Farkalit Usman

vXebia@usmanuat.com

|  |  |  |
| --- | --- | --- |
| **Version** | **Name** | **Description** |
| 1.0 | Farkalit Usman | Initial document for Usman bank architecture |
|  |  |  |
|  |  |  |
|  |  |  |

Abstract

Define the micro services architecture

Usman bank MS Architecture

Micro services Architecture

#### Usman Bank Micro services Architecture

Table of Contents

[Tools and Technology Required for Micro Services: 2](#_Toc19600583)

[High Level Architecture 3](#_Toc19600584)

[Detailed Micro services Layers: 4](#_Toc19600585)

[4](#_Toc19600586)

[One Micro Service Architecture 4](#_Toc19600587)

[Eureka Netflix Registry Server: 5](#_Toc19600588)

[CI/CDs and DevOps 5](#_Toc19600589)

[Deployment Server 5](#_Toc19600590)

# Tools and Technology Required for Micro Services:

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Software/Tools** | **Version** | **Comments** |
| 1 | Java, JRE | 1.8.0 | Java/EE API |
| 2 | Spring Tool Suite 4 | 4.3.0 | API development tool |
| 3 | Maven | 3.6.1 | To build the application |
| 4 | Spring boot | 2.1.2 | Framework to create micro-service |
| 5 | Tomcat Server | 9.0 | To deploy the services |
| 6 | Rest Services | 2.1.2 | Every micro service endpoints can be exposed over HTTP/HTTPS |
| 7 | JPA/Hibernate/  JDBC-native | 2.1.2 | To handle the DAO or repository layer. It directly interact with database(s). |
| 8 | Spring security | 2.1.2 | To handle application authentication and authorization. |
| 9 | Spring cloud / Netflix/Eureka/ZUUL | 2.1.2 | Application can be registered on Spring cloud Netflix Eureka so that internally accessible. *Client-side service discovery* allows services to find and communicate with each other without hard-coding hostname and port. |
| 10 | Oracle Database | 12.c | To store the physical data. |
| 11 | Oracle Developer | 4.1 | To handle the oracle PL/SQL. |
| 12 | JSON Web Token | 2.1.2 | Token for application security. |
| 13 | Swagger UI | 2.9 | To display the API documentation where application can also be executed. |
| 14 | Log4J2 | 2.11 | To save the application logs based on time and size. |
| 15 | JUnit with Mockito | 4.1 | To write the java application test cases. |
| 16 | Sonar Cube/ Lint /Jacoco | 0.8.3 | For test code coverage and to meet java coding standard. |
| 17 | Redis Cache DB | 5.0.5 | For caching the data. https://redis.io/ |
| 18 | Kafka Topic/Queue | 2.12 | For asynchronous messaging. https://kafka.apache.org/ |
| 19 | MS Azure\* |  | Recommended for cloud deployment |
| 20 | Docker\* |  | Recommended to deploy the images of the application for highly availability (HA) |
| 21 | Lombak \* |  | To avoid the boiler plate getter setter code |
| 22 | Hystrix\* |  | Fault Tolerance: To load the default response in case of service response not received. |
| 23 | Apache Cassandra Database\* | 3.x | Recommended for micro-services as it open-source, distributed, wide column store, NoSQL database. http://cassandra.apache.org/ |

**\***Optional

## High Level Architecture

Web Client

API Gateway/Netflix-Eureka&ZUUL

Common-utility

Utility-services

Authentication

Customer

Accounts

Payment

Transfer

DB-2

DB-1

DB-utils

DB-auth

DB-4

DB-3

Utility-services: audit, email and sms etc.

Common-utility: constants, masking, formatting and validation of date, email, phone number etc.

Customer-service: Contain all types of customer related service based on CIF number or customer Id.

Account-service: Contains all types of account related service based on CIF number or account number.

Payment-service: It handles all types of payment like zakat payment, mobile payment etc.

Transfer-service: It handles all types of amount transfer like own transfer, account transfer etc.

Authentication: For each customer it creates a JSON Web Token (JWT) for authorize access of any service.

## Detailed Micro services Layers:

React/Native 

Angular

Mobile

Web/JSP

Rest Client

API Gateway

1-N Services

Oracle DB-1

Microservice-1

Repository/DAO

JPA/Native

Service

Service logic

Rest API https://kafka.apache.org/

Web Services

Validation

Helper

Service Client/Netflix CClientSerer

Service Registry/Eureka

## One Micro Service Architecture

How a single micro service will be interacted with common-utility and utility-services.

Common-utils

Customer-management

Utility-services

DB-util

App-DB

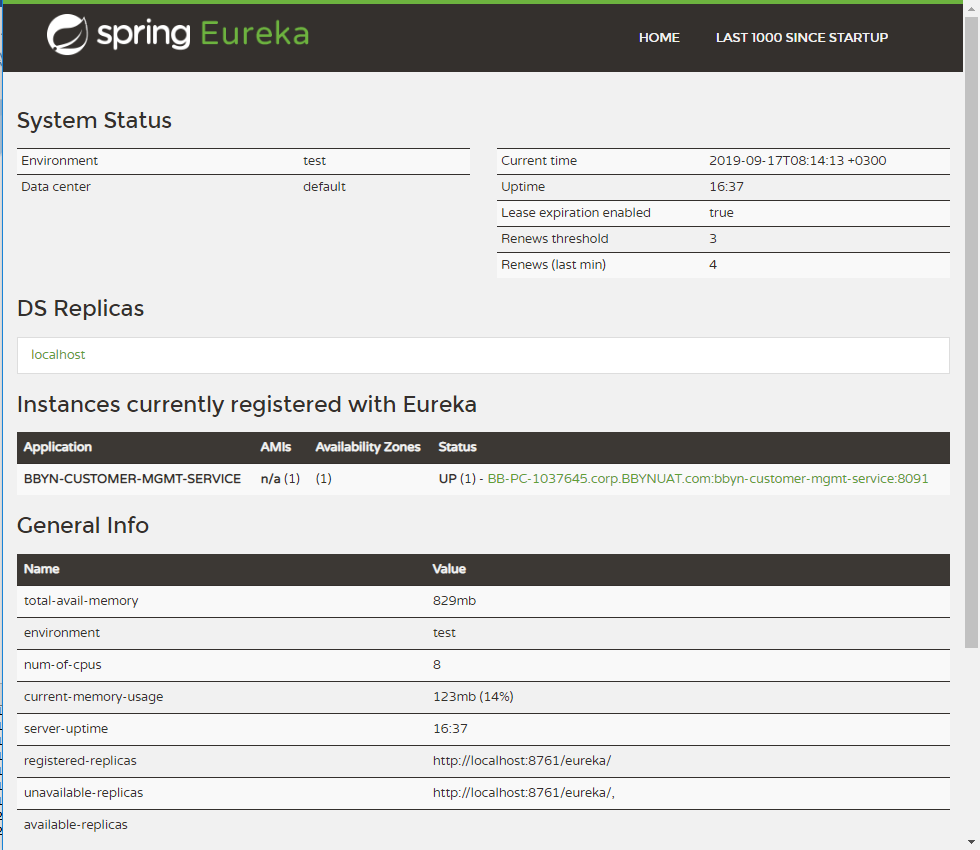
## Eureka Netflix Registry Server:

Eureka server provide a dashboard with its details information. Also we can view all the registered application on it.

Service Registry/Eureka

MS-Customer Eureka Client

MS-Accounts Eureka Client



## NETFLIX

Netflix, a pioneer in the microservices space, has built many such tools. Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).

## EUREKA

Eureka is a service registry, which registers all of a microservices instances and supplies each microservice with instance information to use in discovering others.

## HYSTRIX

Hystrix, the Hystrix Dashboard, and Turbine provide fault tolerance using circuit breakers and enable monitoring of circuits across microservices and instances.

## ZUUL

Zuul is an edge service that provides dynamic routing, monitoring, resiliency, security, and more.

## Spring Cloud Netflix

Eureka is a REST (Representational State Transfer) based service that is primarily used in the AWS cloud ***for locating services for the purpose of load balancing and failover of middle-tier servers***. We call this service, the Eureka Server. Eureka also comes with a Java-based client component, the Eureka Client, which makes interactions with the service much easier. The client also has a built-in load balancer that does basic round-robin load balancing. At Netflix, a much more sophisticated load balancer wraps Eureka to provide weighted load balancing based on several factors like traffic, resource usage, error conditions etc. to provide superior resiliency

@Netflix GitHub

## 

## Spring Cloud Features

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems.

* Distributed/versioned configuration
* Service registration and discovery
* Routing
* Service-to-service calls
* Load balancing
* Circuit Breakers
* Global locks
* Leadership election and cluster state
* Distributed messaging

## Load Balancing with Ribbon and Spring Cloud

It provides client-side load balancing for a microservice application using Netflix Ribbon

Read: <https://spring.io/guides/gs/client-side-load-balancing/>

#Spring boot application name

spring:

application:

name: usman-customer-mgmt

#set the Ribbon list of servers for load balance

usman-customer-mgmt:

ribbon:

eureka:

enabled: true

listOfServers: localhost:8091,localhost:8061,localhost:8071

ServerListRefreshInterval: 15000

@SpringBootApplication

@EnableEurekaClient

//@RibbonClient( name= "usman-customer-mgmt", configuration = RibbonConfiguration.class)

**public** **class** CustomerManagementApp {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CustomerManagementApp.**class**, args);

}

}

**public** **class** RibbonConfiguration {

@Autowired

IClientConfig config;

@Bean

**public** IPing ribbonPing(IClientConfig config) {

**return** **new** PingUrl();

}

@Bean

**public** IRule ribbonRule(IClientConfig config) {

**return** **new** AvailabilityFilteringRule();

}

}

## Uploading Files through Zuul

If you use @EnableZuulProxy, you can use the proxy paths to upload files and it should work, so long as the files are small. For large files there is an alternative path that bypasses the Spring DispatcherServlet (to avoid multipart processing) in "/zuul/\*". In other words, if you have zuul.routes.customers=/customers/\*\*, then you can POST large files to /zuul/customers/\*. The servlet path is externalized via zuul.servletPath. If the proxy route takes you through a Ribbon load balancer, extremely large files also require elevated timeout settings, as shown in the following example:

hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 60000

ribbon:

ConnectTimeout: 3000

ReadTimeout: 60000

## CI/CDs and DevOps

1. Maven and Git-hub with Jenkins/Hudson including Sonar Cube/Lint and Jacoco.

## Deployment Server

1. App Server –Tomcat 9.x
2. Web Server –WebLogic 12.c
3. DB Server –Oracle DB 12.c
4. Redis Server – Version 5.0.4
5. Kafka Server – Version 2.12
6. Docker on Linux – Version 19.03

## Further reading:

Spring Cloud: <https://spring.io/projects/spring-cloud>

Spring Cloud: <https://docs.pivotal.io/spring-cloud-services/1-5/common/index.html>

ZUUL Config: <https://cloud.spring.io/spring-cloud-netflix/multi/multi__router_and_filter_zuul.html>