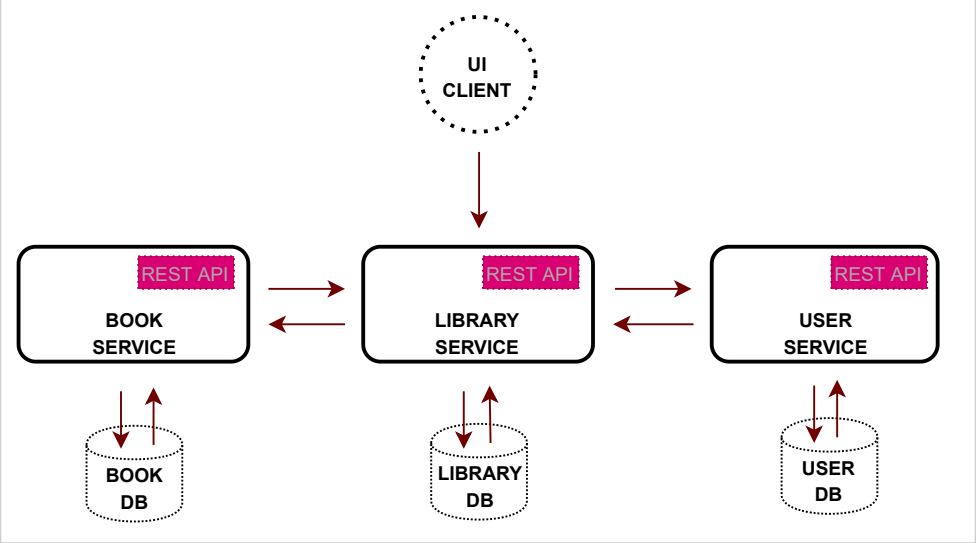
JPOP2 Case study

This article provides the details of the the application that will be built, during the . This application follows common micro services architecture patterns using spring-cloud services.

Functional Services



Book Service REST Endpoints:

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

/books/{book_id}	PUT	Update a book	GENERAL	Update book Author, Category, Description, etc

User Service REST Endpoints:

Endpoint	HTTP Method	Description	Access	Comments
/users	GET	List of all users	GENERAL	
/users/{user_id}	GET	Get user by id	GENERAL	Login, view profile
/users	POST	Add a user	GENERAL	User Registration
/users/{user_id}	DELETE	Delete a user	GENERAL	
/users/{user_id}	PUT	Update a user	GENERAL	Update user profile

Library Service REST Endpoints:

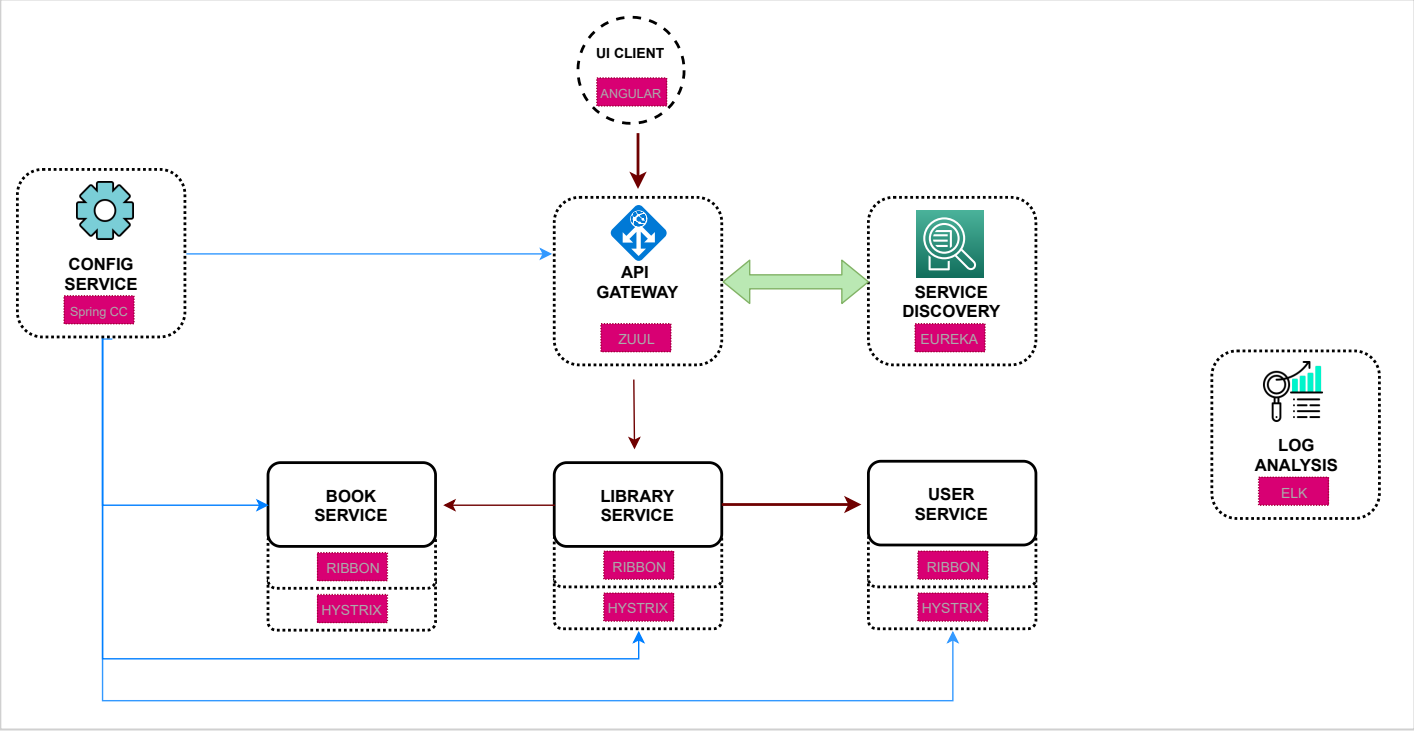
Endpoint	HTTP Method	Description	Access	Comments
/lib	GET	Login	GENERAL	calls user service **GET /users/{user_id}**
/lib/books	GET	List of all books	GENERAL	Calls book service **GET /books** URL
/lib/books/{book_id}	GET	Get a book's details	GENERAL	Calls book service **GET /books/{book_id}** URL
/lib/books/{book_id}	POST	Add a new book	ADMIN	Calls book service **POST /books/{book_id}** URL
/lib/books/{book_id}	DELETE	Delete a book	ADMIN	Delete book association with user in library table and calls book service **DELETE /books/{book_id}** URL
/lib/users	GET	List of all users	GENERAL	Calls user service **GET /users** URL
/lib/users/{user_id}	GET	View user profile with all issued books	GENERAL	calls user service **GET /users/{user_id},** fetches book ids on user_id from library table and calls **GET /books/{book_id}** for all books. Displays consolidated data

