May 28, 2016 By Abhisek Jana — 5 Comments (Edit)

Develop Microservices using Netflix OSS and Spring Boot

With a rapid growth in popularity, Microservices are becoming ubiquitous in any modern application architecture. It has become important to understand the concept of Microservices and how to implement. In this How to Develop Microservices using Netflix OSS and Spring Boot tutorial series, I will try to explain the fundamentals of Microservices and implement the same using sample examples.

Prerequisite:

You need to have prior knowledge of application design & development. Since I will be using `Spring Boot` all my examples will in Java, so I am expecting you to have some knowledge on Java.

How to Develop Microservices using Netflix OSS and Spring Boot is a continuation of the **An Introduction to Spring Boot series** (It has 4 parts, you can find the links at the end of each part) It provides an introduction on Spring Boot and how to get started with it. In case you are new to Spring Boot, I highly recommend to read all 4 chapters.

An Introduction to Spring Boot

Spring has been a great framework for years however it had few drawbacks. In this tutorial, An Introduction to Spring Boot we will see how Spring Boot has not only addressed the drawbacks but also supports modern software architecture. Spring Boot is around for sometime now, I had started working using spring boot 1.5 years ... Continue reading

A Developer Diary 10

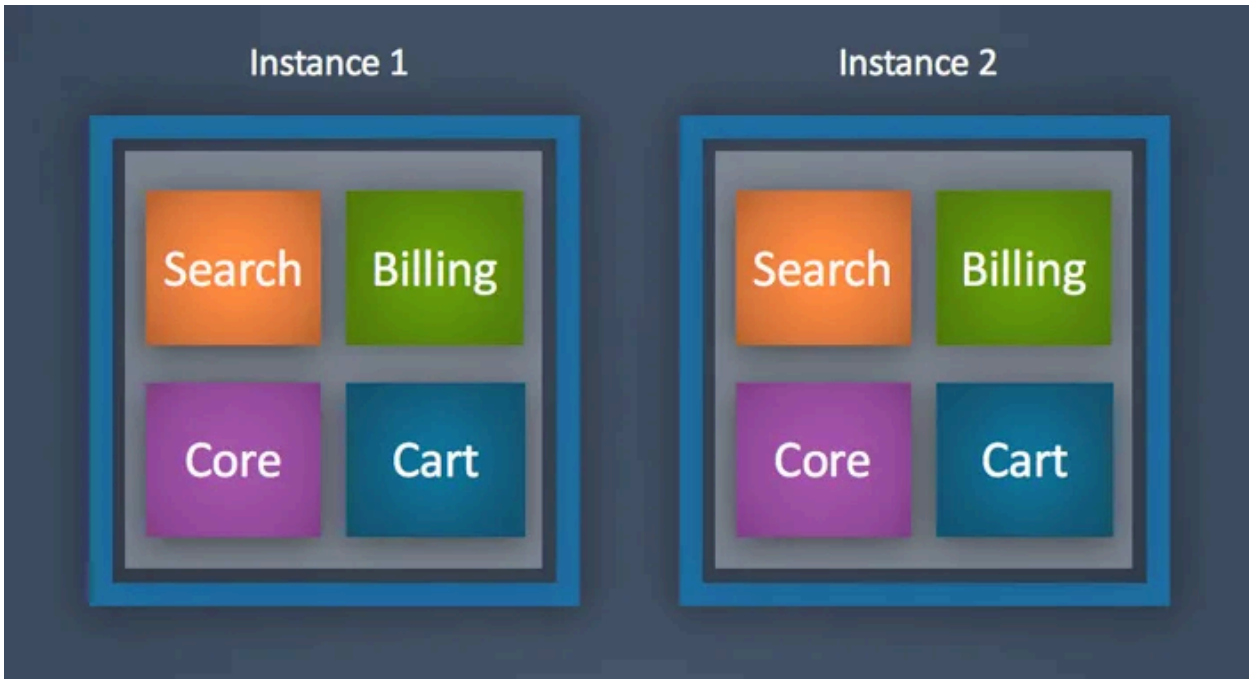
What is Microservices based Architecture and why to use it:

As you can understand from the name itself, in a Microservices based architecture you should create small `autonomous component` of any `business functions`. Let's understand this by example.

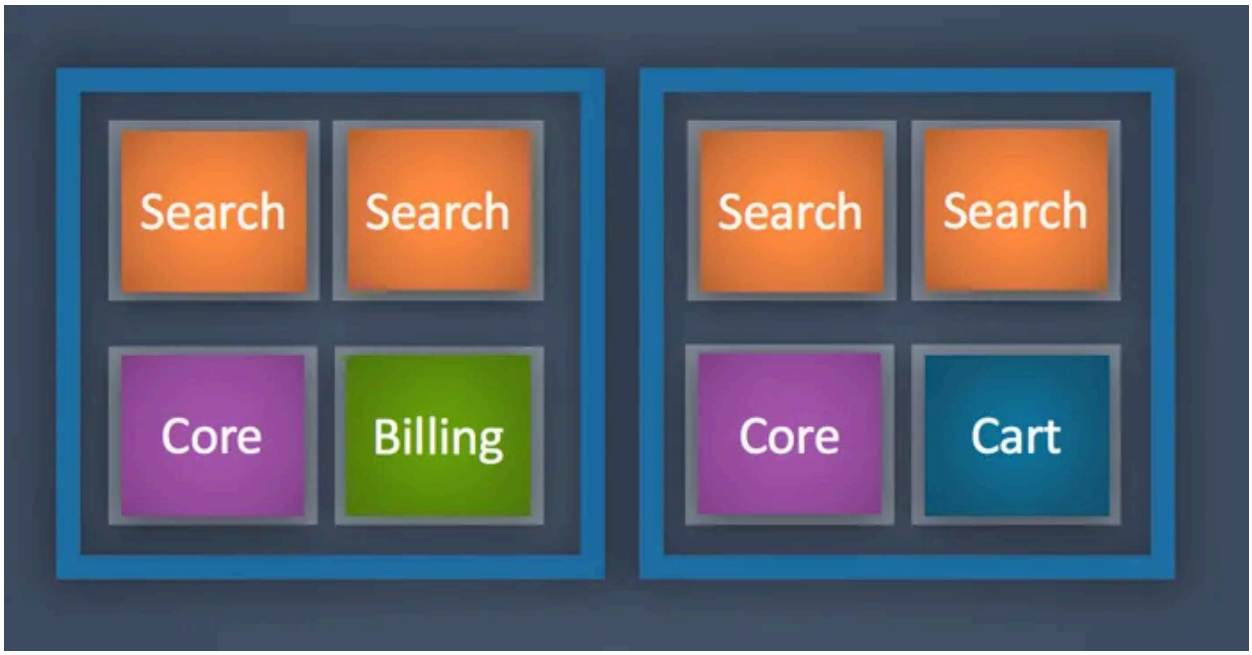
Say, you have an e-commerce application.It has thousands of products that users can search and then purchase. There would be several components, such as Product Search/Catalog, Core Function, Billing, Shipping & Shopping Cart etc. Generally we will have all these functionalities developed as one package (EAR in Java) then deployed to one or more application server in production environment. This is a typical monolithic application architecture style.

Now all the functionalities you have developed are not evenly used by the users. The e-commerce application might have 1000s of products, the search should be the most used functions. So in case if you want to increase the throughput of search, you need to deploy the entire application to another application server with all the functionalities, which is not really the objective though.

Scalability is a major obstacle in Monolithic application architecture. Also there is risk, any issue with Billing (for an example) would impact the entire application and user won't be able to use search, since all the functions are part of a single package/container/JVM. Even if, say the billing component is not working we should allow users to search products and store them in the cart.



In a Microservices based Architecture the idea is to deploy autonomous business functions such as Search, Billing, Cart as separate service. Each service can communicate using a lightweight api like JSON. So now we can have more number of Search instances. This helps us to scale the application based on functions used. Again, in case any service is down it shouldn't impact the other functions since all of them are packaged and deployed separately.



