**Micro Services:**

1. **Monolith Architecture:**

=> Develop all functionalities in single app

=> Application will be packaged as one fat jar/fat war

=> App will be deployed in single sever.

**2. Drawbacks:**

=> Single point of failure

=>Re-Deploy entire of the app

=>Maintenance of the app

=>Burdon on server

**3. Micro Services**:

=>Micro Services is not Technologies

=>Micro Services is not Framework

=> Micro Services is not API

=>Micro Services is Architectural Design Pattern and it’s universal

=> Micro Services based application means collections of rest apis.

=>Micro Services means independently and executed services.

**4. Benefits:**

=> Loosely Coupled

=> Easy Maintenance

=> Load will be Independency

=> Technology Independency

=> High Availability

**5. Challenges:**

=> Bounded Context (deciding no.of rest APIs to develop)

=> Duplicate Configuration

=> Visibility

**6. Micro Services Architecture:**

=> There is no standard architecture for Micro Service Development

=> People are customizing micro services project architecture according to their requirement.

1) Service Registry

2) Admin Server

3) Zipkin Server

4) Backend Services (Rest APIs)

5) API Gateway

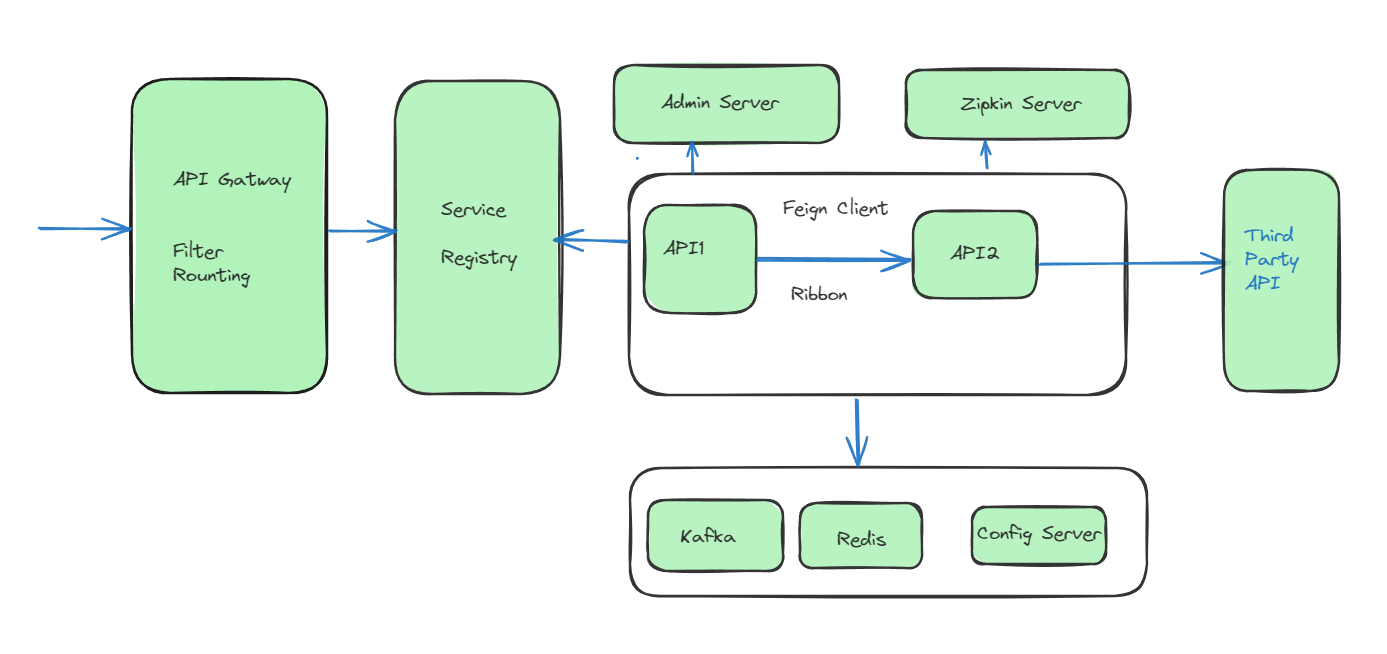
6) Feign Client

7) Config Server

8) Apache Kafka

9) Redis Cache

10) Docker



1. **Service Registry:**

=> Service Registry is used to maintain list of services available in the project.

=>It provides information about registered services like Name of service, url of service, status of service

=>It provides no.of instances available for each service.

=>We can use Eureka Server as a service registry.

=>Eureka server provided by Spring Cloud Netflix library

**2. Admin Server:**

=>Actuators are used to monitor and manage our applications

=>Monitoring and managing all the APIs separately is a challenging task

=>Admin Server Provides and user interface to monitor and manage all the APIs at one place using actuator endpoints.

**3. Zipkin Server:**

=> It is used for Distributed tracing

=>Using zipkin server, we can monitor which API is taking more time to process request.

=>Using Zipkin we can understand how many APIs involved in request processing.

**4. Backend APIs:**

=>Backend APIs contains business logic

=> Backend APIs are also called as REST APIs / services / micro services

Ex: payment-api, cart-api, flights-api, hotels-api

=> Note: Backend API can register as client for Service Registry, Admin server & Zipkin server (It is optional)

**5. Feign Client:**

=> It is provided by spring cloud libraries

=> It is used for Inter Service Communication

=> Inter service communication means one api is accessing another api using Service Registry.

Note: External communication means accessing third party APIs.

=>When we are using Feign Client we no need mention URL of the API to access. Using service name feign client will get service URL from service registry.

=> Feign Client uses Ribbon to perform Client side load balancing.

**6.** **API Gateway:**

=> API Gateway is used to manage our project backend APIs

=> API Gateway acts as mediator between user requests and backend APIs

=> API Gateway acts as entry point for all backend APIs

=>In API Gateway we will have 2 types of logics

1) Request Filter : To validate the request (go / no-go)

2) Request Router: forward request to particular backend-api based on URL Pattern

/hotels => hotels - API

/flights => flights - API

/trains => trains – API

**7.** **Config Server:**

=> Config Server is part of Spring Cloud Library

=> Config Server is used to externalize config properties of application

Note: In real-time we will keep app config properties outside of the project to simplify application maintenance.

**8.** **Apache Kafka:**

=> Kafka is a message broker

=> Kafka works based on Publisher - Subscriber model

=> To send msgs from one app to another app we will use Kafka as a mediator.

=> Using Kafka we can develop Event Driven Micro Services based applications.

**9. Redis Cache:**

=> In our application we will have 2 types of tables

1) Transaction tables (app will insert/update/delete records)

2) Non-Transactional tables (app will only retrieve records)

Note: It is not recommended to load non-transactional tables data from DB every time.

=> To reduce no.of round trips between Java app and Database we will use cache.

=> Redis is used for distributed cache implementation.

**Steps to develop WELCOME-API:**

1) Create Spring Boot application with below dependencies

- eureka-discovery-client

- starter-web

- devtools

- actuator

- zipkin

- admin-client

2) Configure @EnableDiscoveryClient annotation at boot start class

3) Create Rest Controller with required method

4) Configure below properties in application.yml file

**application.yml:**

|  |
| --- |
| server:  port: 1111  spring:  application:  name: WELCOME-API  boot:  admin:  client:  url: http://localhost:9090/  eureka:  client:  serviceUrl:  defaultZone: http://localhost:8761/eureka  management:  endpoints:  web:  exposure:  include: '\*' |

5) Run the application and check in Eureka Dashboard (It should display in eureka dashboard)

6) Check Admin Server Dashboard (It should display) (we can access application details from here)

Ex: Beans, loggers, heap dump, thred dump, metrics, mappings etc...

7) Send Request to REST API method

8) Check Zipkin Server UI and click on Run Query button

(it will display trace-id with details)

**Steps to develop GREET-API:**

1) Create Spring Boot application with below dependencies

- eureka-discovery-client

- starter-web

- devtools

- actuator

- zipkin

- admin-client

- openfeign

2) Configure @EnableDiscoveryClient annotation at boot start class

3) Create Rest Controller with required method

4) Configure below properties in application.yml file

**application.yml:**

|  |
| --- |
| server:  port: 2222  spring:  application:  name: GREET-API  boot:  admin:  client:  url: http://localhost:9090/  management:  endpoints:  web:  exposure:  include: '\*' |

5) Run the application and check in Eureka Dashboard (It should display in eureka dashboard)

6) Check Admin Server Dashboard (It should display) (we can access application details from here)

Ex: Beans, loggers, heap dump, thred dump, metrics, mappings etc...

7) Send Request to REST API method

8) Check Zipkin Server UI and click on Run Query button

(it will display trace-id with details)

**Interservice communication:**

|  |
| --- |
| => Add @EnableFeignClients dependency in GREET-API boot start class  => Create FeignClient interface like below  @FeignClient(name = "WELCOME-API")  public interface WelcomeApiClient {    @GetMapping("/welcome")  public String invokeWelcomeMsg();  }  => Inject feign client into GreetRestController like below  @RestController  public class GreetRestController {    @Autowired  private WelcomeApiClient welcomeClient;    @GetMapping("/greet")  public String getGreetMsg() {    String welcomeMsg = welcomeClient.invokeWelcomeMsg();    String greetMsg = "Good Morning, ";    return greetMsg.concat(welcomeMsg);  }  }  => Run the applications and access greet-api method  **(It should give combined response)** |