Why Spring 2?

1.**Java 8** is minimum version
2.**Tomcat** version **8.5** is minimum
3.Hibernate version **5.2** is minimum
4.**Gradle** version **3.4** is minimum
5.Added SpringBoot Starters for **WebFlux** and reactive support for **Cassandra**, **MongoDB** and **Redis**.
6.**AutoConfiguration**
7. Actuator Endpoint change:

   Before 2.*: http://localhost:8080/business-customer/profile/**env** will give the details.
   From 2.*: http://localhost:8080/business-customer/profile/**actuator/env** will give the details.
8. **Endpoint properties** in **application.properties** (to enable all endpoints)
   management.endpoints.web.exposure.include=*
   management.endpoints.web.exposure.exclude=loggers
9. **Connection Pool** by default:
   Before 2.*: tomcat CP

   After 2.*: **HikariCP** (from SpringBoot 2.* You don't need to add HikariCP dependency and its config bean creation and its properties changes.)
10. **Migration**: https://spring.io/blog/2018/03/12/upgrading-start-spring-io-to-spring-boot-2

What is Spring Boot?

Spring Boot is another Java framework from Spring umbrella which aims to simplify the use of Spring Framework for Java development. It removes most of the pain associated with dealing with Spring e.g. a lot of configuration and dependencies and a lot of manual setups.

Why should you use it? Well, Spring Boot not only provides a lot of convenience by auto-configuration a lot of things for you but also improves the productivity because it lets you focus only on writing your business logic.

What is Spring Boot?

What is Spring Boot?

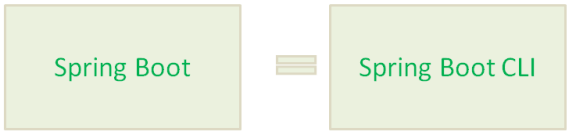
1. Starter dependency
This feature aggregates common dependencies together. For example, if you want to develop Spring MVC based RESTful services then instead of including Spring MVC JAR and Jackson JAR file into classpath you can just specify spring-boot-web-starter and it will automatically download both those JAR files. Spring Boot comes with many such starter dependencies to improve productivity.

2. Auto-Configuration
This is another awesome features of Spring Boot which can configure many things for you. For example, If you are developing Spring web application and Thymeleaf.jar is present on the classpath then it can automatically configure Thymeleaf template resolver, view resolver, and other settings. A good knowledge of auto-configuration is required to become an experienced Spring Boot developers.

3. Spring Initializer
A web application which can create initial project structure for you. This simplifies initial project setup part.

4. Spring Actuator
This feature provides a lot of insights of a running Spring boot application. For example, you can use Actuator to find out which beans are created in Spring's application context and which request path are mapped to controllers.

5. Spring CLI
This is another awesome feature of Spring Boot which really takes Spring development into next level. It allows you to use Groovy for writing Spring boot application which means a lot more concise code.



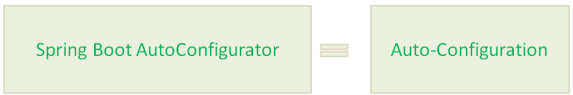
Why we need Spring Boot?

1. Spring Framework aims to simplify Java Applications Development.
2. Spring Boot Framework aims to simplify Spring Development.

Spring Boot Components
Spring Boot Framework has the following components:

1. Spring Boot Starter
2. Spring Boot AutoConfigurator
3. Spring Boot Actuator
4. Spring Boot CLI
5. Spring Boot Initilizr

What is auto-configuration in Spring boot? how does it help? Why Spring Boot is called opinionated?
it automatically configures a lot of things based upon what is present in the classpath.
Spring Boot AutoConfigurator is used by Spring Boot Framework to provide “Auto-Configuration”.



For example, it can configure JdbcTemplate if its present and a DataSource bean are available

in the classpath. It can even do some basic web security stuff if Spring security is present in the classpath.

Anyway, the point is auto-configuration does a lot of work for you with respect to configuring beans, controllers, view resolvers etc, hence it helps a lot in creating a Java application.

Now, the big questions come, why it's considered opinionated? Well because it makes a judgment on its own. Sometimes it imports things which you don't want, but don't worry, Spring Boot also provides ways to override auto-configuration settings.

It's also disabled by default and you need to use
either @SpringBootApplication or @EnableAutoConfiguration annotations on the Main class to enable the auto-configuration feature. See Spring Boot Essentials for learning more about them.

What is Spring Boot Starter? starter dependency in Spring Boot
Spring Boot Starters are just JAR Files. They are used by Spring Boot Framework to provide "Auto-Dependency Resolution".

As the name suggests, starter dependency deal with dependency management.
After examining several Spring-based projects Spring guys notice that there is always some set of libraries which are used together e.g. Spring MVC with Jackson for creating RESTful web services. Since declaring a dependency in Maven's pom.xml is the pain, they combined many libraries into one based upon functionality and created this starter package.

This not only frees you from declaring many dependencies but also frees you from compatibility and version mismatch issue. Spring Boot starter automatically pulls compatible version of other libraries so that you can use them without worrying about any compatibility issue.

What is the difference between @SpringBootApplication and @EnableAutoConfiguration annotation?
Even though both are essential Spring Boot application and used in the Main class or Bootstrap class there is a subtle difference between them. The @EnableAutoConfiguration is used to enable auto-configuration but @SpringBootApplication does a lot more than that.

It also combines @Configuration and @ComponentScan annotations to enable Java-based configuration and component scanning in your project.

The @SpringBootApplication is in fact combination
of @Configuration, @ComponentScan and @EnableAutoConfiguration annotations. You can also check Spring Boot MasterClass to learn more about this annotation and it's used.

What is Spring Initializer? why should you use it?
It's nothing but a web application which helps you to create initial Spring boot project structure and provides Maven or Gradle build file to build your code.

Spring Boot Initilizr is a Spring Boot tool to bootstrap Spring Boot or Spring Applications very easily.
Spring Boot Initilizr comes in the following forms:

Spring Boot Initilizr With Web Interface
Spring Boot Initilizr With IDEs/IDE Plugins
Spring Boot Initilizr With Spring Boot CLI
Spring Boot Initilizr With ThirdParty Tools

What is Spring Actuator? What are its advantages?
Spring Boot Actuator is used by Spring Boot Framework to provide "Management EndPoints" to see Application Internals, Metrics etc.

Spring Actuator is another cool Spring Boot feature which allows seeing inside a running application.

Yes, you read it correctly. It allows you to see inside an application. Since Spring Boot is all about auto-configuration it makes debugging difficult and at some point in time, you want to know which beans are created in Spring's Application Context and how Controllers are mapped. Spring Actuator provides all that information.

It provides several endpoints e.g. a REST endpoint to retrieve this kind of information over the web. It also provides a lot of insight and metrics about application health e.g. CPU and memory usage, number of threads etc.

It also comes with a remote shell which you can use to securely go inside Spring Boot application and run some command to expose the same set of data. You can even use JMX to control this behavior at runtime.

Btw, it's important to secure your Spring Actuator endpoints because it exposes a lot of confidential information and a potentially dangerous one-two. For example, by using /showdown endpoint you can kill a Spring Boot application.

What is Spring Boot CLI? What are its benefits?
Spring Boot CLI is a command line interface which allows you to create Spring-based Java application using Groovy. Since it's used Groovy, it allows you to create Spring Boot application from the command line without ceremony e.g. you don't need to define getter and setter method, or access modifiers, return statements etc.

It's also very powerful and can auto-include a lot of library in Groovy's default package if you happen to use it.

For example, if you use JdbcTempalte, it can automatically load that for you.
No Semicolons

- No Public and private access modifiers
- No Imports(Most)

- No “return” statement
- No setters and getters
- No Application class with main() method(It takes care by SpringApplication class).
- No Gradle/Maven builds.
- No separate HTTP Servers.

Where do you define properties in Spring Boot application?
You can define both application and Spring boot related properties into a file called application.properties. You can create this file manually or you can use Spring Initializer to create this file, albeit empty.

You don't need to do any special configuration to instruct Spring Boot load this file. If it exists in classpath then Spring Boot automatically loads it and configure itself and application code according.

Can you change the port of Embedded Tomcat server in Spring boot? If Yes, How?
Yes, we can change the port of Embedded Tomcat Server in Spring Boot by adding a property called server.port in the application.properties file.

What is the difference between an embedded container and a WAR?
The main difference between an embedded container and a WAR file is that you can Spring Boot application as a JAR from the command prompt without setting up a web server. But to run a WAR file, you need to first set up a web server like Tomcat which has Servlet container and then you need to deploy WAR there.

What embedded containers does Spring Boot support?
Spring Boot support three embedded containers: Tomcat, Jetty, and Undertow. By default, it uses Tomcat as embedded containers but you can change it to Jetty or Undertow.

What are some common Spring Boot annotations?
Some of the most common Spring Boot annotations
are @EnableAutoConfiguration, @SpringBootApplication, @SpringBootConfiguration, and @SpringBootTest.

@EnableAutoConfiguration is used to enable auto-configuration on Spring Boot
@SpringBootApplication is used on the Main class to allow it to run a JAR file.
@SpringBootTest is used to run unit test on Spring Boot environment.

Can you name some common Spring Boot Starter POMs?
Some of the most common Spring Boot Start dependencies or POMs are spring-boot-starter, spring-boot-starter-web, spring-boot-starter-test. You can use spring-boot-starter-web to enable Spring MVC in Spring Boot application.

Can you control logging with Spring Boot? How?
Yes, we can control logging with Spring Boot by specifying log levels on application.properties file. Spring Boot loads this file when it exists in the classpath and it can be used to configure both Spring Boot and application code.

Spring Boot uses Commons Logging for all internal logging and you can change log levels by adding following lines in the `application.properties` file

```
logging.level.org.springframework=DEBUG
logging.level.com.demo=INFO
```

Can we disable the default web server in the Spring Boot application?
The major strong point in Spring is to provide flexibility to build your application loosely coupled. Spring provides features to disable the web server in a quick configuration. Yes, we can use the application.properties to configure the web application type, i.e.
spring.main.web-application-type=none.

26) How can you enable auto reload of application with Spring Boot?
You can enable auto-reload/LiveReload of spring boot application by adding the **spring-boot-devtools** dependency in the **pom.xml** file.

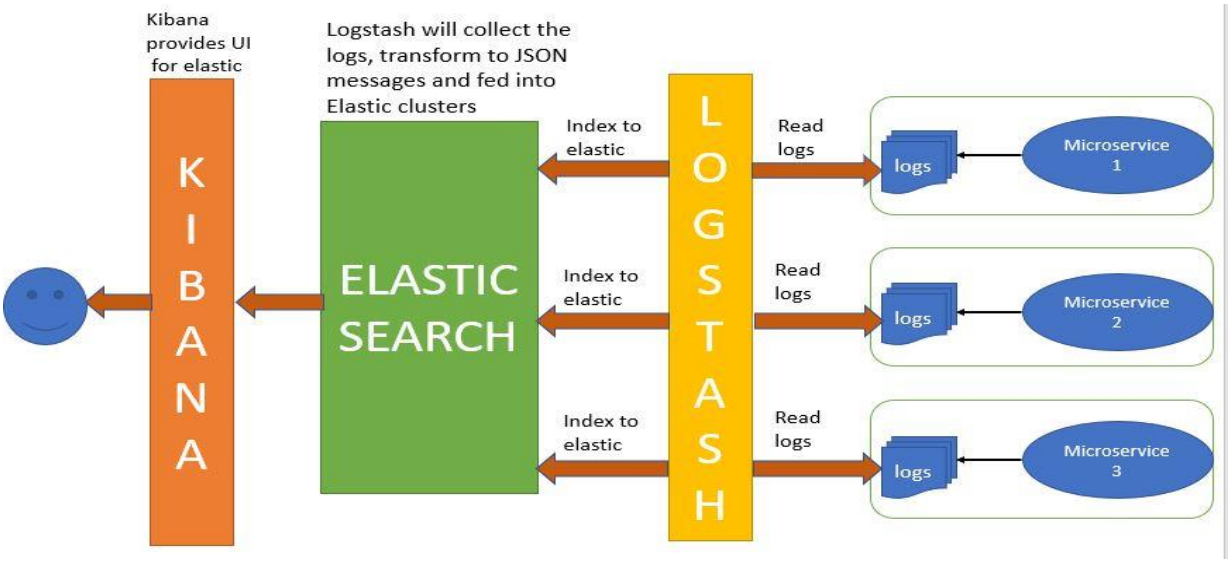
```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <optional>true
</dependency>
```

Note: please restart your application for immediate effects.

What is the difference between RequestMapping and GetMapping in Spring Boot?

Both @GetMapping and @RequestMapping are annotations for mapping HTTP GET requests onto specific handler methods in Spring boot. @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping. @GetMapping is the newer annotation.

What is ELK stack?How to use it with Spring Boot?
The ELK Stack consists of three open-source products - Elasticsearch, Logstash, and Kibana from Elastic.



- Elasticsearch is a NoSQL database that is based on the Lucene search engine.

 Logstash is a log pipeline tool that accepts inputs from various sources, executes different transformations, and exports the data to various targets. It is a dynamic data collection pipeline with an extensible plugin ecosystem and strong Elasticsearch synergy
 Kibana is a visualization UI layer that works on top of Elasticsearch.These three projects are used together for log analysis in various environments. So Logstash collects and parses logs, Elastic search indexes and store this information while Kibana provides a UI layer that provide actionable insights.
Spring Boot + ELK stack

How to reload my changes on Spring Boot without having to restart server?
This can be achieved using DEV Tools . With this dependency any changes you save, the embedded tomcat will restart. Spring Boot has a Developer tools (DevTools) module which helps to improve the productivity of developers. One of the key challenge for the Java developers is to auto deploy the file changes to server and auto restart the server. Developers can reload changes on Spring Boot without having to restart my server. This will eliminates the need for manually deploying the changes every time. Spring Boot doesn't have this feature when it has released it's first version. This was a most requested features for the developers. The module DevTools does exactly what is needed for the developers. This module will be disabled in the production environment. It also provides H2-database console for better testing the application. The following dependency is used

How to run Spring boot application to custom port?

In order to run a spring boot application on a custom port you can specify the port in application.properties.
server.port=8090

Have you written Test cases using Spring Boot?
Spring Boot provides the @SpringBootTest for writing Unit Test Cases
Spring Boot Unit Test Simple Example

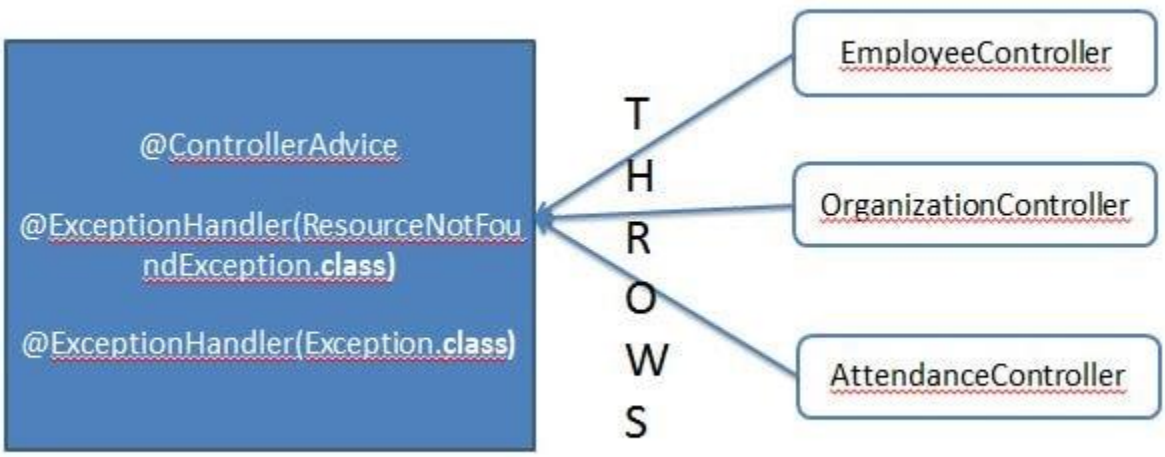
What is YAML?
YAML is a human-readable data serialization language. It is commonly used for configuration files.
Compared to properties file, YAML file is much more structured and less confusing in case we want to add complex properties in the configuration file. As can be seen YAML has hierarchical configuration data.
Use YAML properties in Spring Boot

How to implement Pagination and Sorting with Spring Boot?
Using Spring Boot achieving pagination is very simple. Using the Spring Data-JPA this is achieved passing pageable org.springframework.data.domain.Pageable to the repository methods.
Spring Boot pagination explanation

What is Spring Batch? How do you implement it using Spring Boot?
Spring Boot Batch provides reusable functions that are essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management. It also provides more advanced technical services and features that will enable extremely high-volume and high performance batch jobs though optimization and partitioning techniques. Simple as well as complex, high-volume batch jobs can leverage the framework in a highly scalable manner to process significant volumes of information.
Spring Boot + Batch

What is Spring Profiles? How do you implement it using Spring Boot?
Spring Profiles allows users to register beans depending on the profile(dev, test, prod etc). So when the application is running in DEVELOPMENT only certain beans can be loaded and when in PRODUCTION certain other beans can be loaded. Suppose our requirement is that the Swagger documentation be enabled only for the QA environment and disabled for all others. This can be done using Profiles. Spring Boot makes using Profiles very easy.
Spring Boot + profiles

How to implement Exception Handling using Spring Boot?
Spring provides a very useful way **to** handle exceptions using ControllerAdvice.
We will be implementing a ControlerAdvice class which will handle all exceptions thrown by the controller class.
Spring Boot Exception Handling



What is caching? Have you used any caching framework with Spring Boot?
A cache is an area of local memory that holds a copy of frequently accessed data that is otherwise expensive to get or compute. Have used Hazelcast for caching.
Spring Boot + Hazelcast Example

How to upload a file using Spring?

Spring Boot + File Upload Example

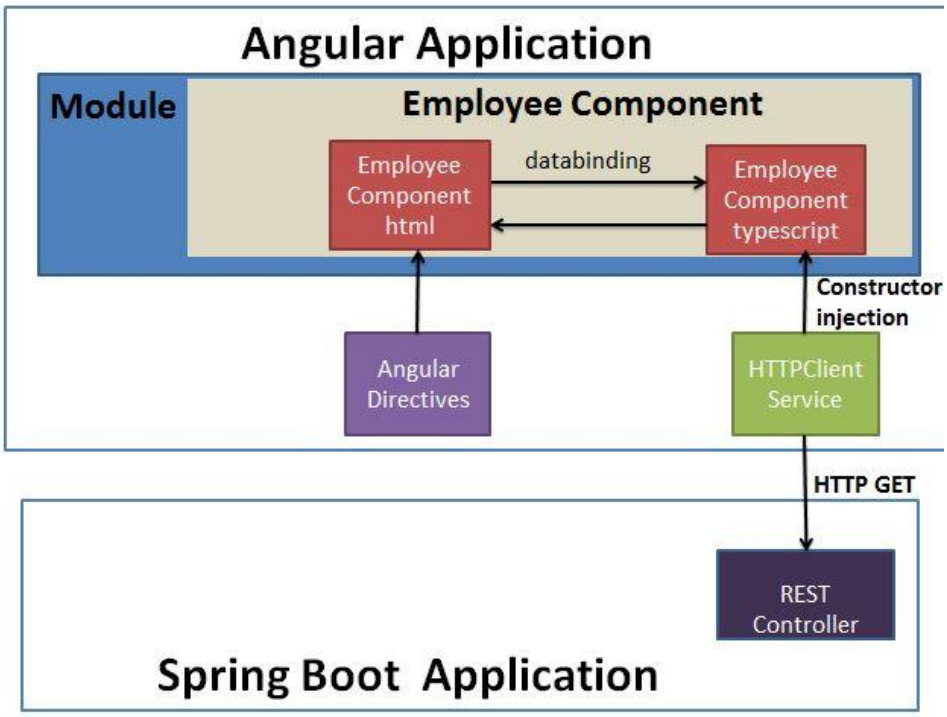
How to implement interceptors with Spring Boot?

Using Spring MVC HandlerInterceptor with Spring Boot

How to use schedulers with Spring Boot?

Spring Boot Task Scheduler Example

How to develop a full stack application using Spring Boot and Angular?
In full stack application we expose the back end point to get the data. This data can then be used by any application or device as per the need. In future even if another front end device is to be used, there will not be much change and the new device will need to consume these end points.



Angular 7 + Spring Boot Tutorials

Which all starter maven dependencies have you used?

Have used different starter dependencies like spring-boot-starter-activemq dependency, spring-boot-starter-security dependency, spring-boot-starter-web dependency.
This helps in adding less number of dependencies and also in reducing version conficts.
Spring Boot Security example and explanation

What is CSRF attack? How to enable CSRF protection against it?

CSRF stands for Cross-Site Request Forgery. It is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
Spring Boot Security - Enabling CSRF Protection

How to use Form Login Authentication using Spring Boot?

Spring Boot Form Security Login Hello World Example

What is OAuth2? How to implement it using Spring Boot?

Spring Boot + OAuth2 implementation

What is GZIP?How to implement it using Spring Boot?

gzip is a file format and a software application used for file compression and decompression.
Spring Boot + GZIP Compression

Have you used any integration framework with Spring Boot?

Have integrated Apache Camel with Spring Boot. Made use of Apache Camel Spring Boot starter dependency. Spring Boot +Apache Camel

What is Apache Freemarker? When to use it instead of JSP? How to integrate it with Spring Boot?

JSP is tailor made for Web pages, Freemarker template is a more generic templating language - it can be used to generate html, plain text, emails, etc.
Spring Boot + FreeMarker Example

When will you use WebSockets? How to implement it using Spring Boot?

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.

- **WebSocket are bi-directional -** Using WebSocket either client or server can initiate sending a message.
- **WebSocket are Full Duplex -** The client and server communication is independent of each other.
- **Single TCP connection -** The initial connection is using HTTP, then this connection gets upgraded to a socket based connection. This single connection is then used for all the future communication
- **Light -** The WebSocket message data exchange is much lighter compared to http.

Spring Boot + WebSockets Example

What is AOP? How to use it with Spring Boot?

During software development, functions that span multiple points of an application are called cross-cutting concerns. These cross-cutting concerns differ from the application's main business logic. Hence ,separating these cross-cutting concerns from the business logic is where aspect-oriented programming (AOP) comes into picture.
Spring Boot + AOP Example

What is Apache Kafka? How to integrate it with Spring Boot?

Apache Kafka is a distributed publish-subscribe messaging system. It is a scalable, fault-tolerant, publish-subscribe messaging system which enables us to build distributed applications. It is an Apache Top Level project. Kafka is suitable for both offline and online message consumption.
Spring Boot + Apache Kafka Example

How can we monitor all the Spring Boot Microservices?

Spring Boot provides actuator endpoints to monitor metrics of individual microservices. These endpoints are very helpful for getting information about applications like if they are up, if their components like database etc are working good. But a major drawback or difficulty about using actuator enpoints is that we have to individually hit the enpoints for applications to know their status or health. Imagine microservices involving 50 applications, the admin will have to hit the actuator endpoints of all 50 applications. To help us deal with this situation, we will be using open source project located at https://github.com/codecentric/spring-boot-admin.
Built on top of Spring Boot Actuator, it provides a web UI to enable us visualize the metrics of multiple applications.
Spring Boot Admin

Spring Annotation List:

- Spring framework implements and promotes the principle of control inversion (IOC) or dependency injection (DI) and is in fact an IOC container.

- Traditionally, Spring allows a developer to manage bean dependencies by using XML-based configuration.
- There is an alternative way to define beans and their dependencies. This method is a Java-based configuration.
- Unlike the XML approach, Java-based configuration allows you to manage bean components programmatically. That's why Spring annotations were introduced.

Spring Annotations List

1. @Configuration: Used to indicate that a class declares one or more @Bean methods. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.
2. @Bean: Indicates that a method produces a bean to be managed by the Spring container. This is one of the most used and important spring annotation. @Bean annotation also can be used with parameters like name, initMethod and destroyMethod.