Since there isn't any clear definition of microservices available, we call only call out the most obvious characteristics, such as:

1. Modularization by services.
2. Autonomous endpoints loosely coupled with simplified lightweight API like REST.
3. Build capabilities/functions and not projects/application.
4. Decentralized Data Strategy.
5. Fault tolerance by design.
6. The final architecture can change\evolve over a period of time.
7. DevOps – Developers friendly automated infrastructure.

Next, lets find out few advantages of Microservices based design/architecture.

Advantages of Microservices:

Independent and Dynamic Scaling:

As we found out, each services can be scaled differently than others. This is a huge benefit towards cost, infrastructure and operation. In case Cloud Native

Application, you can dynamically bring more instances of any services when the demand it high. This is big.

Use of any Technology:

You can have different technologies for each of the services. In monolithic application you need to choose a single language/technology for all the functions, however you can choose any technology for the Microservices as long as they can communicate using a lightweight API like REST.

Continuous Delivery & Integration:

Microservices supports continuous delivery and integration. You can also have different small teams working on these services. This helps to develop faster and better software. Project management and tracking becomes very easy with Microservices.

Easy to Replace:

You can easily replace part of your functionality without impacting the entire application. In case of monolithic application change of any business functions imposes a big risk and we end up retesting the entire application to make sure none of the other functions are impacted. However in case of Microservices based application, you can have legacy technology in your application and you can easily replace them slowly without impacting other part of the application.

Time to market:

After the initial rollout, you can have small releases, addition of more functions/services deployed much faster then its possible to a monolithic application since the impacts are localized and far less risky.

What Spring Boot brings to the Table ?

If you look at purely from a development standpoint, Microservices is fairly simple to understand. However it gets complex when we think about adding 100s of independently scalable services. Let's talk about few of the best practices and see how spring boot helps.

Framework Standardization:

In real world, we may have many small teams (or groups) working on developing the services.Now there could be a situation where each team develops the services very differently than each other. Later when we switch/rebuild teams its difficult to understand the code since each of the services are structured in different way. Spring Boot comes as the choice of a single framework so that the structure of the each services looks similar even though the business function will be different. Spring Boot helps with standardization of the services.

Convention over Configuration:

As you have seen in the Introduction of the Spring Boot section, Spring Boot has replaced all the XML configuration of Spring by simple Annotation. However you should be able to customize the configurations using annotation as well.

Integrated Server for Development:

Spring Boot attaches a Tomcat/Jetty server with the compiled Jar using Maven/Gradle. This helps the developer to run the application easily without going to the deployment process.

Centralized Cloud based Configuration:

Since we need to replicate all the configurations across multiple instances of the services, a Centralized Cloud based Configuration is a must needed

function. Spring Boot provides a Cloud Server to manage and push the configurations to the services.

12 Factor App style Configuration:

Spring Boot also support the 12 Factor App style Configuration. If you want to read more about it, read it here—>

Centralized Logging:

You should have Centralized Logging with Microservices. Spring Boot has few Toolsets however your company might already have an enterprise solution for Centralized Logging.

3rd Party Library Support:

Spring Boot has taken a significant step and widen support for 3rd Party Open Source Library like Netflix OSS, No-SQL DB, Distributed Cache etc.

Security:

Again security is another aspect where Spring Boot has support for OAuth 2.0 based authentication.

So all these are good points, however not enough to build the microservices. The above points on how Spring Boot helps is kind of generic and applicable to any application not just Microservices. Let's talk about Netflix OSS.

What is Netflix OSS?

Netflix is very popular with their online streaming services.They had to scale the services since their customer base was increasing and also had plan for expanding across geographic. They adapted Microservices architecture style to address the scalability, fault tolerance & availability. They created many

softwares/tools to support the development which later they open sourced as part of Netflix Open Source Software (OSS).

