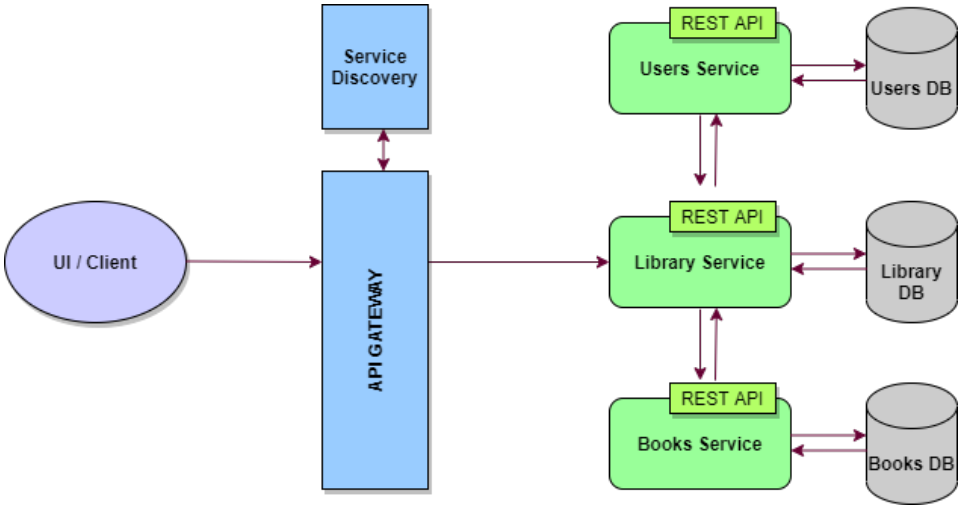
RD Java - Microservices project workflow

This page provides the details of the microservices project workflow that will be built during the microservices-basics learning.

High level Architecture of services interaction



Books Service endpoints:

BookEntity : id, name, publisher, author

Endpoint	HTTP Method	Description
/books	GET	List of all books
/books/{book_id}	GET	Get book by id
/books	POST	Add a book
/books/{book_id}	DELETE	Delete a book
/books/{book_id}	PUT	Update a book

Users Service endpoints:

UserEntity :username, email, name

Endpoint	HTTP Method	Description
/users	GET	List of all users
/users/{username}	GET	Get user by username
/users	POST	Add a user
/users/{username}	DELETE	Delete a user
/users/{username}	PUT	Update a user

Library Service endpoints:

LibraryEntity : id, username, bookId

Endpoint	HTTP Method	Description	Comments
/library/books	GET	List of all books	Calls book service **GET /books** URI
/library/books/{bookId}	GET	Get a book's details	Calls book service **GET /books/{bookId}** URI
/library/books/{bookId}	POST	Add a new book	Calls book service **POST /books/{bookId}** URI
/library/books/{bookId}	PUT	Update books details	Calls book service **PUT /books/{book_id}** URI
/library/books/{bookId}	DELETE	Delete a book	Delete book association with user in library table and calls book service **DELETE /books/{bookId}** URI
/library/users	GET	List of all users	Calls user service **GET /users** URI
/library/users/{username}	GET	View user profile with all issued books	calls user service **GET /users/{username},** fetches book ids on user_id from library table and calls **GET /books/{book_id}** for all books. Displays consolidated data
/library/users/{username}	POST	Add a user	User Registration, calls user service **POST /users/{username}**
/library/users/{username}	DELETE	Release all books for that user_id in library table and delete user.	Calls **DELETE users/{username}** URI
/library/users/{username}	PUT	Update user's account details	Calls **PUT users/{username}** URI
/library/users/{username}/books/{bookId}	POST	Issue a book to a user	Updates library table with book-user association.
/library/users/{username}/books/{bookId}	DELETE	Release the book for the user_id	Delete book-user association in library table.

Short notes:

API Gateway:

It is a single entry point into the system, used to handle requests by routing them to the appropriate backend service or by invoking multiple backend services and aggregating the results.

Netflix open sourced such an edge service (**ZUUL**), and now with Spring Cloud we can enable it with one @EnableZuulProxyannotation.

Service Discovery:

It allows automatic detection of network locations for service instances, which could have dynamically assigned addresses because of auto-scaling, failures, and upgrades. The key part of service discovery is the registry. We will use Netflix **Eureka** for this project. Eureka is a good example of the client-side discovery pattern, when the client is responsible for determining the locations of available service instances (using a registry server) and load balancing requests across them.

With Spring Boot, we can easily build Eureka Registry with a spring-cloud-starter-eureka-server dependency, @EnableEurekaServer annotation, and simple configuration properties. Client support is enabled with @EnableDiscoveryClient annotation and bootstrap.yml with application name

Feign:

Feign is a declarative HTTP client, which seamlessly integrates with **Ribbon** and **Hystrix**. Actually, with one spring-cloud-starter-feign dependency and @EnableFeignClients annotation you have a full suite of a load balancer, circuit breaker, and HTTP client with a sensible ready-to-go default configuration.

Log Analysis:

Centralized logging can be very useful when attempting to identify problems in a distributed environment. **Elasticsearch, Logstash, and Kibana (ELK)** stack lets you search and analyze your logs, utilization and network activity data with ease

Ribbon:

Ribbon is a client side load balancer which gives you a lot of control over the behavior of HTTP and TCP clients. Compared to a traditional load balancer, there is no need of an additional hop for every over-the-wire invocation — you can contact the desired service directly.

Out of the box, it natively integrates with Spring Cloud and Service Discovery. Eureka Client provides a dynamic list of available servers so Ribbon could balance between them.

Hystrix:

Hystrix is the implementation of a Circuit Breaker pattern, which gives a control over latency and failure from dependencies accessed over the network. The main idea is to stop cascading failures in a distributed environment with a large number of microservices. That helps to fail fast and recover as soon as possible — important aspects of fault-tolerant systems that self-heal.

Besides circuit breaker control, with Hystrix you can add a fallback method that will be called to obtain a default value in case the main command fails.

Moreover, Hystrix generates metrics on execution outcomes and latency for each command, that we can use to monitor system behavior.

Config Service:

Spring Cloud Config is horizontally scalable centralized configuration service for distributed systems. It uses a pluggable repository layer that currently supports local storage, Git, and Subversion. In this project, we use native profile, which simply loads config files from the local classpath.