- name – allows you give name for bean
- initMethod – allows you to choose method which will be invoked on context register
- destroyMethod – allows you to choose method which will be invoked on context shutdown

For example:

```
@Configuration
public class AppConfig {

    @Bean(name = "comp", initMethod = "turnOn", destroyMethod = "turnOff")
    Computer computer(){
        return new Computer();
    }
}

public class Computer {

    public void turnOn(){
        System.out.println("Load operating system");
    }
    public void turnOff(){
        System.out.println("Close all programs");
    }
}
```

3. @PreDestroy and @PostConstruct are alternative way for bean initMethod and destroyMethod. It can be used when the bean class is defined by us. For example;

```
public class Computer {
```

```
    @PostConstruct
    public void turnOn(){
        System.out.println("Load operating system");
    }

    @PreDestroy
    public void turnOff(){
        System.out.println("Close all programs");
    }
}
```

4. @ComponentScan: Configures component scanning directives for use with @Configuration classes. Here we can specify the base packages to scan for spring components.
5. @Component: Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
6. @PropertySource: provides a simple declarative mechanism for adding a property source to Spring’s Environment.
7. @Service: Indicates that an annotated class is a “Service”. This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning.
8. @Repository: Indicates that an annotated class is a “Repository”. This annotation serves as a specialization of @Component and advisable to use with DAO classes.
9. @Autowired: Spring @Autowired annotation is used for automatic injection of beans.
10. @Qualifier annotation is used in conjunction with Autowired to avoid confusion when we have two of more bean configured for same type.

Spring MVC Annotations

1. **@Controller** - @Controller annotation is used to indicate that it’s a web controller class.
2. **@RequestMapping** - is used with classes and methods to redirect the client request to specific handler method. Notice that handler methods are returning String, this should be the name of view page to be used as the response.
3. **@PathVariable** - is used to get the request parameters from URL, also known as query parameters
4. @**RequestParam** extracts values from URI.

For example, if the incoming HTTP request to retrieve a book on topic "Java" is `http://localhost:8080/shop/order/1001/receipts?date=12-05-2017`, then use `@PathVariable` to extract the `orderId` i.e. "1001" as shown below:

```
@RequestMapping(value="/order/{orderId}/receipts", method =
RequestMethod.GET)
public List listUsersInvoices( @PathVariable("orderId") int order,
@RequestParam(value = "date", required = false) Date dateOrNull) {
...
}
```

5. @ModelAttribute refers to a property of the Model object (the M in MVC ;)
6. @RequestBody and @ResponseBody
7. @RequestHeader and @ResponseHeader

Spring Transaction Management Annotations

@Transactional is the spring declarative transaction management annotation, read more at Spring MVC Hibernate.

Spring Security Annotations

@EnableWebSecurity is used with @Configuration class to have the Spring Security configuration defined

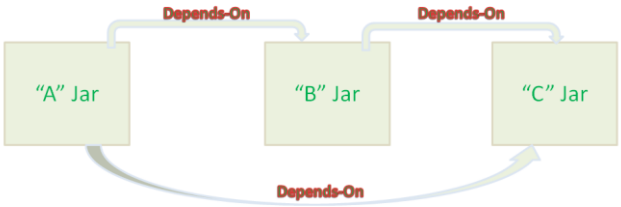
Spring Boot Annotations

1. @SpringBootApplication
2. @EnableAutoConfiguration

Spring Boot With Maven/Gradle?

Spring Boot Framework uses one of the greatest features of Maven/Gradle build tools: **"Transitively Dependency Resolution Management"**.

What is "Transitively Dependency Resolution Management"?
"Transitively Dependency Resolution Management" means: If we define an "A" dependency in build scripts, "A" is dependent on "B" and "B" is dependent on "C", That means "A" is also dependent on "C".
Then Build Tools will download and add all Three Jar files "A", "B" and "C" to our application classpath.



How Is Spring Security Implemented In a Spring Boot Application?
Minimal configuration is needed for implementation. All you need to do is add thespring-boot-starter-securitystarter in the pom.xml file. You will also need to create a Spring config class that will override the required method while extending the WebSecurityConfigurerAdapter to achieve security in the application. Here is some example code:

```
package com.gkatzioura.security.securityendpoints.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
@Override
protected void configure(HttpSecurity http) throws Exception {
http.authorizeRequests()
.antMatchers("/welcome").permitAll()
.anyRequest().authenticated()
.and()
.formLogin()
.permitAll()
.and()
.logout()
.permitAll();
}
}
```

What Is Semantic Monitoring?

It combines monitoring of the entire application along with automated tests. The primary benefit of Semantic Monitoring is to find out the factors which are more profitable to your business.

Semantic monitoring along with service layer monitoring approaches monitoring of microservices from a business point of view. Once an issue is detected, they allow faster isolation and bug triaging, thereby reducing the main time required to repair. It triages the service layer and transaction layer to figure out the transactions affected by availability or poor performance.

How Can You Set Up Service Discovery?

There are multiple ways to set up service discovery. I'll choose the one that I think to be most efficient, Eureka by Netflix. It is a hassle free procedure that does not weigh much on the application. Plus, it supports numerous types of web applications.

Eureka configuration involves two steps – client configuration and server configuration.

Client configuration can be done easily by using the property files. In the clas spath, Eureka searches for a eureka-client.properties file. It also searches for overrides caused by the environment in property files which are environment specific.

For server configuration, you have to configure the client first. Once that is done, the server fires up a client which is used to find other servers. The Eureka server, by default, uses the Client configuration to find the peer server.

Why Would You Opt for Microservices Architecture?

This is a very common microservices interview question which you should be ready for! There are plenty of pros that are offered by a microservices architecture. Here are a few of them:

- Microservices can adapt easily to other frameworks or technologies.
- Failure of a single process does not affect the entire system.
- Provides support to big enterprises as well as small teams.
- Can be deployed independently and in relatively less time.

How Does PACT Work?

PACT is an open source tool. It helps in testing the interactions between consumers and service providers. However, it is not included in the contract, increasing the reliability of the application. The consumer service developer starts by writing a test which defines a mode of interaction with the service provider. The test includes the provider's state, the request body, and the response that is expected. Based on this, PACT creates a stub against which the test is executed. The output is stored in a JSON file.

Define Domain Driven Design

The main focus is on the core domain logic. Complex designs are detected based on the domain's model. This involves regular collaboration with domain experts to resolve issues related to the domain and improve the model of the application. While answering this microservices interview question, you will also need to mention the core fundamentals of DDD. They are:

- DDD focuses mostly on domain logic and the domain itself.
- Complex designs are completely based on the domain's model.
- To improve the design of the model and fix any emerging issues, DDD constantly works in collaboration with domain experts.

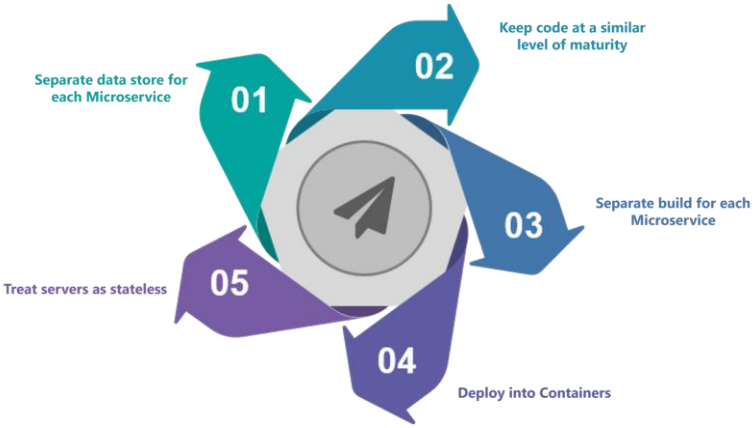
List down the advantages of Microservices Architecture.

Advantages of Microservices Architecture	
Advantage	**Description**
Independent Development	All microservices can be easily developed based on their individual functionality
Independent Deployment	Based on their services, they can be individually deployed in any application
Fault Isolation	Even if one service of the application does not work, the system still continues to function
Mixed Technology Stack	Different languages and technologies can be used to build different services of the same application
Granular Scaling	Individual components can scale as per need, there is no need to scale all components together

What are the features of Microservices?

- **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
- **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
- **Business Capabilities** – Microservices are very simple and focus on a single capability
- **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
- **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
- **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
- **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
- **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

What are the best practices to design Microservices?



What is Cohesion?
The degree to which the elements inside a module belong together is said to be **cohesion**.

What is Coupling?
The measure of the strength of the dependencies between components is said to be **coupling**. A good design is always said to have **High Cohesion** and **Low Coupling**.

What are different types of Tests for Microservices?

While working with microservices, testing becomes quite complex as there are multiple microservices working together. So, tests are divided into different levels.

- At the **bottom level**, we have **technology-facing tests** like- unit tests and performance tests. These are completely automated.
- At the **middle level**, we have tests for **exploratory testing** like the stress tests and usability tests.
- At the **top level,** we have **acceptance tests** that are few in number. These acceptance tests help stakeholders in understanding and verifying software features.

What do you understand by Distributed Transaction?

Distributed Transaction is any situation where a single event results in the mutation of two or more separate sources of data which cannot be committed atomically. In the world of microservices, it becomes even more complex as each service is a unit of work and most of the time multiple services have to work together to make a business successful.

What is an Idempotence and where it is used?

Idempotence is the property of being able to do something twice in such a way that the end result will remain the same i.e. as if it had been done once only.
Usage: Idempotence is used at the remote service, or data source so that, when it receives the instruction more than once, it only processes the instruction once.

What are Client certificates?

A type of digital certificate that is used by client systems to make authenticated requests to a remote server is known as the **client certificate**. Client certificates play a very important role in many mutual authentication designs, providing strong assurances of a requester's identity.

Q30. What is the use of PACT in Microservices architecture?

PACT is an open source tool to allow testing interactions between service providers and consumers in isolation against the contract made so that the reliability of Microservices integration increases.

Usage in Microservices:

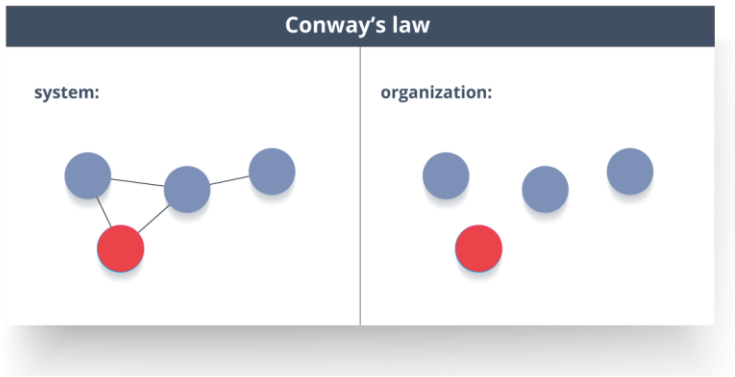
- Used to implement Consumer Driven Contract in Microservices.
- Tests the consumer-driven contracts between consumer and provider of a Microservice.

What is OAuth?

OAuth stands for open authorization protocol. This allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as third-party providers Facebook, GitHub, etc. So with this, you can share resources stored on one site with another site without using their credentials.

What is Conway's law?

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." – ***Mel Conway***



This law basically tries to convey the fact that, in order for a software module to function, the complete team should communicate well. Therefore the structure of a system reflects the social boundaries of the organization(s) that produced it.

What is DRY in Microservices architecture?

DRY stands for **Don't Repeat Yourself**. It basically promotes the concept of reusing the code. This results in developing and sharing the libraries which in turn result in tight coupling.

Q37. What is a Consumer-Driven Contract (CDC)?

This is basically a pattern for developing Microservices so that they can be used by external systems. When we work on microservices, there is a particular provider who builds it and there are one or more consumers who use Microservice.
Generally, providers specify the interfaces in an XML document. But in Consumer Driven Contract, each consumer of service conveys the interface expected from the Provider.

What do you understand by Semantic monitoring in Microservices architecture?

Semantic monitoring, also known as **synthetic monitoring** combines automated tests with monitoring the application in order to detect business failing factors.

Can we create State Machines out of Microservices?

As we know that each Microservice owning its own database is an independently deployable program unit, this, in turn, lets us create a State Machine out of it. So, we can specify different states and events for a particular microservice.
For Example, we can define an Order microservice. An Order can have different states. The transitions of Order states can be independent events in the Order microservice.

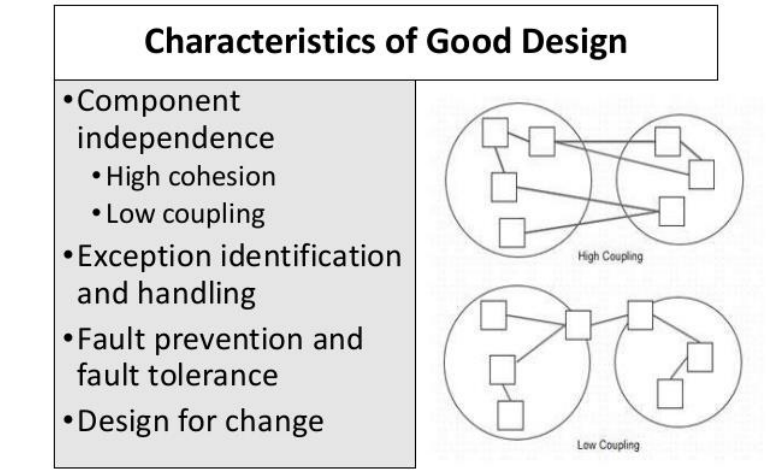
What are Reactive Extensions in Microservices?

Reactive Extensions also are known as Rx. It is a design approach in which we collect results by calling multiple services and then compile a combined response. These calls can be synchronous or asynchronous, blocking or non-blocking. Rx is a very popular tool in distributed systems which works opposite to legacy flows.

What is the role of an architect in Microservices architecture?

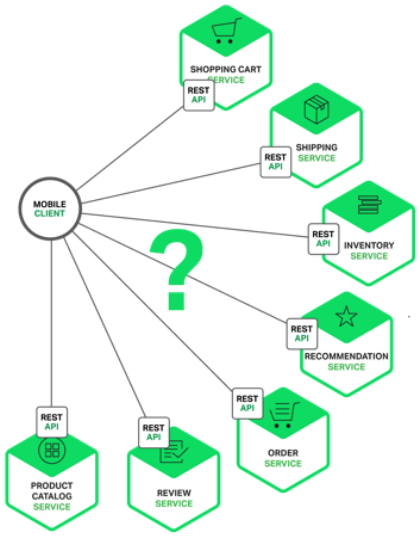
An architect in microservices architecture plays the following roles:

- Decides broad strokes about the layout of the overall software system.
- Helps in deciding the zoning of the components. So, they make sure components are mutually cohesive, but not tightly coupled.
- Code with developers and learn the challenges faced in day-to-day life.
- Make recommendations for certain tools and technologies to the team developing microservices.
- Provide technical governance so that the teams in their technical development follow principles of Microservice.



API GATEWAY

Direct client-to-microservice communication

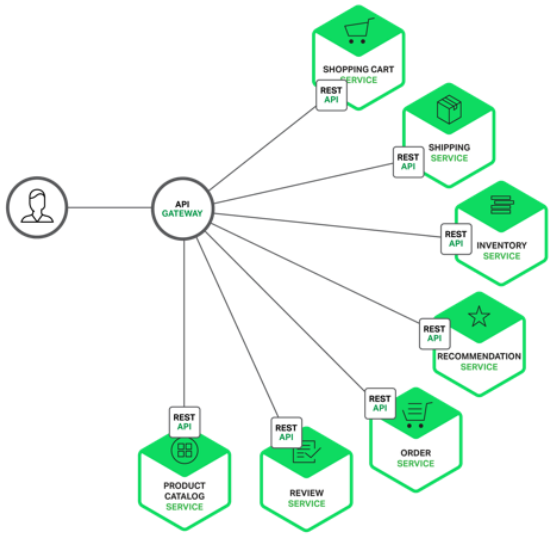


In theory, a client could make requests to each of the microservice directly. Each microservice would have a public endpoint (https://<serviceName>.api.company.name). This URL would map to the microservice's load balancer, which distributes requests across the available instances. To retrieve the product details the mobile client would make requests to each of the N services listed above.
Unfortunately, there are challenges and limitations with this option. One problem is the mismatch between the needs of the client and the fine-grained APIs exposed by each of the microservices. The client in this example has to make 7 separate requests. In more complex applications it might to make many more. For example, Amazon describe how 100s of services are involved in rendering their product page. While a client could make that many requests over a LAN, it would probably be too inefficient over the public Internet and would definitely be impractical over a mobile network. This approach also makes the client code much more complex.
Another problem with the client directly calling the microservices is that some might use protocols that are not web friendly. One service might use Thrift binary RPC while another service might use the AMQP messaging protocol. Neither protocol is particularly browser or firewall friendly and is best used internally. An application should use protocols such as HTTP and WebSockets outside of the firewall.
Another drawback with this approach is that it makes it difficult to refactor the microservices. Over time we might want to change how the system is partitioned into services. For example, we might merge two services or split a service into two or more services. If, however, clients communicate directly with the services then performing this kind of refactoring can be extremely difficult.
Because of these kinds of problems it rarely makes sense for clients to talk directly to microservices.

Using an API Gateway

Usually a much better approach is to use what is known as an API Gateway. An API Gateway is a server that is the single entry point into the system. It is similar to the Facade pattern from object-oriented design. The API Gateway encapsulates the internal system architecture and

provides an API that is tailored to each client. It might have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.
The following diagram shows how an API Gateway typically fits into the architecture:



The API Gateway is responsible for request routing, composition, and protocol translation. All requests from clients first go through the API Gateway. It then routes requests to the appropriate microservice. The API Gateway will often handle a request by invoking multiple microservices and aggregating the results. It can translate between web protocols such as HTTP and WebSockets and web unfriendly protocols that are used internally.
The API Gateway can also provide each client with a custom API. It typically exposes a coarse-grained API for mobile clients. Consider, for example, the product details scenario. The API Gateway can provide an endpoint (/productdetails?productid=xxx) that enables a mobile client to retrieve all of the product details with a single request. The API Gateway handles the request by invoking the various services – product info, recommendations, reviews, etc – and combining the results.
A great example of an API Gateway is the Netflix API Gateway. The Netflix streaming service is available on hundreds of different kinds of devices including televisions, set-top boxes, smartphones, gaming systems, tablets, etc. Initially, Netflix attempted to provide a one-size-fits-all API for their streaming service. However, they discovered that it didn't work well because of the diverse range of devices and their unique needs. Today, they use an API Gateway that provides an API tailored for each device by running device specific adapter code. An adapter typically handles each request by invoking on average 6-7 backend services. The Netflix API Gateway handles billions per day.

Benefits and Drawbacks of an API Gateway

As you might expect, using an API Gateway has both benefits and drawbacks. A major benefit of using an API Gateway is that it encapsulates internal structure of the application. Rather than having to invoke specific services, clients simply talk to the gateway. The API Gateway provides each kind of client with a specific API. This reduces the number of round trips between the client and application. It also simplifies the client code.
The API Gateway also has some drawbacks. It is yet another highly available component that must be developed, deployed, and managed. There is also a risk that the API Gateway becomes

a development bottleneck. Developers must update the API Gateway in order to expose each microservice's endpoints. It is important that the process for updating the API Gateway is as lightweight as possible. Otherwise, developers will be forced to wait in line in order to update the gateway. Despite these drawbacks, however, for most real world applications it makes sense to use an API Gateway.

Implementing an API Gateway

Now that we have looked at the motivations and the trade-offs for using an API Gateway lets now look at various design issues you need to consider.

Performance and Scalability

Only a handful of companies operates at the scale of Netflix and needs to handle billions of requests per day. However, for most applications the performance and scalability of the API Gateway is usually very important. It makes sense, therefore, to build the API Gateway on a platform that supports asynchronous, non-blocking I/O. There are a variety of different technologies that can be used to implement a scalable API Gateway. On the JVM you can use one of the NIO-based frameworks such Netty, Vertx, Spring Reactor, or JBoss Undertow. One popular non-JVM option is NodeJS, which is a platform built on Chrome's JavaScript engine. Another option is to use NGINX Plus. NGINX offers a mature, scalable, high-performance server and reverse proxy that is easily deployed, configured, and programmed. NGINX Plus can manage authentication, access control, load balancing requests, caching responses, and provides application-aware health checks and monitoring.

Using a Reactive Programming Model

The API Gateway handles some requests by simply routing them to the appropriate backend service. It handles other requests by invoking multiple backend services and aggregating the results. With some requests, such as product details request, the request to backend services are independent of one another. In order to minimize response time, the API Gateway should perform independent requests concurrently. Sometimes, however, there are dependencies between requests. The API Gateway might first need to validate the request by calling an authentication service, before routing the request to a backend service. Similarly, to fetch information about the products in a customer's wish list the API Gateway must first retrieve the customer's profile containing that information and then retrieve the information for each product. Another interesting example of API composition is the Netflix Video Grid.
Writing API composition code using the traditional asynchronous callback approach quickly leads you to callback hell. The code will be tangled, difficult to understand and error-prone. A much better approach is to write API Gateway code in a declarative style using a reactive approach. Examples of reactive abstractions including Scala Futures, Java 8 CompletableFutures, and JavaScript promises. There are also Reactive Extensions (Rx), which was originally developed by Microsoft for the .NET platform. Netflix created RxJava for the JVM specifically to use in their API Gateway. There is also RxJS for JavaScript, which runs in both the browser and NodeJS. Using a reactive approach will enable you to write simple, yet efficient API Gateway code.

Service Invocation

A microservices-based application is a distributed system and must use an inter-process communication mechanism. There are two styles of inter-process communication. One option

is to use an asynchronous, messaging-based mechanism. Some implementations use a message broker such as JMS or AMQP. Other such as Zeromq are broker-less and the services communicate directly. The other style of inter-process communication is a synchronous mechanism such as HTTP or Thrift. A system will typically use both asynchronous and synchronous styles. It might even use multiple implementations of each style. Consequently, the API Gateway will need to support a variety of communication mechanisms.

Service Discovery

The API Gateway needs to know the location (IP address and port) of each microservice with which it communicates. In a traditional application, you could probably hardwire the locations, but in a modern, cloud-based microservices application this is a non-trivial problem. Infrastructure services, such as a message broker, will usually have a static location, which can be specified via OS environment variables. However, determining the location of an application service is not so easy. Application services have dynamically assigned locations. Also, the set of instances of a service changes dynamically because of autoscaling and upgrades. Consequently, the API Gateway like any other service client in the system needs to use the system's service discovery mechanism: either Server-Side Discovery or Client-Side Discovery. A later article will describe service discovery in more detail. For now, it is worthwhile to note that if the system uses Client-Side Discovery then the API Gateway must be able to query the Service Registry, which is a database of all microservice instances and their locations.

Handling Partial Failures

Another issue you have to address when implementing an API Gateway is the problem of partial failure. This issue arises in all distributed systems whenever one service calls another service that is either responding slowly or unavailable. The API Gateway should never block indefinitely waiting for a downstream service. However, how it handles the failure depends on the specific scenario and which service is failing. For example, if the recommendation service is unresponsive in the product details scenario, the API Gateway should return the rest of the product details to the client since they are still useful to the user. The recommendations could either be empty or replaced by, for example, a hardwired top ten list. If, however, the product information service is unresponsive then API Gateway should return an error to the client.
The API Gateway could also return cached data if that was available. For example, since product prices change infrequently the API Gateway could return cached pricing data if the pricing service is unavailable. The data can be cached by the API Gateway itself or be stored in an external cache such as Redis or Memcached. By returning either default data or cached data the API Gateway ensures that system failures do not impact the user experience.
Netflix Hystrix is an incredibly useful library for writing code that invokes remote services. Hystrix times out calls that exceed the specified threshold. It implements the Circuit Breaker pattern, which stops the client from waiting needlessly for an unresponsive service. If the error rate for a service exceeds a specified threshold, Hystrix trips the circuit breaker and all requests will fail immediately for a specified period of time. Hystrix let's you define a fallback action when a request fails such read from a cache or return a default value. If you are using the JVM you should definitely consider using Hystrix. And, if you are running in a non-JVM environment you should use an equivalent library.

Summary

For most microservice-based applications, it makes sense to implement an API Gateway, which acts as a single entry point into a system. The API Gateway is responsible for request routing, composition and protocol translation. It provides each of the application's clients with a custom API. The API Gateway can also mask failures in the backend services by returning cached or default data. In the next article in the series, we will look at communication between services.

Spring Security: