```

REALIZZAZIONE SECONDO MICROSERVIZIO (product-service-date) - MAVEN COME package manager

1. Starters:web

2. configurazione application.properties
   spring.application.name=product-service-date
   server.port=8301

3. Implementazione JavaBean Product

```
public class Product {

 private String code;

 private String name;

 private String origin;

 private Date date;

 public String getCode() {
        return code;
}

 public void setCode(String code) {
        this.code = code;
}

 public String getName() {
        return name;
}

 public void setName(String name) {
        this.name = name;
}

 public String getOrigin() {
        return origin;
}


 public void setOrigin(String origin) {
        this.origin = origin;
}


 public Date getDate() {
        return date;
}

 public void setDate(Date date) {
        this.date = date;
}

 protected Product() {

 }
 public Product(String code, String name, String origin, Date date) {
```

```
            this.code=code;
            this.name = name;
            this.origin = origin;
            this.date = date;
        }


}
```

4. Implementazione @RestController

```
@RestController
@RequestMapping("/rest/api/products/dates")
public class ProductDateController {

        @PostMapping
        public @ResponseBody Object addDateToProduct(@RequestBody Product product){


            Product receivedProduct = product;
            receivedProduct.setDate(new Date(System.currentTimeMillis()));

            return receivedProduct;

        }

}
```

DOCKER E' UNA TECNOLOGIA CHE CONSENTE DI CREARE DEI DOCKER CONTAINERS PER ESEGUIRE IN MANIERA PERFORMANTE, VELOCE E TRASPARENTE DAL SISTEMA OPERATIVO, APPLICAZIONI E MICROSERVIZI. DOCKER SI ADATTA INFATTI AL SISTEMA OPERATIVO IN AUTONOMIA; EVITANDO COMPLESSE CONFIGURAZIONI, SUPERANDO DI FATTO LA TECNOLOGIA HYPERVISOR DELLE MACCJINE VIRTUALI, PIU' ONEROSA E "PESANTE".

INSTALLANDO IL SOFTWARE DOCKER, ABBIAMO A DISPOSIZIONE UN DOCKER CLIENT, CHE SI CONNETTE AL DOCKER HUB DANDO LA POSSIBILITA' DI SCARICARE DOCKER IMAGES GIA' PRONTE, O DI SCARICARE PIU' DOCKER IMAGES ASSEMBLANDOLE IN UNA DOCKER IMAGE CUSTOM.

UNA DOCKER IMAGE E' UN TEMPLATE IN CUI SI POSSONO CONFIGURARE DIPENDENZE, RUNTIME ENVIRONMENT COME LA JVM, E SOFTWARE DI VARIO GENERE.

LA DOCKER IMAGE NON HA UNA VITA REALE, MA A PARTIRE DA UNA DOCKER IMAGE OCCORRE ISTANZIARE UN DOCKER CONTAINER SUL QUALE SI PUO' ESEGUIRE L'ARTIFACT DI UNA APPLICAZIONE MONOLITICA O UN MICROSERVIZIO.

L'ARTIFACT DI UN PROGETTO SPRING BOOT E' RAPPRESENTATO DA UN FILE .jar, DETTO ANCHE PACKAGE OF DEPLOYMENT. IL FILE .jar E' L'INSIEME DEI FILE COMPILATI DELL'APPLICAZIONE, OVVERO DEI FILE .class GENERATI DAL COMPILATORE.

E' POSSIBILE ESEGUIRE UN ARTIFACT .jar GENERATO DA UN MICROSERVIZIO SPRING BOOT SU UN DOCKER CONTAINER, A PARTIRE DA UNA DOCKER IMAGE CONFIGURATA TRAMITE FILE Dockerfile.

PROCESSO DI DOCKERIZZAZIONE DELL'ARCHITETTURA APPLICATIVA

1. CREAZIONE Dockerfile all'interno del Progetto Spring relativo a product-service

```
FROM openjdk:17-jdk-slim
COPY ./target/product-service-0.0.1-SNAPSHOT.jar product-service-0.0.1-SNAPSHOT.jar
CMD ["java","-jar","product-service-0.0.1-SNAPSHOT.jar"]
```

2. GENERAZIONE JAR (ARTIFACT) product-service

```
mvn clean package && java -jar target/product-service-0.0.1-SNAPSHOT.jar
```

3. BUILD IMMAGINE DOCKER product-service

```
docker image build -t microservice1 .
```

4. CREAZIONE Dockerfile all'interno del Progetto Spring relativo a product-service-date

```
FROM openjdk:17-jdk-slim
COPY ./target/product-service-date-0.0.1-SNAPSHOT.jar product-service-date-0.0.1-SNAPSHOT.jar
CMD ["java","-jar","product-service-date-0.0.1-SNAPSHOT.jar"]
```

5. GENERAZIONE JAR (ARTIFACT) product-service-date

```
mvn clean package && java -jar target/product-date-service-0.0.1-SNAPSHOT.jar
```

6. BUILD IMMAGINE DOCKER product-service-date

```
docker image build -t microservice2 .
```

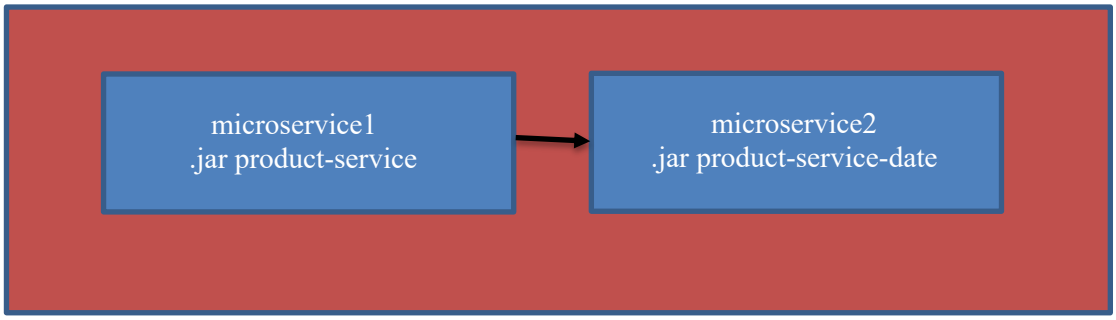
7. CREAZIONE NETWORK DOCKER

```
docker network create microservice1-microservice2
```

8. CREAZIONE CONTAINERS RELATIVI ALLE DUE IMMAGINI CREATE PRECEDENTEMENTE

```
docker container run --network microservice1-microservice2 --name microservice1 -p 8300:8300 -d microservice1
docker container run --network microservice1-microservice2 --name microservice2 -p 8301:8301 -d microservice2
```

ARCHITETTURA CONTAINERIZZATA



DOCKER NETWORK microservice1-microservice2

TEST FINALE CON POSTMAN

POST http://localhost/8300/rest/api /products

con JSON

{

"code":"xyz",

"name":"laptop",

"origin":"Italy