Guided Learning Java Microservices

Description: This document is a guided learning path for Java Microservices using Netflix OSS and Spring Cloud

Document Control

Version	1.0
Created by	Chand Ali
Updated by	Chand Ali
Reviewed by	Vinil M
Date	December 2019

Table of Contents

1. Pre-Requisites

- Basic Knowledge of Spring and Spring Boot.
- To know more about SpringBoot refer below Tutorials .
  - Spring
  - SpringBoot Quick Start

2. Hardware-Software Requirements

The learner will require a computer (PC / Laptop) running Windows 7 and above with at least 4 GB of RAM. Additionally, the following is also required:-

- Jdk 1.8
- SpringTool Suite
- Maven
- AWS Free Tier Accout

3. Microservices Architecture

What are Microservices?

- There is presently a lot of hype. This hype (microservices) is at the peak of inflated expectations in the Gartner Hype Cyle Model.
- Best described as
  - An Architecure Style
  - An alternative to more traditional 'monolithic" applications
  - Decomposition of Single system into a suite of small services, each running as independent processes and intercommunicating via open Protocols (HTTP/Rest or messaging)

Definition form Experts:
Developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often HTTP resource API –**Martin Fowler**

Fined-grained SOA -Adrian Cockcroft -**Netflix**

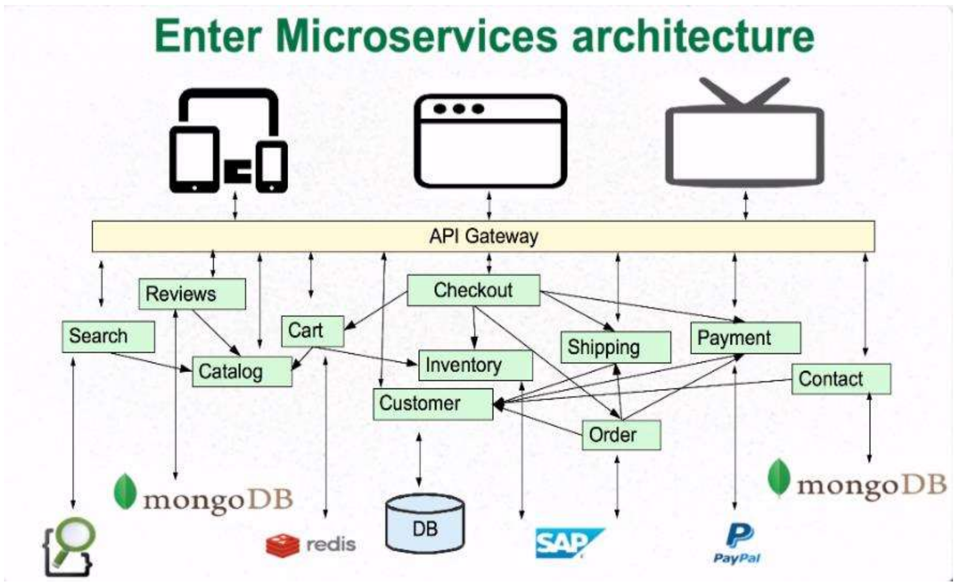
Microservices -Working Definition

- Composing a single application using a suite of small services
  - rather than single, monolithic application
- Each running as independent process
  - not merely modules/componennts with a single executable
- Intercommunication via open protocols
  - Like HTTP/REST or messaging
- Seperatley written, deployed, scaled and maintained
  - potentially in different languages
- Services encapsulate business capability
  - rather than language constructs (classes, packages) as primary way to encapsulate.
- Services are independently replacable and upgradable

Microserices are not

- The same as SOA
  - SOA is about interating various enterprise applications.Microservices are mainly about decomposing single applications
- A Silver Bullet
  - Microservices approach involves drawbacks and risks
- New ! You may be using microservices now and not know it

Sample Architechure:



Video : Explaining Microservices Architechure

4. Principles of Microservices

Now let's examine "must have" principles of a microservice.

- **Single responsibility principle :** The single responsibility principle is one of the principles defined as part of the SOLID design pattern. It implies that a unit, either a class, a function, or a microservice, should have one and only one responsibility.

At no point of time, one microservice should have more than one responsibility.

- **Built around business capabilities : Microservices should focus on certain business function** and ensure that it helps in getting things done. A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose.

  This is often the constraint when we design monolithic applications where we try to solve multiple business solutions with some compromises in some areas. Microservices enable you to choose whats best for the problem in hand.

- **You build it, you own it! :** Another important aspect of such design is related to responsibilities pre-and-post development. In large organization, usually one team develops the app location, and after some knowledge transfer sessions it hands over the project to maintenance team. In microservices, the team which build the service – owns it, and is responsible for maintaining it in future.
  You build it, you own it !!

  This brings developers into contact with the day-to-day operation of their software and they better understand how their built product is used by customers in real world.

- **Design for Failure :** A microservice shall be designed with failure cases in mind. What if the service fails, or go down for some time. These are very important questions and must be solved before actual coding starts – to clearly estimate **how service failures will affect the user experience**.

  Fail fast is another concept used to build fault-tolerant, resilient systems. This philosophy advocates systems that expect failures versus building systems that never fail. Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service.

- **Infrastructure Automation :** Preparing and building infrastructure for microservices is another very important need. **A service shall be independently deployable** and shall bundle all dependencies, including library dependencies, and even execution environments such as web servers and containers or virtual machines that abstract physical resources.

  In traditional application developments, we build a WAR or an EAR, then deploy it into a JEE application server, such as with JBoss, WebLogic, WebSphere, and so on. We may deploy multiple applications into the same JEE container. In ideal scenario, in the microservices approach, each microservice will be built as a fat Jar, embedding all dependencies and run as a standalone Java process.

5. Benefits of Microservices :

Microservices offer a number of benefits over the traditional multi-tier, monolithic architectures. Let's list down them:

- Polygot programming - With microservices, architects and developers can choose fit for purpose architectures and technologies for each microservice (polyglot architecture). This gives the flexibility to design better-fit solutions in a more cost-effective way.

- Ability to Experiment - As services are fairly simple and smaller in size, enterprises can afford to experiment new processes, algorithms, business logic, and so on. It enables enterprises to do disruptive innovation by offering the ability to experiment and fail fast.

- Scalability - Microservices enable to implement selective scalability i.e. each service could be independently scaled up or down and cost of scaling is comparatively less than monolithic approach.

- Easy Rollouts and Rollbacks : Microservices are self-contained, independent deployment modules enabling the substitution of one microservice with another similar microservice, when second one is not performing as per our need. It helps in taking right buy-versus-build decisions which are often the challenge for many enterprises.

- Supports organic Systems : Microservices help us build systems that are organic in nature (Organic systems are systems that grow laterally over a period of time by adding more and more functions to it). Because microservices are all about independently manageable services – it enable to add more and more services as the need arises with minimal impact on the existing services.

- No Language/Framework Lock : Technology changes are one of the barriers in software development. With microservices, it is possible to change or upgrade technology for each service individually rather than upgrading an entire application.

- Mutiple flavours of Same service can be used : As microservices package the service runtime environment along with the service itself, this enables having multiple versions of the service to coexist in the same environment.

- Easire to Scale staff : Finally, microservices also enable smaller, focused agile teams for development. Teams will be organized based on the boundaries of microservices. Flavours Amazon Two-PIZAA Rule .

6. Java Microservices Technology Stack

This section describes in detail the benefits and roles of each technology and and their purpose. Each technology description will follow the template below.

Name	**Name of the Technology**
Description	An introductory description of the technology.
Role/Purpose	Describe the role of the technology and how the current architecture can benefit with its inclusion
Additional Considerations	Describe the advantages of the technology and its benefits associated with its functionalities.
DownSides (if applicable)	Describe in which scenarios the technology isn't best suited

Spring Cloud:

Name	Spring Cloud
Description	Open-source tools to implement common patterns in distributed systems.
Role/Purpose	Spring Cloud provides a set of tools as dependencies that can be integrated with a Spring solution. These tools provide some out-of-the-box functionalities such as configuration management, service discovery, circuit breakers, intelligent routing, etc.
Additional Considerations	A distributed system such as a Microservice leads to boilerplate patterns. Instead of manually implementing those patterns, Spring Cloud provides a set of services as dependencies that can be leveraged. These patterns are open-source, commonly used in similar architectures and supported by major players in the Microservices scene such as Netflix

Eureka:

Name	Eureka
Description	Eureka is a REST-based service used for locating services (Service Discovery)
Role/Purpose	Eureka, or Eureka Server, lives on the premise of registering and discovering services for the purposes of load-balancing, dynamic configurations and failover of middle-tier services.
Additional Considerations	Setup: easy setup and integration with Spring Cloud; Health: provides health checking metrics; Monitoring: uses servo (interface to display application metrics) to track performance in client and server sides, monitoring and alerting

Hystrix:

Name	Hystrix
Description	Library for enabling resilience in micoservices. It provides more control in the interaction between services by enabling fault tolerance and latency tolerance.
Role/Purpose	Hystrics wraps calls to external dependencies and monitors metrics in real time. Invokes failover method when encountering exceptions, timeouts, thread pool exhaustion etc. Hystrix helps to improve the overall resilience of the system by isolating the failure services and stopping the cascading effect of failures.
Additional Considerations	• **Latency and Fault Tolerance:** proves protection and control over latency and failure from dependencies, most commonly those accessed over a network. It helps in stopping cascading failures and enables developers to fail fast and rapidly recover, or fallback and gracefully degrade • **Thread and semaphore isolation with circuit breakers:** Each dependency and rejects requests (instead of queuing requests) if the thread-pool becomes exhausted, it provides circuit-breaker

	functionality so all requests can be stopped to a dependency. Fallback logic can also be implemented, e.g.: when a request fails, is rejected, timed-out or short-circuited • **Real-time Operations:** Watch changes on properties and on services take effect in real-time monitoring and configuration changes • **Concurrency:** Hystrix uses the bulkhead pattern to isolate dependencies and limit concurrent access. Separate thread pools are used per dependency so that concurrent requests are constrained.

Zull:

Name	**Zull**
Description	Zuul is a JVM based router and server side load balancer by Netflix and Spring Cloud has a nice integration with an embedded Zuul proxy
Role/Purpose	Zuul makes use of a wide range of filters to dynamically route requests and enables developers to apply functionality to edge services such as Authentication and Security; Insight and Monitoring; Dynamic Monitoring; Stress Testing; Load Shedding; Static Response Handling.
Additional Consederations	• **Authentication and Security:** Identifying authentication requirements for each resource and rejecting requests that do not satisfy them • **Insights and Monitoring:** Tracking key data and statistics at the edge service in order to present a real-time and accurate view of the services in production • **Dynamic Routing**: Dynamically routing requests to different backend clusters as needed • **Stress Testing:** Gradually increasing the traffic to a cluster in order to gauge performance • **Load Shedding:** Allocating capacity for each type of request and dropping requests that go over the limit • **Static Response handling:** Building some responses directly at the edge services instead of forwarding them to an internal cluster.

Ribbon:

Name	**Ribbon**
Description	Ribbon is a client-side load balancer that gives a lot of control over the behavior of HTTP and TCP clients.
Role/Purpose	Some of the load balancing strategies offered are listed below: • Simple Round Robin LB • Weighted Response Time LB • Zone Aware Round Robin LB • Random LB

Additional Considerations	Setup: easy setup and integration with Spring Cloud Health: provides health checking metrics Monitoring: uses servo interface to display application metrics) to track performance in client and server sides, monitoring and alerting

7. Examples to Practice(Every Example is supported by Video and Code)

- Spring Cloud Config Server Example :
  - Spring Cloud Config Server Using Git Simple Example
- Spring Cloud Config Server using Git Example :
  - Spring Cloud Config Server Using Git Simple Example
- Service Discovery using Eureka Examples :
  - http://www.baeldung.com/spring-cloud-netflix-eureka
  - Microservice Registration and Discovery with Spring cloud using Netflix Eureka- Part 1.
  - Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 2.
  - Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 3.
  - Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 4.
- Client Side Load balancing using Ribbon Example :
  - http://www.baeldung.com/spring-cloud-rest-client-with-netflix-ribbon
  - Spring Cloud- Netflix Eureka + Ribbon Simple Example
- Example using Standard Client Interface Feign:
  - Spring Cloud- Netflix Feign REST Client Simple Example
- Example for Hystrix :
  - http://www.baeldung.com/introduction-to-hystrix
- CircuitBreaker Example :
  - Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example
- Api Gateway Demo using Zull and Eureka
  - Spring Cloud- Netflix Zuul +Eureka Simple Example
  - Spring Cloud – Create API Gateway with Netflix Zuul

Other Refrence Document: Microservice Architecure with Spring Cloud

8. Use cases

- If we try searching Google about the adoption of this architecture, we can see several articles floating around on the successful implementation of it. Some of the products and companies who had implemented it:
- Netflix (Ref)
- eBay (Ref)
- Amazon (Ref)
- Microservices for Digital Transformation - case studies across ERP, Entertainment, Finance, Search, Auctions, and Cloud Industries