

PROGETTO MICROSERVICES

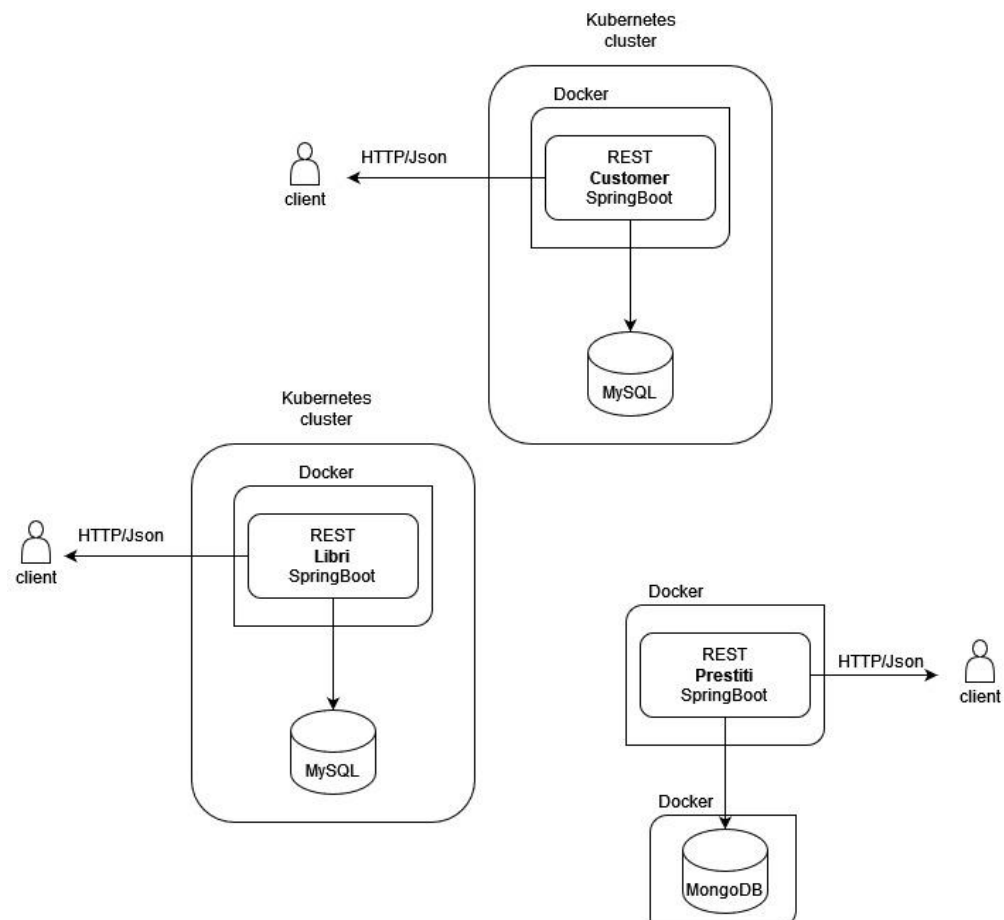
Microservices can be developed in any any technology , but they must comply with following constraints:

- Use HTTP/REST for synchronous communication
- Use at least two different database technology (RDBMS and NoSQL).
- Use a message broker (Kafka, Active MQ, Rabbit MQ) for asynchronous communications (ie: calling the Notification service)

Evaluation criteria:

- Microservices (0 to 5 points)
- Design patterns (0 to 5 points)
- Testing (0 to 5 points)
- Logging and tracing (0 to 5 points)
- CI/CD (0 to 5 points)
- Docker and Kubernetes (0 to 5 points)

□ SCHEMA PROGETTO FINALE

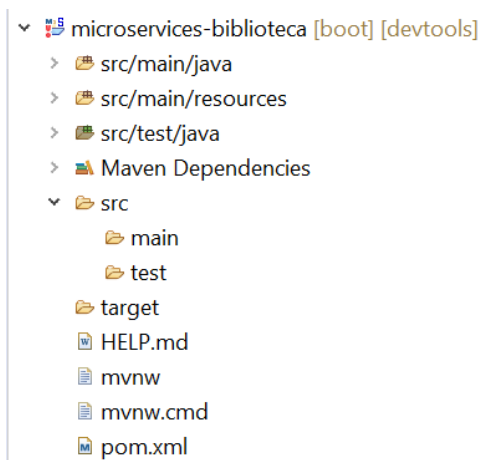


□ CREAZIONE DEL PROGETTO SPRING SU ORACLE ECLIPSE

SpringBoot → framework

Durante la creazione del progetto springboot/maven utilizzo come dependencies:

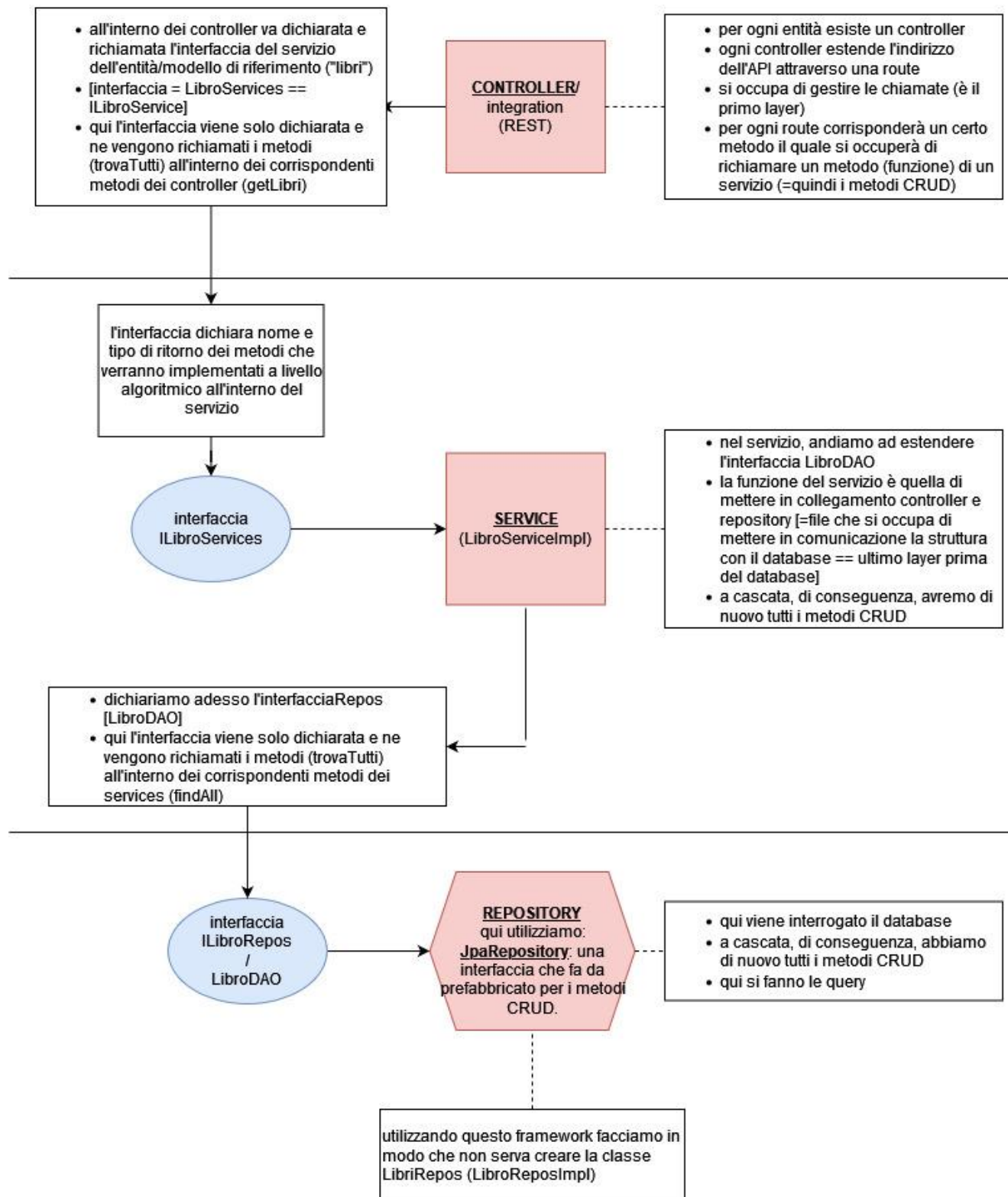
- spring boot dev tools → per deployare direttamente sul server
- spring web
- MySQL driver
- spring data JPA → per le operazioni CRUD



file **POM.XML**: file di configurazione, Project Object Model.

Infrastruttura in sottopacchetti del main package "biblioteca"

- creo il package che conterrà i modelli: "**biblioteca.entities**"
- creo il package che conterrà l'accesso al dato: "**biblioteca.repos**". Il repos contiene il **DAO** (Data Access Object)
- creare il package che conterrà il collegamento al DAO: "**biblioteca.services**"
- creo un package "**biblioteca.integration**", contenente i/controller.



→ `private LibroService ls;`

LibroService è l'oggetto "interfaccia" chiamato ls.

Infatti `trovaTutti()` non è un metodo definito in questa classe! Viene definito nell'interfaccia e nell'implementazione dell'interfaccia!

```
LibriREST.java LibroService.java LibroServiceImpl.java
1 package biblioteca.integration;
2
3 import java.util.Collections;
18
19 @RestController
20 @RequestMapping("/api/libri")
21 public class LibriREST {
22
23     @Autowired
24     private LibroService ls;
25
26     @GetMapping("")
27     List<Libro> getLibri(){
28
29         List<Libro> trovaTutti = ls.trovaTutti();
30         Collections.sort(trovaTutti);
31         return trovaTutti;
32     }
33 }
```

→ interfaccia

```
LibriREST.java LibroService.java LibroServiceImpl.java
1 package biblioteca.services;
2
3 import java.util.List;
6
7 public interface LibroService {
8
9     List<Libro> trovaTutti();
10     Libro trovaUno(int id);
11     Libro addLibro(Libro l);
12     Libro updateLibro(Libro libro);
13     void deleteLibro(int id);
14
15 }
```

→ implementazione dell'interfaccia

```
LibriREST.java LibroService.java LibroServiceImpl.java
11 @Service
12 public class LibroServiceImpl implements LibroService {
13
14     @Autowired
15     private LibroDAO repo;
16
17     @Override
18     public List<Libro> trovaTutti() {
19
20         return repo.findAll();
21     }
22 }
```

→ Ho aggiunto le annotazioni per mappare l'entità in base al database di **xampp** che ho deciso di utilizzare.

```
3 import javax.persistence.Entity;  
4 import javax.persistence.Table;  
5  
6 @Entity  
7 @Table (name = "libri")  
8
```

“table” serve per indicare il nome della tabella di riferimento

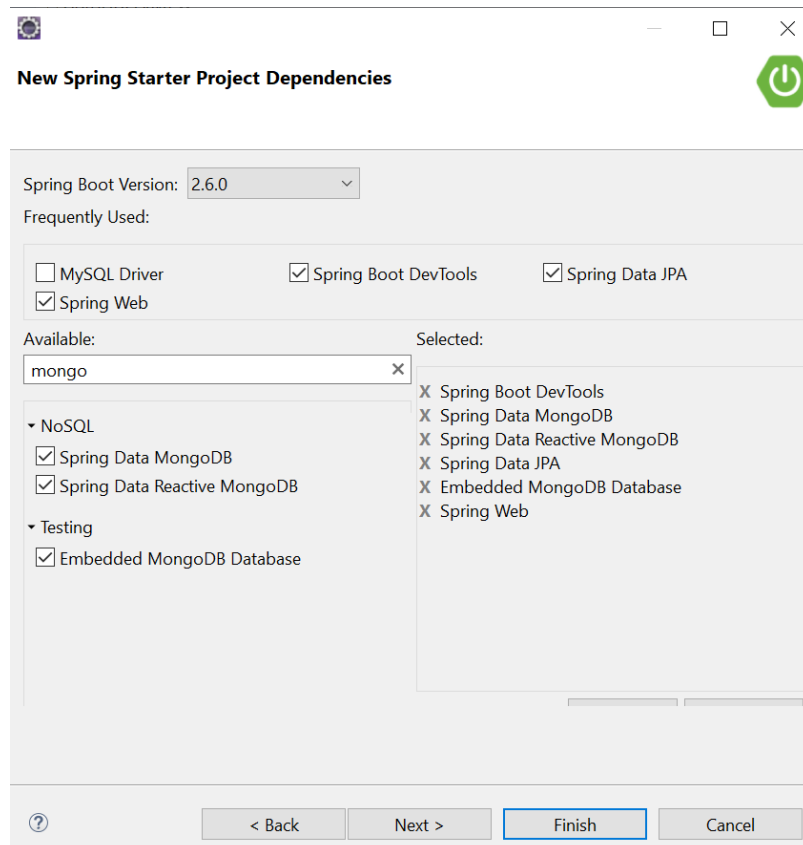
ATTENZIONE!: macchina virtuale linux e macchina fisica windows usano due versioni di java diverse! per avviare il progetto è sufficiente cambiarla dal file pom.xml

UTILIZZO DI DATABASE RELAZIONALI E NON RELAZIONALI

LIBRI E CUSTOMER: database relazionale, **mySQL** → JPA repository

PRESTITI: database non relazionale, **MongoDB** → MongoRepository

[di seguito le dipendenze inserite nella creazione del progetto con l'utilizzo del database non relazionale]



Dopo aver creato i servizi in modo che siano collegabili a database relazionali e non, il collegamento effettivo va fatto nella sezione “application properties”

→ non relazionale, MONGODB “prestiti”:

```

1 #Local MongoDB config
2 spring.data.mongodb.authentication-database=admin
3 spring.data.mongodb.username=root
4 spring.data.mongodb.password=root
5 spring.data.mongodb.database=customer_db
6 spring.data.mongodb.port=27017
7 spring.data.mongodb.host=customer-mongodb
8
9 # App config
10 server.port=8102
11 spring.application.name=CustomerMicroservice
12 server.servlet.context-path=/customers-service
13
14 # Log config
15 logging.level.org.springframework.web.servlet.DispatcherServlet=TRACE
16
17 # Management config
18 management.endpoint.info.enabled=true
19 management.security.enabled=false
  
```

→ relazionale, MYSQL “libri” e “customers”

Tabella	Azione	Righe	Tipo	Codifica caratteri	Dimensione
<input type="checkbox"/> autori		585	InnoDB	utf8mb4_general_ci	48.0 KiB
<input type="checkbox"/> customers		3	InnoDB	utf8mb4_general_ci	16.0 KiB
<input type="checkbox"/> editori		443	InnoDB	utf8mb4_general_ci	48.0 KiB
<input type="checkbox"/> libri		1,147	InnoDB	utf8mb4_general_ci	112.0 KiB
4 tabelle	Totali	2,178	InnoDB	utf8mb4_general_ci	224.0 KiB

```

1 spring.datasource.url=jdbc:mysql://localhost:3306/libreria
2 spring.datasource.username=root
3 spring.datasource.password=
4
5 spring.jpa.show-sql=true
6
7 server.port=9000
  
```

[
comandi **xampp** su **linux**:

- con il comando: `sudo /opt/lampp/lampp start`
che avvierà automaticamente tutti i servizi di XAMPP senza passare per il pannello di controllo grafico.
- oppure avviando il pannello di controllo utilizzando il comando:
`sudo /opt/lampp/manager-linux-x64.run`
e poi avviare i servizi di XAMPP dalla scheda "Manage Servers"

]

Dopo aver collegato i database a tutte le entità (i database in locale li ho allocati sia sulla mia macchina fisica che sulla mia macchina virtuale), ho provato a testare i vari verbi HTTP su **postman**, inserendo le route che ho indicato nelle varie classi.

→ molto importante mappare correttamente le varie entità!

```
Prestiti.java application.... application.... LibriREST.java Libri/pom
1 package biblioteca.entities;
2 import javax.persistence.Entity;
3
4 @Entity
5 @Table(name = "libri")
6 public class Libro implements Comparable<Libro>{
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10
11     private int id;
12     private String titolo;
13     private String autori;
14 }
```

postman:

The screenshot shows the Postman interface with a GET request to `http://localhost:9000/api/libri/autore`. The response is a JSON array of three book objects, each with an id, titolo, and autori field.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
id	774			
titolo	"DIARIO DI ESTERINA "			
autori	" "			
id	775			
titolo	"IL COMPITO DI LATINO "			
autori	" "			
id	776			
titolo	"MAGISTRE DI ESTERINA DI UN "			
autori	" "			

http://localhost:9002/api/customer

GET http://localhost:9002/api/customer Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit

Body Cookies Headers (5) Test Results 200 OK 349 ms 300 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": 3456,
3    "nome": "Denis",
4    "cognome": "Maggi"
5  },
6  {
7    "id": 1234,
8    "nome": "Giada",
9    "cognome": "Russo"
10 },
11 {
12   "id": 2345,
13 }

```

La parte relativa ai database relazionali adesso è finita.

→ Per la parte del database non relazionale utilizzo Atlas di MongoDB per creare il database “Prestiti”. (soluzione 1)

- Creo la collection:

`db.createCollection("Prestiti")`

- Inserisco tre elementi all'interno:

`db.Prestiti.insert({orderId: "4444", customer: "Femia", libro: "Il signore degli anelli"});`

`db.Prestiti.insert({orderId: "5555", customer: "Russo", libro: "Piccole donne"});`

`db.Prestiti.insert({orderId: "6666", customer: "Maggio", libro: "Shining"});`

```

Atlas atlas-aedy5b-shard-0 [primary] myFirstDatabase> db.createCollection("Prestiti")
{ ok: 1 }
Atlas atlas-aedy5b-shard-0 [primary] myFirstDatabase> db.Prestiti.insert({orderId: "4444", customer: "Femia", libro: "Il signore degli anelli"});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
  acknowledged: true,
  insertedIds: { '_id': ObjectId("61ae25a2995d1f8e230b7c07") }
Atlas atlas-aedy5b-shard-0 [primary] myFirstDatabase> db.Prestiti.insert({orderId: "5555", customer: "Russo", libro: "Piccole donne"});
  acknowledged: true,
  insertedIds: { '_id': ObjectId("61ae2678995d1f8e230b7c08") }
Atlas atlas-aedy5b-shard-0 [primary] myFirstDatabase> db.Prestiti.insert({orderId: "6666", customer: "Maggio", libro: "Shining"});
  acknowledged: true,
  insertedIds: { '_id': ObjectId("61ae269d995d1f8e230b7c09") }
Atlas atlas-aedy5b-shard-0 [primary] myFirstDatabase>

```



```
_id: ObjectId("61ae25a2995d1f8e230b7c07")
orderId: "4444"
customer: "Femia"
libro: "Il signore degli anelli"
```

```
_id: ObjectId("61ae2678995d1f8e230b7c08")
orderId: "5555"
customer: "Russo"
libro: "Piccole donne"
```

```
_id: ObjectId("61ae269d995d1f8e230b7c09")
orderId: "6666"
customer: "Maggio"
libro: "Shining"
```

❑ PUSH DELL'IMMAGINE DEL MICROSERVIZIO LIBRI SU DOCKERHUB

→ compilo il dockerfile:

```
FROM maven:3.6.3-jdk-11-slim as builder
ARG MVN_ARGS
ADD ./src/ /app/src/
ADD ./pom.xml /app/pom.xml
WORKDIR /app
RUN mvn --batch-mode package ${MVN_ARGS}
```

```
FROM openjdk:11
WORKDIR /app
COPY --from=builder /app/target/*.jar app.jar
EXPOSE 8102
ENTRYPOINT ["java", "-jar", "app.jar"]
```

→ diamo il comando da terminale per **creare l'immagine**:

```
sudo docker build --build-arg MVN_ARGS=-DskipTests -t rossellafemia/libri:1.0-SNAPSHOT .
```

```

Dockerfile U x pom.xml M
Libri > Dockerfile > ...
1 FROM maven:3.6.3-jdk-11-slim as builder
2 ARG MVN_ARGS
3 ADD ./src/ /app/src/
4 ADD ./pom.xml /app/pom.xml
5 WORKDIR /app
6 RUN mvn --batch-mode package ${MVN_ARGS}
7
8 FROM openjdk:11
9 WORKDIR /app
10 COPY --from=builder /app/target/*.jar app.jar
11 EXPOSE 8102
12 ENTRYPOINT ["java", "-jar", "app.jar"]
13
PROBLEMS 8 OUTPUT TERMINAL DEBUG CONSOLE
---> 2db0ed407a98
Step 4/11 : ADD ./pom.xml /app/pom.xml
---> Using cache
---> 798cde5d4db0
Step 5/11 : WORKDIR /app
---> Using cache
---> 13b53bba44dc
Step 6/11 : RUN mvn --batch-mode package ${MVN_ARGS}
---> Using cache
---> 4cb19635df3f
Step 7/11 : FROM openjdk:11
---> 0719902862f3
Step 8/11 : WORKDIR /app
---> Using cache
---> d6f748c758f8
Step 9/11 : COPY --from=builder /app/target/*.jar app.jar
---> 37c1682134f9
Step 10/11 : EXPOSE 8102
---> Running in e75200efdb08
Removing intermediate container e75200efdb08
---> cf71cec10dca
Step 11/11 : ENTRYPOINT ["java", "-jar", "app.jar"]
---> Running in 2e7c2b478633
Removing intermediate container 2e7c2b478633
---> cb38e0ae9ca8
Successfully built cb38e0ae9ca8
Successfully tagged rossellafemia/libri:1.0-SNAPSHOT
rossella@rossella-VirtualBox:~/Documents/Microservizi/Libri$

```

→ push su **DockerHub** l'immagine del mio microservizio:

docker hub

[username: rossellafemia

password: Unitofaschifo!1]

\$ docker push rossellafemia/libri:1.0-SNAPSHOT

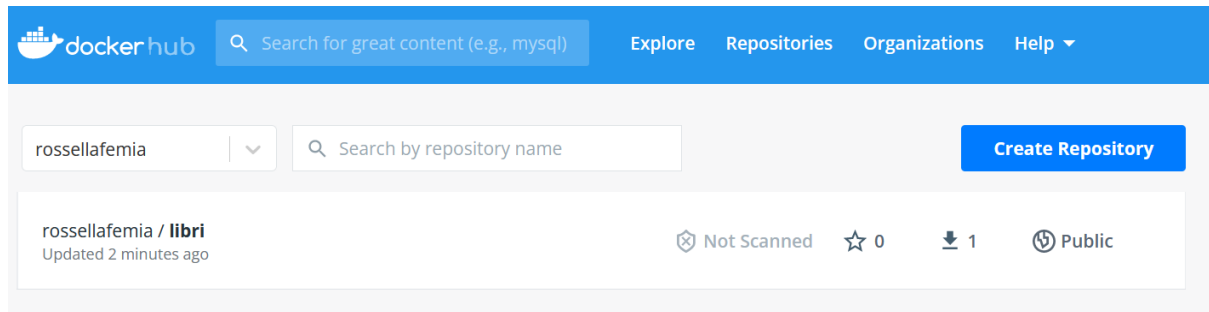
```

EXPLORER MICROSERVIZI
  > vscode
  > Customers
  > Libri
    > src
    > main
      > java/biblioteca
      > entities
      > integration
      > repos
      > services
      > LibriApplication.java
      > resources
      > test
      > target
    Dockerfile
    HELP.md
    mwnw
    mwnw.cmd
    pom.xml
    Prestiti
    > src
    > target
    > HELP.md
    > mwnw
    > mwnw.cmd
    > pom.xml
    > README.md
  > OUTLINE

Dockerfile x pom.xml M
Libri > Dockerfile > ...
1 FROM maven:3.6.3-jdk-11-slim as builder
2 ARG MVN_ARGS
3 ADD ./src/ /app/src/
4 ADD ./pom.xml /app/pom.xml
5 WORKDIR /app
6 RUN mvn --batch-mode package ${MVN_ARGS}
7
8 FROM openjdk:11
9 WORKDIR /app
10 COPY --from=builder /app/target/*.jar app.jar
11 EXPOSE 8102
12 ENTRYPOINT ["java", "-jar", "app.jar"]
13
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
rossella@rossella-VirtualBox:~/Documents/Microservizi/Libri$ cd Libri/
rossella@rossella-VirtualBox:~/Documents/Microservizi/Libri$ docker push rossellafemia/libri:1.0-SNAPSHOT
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/rossellafemia/l
rossella@rossella-VirtualBox:~/Documents/Microservizi/Libri$ sudo docker push rossellafemia/libri:1.0-SNAPSHOT
[sudo] password for rossella:
The push refers to repository [docker.io/rossellafemia/libri]
31d9ef90994c: Pushed
364aa091b872: Pushed
5c31f9330d99: Mounted from library/openjdk
927f9fcef4cf: Mounted from library/openjdk
a01f1846a0d2: Mounted from library/openjdk
3b4417cb4b0b: Mounted from library/openjdk
d3710d004cb3: Mounted from library/openjdk
91f73360bfff: Mounted from library/openjdk
c20bc32ed777: Mounted from library/openjdk
1.0-SNAPSHOT: digest: sha256:1c006c704b4a3fcd08dc2f6c35c68fd3d89f563ed063ab03a441432349743d54 size: 2214
rossella@rossella-VirtualBox:~/Documents/Microservizi/Libri$

```

Adesso l'immagine sarà presente sul mio **DockerHub**.



PUSH DELL'IMMAGINE DEL MICROSERVIZIO CUSTOMERS SU DOCKERHUB

→ compiliamo il dockerfile:

```
FROM maven:3.6.3-jdk-11-slim as builder
ARG MVN_ARGS
ADD ./src/ /app/src/
ADD ./pom.xml /app/pom.xml
WORKDIR /app
RUN mvn --batch-mode package ${MVN_ARGS}
```

```
FROM openjdk:11
WORKDIR /app
COPY --from=builder /app/target/*.jar app.jar
EXPOSE 8102
ENTRYPOINT ["java", "-jar", "app.jar"]
```

→ diamo il comando da terminale per creare l'immagine:

```
sudo docker build --build-arg MVN_ARGS=-DskipTests -t rossellafemia/customers:1.0-SNAPSHOT .
```

```
Dockerfile u x pom.xml M
Customers > Dockerfile > ...
1 FROM maven:3.6.3-jdk-11-slim as builder
2 ARG MVN_ARGS
3 ADD ./src/ /app/src/
4 ADD ./pom.xml /app/pom.xml
5 WORKDIR /app
6 RUN mvn --batch-mode package ${MVN_ARGS}
7
8 FROM openjdk:11
9 WORKDIR /app
10 COPY --from=builder /app/target/*.jar app.jar
11 EXPOSE 8102
12 ENTRYPOINT ["java", "-jar", "app.jar"]
13

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-lang3/3.7/commons-lang3-3.7.jar (500 kB at 398 kB/s)
[INFO] Replacing main artifact with repackaged archive
[INFO] BUILD SUCCESS
[INFO] Total time: 34.625 s
[INFO] Finished at: 2021-12-14T09:53:17Z
[INFO]
Removing intermediate container 1c2dce63f0f9
--> 7424d79904ee
Step 7/11 : FROM openjdk:11
--> 07199e2862f3
Step 8/11 : WORKDIR /app
--> Using cache
--> d6f748c758f8
Step 9/11 : COPY --from=builder /app/target/*.jar app.jar
--> f30cd9a9e733
Step 10/11 : EXPOSE 8102
--> Running in acd79988808e
Removing intermediate container acd79988808e
--> 7319ae37d180
Step 11/11 : ENTRYPOINT ["java", "-jar", "app.jar"]
--> Running in 42feb52c6921
Removing intermediate container 42feb52c6921
--> feld1ee66415
Successfully built feld1ee66415
Successfully tagged rossellafemia/customers:1.0-SNAPSHOT
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$
```

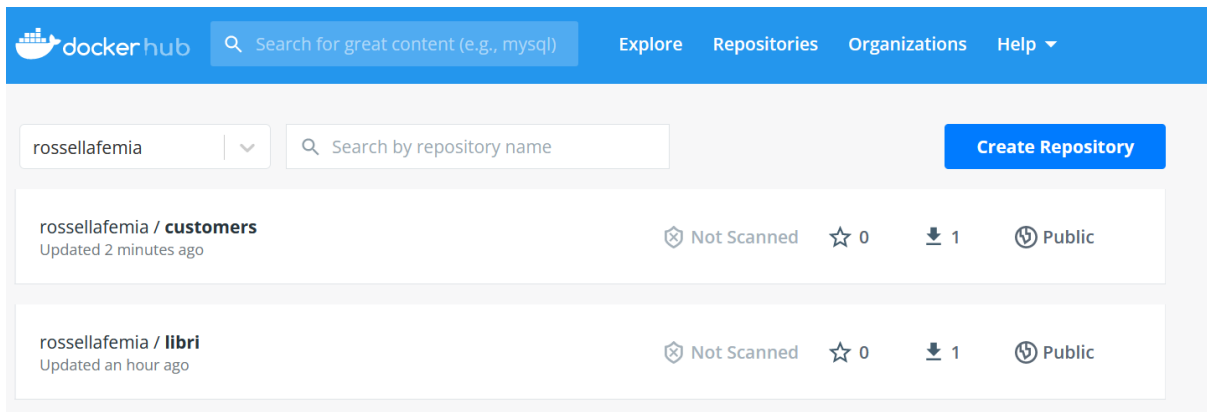
→ adesso bisogna pushare su **DockerHub** l'immagine del mio microservizio:

\$ docker push rossellafemia/customers:1.0-SNAPSHOT

```
Dockerfile u x pom.xml M
Customers > Dockerfile > ...
1 FROM maven:3.6.3-jdk-11-slim as builder
2 ARG MVN_ARGS
3 ADD ./src/ /app/src/
4 ADD ./pom.xml /app/pom.xml
5 WORKDIR /app
6 RUN mvn --batch-mode package ${MVN_ARGS}
7
8 FROM openjdk:11
9 WORKDIR /app
10 COPY --from=builder /app/target/*.jar app.jar
11 EXPOSE 8102
12 ENTRYPOINT ["java", "-jar", "app.jar"]
13

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
bash: /opt/ros/melodic/setup.bash: No such file or directory
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi$ cd Customers/
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ docker push rossellafemia/customers:1.0-SNAPSHOT
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/rossellafemia/c
ustomers/push?tag=1.0-SNAPSHOT": dial unix /var/run/docker.sock: connect: permission denied
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ sudo docker push rossellafemia/customers:1.0-SNAPSHOT
[sudo] password for rossella:
Sorry, try again.
[sudo] password for rossella:
The push refers to repository [docker.io/rossellafemia/customers]
4f5bacc4ec6b: Pushed
364aa091b872: Mounted from rossellafemia/libri
5c81f9330d99: Mounted from rossellafemia/libri
927f9fce4ef: Mounted from rossellafemia/libri
a81f1846a0d2: Mounted from rossellafemia/libri
3b441d7cb46b: Mounted from rossellafemia/libri
d3710de04cb3: Mounted from rossellafemia/libri
91f7336bbfff: Mounted from rossellafemia/libri
e2ebc39ebf77: Mounted from rossellafemia/libri
1.0-SNAPSHOT: digest: sha256:641154a448b95d9d5ba5842bba5aced0ab4056c3d3bcc6a6683b2d7fa390bbb size: 2214
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$
```

Adesso l'immagine sarà presente sul mio **DockerHub**.



❑ INSTALLARE I COMPONENTI DELL'INFRASTRUTTURA SU KUBERNETES

Installiamo:

- Minikube (3)
(guida: <https://phoenixnap.com/kb/install-minikube-on-ubuntu#ftoc-heading-1>)
- Kubectl (2)
- Helm (installa software su kubernetes) (1)

→ **LIBRI**

→ Start **minikube**:

```
sudo chmod 777 /var/run/docker.sock
```

```
minikube start --driver=docker --memory=4G --cpus=2
```

oppure

```
minikube start --driver=docker --memory=4G --cpus=2 --profile=minikube
```

→ Controllare la configurazione di **KUBECTL**

```
kubectl get nodes
```

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ sudo chmod 777 /var/run/docker.sock
[sudo] password for rossella:
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ minikube start --driver=docker --memory=4G --cpus=2 --profile=minikube
🐳 minikube v1.24.0 on Ubuntu 21.04
🔧 Using the docker driver based on existing profile
❗ Your cgroup does not allow setting memory.
   More information: https://docs.docker.com/engine/install/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities
❗ Your cgroup does not allow setting memory.
   More information: https://docs.docker.com/engine/install/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities
👉 Starting control plane node minikube in cluster minikube
📶 Pulling base image ...
🔄 Restarting existing docker container for "minikube" ...
🔧 Preparing Kubernetes v1.22.3 on Docker 20.10.8 ...
🔍 Verifying Kubernetes components...
   ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: default-storageclass, storage-provisioner
🏁 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane,master	18d	v1.22.3

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$
```

→ deployare infrastruttura **KAFKA**

- Creare il file kafka-values.yaml all'interno del progetto
- digitare il comando

```
helm install --values kafka-values.yaml kafka bitnami/kafka
```

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ helm install --values kafka-values.yaml kafka bitnami/kafka
NAME: kafka
LAST DEPLOYED: Wed Jan  5 15:46:45 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: kafka
CHART VERSION: 14.7.1
APP VERSION: 2.8.1
```

- verificare che KAFKA sia installato correttamente, digitare il comando:

```
kubectl get pods
```

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
kafka-0                             0/1     ImagePullBackOff    0           4m16s
kafka-zookeeper-0                   1/1     Running             0           4m16s
prestiti-mongodb-6f4bdbfd84-hkqll  1/1     Running             1 (11m ago) 18d
```

→ deployare l'infrastruttura su **ELASTICSEARCH**

- aggiungere il repository elastic Helm
`helm repo add elastic https://helm.elastic.co`
- installare Elasticsearch (prima creare il file elasticsearch-values.yaml secondo le indicazioni del docente)
`helm install --values elasticsearch-values.yaml elasticsearch elastic/elasticsearch`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ helm install --values elasticsearch-values.yaml elasticsearch elastic/elasticsearch
W0105 15:59:26.691115 21625 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
W0105 15:59:26.743981 21625 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
NAME: elasticsearch
LAST DEPLOYED: Wed Jan 5 15:59:26 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Watch all cluster members come up.
   $ kubectl get pods --namespace=default -l app=elasticsearch -w2. Test cluster health using Helm test.
   $ helm --namespace=default test elasticsearch
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$
```

- controllare se funziona tutto correttamente
`kubectl get pods --namespace=default -l app=elasticsearch-master`
- installare **Logstash** (creare il file logstash-values.yaml)
`helm install --values logstash-values.yaml logstash elastic/logstash`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ helm install --values logstash-values.yaml logstash elastic/logstash
W0105 16:06:44.108246 27663 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
W0105 16:06:44.141590 27663 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
NAME: logstash
LAST DEPLOYED: Wed Jan 5 16:06:43 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all cluster members come up.
   $ kubectl get pods --namespace=default -l app=logstash -w
```

- controllare se funziona correttamente
`kubectl get pods --namespace=default -l app=logstash-logstash`
- installa **Filebeat** come DaemonSet (prima creare il file filebeat-values.yaml)
`helm install --values filebeat-values.yaml filebeat elastic/filebeat`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ helm install --values filebeat-values.yaml filebeat elastic/filebeat
NAME: filebeat
LAST DEPLOYED: Wed Jan 5 16:15:39 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all containers come up.
   $ kubectl get pods --namespace=default -l app=filebeat -w
```

- controllare se funziona tutto correttamente
`kubectl get pods --namespace=default -l app=filebeat`
- installare **KIBANA** (prima creare il file kibana-values.yaml)
`helm install --values kibana-values.yaml kibana elastic/kibana`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ helm install --values kibana-values.yaml kibana elastic/kibana
NAME: kibana
LAST DEPLOYED: Wed Jan 5 16:20:43 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

- controllare che tutto funzioni correttamente
`kubectl get pods --namespace=default -l app=kibana`

→ CUSTOMERS

- KAFKA

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ helm install --values kafka-values1.yaml kafka1 bitnami/kafka
NAME: kafka1
LAST DEPLOYED: Wed Jan 5 16:45:43 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: kafka
CHART VERSION: 14.7.1
APP VERSION: 2.8.1
```

- ELASTICSEARCH

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ helm install --values elasticsearch-values.yaml elasticsearch1 elastic/elasticsearch
W0105 17:15:29.280821 117216 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
W0105 17:15:29.308088 117216 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
NAME: elasticsearch1
LAST DEPLOYED: Wed Jan 5 17:15:29 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Watch all cluster members come up.
   $ kubectl get pods --namespace=default -l app=elasticsearch1 -w2. Test cluster health using Helm test.
   $ helm --namespace=default test elasticsearch1
```

- LOGSTASH

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ helm install --values logstash-values.yaml logstash1 elastic/logstash
W0105 17:16:36.274449 118971 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
W0105 17:16:36.308354 118971 warnings.go:70] policy/v1beta1 PodDisruptionBudget is deprecated in v1.21+, unavailable in v1.25+; use policy/v1 PodDisruptionBudget
NAME: logstash1
LAST DEPLOYED: Wed Jan 5 17:16:36 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all cluster members come up.
   $ kubectl get pods --namespace=default -l app=logstash1 -w
```

- FILEBEAT

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ helm install --values filebeat-values.yaml filebeat1 elastic/filebeat
NAME: filebeat1
LAST DEPLOYED: Wed Jan 5 17:17:35 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch all containers come up.
   $ kubectl get pods --namespace=default -l app=filebeat1 -w
```

- KIBANA

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Customers$ helm install --values kibana-values.yaml kibana1 elastic/kibana
NAME: kibana1
LAST DEPLOYED: Wed Jan 5 17:18:34 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

❑ LOGGING AND TRACING

→ grazie a **Kubectl** sarà possibile monitorare i log dei pod di Kubernetes.

- innanzitutto bisogna visualizzare i pod con il comando:

`kubectl get pods`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
elasticsearch-0                     0/1     Pending   0           4d23h
elasticsearch-1                     0/1     Pending   0           4d23h
elasticsearch1-0                    0/1     Pending   0           4d22h
elasticsearch1-1                    0/1     Pending   0           4d22h
filebeat-pjsrb                     1/1     Running   2 (7m50s ago)  4d23h
filebeat1-tsjr9                    1/1     Running   3 (6m51s ago)  4d22h
kafka-0                             1/1     Running   3 (6m58s ago)  4d23h
kafka-zookeeper-0                  1/1     Running   1 (8m43s ago)  4d23h
kafka1-0                            1/1     Running   3 (7m16s ago)  4d22h
kafka1-zookeeper-0                 1/1     Running   1 (8m43s ago)  4d22h
kibana-5b7b9b757b-2dz2s            0/1     Running   1 (8m43s ago)  4d23h
kibana1-5748b8ddc5-m5hv5           0/1     Pending   0           4d22h
logstash-0                          1/1     Running   1 (8m43s ago)  4d23h
logstash1-0                        1/1     Running   1 (8m43s ago)  4d22h
prestiti-mongodb-6f4bdbfd84-hkqll  1/1     Running   2 (8m43s ago)  23d
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$
```

- poi sarà sufficiente inviare il comando
`kubectl logs pod-name`
specificando il nome del pod che interessa.
- dato che il pod di elasticsearch risulta in PENDING devo utilizzare il comando
`kubectl describe pod nome-pod`
per cercare di analizzare il problema
- problema del "pending" risolto.

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
elasticsearch-0                     0/1     Init:0/1   0           10s
elasticsearch-1                     0/1     Init:0/1   0           10s
elasticsearch1-0                    0/1     Pending   0           8d
elasticsearch1-1                    0/1     Pending   0           8d
filebeat-pjsrb                     1/1     Running   5 (47m ago)  8d
filebeat1-tsjr9                    1/1     Running   6 (47m ago)  8d
kafka-0                             1/1     Running   7 (47m ago)  8d
kafka-zookeeper-0                  1/1     Running   2 (50m ago)  8d
kafka1-0                            1/1     Running   7 (47m ago)  8d
kafka1-zookeeper-0                 1/1     Running   2 (50m ago)  8d
kibana-5b7b9b757b-2dz2s            0/1     Running   2 (50m ago)  8d
kibana1-5748b8ddc5-m5hv5           0/1     Pending   0           8d
logstash-0                          1/1     Running   2 (50m ago)  8d
logstash1-0                        1/1     Running   2 (50m ago)  8d
prestiti-mongodb-6f4bdbfd84-hkqll  1/1     Running   3 (50m ago)  27d
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$
```

- adesso per visualizzare i logs ridare il comando:
`kubectl logs pod-name`
`kubectl logs logstash-0` (ad esempio)

→ Visualizzare i log sull'interfaccia di **kibana**:

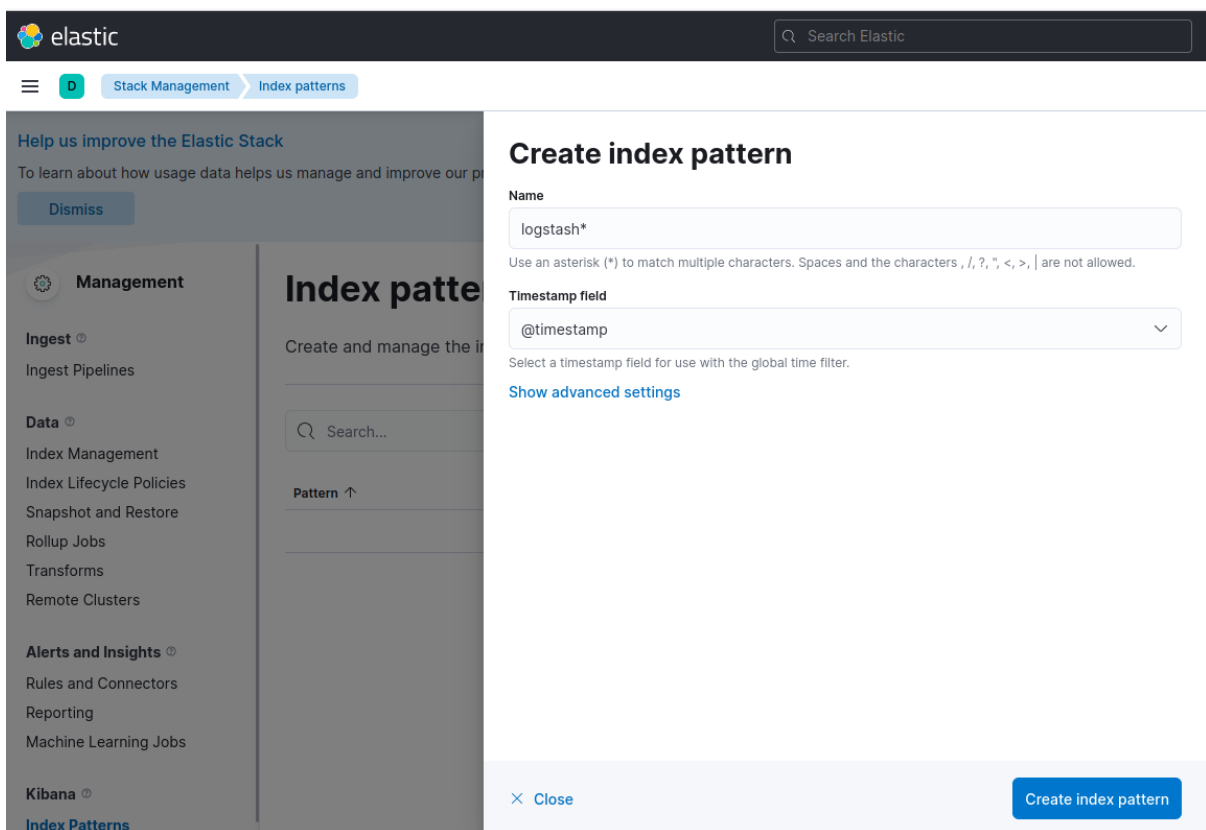
è necessario uscire fuori dal cluster! quindi dobbiamo fare un port-forward:

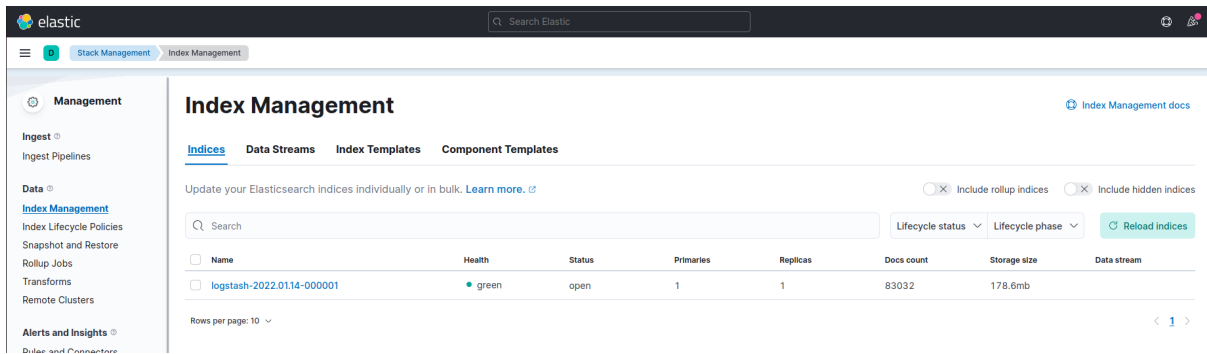
- utilizzare il comando
`kubectl port-forward svc/kibana 5601:5601`
- andare su
`localhost:5601`

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Libri$ kubectl port-forward svc/kibana 5601:5601
Forwarding from 127.0.0.1:5601 -> 5601
Forwarding from [::1]:5601 -> 5601
```

Sull'interfaccia di kibana creare un **index pattern**:

- stack management (nella barra laterale a sinistra)
- index patterns (da aggiungere "logstash*")

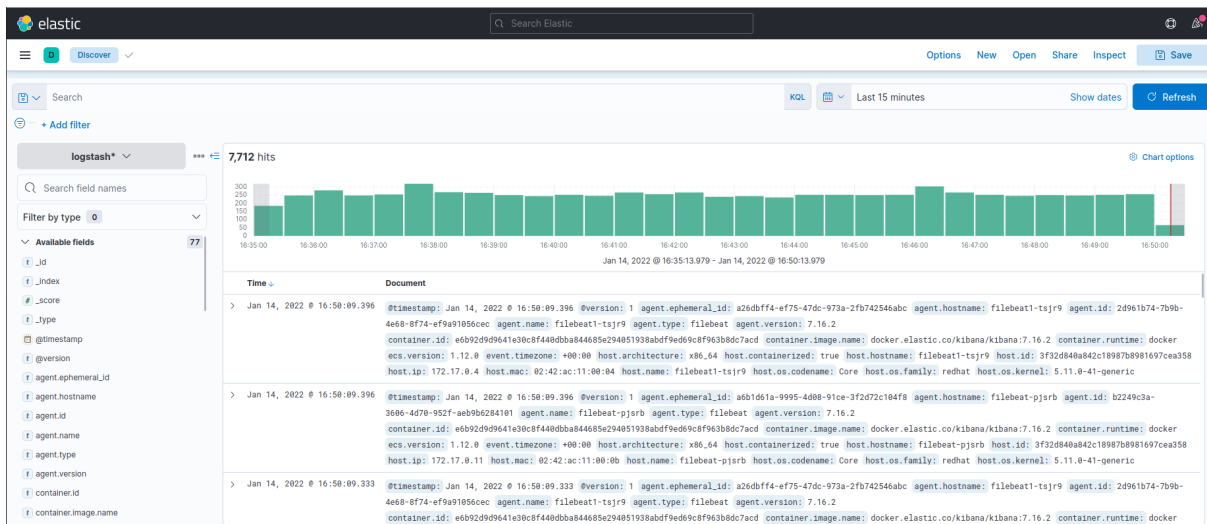




Andare su:

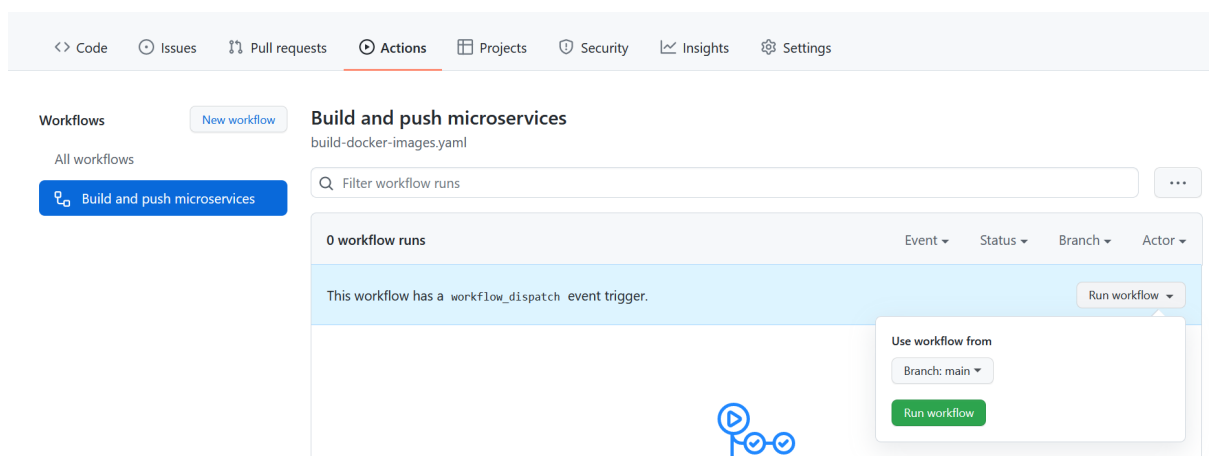
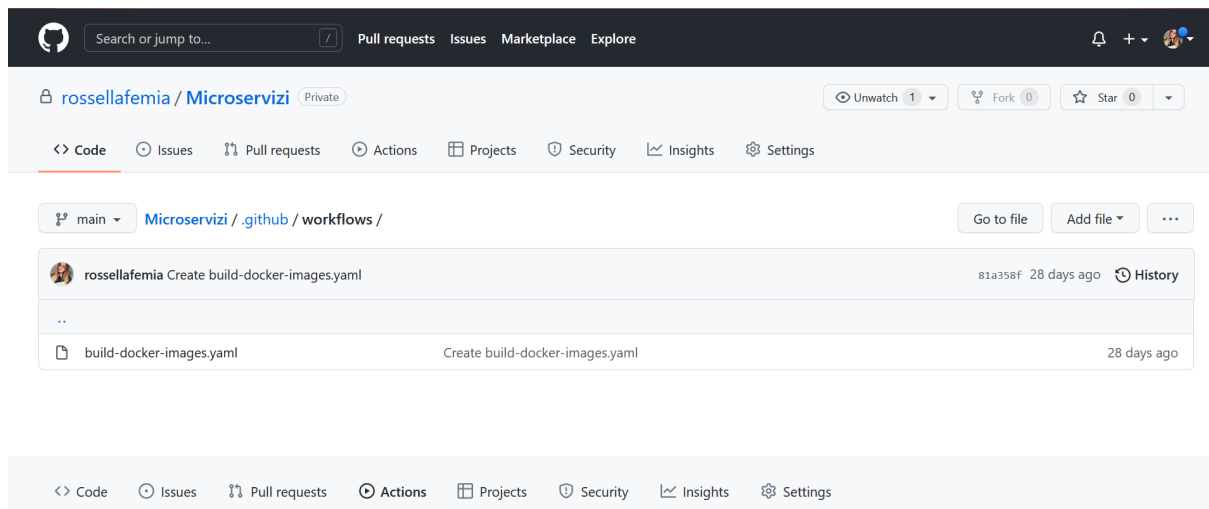
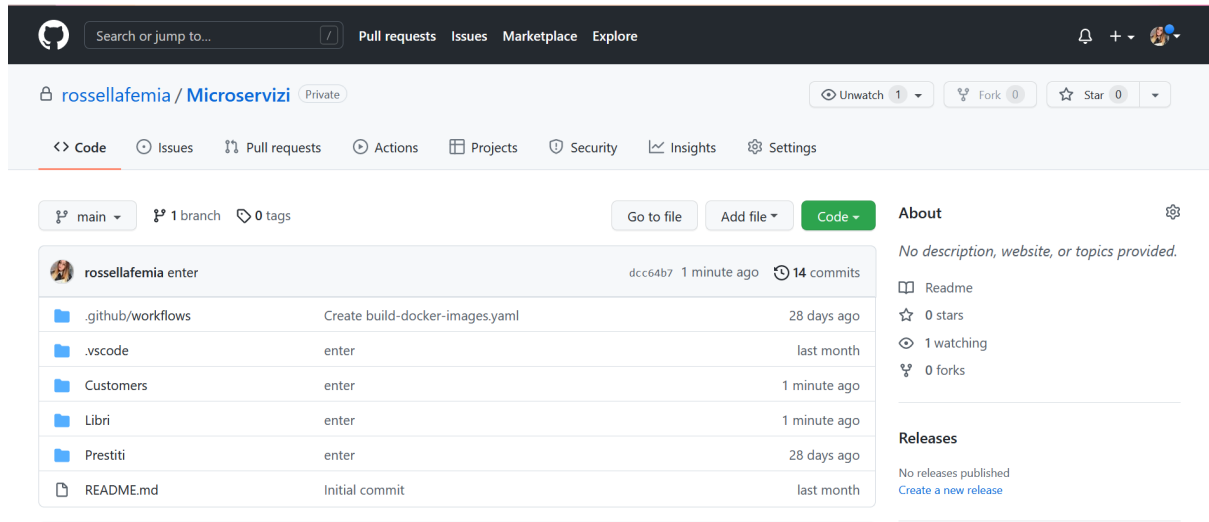
- Analytics
- Discover

da qui si possono visualizzare parte dei log



CONTINUOUS DELIVERY / CONTINUOUS INTEGRATION

Per affrontare questa parte è necessario creare un file `create-docker-images.yaml` all'interno del mio progetto github con il path `.github/workflows`. Fare poi il run del workspace appena creato.



Workflow run was successfully requested. ✕

🔒 rossellafemia / **Microservizi** Private

👁️ Unwatch 1 ▼👤 Fork 0🌟 Star 0 ▼

<> Code🕒 Issues🔗 Pull requests🎯 **Actions**📁 Projects🛡️ Security📈 Insights⚙️ Settings

Workflows

New workflow

All workflows

🔗 Build and push microservices

Build and push microservices
build-docker-images.yaml

⋮

0 workflow runs

Event ▼Status ▼Branch ▼Actor ▼

This workflow has a workflow_dispatch event trigger.

Run workflow ▼

✖ **Build and push microservices**

Build and push microservices #1: Manually run by rossellafemia

📅 6 minutes ago

🕒 1m 31s

⋮

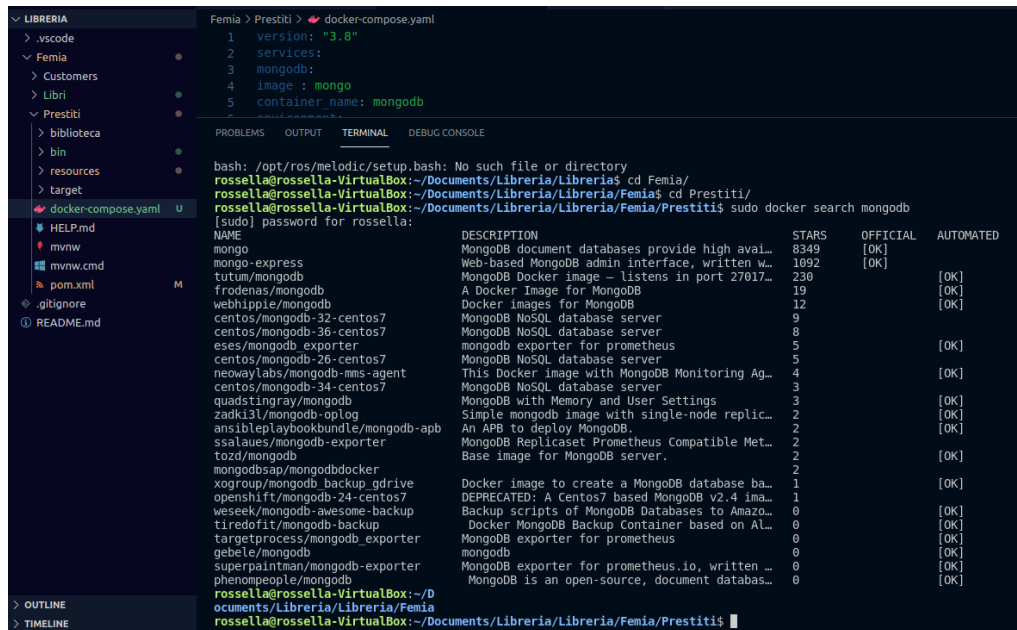
❑ UTILIZZO DI DATABASE NON-RELAZIONALI

soluzione 1: fallita ❌

→ collegare il database di MongoDB **Atlas** sulla mia applicazione SpringBoot: errori durante la compilazione.

soluzione 2: fallita ❌

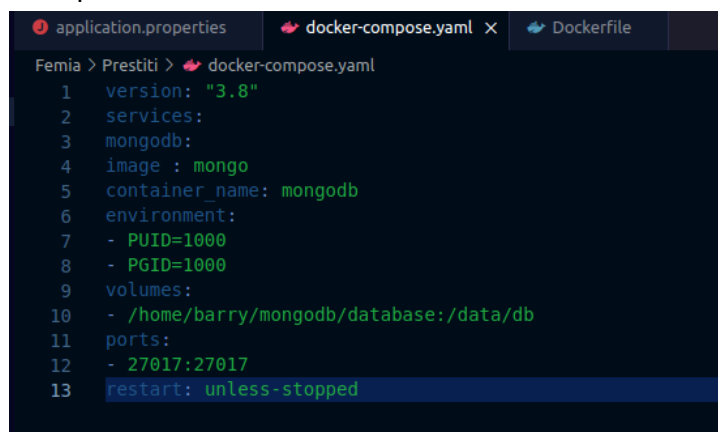
→ Istanziare un database tramite docker-compose su Visual Studio Code.
Inizializzare **MongoDB** con il comando:



```
Femia > Prestiti > docker-compose.yaml
1  version: "3.8"
2  services:
3  mongodb:
4    image : mongo
5    container_name: mongodb

bash: /opt/ros/melodic/setup.bash: No such file or directory
rossella@rossella-VirtualBox:~/Documents/Libreria/Libreria$ cd Femia/
rossella@rossella-VirtualBox:~/Documents/Libreria/Libreria/Femia$ cd Prestiti/
rossella@rossella-VirtualBox:~/Documents/Libreria/Libreria/Femia/Prestiti$ sudo docker search mongodb
[sudo] password for rossella:
NAME                DESCRIPTION                STARS     OFFICIAL   AUTOMATED
mongo               MongoDB document databases provide high avail... 8349      [OK]
mongo-express       Web-based MongoDB admin interface, written w... 1092      [OK]
tutum/mongodb       MongoDB Docker Image - listens in port 27017... 230
frodenas/mongodb    A Docker Image for MongoDB 19
webhippie/mongodb   Docker images for MongoDB 12
centos/mongodb-32-centos7 MongoDB NoSQL database server 9
centos/mongodb-36-centos7 MongoDB NoSQL database server 8
eses/mongodb_exporter MongoDB exporter for prometheus 5 [OK]
centos/mongodb-26-centos7 MongoDB NoSQL database server 5
newwaylabs/mongodb-mms-agent This Docker image with MongoDB Monitoring Ag... 4 [OK]
centos/mongodb-34-centos7 MongoDB NoSQL database server 3
quadstingray/mongodb MongoDB with Memory and User Settings 3 [OK]
zadki31/mongodb-oplog Simple mongodb image with single-node replic... 2 [OK]
ansibleplaybookbundle/mongodb-apb An APB to deploy MongoDB. 2 [OK]
ssalaues/mongodb-exporter MongoDB Replicaset Prometheus Compatible Met... 2
tozd/mongodb        Base image for MongoDB server. 2 [OK]
mongodbsap/mongodbdocker Docker image to create a MongoDB database ba... 2 [OK]
xogroup/mongodb-backup-gdrive DEPRECATED: A Centos7 based MongoDB v2.4 ima... 1
opnshift/mongodb-24-centos7 DEPRECATED: A Centos7 based MongoDB v2.4 ima... 1
weseek/mongodb-awesome-backup Backup scripts of MongoDB Databases to Amazo... 0 [OK]
tiredofit/mongodb-backup Docker MongoDB Backup Container based on AL... 0 [OK]
targetprocess/mongodb_exporter MongoDB exporter for prometheus 0 [OK]
gebele/mongodb      mongodb 0 [OK]
superpaintman/mongodb-exporter MongoDB exporter for prometheus.io, written ... 0 [OK]
phenompeople/mongodb MongoDB is an open-source, document databas... 0 [OK]
```

compilare il docker-compose:



```
Femia > Prestiti > docker-compose.yaml
1  version: "3.8"
2  services:
3  mongodb:
4    image : mongo
5    container_name: mongodb
6    environment:
7      - PUID=1000
8      - PGID=1000
9    volumes:
10     - /home/barry/mongodb/database:/data/db
11    ports:
12     - 27017:27017
13    restart: unless-stopped
```

soluzione 3: ✅

Utilizzare **Kubernetes** per istanziare un database, senza l'intralcio del docker-compose.

→ Installare:

- Minikube (3)

(guida: <https://phoenixnap.com/kb/install-minikube-on-ubuntu#ftoc-heading-1>)

- Kubectl (2)
- Helm (installa software su kubernetes) (1)

```
rossella@rossella-VirtualBox:~$ minikube version
minikube version: v1.24.0
commit: 76b94fb3c4e8ac5062daf70d60cf03ddcc0a741b
```

→ creare un file **mongodb-values.yaml**

auth:

```
enabled: true
rootPassword: root
username: root
password: root
database: prestiti_db
```

→ seguire i comandi del laboratorio 6 del github del docente:

Deploy MongoDB infrastructure

Install MongoDB standalone instance for Customer microservice

```
$ helm install --values mongodb-values.yaml customer-mongodb bitnami/mongodb
NAME: customer-mongodb
LAST DEPLOYED: Sun Jan 3 09:08:08 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
...
```

To check if MongoDB has been started correctly (it may take a while to get Running and Ready):

```
$ kubectl get pods | grep mongo
customer-mongodb-64df6b7549-4ms2r    1/1    Running    0    100s
```

- Start **minikube**:

```
sudo chmod 777 /var/run/docker.sock
```

```
minikube start --driver=docker --memory=4G --cpus=2
```

oppure

```
minikube start --driver=docker --memory=4G --cpus=2 --profile=minikube
```

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Prestiti$ minikube start --driver=docker --memory=4G --cpus=2 --profile=minikube
minikube v1.24.0 on Ubuntu 21.04
Using the docker driver based on user configuration
Your cgroup does not allow setting memory.
  More information: https://docs.docker.com/engine/install/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities
Starting control plane node minikube in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=4096MB) ...
Preparing Kubernetes v1.22.3 on Docker 20.10.8 ...
  Generating certificates and keys ...
  Booting up control plane ...
  Configuring RBAC rules ...
Verifying Kubernetes components...
  Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: default-storageclass, storage-provisioner
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Prestiti$
```

- proseguire con i comandi della guida:

helm install --values mongodb-values.yaml prestiti-mongodb bitnami/mongodb

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Prestiti$ helm install --values mongodb-values.yaml prestiti-mongodb bitnami/mongodb
NAME: prestiti-mongodb
LAST DEPLOYED: Fri Dec 17 16:55:38 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: mongodb
CHART VERSION: 10.30.6
APP VERSION: 4.4.10

** Please be patient while the chart is being deployed **

MongoDB&reg; can be accessed on the following DNS name(s) and ports from within your cluster:

    prestiti-mongodb.default.svc.cluster.local

To get the root password run:

    export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace default prestiti-mongodb -o jsonpath="{.data.mongodb-root-password}" | base64 --decode)

To get the password for "root" run:

    export MONGODB_PASSWORD=$(kubectl get secret --namespace default prestiti-mongodb -o jsonpath="{.data.mongodb-passwords}" | base64 --decode | awk -F',' '{print $1}')

To connect to your database, create a MongoDB&reg; client container:

    kubectl run --namespace default prestiti-mongodb-client --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" --image docker.io/bitnami/mongodb:4.4.10-debian-10-r44 --command -- bash

Then, run the following command:

    mongo admin --host "prestiti-mongodb" --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/prestiti-mongodb 27017:27017 &
    mongo --host 127.0.0.1 --authenticationDatabase admin -p $MONGODB_ROOT_PASSWORD
```

kubectl get pods | grep mongo

```
rossella@rossella-VirtualBox:~/Documents/Microservizi/Microservizi/Prestiti$ kubectl get pods | grep mongo
prestiti-mongodb-6f4bdf84-hkqll 1/1 Running 0 9m22s
```

→ Aggiungere una parte di build al pom.xml per far partire l'applicazione con **exec:exec**

```
58     <build>
59       <plugins>
60         <plugin>
61           <groupId>org.springframework.boot</groupId>
62           <artifactId>spring-boot-maven-plugin</artifactId>
63         </plugin>
64         <plugin>
65           <groupId>org.codehaus.mojo</groupId>
66           <artifactId>exec-maven-plugin</artifactId>
67           <version>3.0.0</version>
68           <executions>
69             <execution>
70               <goals>
71                 <goal>exec</goal>
72               </goals>
73             </execution>
74           </executions>
75           <configuration>
76             <executable>java</executable>
77             <arguments>
78               <!-- <argument></argument> -->
79               <!-- Classpath -->
80               <argument>-classpath</argument>
81               <classpath/>
82               <!-- Main class -->
83               <argument>biblioteca.PrestitiApplication</argument>
84             </arguments>
85           </configuration>
86         </plugin>
87       </plugins>
88     </build>
```

ERRORE! ❌ qui quando si prova a far eseguire l'applicazione si rompe.

□ GLOSSARIO

- **PATTERN A MICROSERVIZI**: Con un'architettura basata su microservizi, un'applicazione è realizzata da componenti indipendenti che eseguono ciascun processo applicativo come un servizio. I microservizi sono uno stile architetturale in cui il processo di sviluppo di un'applicazione avviene costruendola come un insieme di piccoli servizi, ciascuno dei quali funziona in maniera indipendente e comunica con gli altri tramite API (application programming interface).
- **HTTP REST**: Representational state transfer (REST) è uno stile architetturale per sistemi distribuiti. Il termine REST rappresenta un sistema di trasmissione di dati su HTTP senza ulteriori livelli.
- **SPRINGBOOT**: Framework per creare applicazioni basate sul framework Java Spring che sono subito pronte per ambienti di produzione e sono maggiormente utilizzate per creare microservizi.
- **MAVEN**: è uno strumento di compilazione e gestione dei progetti generalmente utilizzato nei framework creati in Java.
- **DATABASE RELAZIONALE**: I dati vengono salvati in delle tabelle seguendo un preciso schema che ogni dato deve seguire per poter essere salvato.
- **DATABASE NON RELAZIONALE**: Anche dati flessibili e scalabili e che vengono utilizzati in modo massiccio dai social, dai cloud computing e dalle applicazioni per computer mobili. I dati vengono salvati in documenti e non si deve seguire per forza uno schema ben preciso.
- **IMMAGINE DOCKER**: La virtualizzazione a container si basa fondamentalmente sulle immagini, ovvero i file reperibili sul Docker Hub e utilizzate per la creazione e l'inizializzazione di una applicazione in un nuovo container Docker. Ogni immagine è definita da un Dockerfile, un file di configurazione che contiene tutti i comandi che un utente deve eseguire per assemblare l'immagine. Un'immagine Docker contiene tutto il necessario per eseguire il software: il codice, un runtime (ad esempio, Java Virtual Machine), driver, strumenti, script, librerie, implementazioni e altro. Un **container** Docker è un'istanza in esecuzione di un'immagine Docker.
- **DOCKERHUB**: Docker Hub è un registro Docker ospitato gestito da Docker. Docker Hub ha oltre 100.000 immagini di container da fornitori di software, progetti open source e dalla community.
- **DOCKER COMPOSE**: Docker Compose è uno strumento sviluppato per facilitare la definizione e la condivisione di applicazioni multi-container. Con Compose è possibile creare un file YAML per definire i servizi e, con un singolo comando, è possibile ruotare tutto o eliminare tutto.

- **KUBERNETES K8S**: Piattaforma open source che automatizza le operazioni dei container Linux. Consente di eliminare molti dei processi manuali coinvolti nel deployment e nella scalabilità di applicazioni containerizzate.
 - **POD K8S**: Un pod Kubernetes è una raccolta di uno o più container Linux, nonché la più piccola unità di un'applicazione Kubernetes. Ciascun pod può includere più container ad alto accoppiamento (scenario di utilizzo avanzato) o un singolo container (scenario di utilizzo più comune).
 - **MINIKUBE**: Banco di prova per tutte le funzioni Kubernetes. Entrambi gli strumenti sono open source con la licenza Apache 2.0. Consente di valutare, attivare, disattivare e supportare effettivamente cluster
 - **HELM**: Strumento di creazione pacchetti open source che consente di installare e gestire il ciclo di vita delle applicazioni Kubernetes. Viene usato per gestire i grafici Kubernetes, ovvero pacchetti di risorse Kubernetes preconfigurate.
 - **KAFKA**: Piattaforma per il data streaming distribuita che permette di pubblicare, sottoscrivere, archiviare ed elaborare flussi di record in tempo reale.
 - **KIBANA**: Strumento di visualizzazione ed esplorazione dei dati utilizzato per analisi dei dati dei registri e serie temporali, monitoraggio delle applicazioni e casi d'uso di intelligenza operativa.
 - **ELASTICSEARCH**: The Elastic Stack (aka ELK) è una soluzione per la ricerca, la gestione dei log e l'analisi dei dati. **Elasticsearch**, **Logstash** e **Kibana**, si combinano per fornire una piattaforma unica per l'archiviazione, il recupero, l'ordinamento e l'analisi dei dati.
 - **LOGSTASH**: Pipeline di elaborazione dati lato server leggera e open source che permette di raccogliere dati da una varietà di fonti, trasformarli e inviarli alla destinazione desiderata. Viene spesso utilizzato come pipeline di dati per Elasticsearch, un motore di ricerca e analisi dei dati open source.
 - **FILEBEAT**: Data-injection dello Stack Elastic che permette la raccolta di dati e l'invio verso lo Stack (direttamente a Elasticsearch o tramite Logstash).
-

Run microservice

Prestiti: \$ mvn exec:exec

Libri e Customers:

\$ mvn package

[\$ mvn -DskipTests=true package]