Here is the link to Netflix OSS:

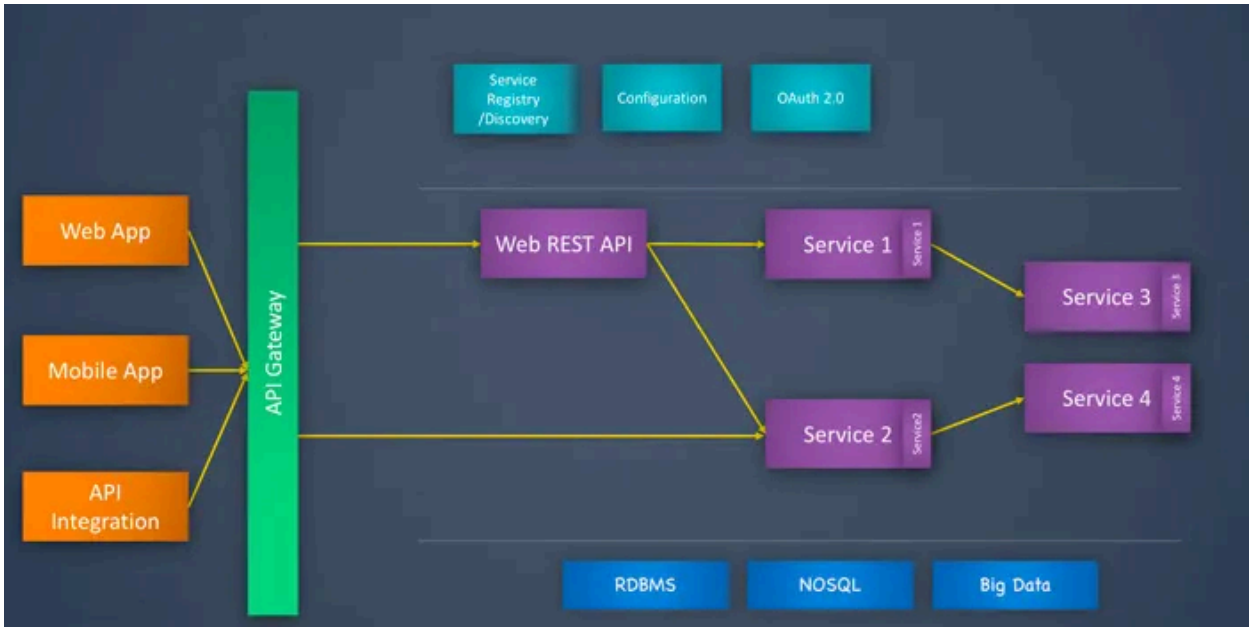
https://netflix.github.io/

There are many tools available, however we will mainly look into the tools available under “Common Runtime Services & Libraries”, such as `Eureka`, `Zuul`, `Ribbon`, `Feign` & `Hystrix` etc. We will explore more about all these tools and how they fit in the Reference Architecture.

Reference Architecture of a Microservices based Application:

It’s very important to understand the Reference Architecture of a Microservices based Application. At this point of time you may not understand each block defined here, however as we progress through the series I will explain the concept before diving deep into the actual tool.

Let me try to explain at the high-level. So an application will have many business function exposed as services, which are defined as purple box below. Each service could also have more than one instance, I have shown them vertically here. There are different UI strategy that we will discuss later, however for now lets assume that all of our UI elements (HTML, CSS, JS etc) are inside the `Web REST API` service which is the UI point of entry.

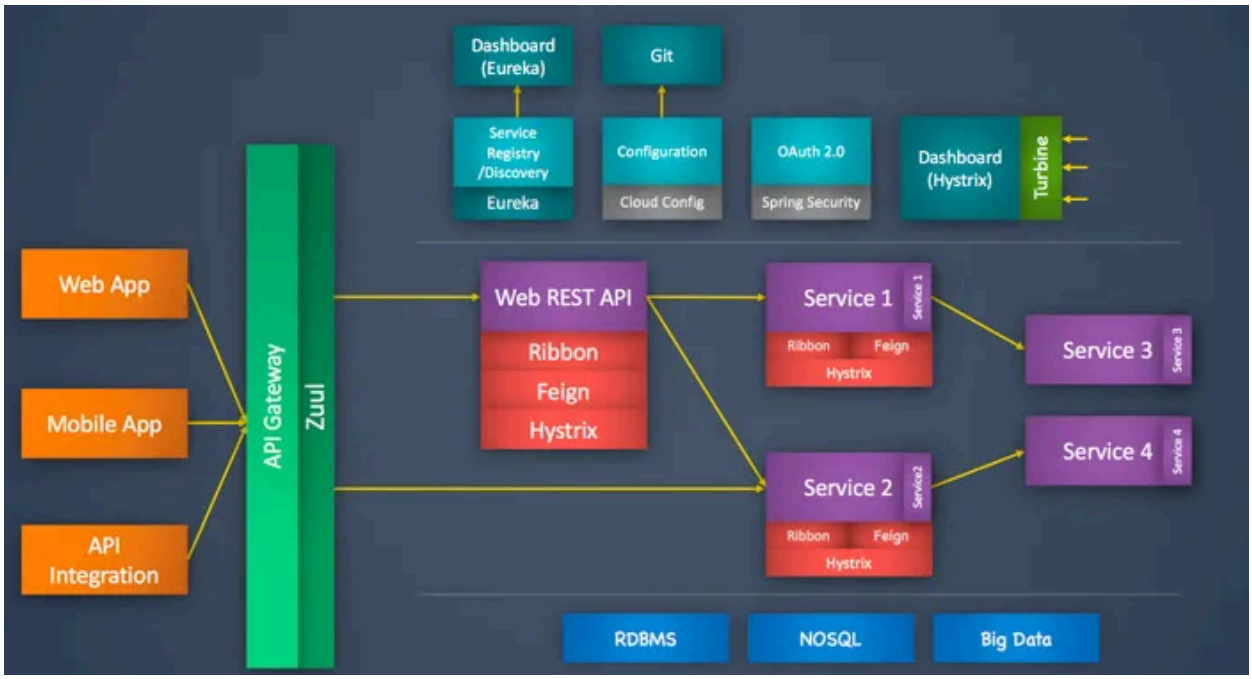


We would need an `API Gateway` to have a single URL at the client end and multiple URLs at the service end. The API Gateway can control the traffic (Server side routing) and route service invocation based on the client (Web App/ Mobile App/ API Integration).

We would also need a `service discovery` so that the system can be aware of all the services and routing can use the information during runtime. In this way you can dynamically add/remove service instances. Centralized configuration and OAuth 2.0 are the other features needed.

Microservices using Spring Boot:

I have added the `Netflix OSS` components to our Reference Architecture of a Microservices based Application we saw earlier. You can see `Zuul` for the API Gateway, `Eureka` for Service Discovery. There are other components, `Ribbon` (Client Side Routing), `Feign` (Declarative REST Client), `Hystrix` (Latency and Fault Tolerance) we will be discussing about them in subsequence parts.



Conclusion:

We have learnt about Microservices, its advantages, discussed on Netflix OSS and its components. We also saw a reference architecture of Microservices based Application and how Netflix OSS can be used. Next part will be learning about how to configure and use Service Discovery using Eureka.

Related

An Introduction to Spring Boot

In "Spring Boot"

How to deploy Spring Boot application in IBM Liberty and WAS 8.5

In "Spring Boot"

How to Create Spring Boot Application Step by Step

In "Spring Boot"

Filed Under: Microservice, Spring Boot | Tagged With: Architecture, Eureka, Feign, Hystrix, Microservices, Monolithic, Ribbon, Spring Boot, Zuul

Comments

Roopali says
October 25, 2016 at 5:11 am

(Edit)

Pretty neat.Thanks!

How about service to service communication? Does that happen through Zuul or via Eureka directly?

Have you published next post?

Reply

Sivakumaran says
November 6, 2016 at 3:31 am

(Edit)

Still its through Eureka. Though its service to service or client to service , still a particular service has to be discorded by client to client service. Since every service is running independently and no one know each other so that it has to go through Eureka.

Thanks
Siva

Reply

Dhiren Hamal says
April 2, 2017 at 6:48 am

(Edit)

Now I am pretty much confident about microservice architecture. Thanks for the great tutorial.

Reply

Tin says
April 18, 2017 at 6:31 pm

(Edit)

Thank you for the great introduction to microservice architectures. Eagerly anticipating your next post on the topic.

Reply

RK says
June 13, 2017 at 2:21 pm

(Edit)

Thanks for the great post! Very informative

Reply

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment