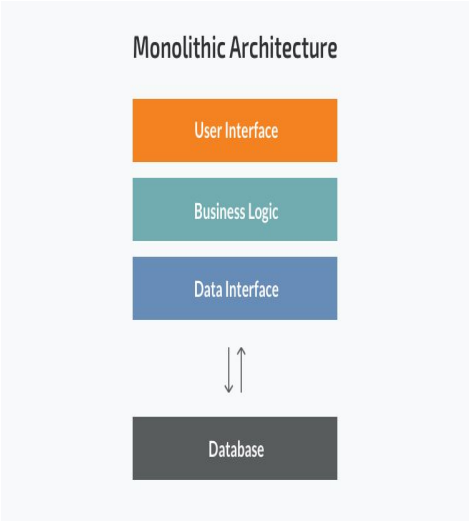
MICROSERVICES WITH SPRING BOOT & SPRING CLOUD

CURRENCY CONVERSION MICROSERVICES WITH SPRING BOOT, SPRING CLOUD AND DOCKER

Monolithic Architecture

- Traditional way of building applications
- Single and indivisible unit(single code base)
- Comprises a client-side user interface, a server side-application, and a database
- It is unified
- Tightly coupled
- All the functions are managed and served in one place
- They lack modularity
- For modification same unified code base is accessed
- Changes are made to the whole stack at once



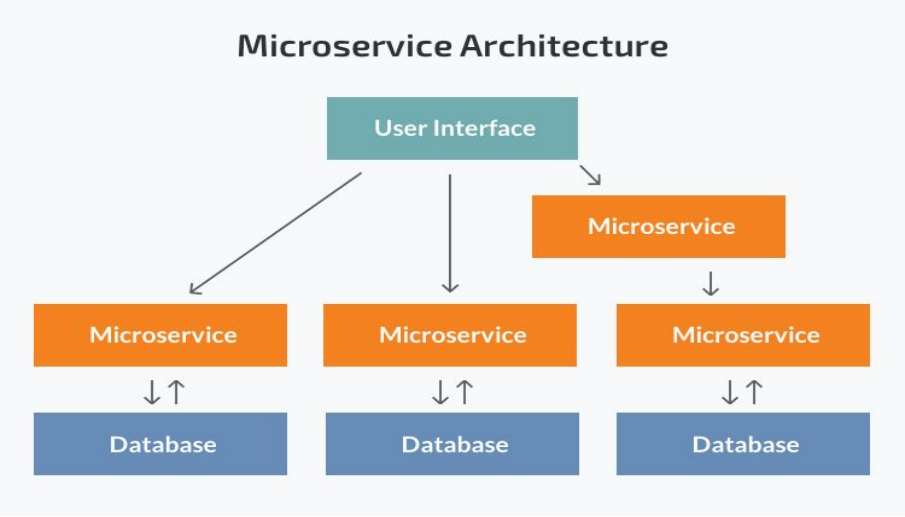
Challenges with Monolithic architecture

LIMITATIONS:

- Limitation in size and complexity
- Difficult to scale when different modules have conflicting resource requirements
- Complex system of code within one application is hard to manage
- Continuous deployment is difficult
- Highly tight coupling
- New technology barriers
- Can lead to extensive manual testing
- bug will impact the availability of the entire application

MICROSERVICES ARCHITECTURE

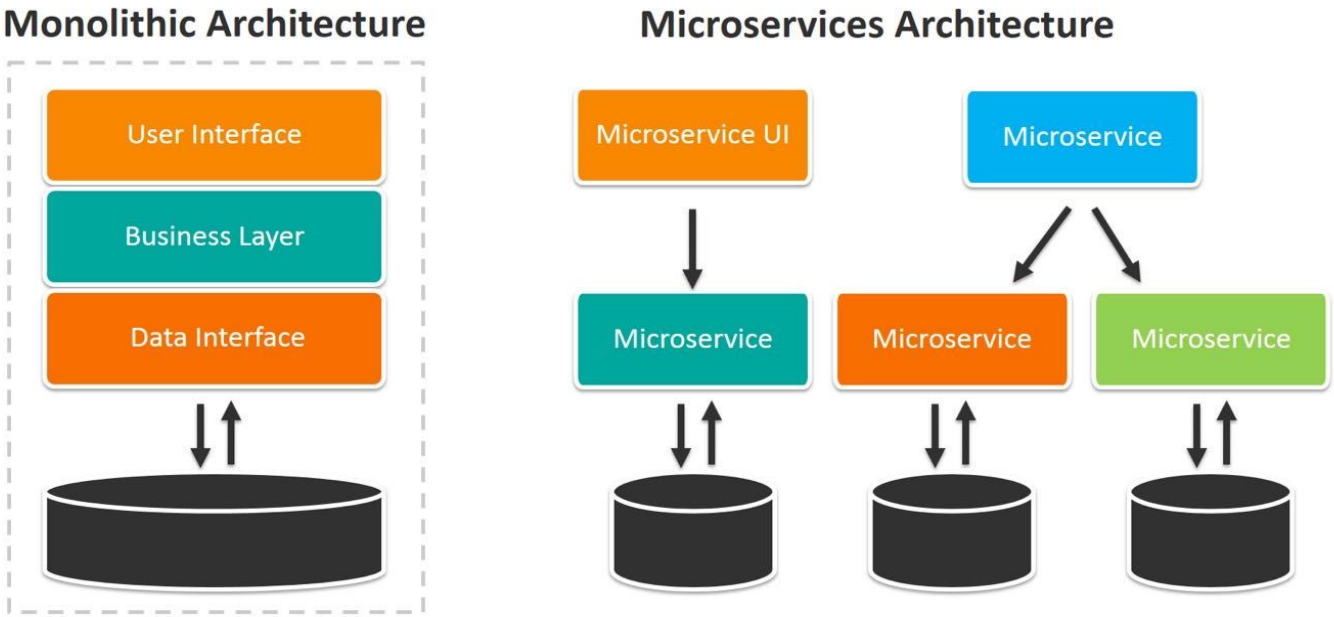
“The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.” - MARTIN FOWLER