/lib/users/{user_id}	POST	Add a user	GENERAL	User Registration, calls user service **POST /users/{user_id}**
		that user_id in library table and delete user.		**users/{user_id}**
/lib/users/{user_id}	PUT	Update user's account details	GENERAL	Calls **PUT users/{user_id}**
/lib/users/{user_id}/books/{book_id}	PUT	Issue a book to a user	GENERAL	Updates library table with book-user association.

UI Client

Angular / ReactJS UI client to access Library system functionalities.

Architecture



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.

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. *What data goes in config(TBD)?*

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

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.

Feign:

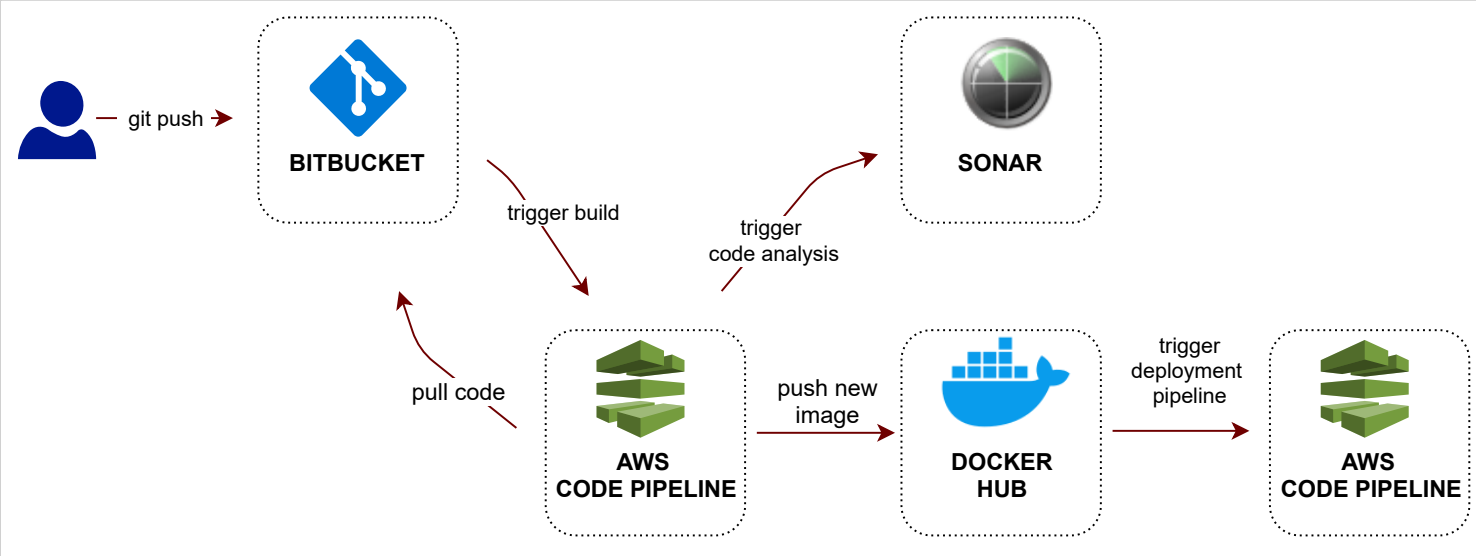
Feign is a declarative HTTP client, which seamlessly integrates with **Ribbon** and **Hystrix**. Actually, with one spring-cloud-starter-feign dependency and @EnableFeignClients annotation you have a full suite of a load balancer, circuit breaker, and HTTP client with a sensible ready-to-go default configuration.

Monitor Dashboard

In this project configuration, each microservice with **Hystrix** on board pushes metrics to Turbine via **Spring Cloud Bus (with AMQP broker)**. The Monitoring project is just a small Spring boot application with **Turbine** and **Hystrix** Dashboard. *Still to decide if we need to put this in our scope of the case study(TBD)*

Infrastructure Automation

Deploying microservices, with their interdependence, is a much more complex process than deploying a monolith application. It is important to have a fully automated infrastructure.



BitBucket

Bitbucket is a GIT code management tool.

AWS Code Pipeline

AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously.

Docker Hub

Docker Hub is a cloud-based repository in which Docker users and partners create, test, store and distribute container images

No labels