1.What is Spring Boot

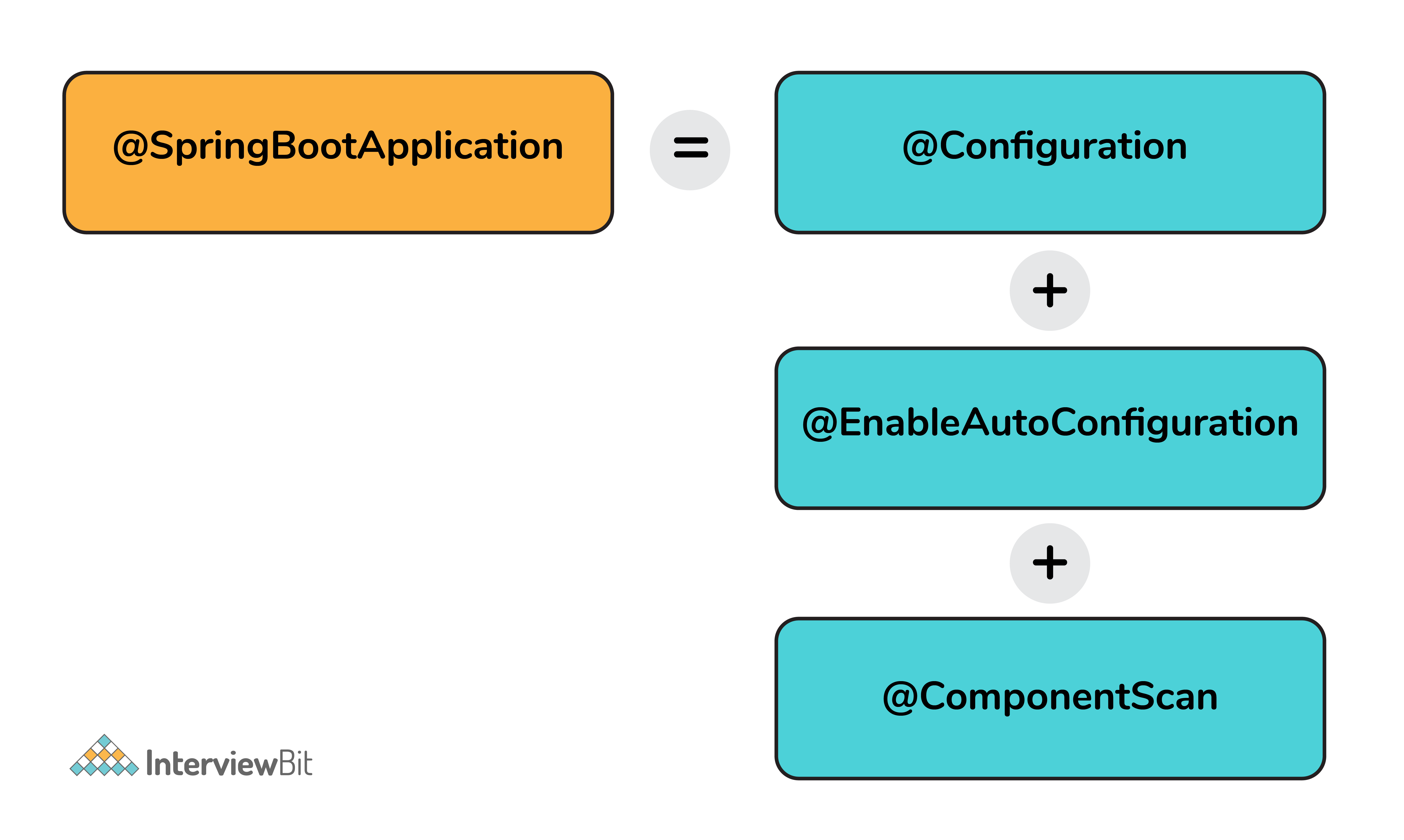
* Spring Boot is a lightweight Spring application framework
* Provides support for RAD- Rapid Application Development
* And gives production-ready applications with a need for very few configurations.
* It has embedded tomcat,
* Provide production-ready features such as metrics, health checks by using Actuator

2. Advantage or Benefits of Using Spring Boot

* Starter Dependency – With the help of this feature, Spring Boot aggregates common dependencies together Eg : spring-boot-starter-web
* Spring Initializer – This is a web application that helps a developer in creating an internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.
* Auto-Configuration – This helps in loading the default configurations according to the project you are working on.
* Spring Actuator – Spring boot uses actuator to provide “Management EndPoints” which helps the developer in going through the Application Internals, Metrics etc.

3. **What does @SpringBootApplication annotation do internally?**

@SpringBootApplication annotation is one point replacement for using @Configuration, @EnableAutoConfiguration and @ComponentScan



* **@AutoConfiguration**: This annotation automatically configures beans in the class path
* **@ComponentScan**: This annotation scans the components (@Component, @Service, etc.) in the package of annotated class and its sub-packages.
* **@Configuration:**This annotation configures the beans and packages in the class path.

4. **How Can We Set Up a Spring Boot Application**

**Using the**[***Spring Initializr***](https://start.spring.io/)**tool**

**5. How does Spring Boot works?**

* Spring Boot automatically configures your application based on the dependencies you have added to the project by using annotation.
* The entry point of the spring boot application is the class that contains @SpringBootApplication annotation and the main method.
* Spring Boot automatically scans all the components included in the project by using @ComponentScan annotation.

6. **What are starter dependencies?**

* Spring boot starter is a maven template that contains a collection of all the relevant transitive dependencies that are needed to start a particular functionality.
* Like we need to import spring-boot-starter-web dependency for creating a web application.

**7. What are the basic Spring Boot Annotations?**

* **@SpringBootApplication:**This is the main annotation used to bootstrap a Spring Boot application. It combines three annotations: **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**. It is typically placed on the main class of the application.
* **@Configuration:**This annotation is used to indicate that a class contains configuration methods for the application context. It is typically used in combination with @Bean annotations to define beans and their dependencies.
* **@Component:**This annotation is the most generic annotation for any Spring-managed component. It is used to mark a class as a Spring bean that will be managed by the Spring container.
* **@RestController:**This annotation is used to define a RESTful web service controller. It is a specialized version of the @Controller annotation that includes the @ResponseBody annotation by default.
* **@RequestMapping:**This annotation is used to map HTTP requests to a specific method in a controller. It can be applied at the class level to define a base URL for all methods in the class, or at the method level to specify a specific URL mapping.
  + @RequestMapping(value = "/pepsico" , method= RequestMethod.***GET***)
  + @RequestMapping(value = "/postTask" , method= RequestMethod.***POST***)
* @Bean – produces bean manged by Spring Container
  + The **@Bean** annotations indicates that a method produces a bean that is to be managed by the Spring container

**@Configuration**

**public** **class** ConfigClass {

**@Bean**

**public** RestTemplate needResttemplate() {

**return** **new** RestTemplate();

}

}

* @Service **– Business Logic**
  + It is used at the class level. It shows that the annotated class is a service class, such as **business basic logic**, and call external APIs.
* @Repository – **Database**
  + It is a Data Access Object (DAO) that accesses the **database**.

@Repository

**public** **interface** DistributionListRepository **extends** JpaRepository<DistributionList, Long>

{

}

* @Controller – **Handle Web Request**
  + indicate that the class is a web request handler & handles web requests.
  + It is most commonly used with @RequestMapping annotation.
* @Autowired
  + used to auto-wire spring bean
  + It injects object dependency implicitly.
  + When we use this annotation, the spring container auto-wires the bean by its matching data type.
* @PathVariable
  + Allows to use dynamic parameters in the Request URI without writing any complex code

@GetMapping("/fetchData**/{id}")**

**public** Optional<DistributionList> fetchDAta**(@PathVariable Long id**) {

**return** distributedlistRepository.findById(id);

}

* @PostMapping:

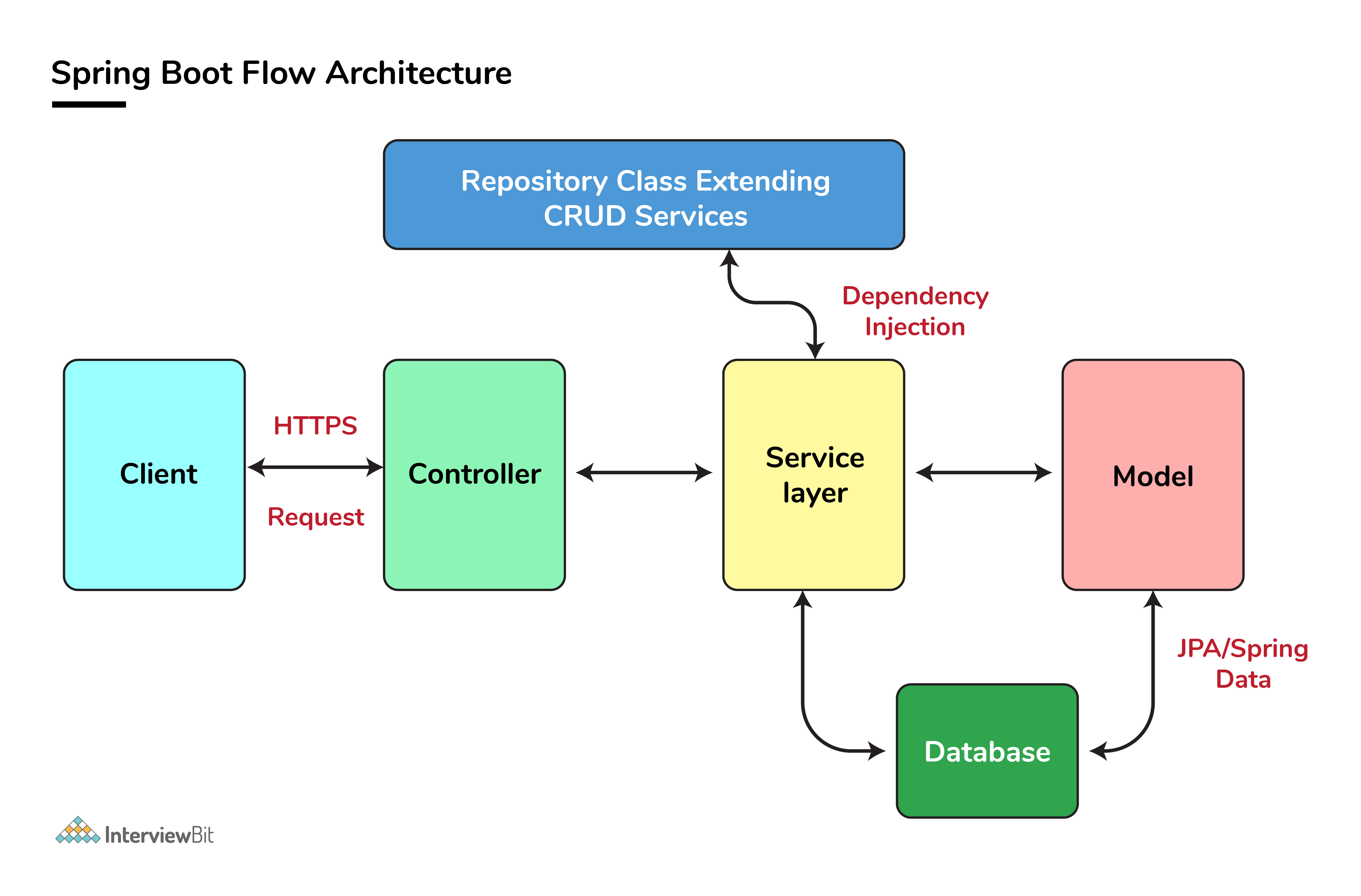
@PostMapping("/postTask2")

public String postTask(@RequestBody Names name) {

return name.getAddress();

}

* @RequestParam
* @Primary
* @Qualifer
* @Patchmapping
* @ControllerAdvice and @ExceptionHandler
* @Profile

8. Spring Boot Flow:

**9.**Difference between RequestMapping and GetMapping?

RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation.

* @RequestMapping(value = "/pepsico" , method= RequestMethod.***GET***)
* @RequestMapping(value = "/postTask" , method= RequestMethod.***POST***)

Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

* @GetMapping("/pepsicoapi/addTask")

10. **Spring Actuator? What are its advantages?**

* An actuator is an additional feature of Spring that helps you to monitor and manage your application when you push it to production.
* These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering, and many more that are automatically applied to your application.

**11. How to enable Actuator in Spring boot application?**

* we need to add the dependency of “spring-boot-starter-actuator” in pom.xml.

12. **What is dependency Injection?**

The process of injecting dependent bean objects into target bean objects is called dependency injection.

* Setter Injection: The IOC container will inject the dependent bean object into the target bean object by calling the setter method.
* Constructor Injection: The IOC container will inject the dependent bean object into the target bean object by calling the target bean constructor.

13. **@RestController annotation in Spring boot**

It is a combination of @Controller and @ResponseBody, used for creating a restful controller. It converts the response to JSON

14. How to **change the port in Spring Boot?**

By using the **server.port=8765** in the **application.properties**.

**15. Explain CORS in Spring Boot**

* [**CORS**](https://howtodoinjava.com/spring-boot2/spring-cors-configuration/)**(*Cross-origin resource sharing*) allows a webpage to request additional resources into the browser from other domains**
* **@CrossOrigin annotation marks the annotated method or type as permitting cross-origin requests.**
* **Eg:**

@CrossOrigin(origins = "\*", allowedHeaders = "\*")

@Controller

**public** **class** HomeController

{

*//*

}

16. **What is Spring Boot’s role in Microservices?**

Spring Boot makes it easier to develop and deploy microservices by providing embedded servers and simplifying configuration with properties. Additionally, it integrates well with Spring Cloud for service discovery, load balancing, and distributed configuration.

17. **What is Dependency Injection?**

Dependency Injection is a design pattern used to implement IoC (Inversion of Control). It allows the creation of dependent objects outside of a class and provides those objects to a class in various ways (constructor, setter method, or field injection).

18. **How do you handle exceptions in a Spring Boot application?**

In a Spring Boot application, you can handle exceptions using @ExceptionHandler methods. These methods are annotated with @ExceptionHandler and can be defined in a controller to handle specific exceptions thrown by the application.

19. **How does the DispatcherServlet work in Spring MVC?**

The DispatcherServlet is the front controller in Spring MVC that receives all incoming requests and dispatches them to the appropriate controllers based on the request URL.

20. What are the different ways to achieve communication between microservices?

* synchronous : by using - HTTP/RESTful API calls
* asynchronous communication : by using
  + Message Queues (e.g., RabbitMQ, ActiveMQ)
  + Publish/Subscribe (e.g., Apache Kafka, AWS SNS/SQS)

21. Explain the differences between synchronous (REST) and asynchronous (message queues) communication.

**Synchronous Communication:**

* In synchronous communication, the client sends a request to the server and waits for a response. The client remains blocked until it receives the server's response, making it a direct and immediate interaction.
* **Example**: A microservice calls a REST API endpoint on another microservice, waits for the response (success or failure), and then continues processing based on that response.

**Asynchronous Communication:**

* In asynchronous communication, the client sends a message to a queue and immediately continues its own processing without waiting for a response. The server retrieves the message from the queue and processes it independently. The response, if needed, may be sent back later via another queue or mechanism.
* **Example**: A microservice places a message in a RabbitMQ queue, which is then picked up and processed by another microservice at a later time. The original microservice doesn't wait for this process to complete.

22.What is service discovery, and how is it implemented in a microservices architecture?

* Service discovery allows microservices to locate each other dynamically.
* Since microservices are often deployed across multiple servers their locations (IP addresses and ports) can change frequently. Service discovery provides a mechanism to manage this dynamic environment, enabling microservices to discover and communicate with each other without hardcoding addresses.

**Implementing Service Discovery in Microservices Architecture**

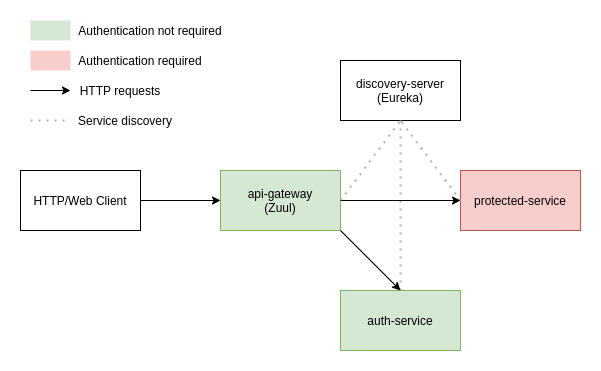
* Using Netflix Eureka (Client-Side Service Discovery)
  + **Service Registry**: Eureka Server acts as the service registry where all microservices register themselves.
  + **Service Registration**: Each microservice (client) uses a Eureka Client to register its instance with the Eureka Server.
  + **Service Discovery**: A service needing to call another service queries the Eureka Server to get the instance information and uses that to make a direct call.

**Spring Boot Example**:

* Add **spring-cloud-starter-netflix-eureka-server** to the **service registry** (Eureka Server).
* Add **spring-cloud-starter-netflix-eureka-client** to the microservices that need to register with the Eureka Server. (Eureka Client)
* The **@EnableEurekaServer** annotation starts the Eureka Server, and **@EnableEurekaClient** enables the microservices to register with it.

23. What is an API Gateway, and why is it used in microservices architecture?

* A single entry point that routes requests to appropriate microservices
* It serves as an intermediary between the clients (such as web applications, mobile apps, or other services) and the backend microservices, providing a single unified interface for accessing various services.



* It takes care of the following:

1. **Client Request**:

* The client (e.g., a web or mobile application) makes a request to the API Gateway instead of directly communicating with the individual microservices.

2. **Routing**:

* The API Gateway examines the incoming request, including the request path, headers, and other metadata. Based on this information, it determines which microservice should handle the request.
* For example, if the client sends a request to /api/users, the API Gateway will route this request to the user management microservice.

3. **Authentication and Authorization**:

* Before forwarding the request to the backend service, the API Gateway may perform authentication to verify the client’s identity (e.g., checking a token) and authorization to ensure the client has permission to access the requested resource.

4. **Load Balancing**:

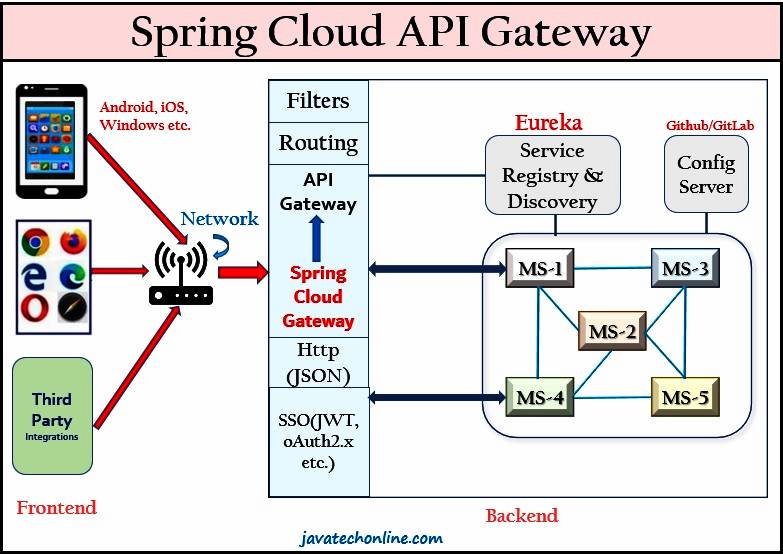
* If multiple instances of a microservice are running, the API Gateway can balance the load by distributing requests among these instances. This prevents any single instance from being overwhelmed with too many requests.

5. **Service Discovery Integration**:

* The API Gateway can integrate with a service discovery mechanism to dynamically route requests to the appropriate microservice instances, even as they scale up or down. This allows it to discover the current locations (IP addresses and ports) of microservices.

6. **Logging and Monitoring**:

* The API Gateway logs all requests and responses, which can be used for monitoring, debugging, and analytics. It provides insights into how the system is being used and helps in identifying performance bottlenecks or security issues.



24. What is a Circuit Breaker? (or) How do you implement resilience and fault tolerance in microservice

It is designed to improve system resilience and prevent cascading failures by managing how services communicate with each other when some services are experiencing issues.

**Circuit Breaker States**:

* **Closed**: In this state, requests are allowed to pass through to the service. The circuit breaker monitors the responses for failures. If the failure rate exceeds a predefined threshold, the circuit breaker transitions to the Open state.
* **Open**: When the failure rate is high, the circuit breaker opens. Requests are not forwarded to the service and are instead immediately failed. This prevents the failing service from being overwhelmed and gives it time to recover.
* **Half-Open**: After a predetermined time, the circuit breaker transitions to the Half-Open state. A few requests are allowed through to test if the service has recovered. If these requests are successful, the circuit breaker transitions back to Closed. If they fail, the circuit breaker returns to Open.

Code:

Annotations:

1. @EnableCircuitBreaker

2. HystrixCommand(fallbackMethod=”fallback”)

Pom.xml:

A screen shot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

A computer screen with text on it

Description automatically generated

A computer screen shot of a program

Description automatically generated

**25. Saga Pattern**

The saga pattern helps **establish consistency** in distributed applications, and **coordinates** transactions **between multiple microservices** to maintain data consistency.

* The application needs to maintain data consistency across multiple microservices without tight coupling.
* We must be ablet to roll back if an operation fails in the sequence.

There are two common saga implementation approaches,

* + - choreography
    - orchestration.

**Choreography**:

* In a choreographed saga, each service involved in the saga knows about the next service in the sequence. Each service publishes events to notify other services of its completion or failure.
* Services are responsible for handling their own compensating transactions.
* This approach reduces the need for a central coordinator and is more decentralized.

How do you deploy microservices to a cloud environment?

 How do you manage configuration properties in Spring Boot?

 What are the different ways to externalize configuration in Spring Boot?