MICROSERVICES ARCHITECTURE

- Application is splitted into a set of smaller, interconnected services
- Loosely coupled
- Faster to develop, much easier to understand & maintain
- It reduces barrier of adopting new technologies
- Enables each microservice to be deployed independently
- Enables each service to be scaled independently.
- The higher level of agility
- Cloud Enabled

MONOLITHIC VS MICROSERVICES



SPRING BOOT for MICROSERVICES

- Spring Boot is an open-source framework, based on the Java platform
- Quickly builds REST based microservices with few simple annotations
- Greatly simplifies mapping HTTP-style verbs (GET, PUT, POST, and DELETE) to URLs
- Simplifies the serialization of the JSON protocol to and from Java objects
- Stand-alone application ready for production
- Embedded tomcat, Jetty or undertow directly
- Provide ‘starter’ dependencies to simplify the build configuration, especially for large projects
- No code generation & no requirement of XML configurations

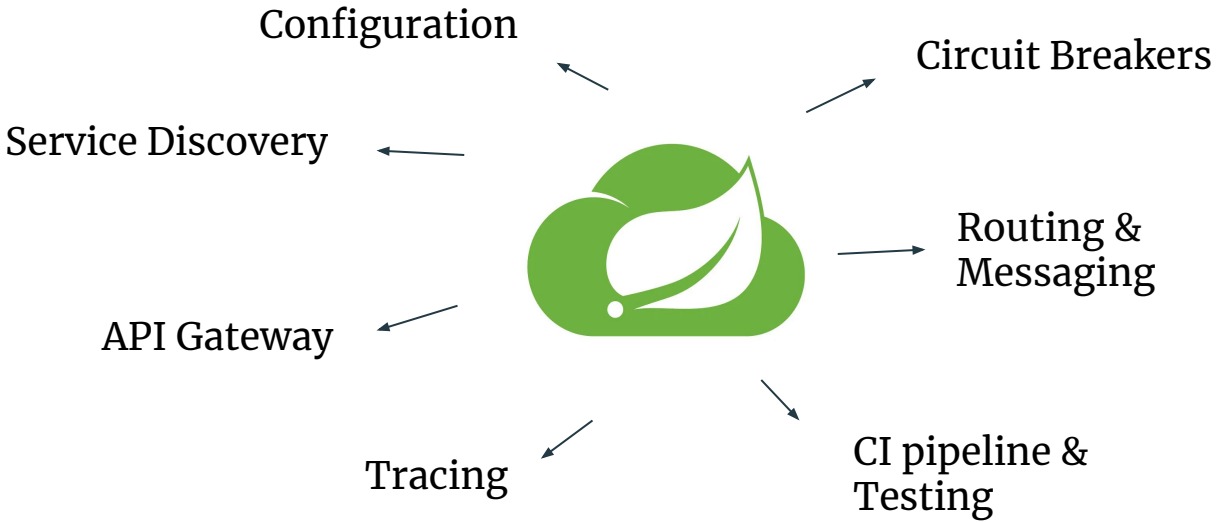
Why the cloud and microservices?

- The advantage of cloud-based microservices centers around the concept of elasticity
- Cloud service providers allow you to quickly spin up new virtual machines and containers
- Massive horizontal scalability during deployment
- More resiliency in applications
- Simplified infrastructure management
- High redundancy through geographic distribution

WHAT IS SPRING CLOUD?

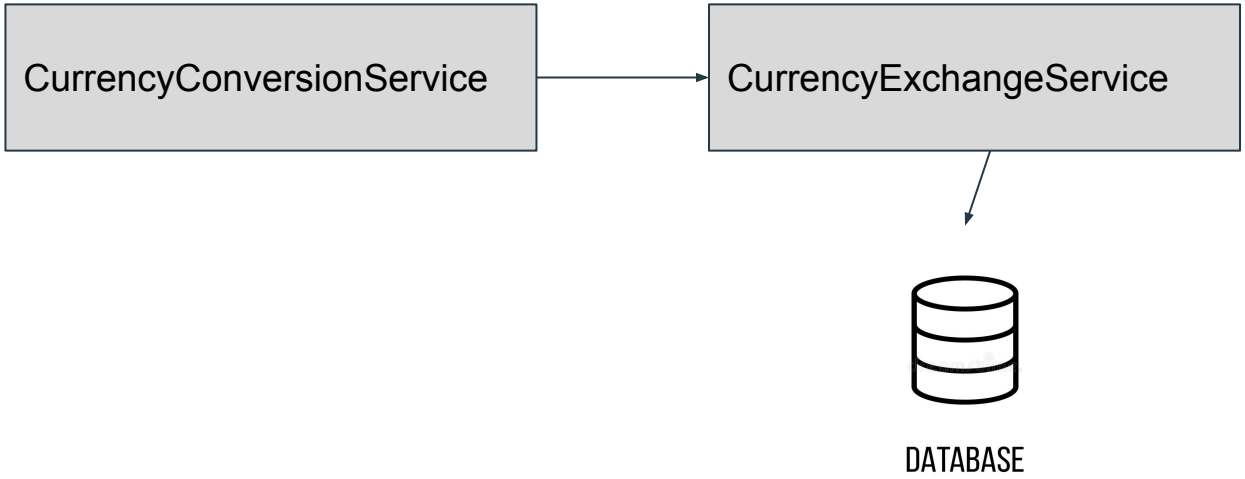
- Spring Cloud wraps the work of open source companies such as PIVOTAL, HashiCorp, and NETFLIX in delivering patterns
- Simplifies setting up and configuring of Spring Cloud projects into a Spring application
- Provides tool for the developer to quickly build some of the common patterns in distributed system
- They work well in distributed environment (developer's laptop, data centres, managed platform)

SPRING CLOUD COMPONENTS



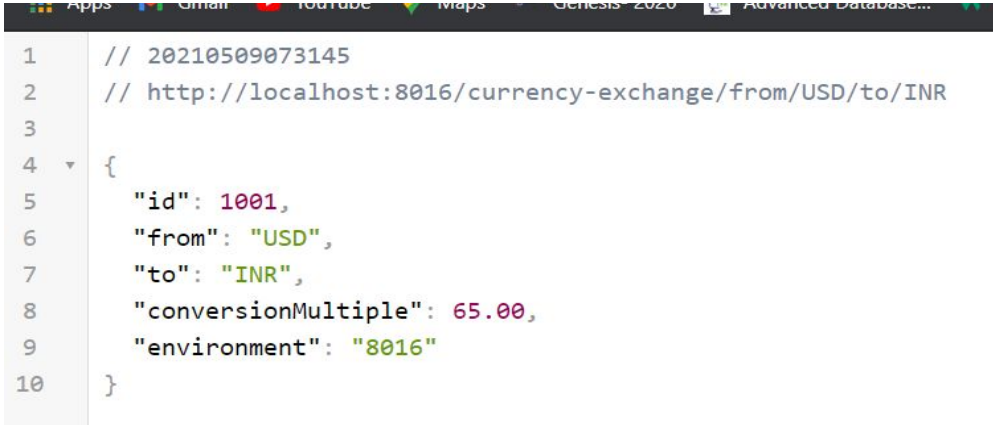
Currency Conversion microservices

- Foreign exchange market oriented microservice API
- Can convert a bucket of currencies into another currency
- Consumer facing microservice that interprets and routes requests to internal services to perform appropriate conversions



CURRENCY-EXCHANGE SERVICE

- Currency exchange service would talk to the database using JPA and return exchange rates for a specific currency.
- It will convert ‘from’ (specific currency) ‘to’ (specific currency)



Port:
8000,8001,8002...

CURRENCY-CONVERSION SERVICE

Currency conversion service makes use of currency exchange service to do the conversion.

It asks currency exchange service to provide exchange rates in order to convert one currency to the other.

```
// 20210509075524
// http://localhost:8765/currency-conversion/from/INR/to/JPY/quantity/99999

{
  "id": 1002,
  "from": "INR",
  "to": "JPY",
  "quantity": 99999,
  "conversionMultiple": 14.00,
  "totalCalculatedAmount": 1399986.00,
  "environment": "8016 rest template"
}
```

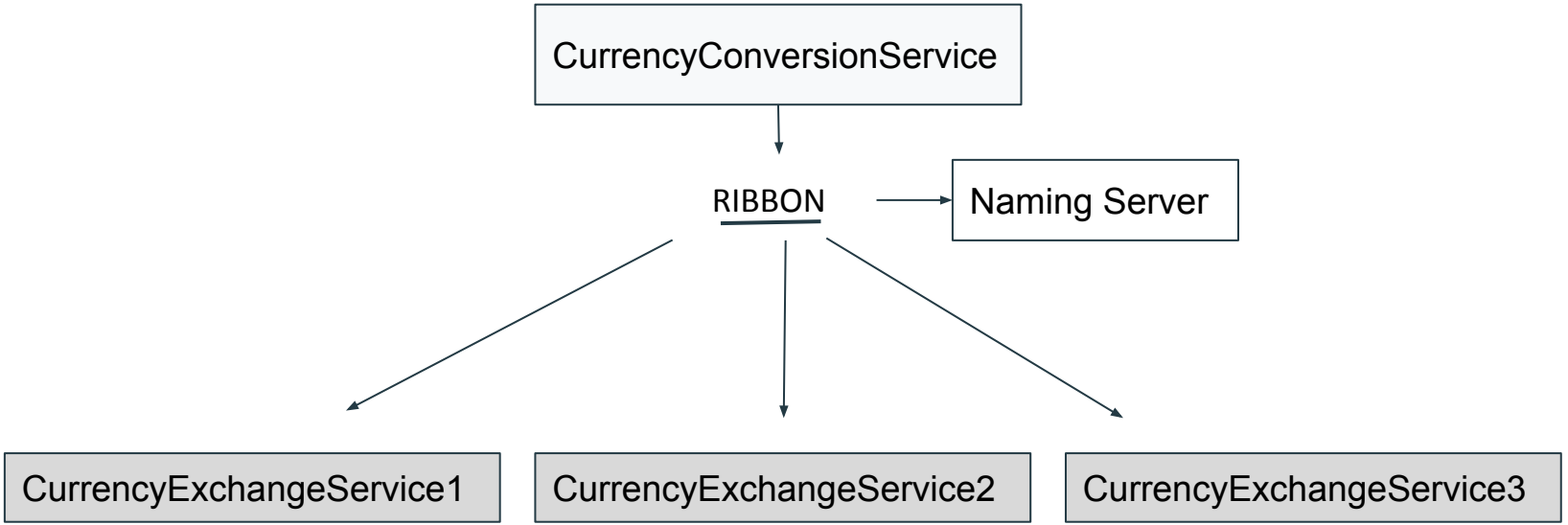
Port:
8100, 8101, 8102...

FEIGN REST CLIENT FOR SERVICE INVOCATION

- HTTP client created by NETFLIX to make HTTP communication easier
- Makes it easy to invoke other microservices
- Feign provides integration with RIBBON, which is a client side load balancing framework
- To use Feign create an interface and annotate it.
- To include Feign in your project use the starter with group org.springframework.cloud and artifact-id spring-cloud-starter-openfeign

```
@SpringBootApplication

@EnableFeignClients

public class Application {

    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);

        }}
```

RIBBON LOAD BALANCING

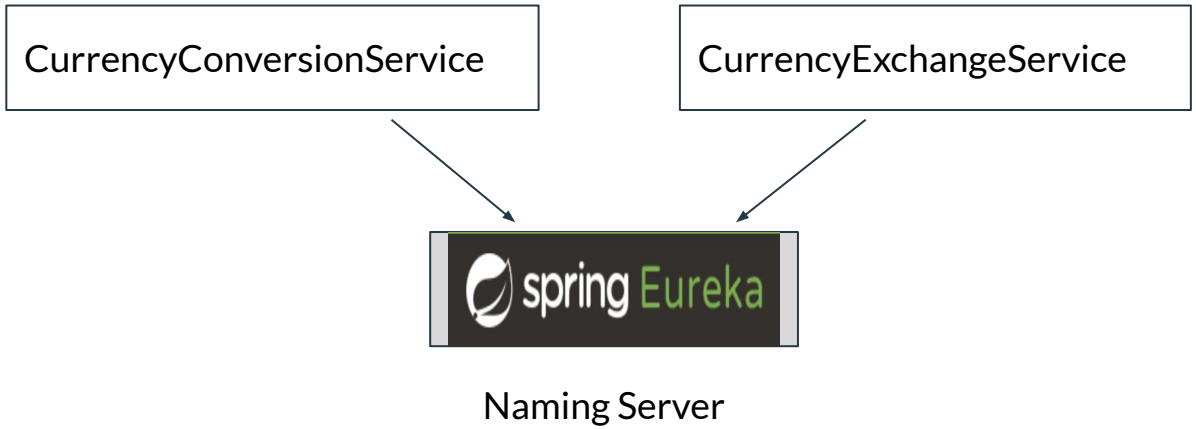


- Enabling RIBBON and distributing the load among different instances of Currency Exchange service

Service Registration with EUREKA

- All microservices will register itself to the naming server(service registration)
- Microservices will first ask the naming server to check the instance details for the request to be made (i.e service discovery)
- Replace list of servers with app property:

Eureka.client.service-url.default-zone = https//localhost:8761/eureka



EUREKA Naming SERVER

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION	n/a (1)	(1)	UP (1) - LAPTOP-9C89MPKT:currency-conversion:8101
CURRENCY-EXCHANGE	n/a (2)	(2)	UP (2) - LAPTOP-9C89MPKT:currency-exchange:8016 , LAPTOP-9C89MPKT:currency-exchange:8003

General Info

Name	Value
total-avail-memory	65mb
num-of-cpus	4
current-memory-usage	52mb (80%)
server-uptime	01:08
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	192.168.43.173
status	UP

IN MEMORY DATABASE: H2

- One of the popular in memory databases
- Spring Boot has very good integration for H2
- Supports a sub set of the SQL standard
- Provides a web console to maintain the database

```
#Enable H2 console
spring.h2.console.enabled= true
spring.datasource.url= jdbc:h2:mem:testdb
spring.data.jpa.repositories.bootstrap-mode=default
```

- Launch up H2 Console at http://localhost:8080/h2-console

SPRING CLOUD API GATEWAY

- Also called Edge Service
- Aims to provide a simple , yet effective way to route to APIs
- Provides a unified interface for a set of microservices so that clients no need to know about all the details of microservices internals
- Microservices should interact with each other via API gateway due to:
    - Common place for Authentication, Authorization & security
    - Rate Limits
    - Fault Tolerance
    - Service Aggregation

Features of Spring Cloud Gateway

- Able to match routes on any request attributes
- Predicates and filters are specific to routes
- Easy to write predicates and filters
- Circuit breaker integration
- Request rate limiting
- Path rewriting
- Spring cloud DiscoveryClient integration

  Spring.cloud.gateway.discovery.client.enabled = true

- Runs on port :

  http://localhost:8765/

CIRCUIT BREAKER

- What if one of these services is down or slow?
  - Impacts entire chain.
- Questions?
  - Can we return a fallback response if the service is down?
  - Can we implement a circuit breaker pattern to reduce the load?
  - Can we retry requests in case of temporary failures?
  - Can we implement rate limiting?

SOLUTION: Circuit Breaker Framework - RESILIENCE4j

RESILIENCE4j

- Lightweight
- Easy to use Fault-Tolerance Library
- Inspired by NETFLIX HYSTRIX
- Designed for Java 8 and functional programming
- Provides higher-order functions (decorators) to enhance
    - functional interface, lambda expression or method reference with a Circuit Breaker, Rate Limiter, Retry or Bulkhead

RESILIENCE4j Modules

Core modules

- resilience4j-circuitbreaker: Circuit breaking
- resilience4j-ratelimiter: Rate limiting
- resilience4j-bulkhead: Bulkheading

Frameworks modules

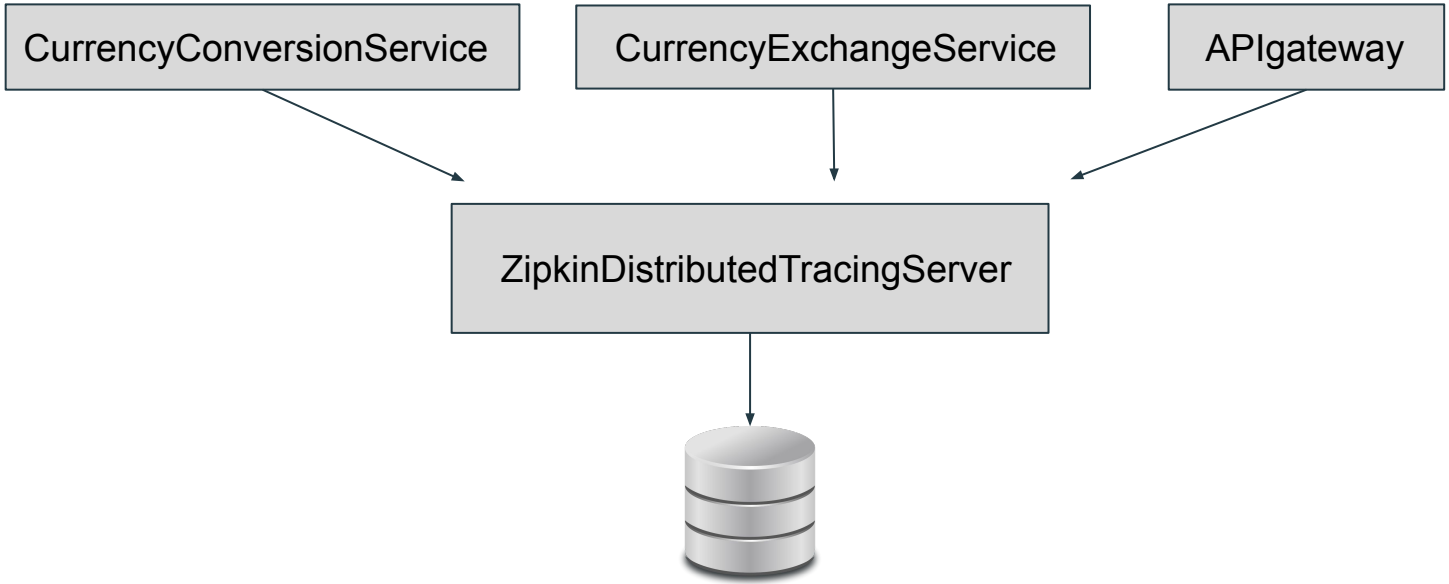
- resilience4j-spring-boot: Spring Boot Starter
- resilience4j-spring-boot2: Spring Boot 2 Starter

All core modules and the Decorators class

- resilience4j-all

DISTRIBUTED TRACING

- Required to track the requests as its going through multiple systems
- Allows to check where the exact failure is.



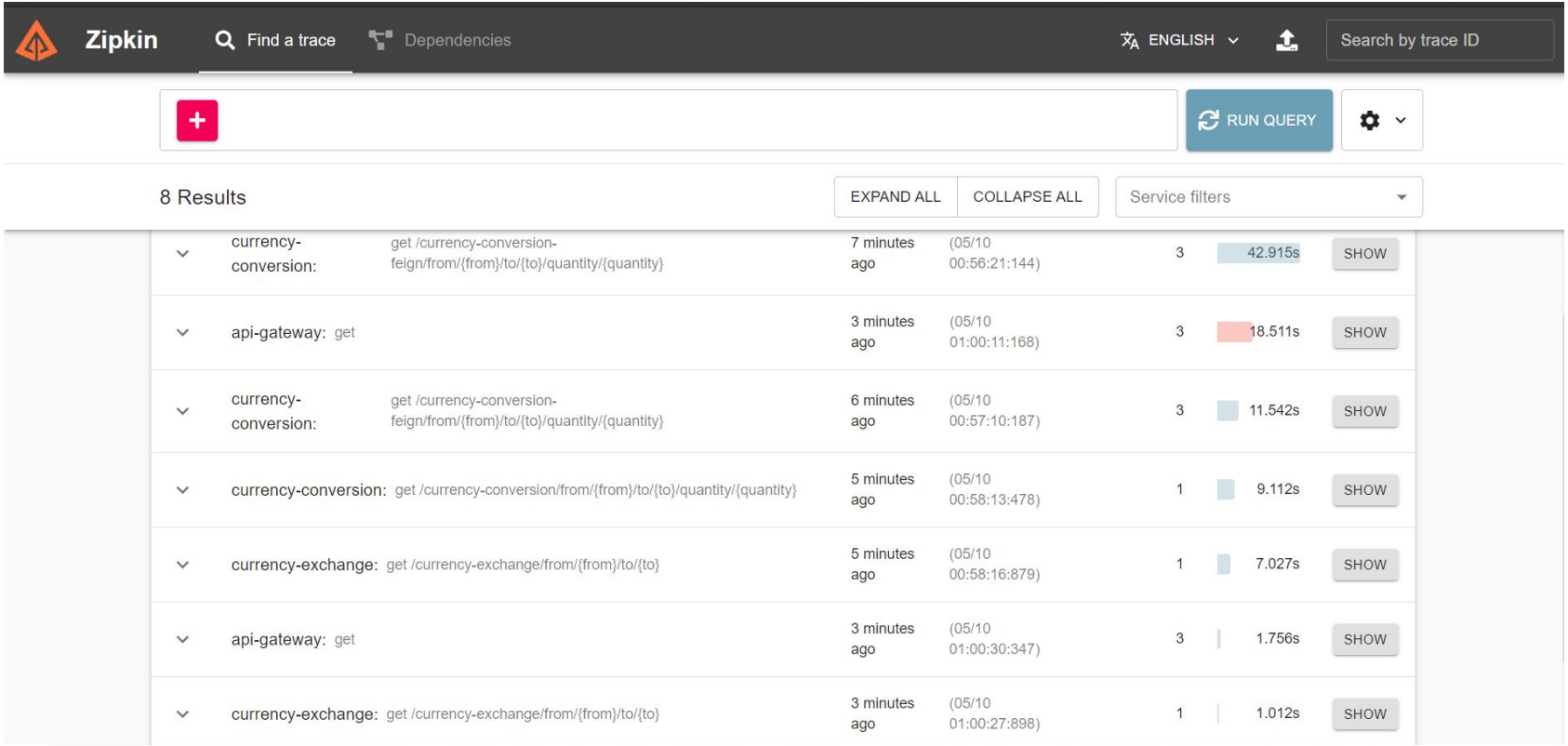
SPRING CLOUD SLEUTH

- Provides Spring Boot auto-configuration for distributed tracing
- Implements distributed tracing solution for spring cloud
- Allows to have unique identifier to each request
- Sleuth provided unique ID will be used in Zipkin Tracing
- A call route to currency conversion:
    - First , User calls the currency conversion API
    - Request goes to the spring cloud log filter to do the logging
    - Then currency conversion microservice tries to call the exchange microservice via the proxy of Spring cloud API gateway
    - Request goes to API gateway log filter
    - Spring cloud API gateway calls the currency exchange microservice

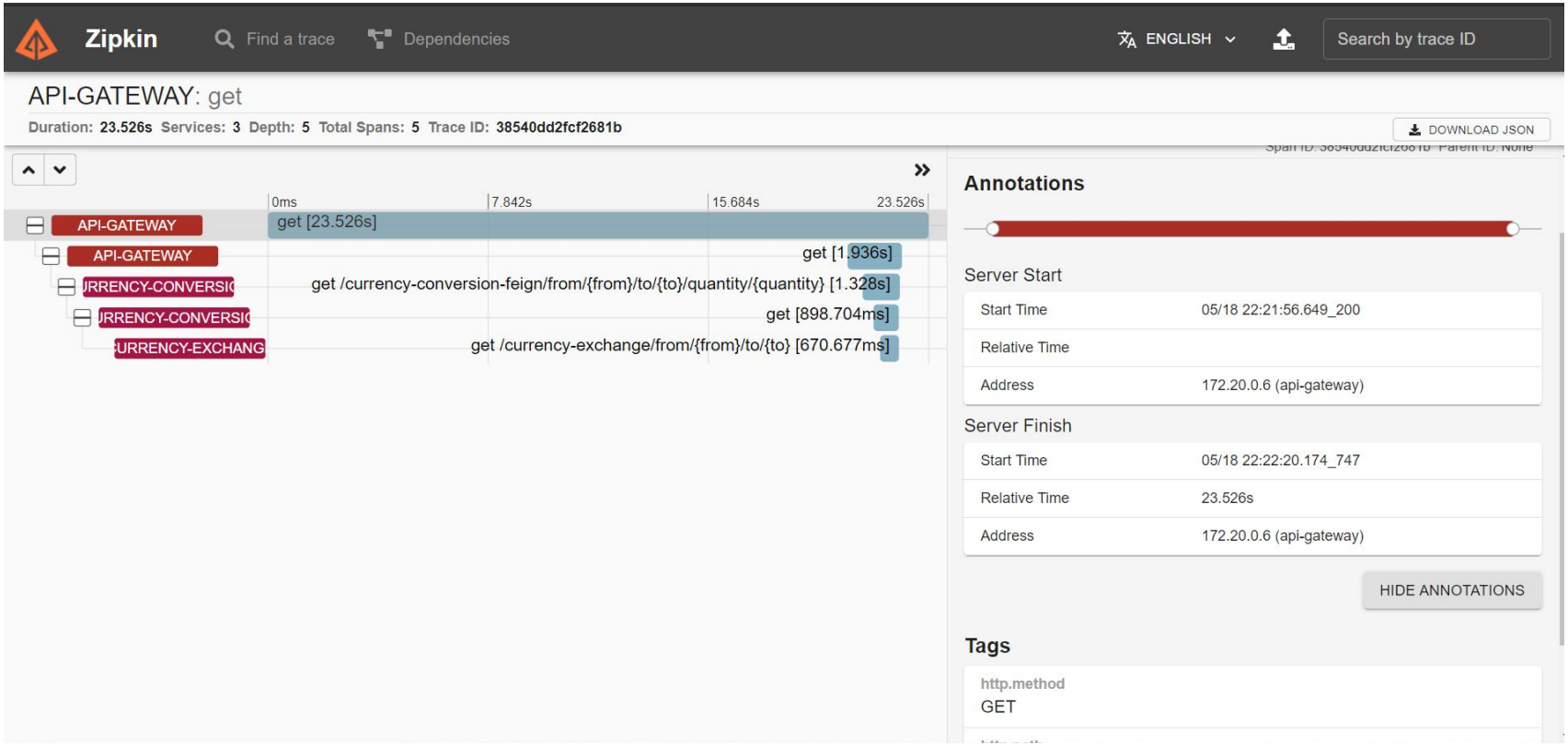
ZIPKIN

- Zipkin is a distributed tracing system
- Helps gather timing data
- Troubleshoot latency problems in service architectures
- Features include both the collection and lookup of this data

Zipkin UI

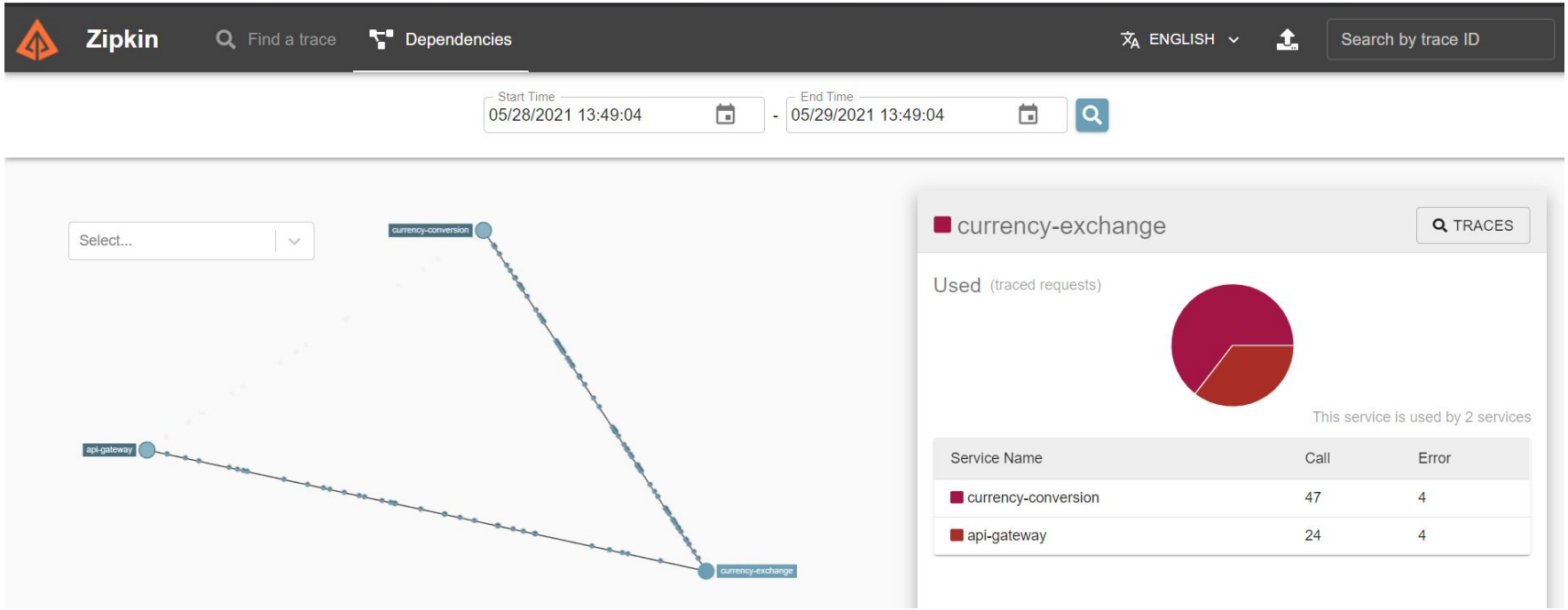

Zipkin
Find a trace
Dependencies
ENGLISH
Search by trace ID
RUN QUERY
8 Results
EXPAND ALL
COLLAPSE ALL
Service filters
currency-conversion: get /currency-conversion-feign/from/{from}/to/{to}/quantity/{quantity} 7 minutes ago (05/10 00:56:21:144) 3 42.915s SHOW
api-gateway: get 3 minutes ago (05/10 01:00:11:168) 3 18.511s SHOW
currency-conversion: get /currency-conversion-feign/from/{from}/to/{to}/quantity/{quantity} 6 minutes ago (05/10 00:57:10:187) 3 11.542s SHOW
currency-conversion: get /currency-conversion/from/{from}/to/{to}/quantity/{quantity} 5 minutes ago (05/10 00:58:13:478) 1 9.112s SHOW
currency-exchange: get /currency-exchange/from/{from}/to/{to} 5 minutes ago (05/10 00:58:16:879) 1 7.027s SHOW
api-gateway: get 3 minutes ago (05/10 01:00:30:347) 3 1.756s SHOW
currency-exchange: get /currency-exchange/from/{from}/to/{to} 3 minutes ago (05/10 01:00:27:898) 1 1.012s SHOW

ZIPKIN UI


Zipkin
Find a trace
Dependencies
ENGLISH
Search by trace ID
API-GATEWAY: get
Duration: 23.526s Services: 3 Depth: 5 Total Spans: 5 Trace ID: 38540dd2fcf2681b
DOWNLOAD JSON
0ms
7.842s
15.684s
23.526s
API-GATEWAY
get [23.526s]
API-GATEWAY
get [1.936s]
URRENCY-CONVERSIO
get /currency-conversion-feign/from/{from}/to/{to}/quantity/{quantity} [1.328s]
URRENCY-CONVERSIO
get [898.704ms]
URRENCY-EXCHANG
get /currency-exchange/from/{from}/to/{to} [670.677ms]
Annotations
Server Start
Start Time 05/18 22:21:56.649_200
Relative Time
Address 172.20.0.6 (api-gateway)
Server Finish
Start Time 05/18 22:22:20.174_747
Relative Time 23.526s
Address 172.20.0.6 (api-gateway)
HIDE ANNOTATIONS
Tags
http.method
GET

ZIPKIN UI - Dependency Diagram

- The Zipkin UI also presents a Dependency diagram
- Helpful for identifying error paths or calls to deprecated services



RABBITMQ

RabbitMQ

- Applications need to be “instrumented” to report trace data to Zipkin
- The data served to the UI are stored in-memory or Backend
- The most popular ways to report data to Zipkin are via HTTP or Kafka or RabbitMQ
- One of the most popular open source message brokers
- Lightweight
- Easy to deploy on premises and in the cloud
- supports multiple messaging protocols

DOCKER

- open platform for developing, shipping, and running applications
- separates applications from infrastructure
- Quick software delivery
- package and run an application in a loosely isolated environment called a container
- Isolation and security
- Lightweight
- Fast, consistent delivery
- Running more workloads on the same hardware

DOCKER COMPOSE

- Tool for defining and running multi-container Docker applications
- Use a YAML file to configure your application's services : *docker-compose.yml*
- Popular commands:
  - *docker-compose up*
  - *docker-compose down*
  - *Docker-compose stop*

Features:

- Multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments