* Spring Boot is an open source Java-based framework used to create a Micro Service. It is developed by Pivotal Team.

## What is Micro Service?

* Micro Service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.
* @ComponentScan annotation is used to find beans and the corresponding injected with @Autowired annotation.
* If you followed the Spring Boot typical layout, no need to specify any arguments for @ComponentScan annotation. All component class files are automatically registered with Spring Beans.
* Application Runner and Command Line Runner interfaces lets you to execute the code after the Spring Boot application is started. You can use these interfaces to perform any actions immediately after the application has started.

@SpringBootApplication

public class DemoApplication implements ApplicationRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(ApplicationArguments arg0) throws Exception {

System.out.println("Hello World from Application Runner");

}

}

* Spring Boot uses Apache Commons logging for all internal logging. Spring Boot’s default configurations provides a support for the use of Java Util Logging, Log4j2, and Logback. Using these, we can configure the console logging as well as file logging.
* @RestController annotation is used to define the RESTful web services. It serves JSON, XML and custom response.
* @RequestMapping annotation is used to define the Request URI to access the REST Endpoints. We can define Request method to consume and produce object. The default request method is GET.
* @RequestBody annotation is used to define the request body content type.
* @PathVariable annotation is used to define the custom or dynamic request URI. The Path variable in request URI is defined as curly braces {}
* @RequestParam annotation is used to read the request parameters from the Request URL. By default, it is a required parameter. We can also set default value for request parameters
* @ControllerAdvice is an annotation, to handle the exceptions globally.
* @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.
* Interceptor in Spring Boot to perform operations under the following situations −

Before sending the request to the controller

Before sending the response to the client

* To work with interceptor, you need to create @Component class that supports it and it should implement the HandlerInterceptor interface.

The following are the three methods you should know about while working on Interceptors −

* preHandle() method − This is used to perform operations before sending the request to the controller. This method should return true to return the response to the client.
* postHandle() method − This is used to perform operations before sending the response to the client.
* afterCompletion() method − This is used to perform operations after completing the request and response.
* Will have to register this Interceptor with InterceptorRegistry by using WebMvcConfigurerAdapter

filter is an object used to intercept the HTTP requests and responses of your application. By using filter, we can perform two operations at two instances –

Before sending the request to the controller,Before sending a response to the client.

* For uploading a file, you can use MultipartFile as a Request Parameter and this API should consume Multi-Part form data value.

@RestController

public class FileUploadController {

@RequestMapping(value = "/upload", method = RequestMethod.POST,

consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public String fileUpload(@RequestParam("file") MultipartFile file) throws IOException {

File convertFile = new File("/var/tmp/"+file.getOriginalFilename());

convertFile.createNewFile();

FileOutputStream fout = new FileOutputStream(convertFile);

fout.write(file.getBytes());

fout.close();

return "File is upload successfully";

}

}

* Thymeleaf is a Java-based library used to create a web application. It provides a good support for serving a XHTML/HTML5 in web applications.
* Cross-Origin Resource Sharing (CORS) is a security concept that allows restricting the resources implemented in web browsers. It prevents the JavaScript code producing or consuming the requests against different origin.
* For example, your web application is running on 8080 port and by using JavaScript you are trying to consuming RESTful web services from 9090 port. Under such situations, you will face the Cross-Origin Resource Sharing security issue on your web browsers.
* @CrossOriginannotation for the controller method. This @CrossOrigin annotation supports specific REST API, and not for the entire application.
* Scheduling is a process of executing the tasks for the specific time period.
* Java Cron expressions are used to configure the instances of CronTrigger, a subclass of org.quartz.Trigger.
* @EnableScheduling annotation is used to enable the scheduler for your application.
* @Scheduled annotation is used to trigger the scheduler for a specific time period.

**@Scheduled(cron = "0 \* 9 \* \* ?")**

* Fixed Rate scheduler is used to execute the tasks at the specific time. It does not wait for the completion of previous task.

**@Scheduled(fixedRate = 1000)**

* Fixed Delay scheduler is used to execute the tasks at a specific time. It should wait for the previous task completion.

**@Scheduled(fixedDelay = 1000, initialDelay = 1000)**

* Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.

@EnableEurekaServer

eureka.client.registerWithEureka = false

eureka.client.fetchRegistry = false

server.port = 8761

# **Service Registration with Eureka**

@EnableEurekaClient

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

eureka.client.instance.preferIpAddress = true

spring.application.name = eurekaclient

# **Zuul Proxy Server and Routing**

Zuul Server is a gateway application that handles all the requests and does the dynamic routing of microservice applications. The Zuul Server is also known as Edge Server.

@EnableZuulProxy

spring.application.name = zuulserver

zuul.routes.products.path = /api/demo/\*\*

zuul.routes.products.url = http://localhost:8080/

server.port = 8111

# **Spring Boot - Cloud Configuration Server**

Spring Cloud Configuration Server is a centralized application that manages all the application related configuration properties.

@EnableConfigServer annotation makes your Spring Boot application act as a Configuration Server.

server.port = 8888

spring.cloud.config.server.native.searchLocations=file:///C:/configprop/

SPRING\_PROFILES\_ACTIVE=native

# **Spring Boot - Cloud Configuration Client**

Some applications may need configuration properties that may need a change and developers may need to take them down or restart the application to perform this. However, this might be lead to downtime in production and the need of restarting the application. Spring Cloud Configuration Server lets developers to load the new configuration properties without restarting the application and without any downtime

@RefreshScope annotation is used to load the configuration properties value from the Config server.

spring.application.name = config-client

spring.cloud.config.uri = http://localhost:8888

# **Spring Boot – Actuator**

Spring Boot Actuator provides secured endpoints for monitoring and managing your Spring Boot application. By default, all actuator endpoints are secured.

# **Spring Boot - Admin Server**

Monitoring your application by using Spring Boot Actuator Endpoint is slightly difficult. Because, if you have ‘n’ number of applications, every application has separate actuator endpoints, thus making monitoring difficult. Spring Boot Admin Server is an application used to manage and monitor your Microservice application.

To handle such situations, CodeCentric Team provides a Spring Boot Admin UI to manage and monitor all your Spring Boot application Actuator endpoints at one place.

@EnableAdminServer annotation is used to make your as Admin Server to monitor all other microservices.