



**Universidad Central del Ecuador**  
Facultad de Ingeniería y Ciencias Aplicadas  
Ingeniería en Sistemas de Información  
Programación Distribuida

**Tarea: 1**

**APELLIDOS Y NOMBRES:** Quishpe Toscano Juan Sebastián

**CARRERA:** Ingeniería en Sistemas de Información

**NIVEL:** 8vo Semestre

**FECHA:** 31-01-2023

**Link del Repositorio:** <https://github.com/SeckQ/DbrProDist1.git>

- **App-Books**

Este proyecto se basó en el que realizamos en clase, comenzamos agregando las dependencias para utilizar DbClient en lugar de JDBI. Lo demás se mantuvo igual a como lo teníamos en clase.

```
dependencies {  
    implementation platform("io.helidon:helidon-dependencies:${project.helidonversion}")  
    implementation 'io.helidon.microprofile.server:helidon-microprofile-server'  
    implementation 'org.glassfish.jersey.media:jersey-media-json-binding'  
  
    compileOnly 'org.projectlombok:lombok:1.18.24'  
    annotationProcessor 'org.projectlombok:lombok:1.18.24'  
  
    testCompileOnly 'org.projectlombok:lombok:1.18.24'  
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.24'  
    implementation 'io.helidon.dbclient:helidon-dbclient'  
    implementation 'io.helidon.dbclient:helidon-dbclient-jdbc'  
  
    implementation 'com.zaxxer:HikariCP:5.0.1'  
    implementation 'org.postgresql:postgresql:42.5.1'  
    implementation group: 'org.flywaydb', name: 'flyway-core', version: '9.12.0'  
    runtimeOnly 'org.jboss:jandex'
```

Lo siguiente que se realizó es utilizar el archivo de configuración "application.yaml" en lugar del archivo de configuración de microprofile. En el archivo de aplicación agregamos la configuración de la base de datos y le indicamos un puerto donde se ejecute la aplicación.



# Universidad Central del Ecuador

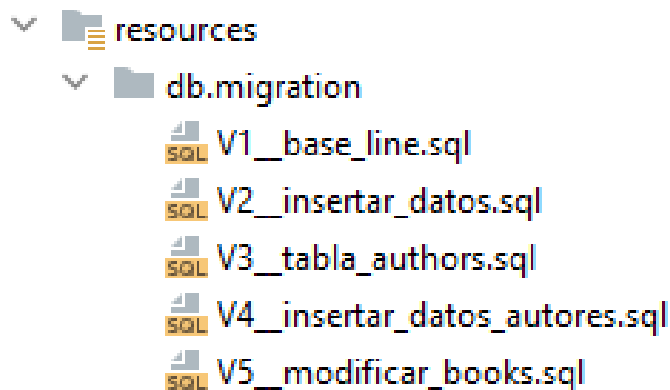
## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
application.yaml | BookRest.java | BookService.java | BookServiceImp
db:
  source: jdbc
  connection:
    url: "jdbc:postgresql://localhost:5432/distribuida"
    username: "postgres"
    password: "postgres"
    poolName: hikariPool
  server:
    port: 6060
```

Otro punto a comentar es la parte de la migración de la base datos, ya teníamos la baseline y otro archivo para insertar datos a la tabla books. Ahora se crearon tres archivos nuevos donde se crea la tabla authors, se insertan datos y se modifica la tabla books. Esto lo hacemos con el mismo método que realizamos en clase de flyway.





# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
@ApplicationScoped
public class AppEventos {

    1 usage
    @Inject
    DataSource dataSource;

    no usages  SeckQ
    public void init(@Observes @Initialized(ApplicationScoped.class) Object event){
        System.out.println("*****MIGRANDO");
        var flyway = Flyway.configure()
            .dataSource(dataSource)
            .load();
        flyway.migrate();
    }
}
```

Para la creación de un objeto de DbClient se usó la clase Config de Helidon para recuperar la configuración del archivo "application.yaml", usamos esa configuración recuperada para la creación del objeto DbClient y para el pool de conexiones.

```
no usages  SeckQ
@Produces
@ApplicationScoped
public DataSource dataSource() {
    Config config = Config.create();
    HikariDataSource ds = new HikariDataSource();
    ds.setDriverClassName("org.postgresql.Driver");
    ds.setJdbcUrl(config.get("db.connection.url").asString().get());
    ds.setUsername(config.get("db.connection.username").asString().get());
    ds.setPassword(config.get("db.connection.password").asString().get());
    return ds;
}
```



**Universidad Central del Ecuador**  
Facultad de Ingeniería y Ciencias Aplicadas  
Ingeniería en Sistemas de Información  
Programación Distribuida

```
no usages  SeckQ
@Produces
@ApplicationScoped
public DbClient dbClient(){
    Config config=Config.create();
    return DbClient.builder()
        .config(config.get("db"))
        .build();
}
```

Finalmente se creó el archivo Dockerfile para esta aplicación

```
>> FROM eclipse-temurin:17.0.5_8-jre-alpine
RUN mkdir /app
WORKDIR /app
COPY build/libs/app-web.jar app.jar
COPY build/libs/libs ./libs
CMD ["java", "-jar", "app.jar"]
```

- **App-Authors:**

Las dependencias que utilizamos para esta aplicación son las de Rest de Resteasy y Resteasy Jackson, aparte de la dependencia para la persistencia que es hibernate panache.

```
dependencies {
    implementation enforcedPlatform("${quarkusPlatformGroupId}:${quarkusPlatformArtifactId}:${quarkusPlatformVersion}")
    implementation 'io.quarkus:quarkus-resteasy'
    implementation 'io.quarkus:quarkus-arc'
    implementation 'io.quarkus:quarkus-resteasy-jackson'
    implementation 'io.quarkus:quarkus-hibernate-orm-rest-data-panache'
    //implementation group: 'com.zaxxer', name: 'HikariCP', version: '5.0.1'
    //implementation group: 'org.jdbi', name: 'jdbi3-core', version: '3.36.0'
    //implementation group: 'org.jdbi', name: 'jdbi3-sqlobject', version: '3.36.0'
    implementation group: 'org.postgresql', name: 'postgresql', version: '42.1.4'
    implementation group: 'io.quarkus', name: 'quarkus-jdbc-postgresql', version: '2.16.0.Final'
    implementation 'io.quarkus:quarkus-flyway'
}
```



# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

Esta aplicación fue realizada con Quarkus, por eso para la configuración de la db se hizo sobre el archivo "application.properties", se indica el tipo, usuario, contraseña, la url, nombre de la aplicación y el puerto donde se ejecutará por defecto.

```
properties
AuthorRepository.java x application.properties x AuthorRest.java x
ibuid 1 #quarkus.extensions.agroal.enabled=true
2 #Configuracion de la base de datos
3 quarkus.datasource.db-kind = postgresql
4 quarkus.datasource.username = postgres
5 quarkus.datasource.password = postgres
6 quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/distribuida
7 quarkus.hibernate-orm.database.generation=none
8 quarkus.http.port=9090
9 quarkus.application.name="app-authors"
10
```

Se definió la clase Author como una entidad para usar Panache y se creó una clase que implemente los métodos de PanacheRepository

```
16 usages SeckQ
@Entity
@Table(name = "authors")
@Data
public class Author{
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String first_name;
    no usages
    private String last_name;
}
```



# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
package com.distribuida.db;  
  
import io.quarkus.hibernate.orm.panache.PanacheRepository;  
  
import javax.enterprise.context.ApplicationScoped;  
  
@ApplicationScoped  
public class AuthorRepository implements PanacheRepository<Author> {  
}
```

Por último se crea el archivo Dockerfile pero en este caso como está realizado con Quarkus el archivo jar a copiar es el que se llama quarkus-run.jar.

```
FROM eclipse-temurin:17.0.5_8-jre-alpine  
RUN mkdir /quarkus-app  
WORKDIR /quarkus-app  
COPY build/quarkus-app .  
CMD ["java", "-jar", "quarkus-run.jar"]
```

- **App-Web:**

Para la aplicación web se agregaron las dependencias de spark y se utilizaron las tareas que ya habíamos realizado en clase como copiar las librerías, generar el jar, etc.



# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
dependencies {
    implementation 'com.sparkjava:spark-core:2.9.3'
    implementation 'org.jboss.weld.se:weld-se-core:5.0.0.SP1'
    implementation 'org.slf4j:slf4j-simple:1.7.21'
    implementation 'com.sparkjava:spark-template-thymeleaf:2.7.1'
    implementation 'org.thymeleaf:thymeleaf:3.0.15.RELEASE'
    implementation group: 'org.jboss.resteasy', name: 'resteasy-client', version: '6.1.0.Final'
    implementation 'org.jboss.resteasy:resteasy-jackson2-provider:6.2.2.Final'
}

sourceSets {
    main {
        output.resourcesDir = file("${buildDir}/classes/java/main")
    }
}

task copyLibs(type: Copy) {
    from configurations.runtimeClasspath
    into 'build/libs/libs'
}

jar {
    archiveFileName = "${project.name}.jar"
    manifest {
        attributes ('Main-Class': "${project.mainClass}",
            'Class-Path': configurations.runtimeClasspath.files.collect { File it -> "libs/${it.name}" }.join(' '))
    }
}
```

Para la configuración del cliente usando spark se creó un objeto tipo cliente y establecemos como objetivo la url base y definimos las rutas que va a usar la aplicación web

```
no usages  🔍 SeckQ
@ApplicationScoped
public class WebConfig {

    1 usage
    private final String BASE_URL = "http://traefik";

    no usages  🔍 SeckQ
    @ApplicationScoped
    @Produces
    public WebTarget webTarget(){
        Client client=ClientBuilder.newClient();
        return client.target(BASE_URL);
    }
}
```



# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
1 usage  ⚡ SeckQ
public void routes(){
    get( path: "/",(request, response) → authService.getAuthors(request,response));
    get( path: "/addAuthor",(request, response) → authService.showAddAuthor(request,response));
    post( path: "/addAuthor",(request, response) → authService.addAuthor(request,response));
    get( path: "/updateAuthor/:id",(request, response) → authService.showUpdateAuthor(request,response));
    get( path: "/deleteAuthor/:id",(request, response) → authService.deleteAuthor(request,response));
    get( path: "/books",(request, response) → bookService.getBooks(request,response));
    get( path: "/addBook",(request, response) → bookService.showAddBook(request,response));
    post( path: "/addBook",(request, response) → bookService.addBook(request,response));
    get( path: "/updateBook/:id",(request, response) → bookService.showUpdateBook(request,response));
    get( path: "/deleteBook/:id",(request, response) → bookService.deleteBook(request,response));
}
```

Finalmente, para el Dockerfile, hicimos un proceso similar a la aplicación de books

```
> FROM eclipse-temurin:17.0.5_8-jre-alpine
RUN mkdir /app
WORKDIR /app
COPY build/libs/app-web.jar app.jar
COPY build/libs/libs ./libs
CMD ["java", "-jar", "app.jar"]
```

- **PostgreSQL**

Para el despliegue de Postgres en el archivo Docker-compose se utilizó lo que hicimos en el proyecto en clase, se definió los puertos y se agregó el healthcheck para comprobar que el servicio esté corriendo y listo para utilizarse.

```
Universidad > ProgramaciónDistribuida > DeberFinal > docker-compose.yml
version: '3'
services:
  postgres-sql:
    image: postgres:alpine
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: distribuida
    ports:
      - 5432:5432
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 10s
      retries: 10
```





# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

- **Traefik**

Para Traefik lo primero es importar la imagen del Docker Hub, le indicamos que se configure como insegura y que el proveedor de configuración será el mismo Docker. Se exponen los puertos 80 y 8081, para el cliente de administración y los servicios respectivamente. Por último, le indicamos que se monte el volumen de la dirección indicada para que acceda la información de los contenedores.

```
traefik:
  image: traefik:v2.9
  command: --api.insecure=true --providers.docker
  ports:
    - 80:80
    - 8081:8080
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

The screenshot shows the Traefik dashboard interface. At the top, there's a navigation bar with 'traefik 2.9.6' and links to 'Dashboard', 'HTTP', 'TCP', 'UDP', and 'Plugins'. Below this, there's a search bar and tabs for 'HTTP Routers' (7), 'HTTP Services' (8), and 'HTTP Middlewares' (4). The main content area displays a table of services with columns: Status, TLS, Rule, Entrypoints, Name, Service, and Provider. The table lists several services, including 'api@internal', 'app-authors@docker', 'app-books@docker', 'app-web-deber1final@docker', 'dashboard@internal', 'postgres-sql-deber1final@docker', and 'traefik-deber1final@docker'. Each row has a green status icon, a 'true' TLS value, a specific rule, an 'http' endpoint, and a corresponding service name and provider.

Status	TLS	Rule	Entrypoints	Name	Service	Provider
●	true	PathPrefix(/api)	traefik	api@internal	api@internal	🔗
●	true	PathPrefix(/app-authors)	http	app-authors@docker	app-authors-deber1final	🔗
●	true	PathPrefix(/app-books)	http	app-books@docker	app-books-deber1final	🔗
●	true	Host(/app-web-deber1final)	http	app-web-deber1final@docker	app-web-deber1final	🔗
●	true	PathPrefix(/)	traefik	dashboard@internal	dashboard@internal	🔗
●	true	Host(/postgres-sql-deber1final)	http	postgres-sql-deber1final@docker	postgres-sql-deber1final	🔗
●	true	Host(/traefik-deber1final)	http	traefik-deber1final@docker	traefik-deber1final	🔗

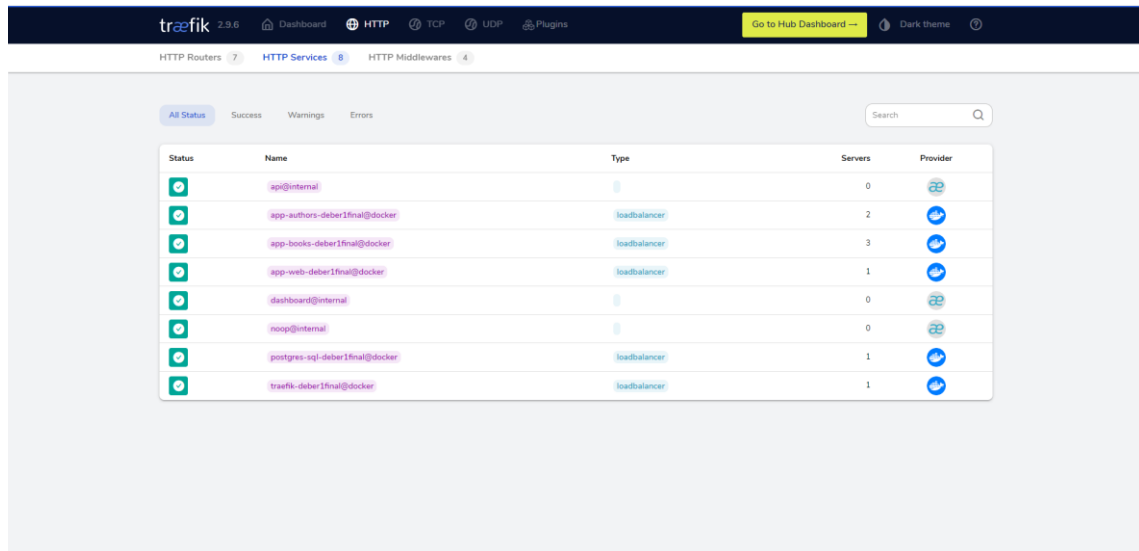


# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

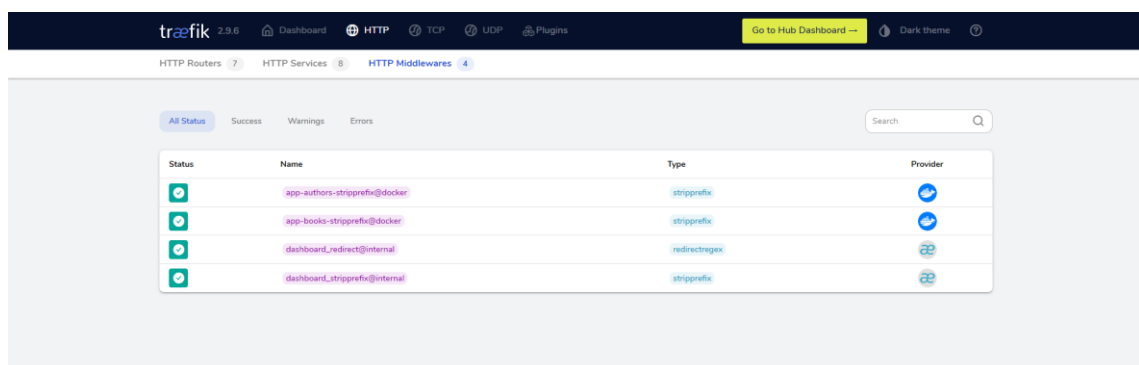


traefik 2.9.6 Dashboard HTTP TCP UDP Plugins Go to Hub Dashboard Dark theme

HTTP Routers 7 HTTP Services 8 HTTP Middlewares 4

All Status Success Warnings Errors Search

Status	Name	Type	Servers	Provider
🟢	api@internal		0	🔗
🟢	app-authors-deber1final@docker	loadbalancer	2	🔗
🟢	app-books-deber1final@docker	loadbalancer	3	🔗
🟢	app-web-deber1final@docker	loadbalancer	1	🔗
🟢	dashboard@internal		0	🔗
🟢	noop@internal		0	🔗
🟢	postgres-sql-deber1final@docker	loadbalancer	1	🔗
🟢	traefik-deber1final@docker	loadbalancer	1	🔗



traefik 2.9.6 Dashboard HTTP TCP UDP Plugins Go to Hub Dashboard Dark theme

HTTP Routers 7 HTTP Services 8 HTTP Middlewares 4

All Status Success Warnings Errors Search

Status	Name	Type	Provider
🟢	app-authors-striprefix@docker	striprefix	🔗
🟢	app-books-striprefix@docker	striprefix	🔗
🟢	dashboard_redirect@internal	redirectregex	🔗
🟢	dashboard_striprefix@internal	striprefix	🔗

- **Despliegue de las aplicaciones:**

Para todas las aplicaciones se realizó un despliegue similar. Se indicó el puerto donde se van a ejecutar, el número de copias de la misma imagen, le indicamos la configuración de ambiente y le agregamos la condición de que dependen de la ejecución de la base de datos y del proxy, si estos no están ejecutándose las aplicaciones no van a levantarse. Lo que también se agregó la regla para el enrutador de Traefik, se encargará de redirigir las solicitudes que cumplan con el prefijo en este caso “/app-books”, la segunda etiqueta es el middleware que se encarga de quitar el prefijo indicado anteriormente y la última etiqueta indica que el middleware se aplicará al enrutador “app-books”.



# Universidad Central del Ecuador

## Facultad de Ingeniería y Ciencias Aplicadas

### Ingeniería en Sistemas de Información

#### Programación Distribuida

```
app-books:
  image: seckq/app-books:2.0
  ports:
    - 6060
  deploy:
    replicas: 3
  environment:
    DB_CONNECTION_URL: jdbc:postgresql://postgres-sql/distribuida
    DB_CONNECTION_USERNAME: postgres
    DB_CONNECTION_PASSWORD: postgres
  depends_on:
    postgres-sql:
      condition: service_healthy
    traefik:
      condition: service_started
  labels:
    - "traefik.http.routers.app-books.rule=PathPrefix(`/app-books`)"
    - "traefik.http.middlewares.app-books-stripprefix.stripprefix.prefixes=/app-books"
    - "traefik.http.routers.app-books.middlewares=app-books-stripprefix"
```