**Note:** Here @RestController = @Controller + @ResponseBody

@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiration.

<https://medium.com/swlh/deploying-spring-boot-applications-15e14db25ff0>

**org.springframework.boot package contains all Spring boot libraries**

**BookStore Project Angular 8 and Spring Boot (**<https://www.youtube.com/watch?v=CrDXtLfiZos&list=PLA7e3zmT6XQXBymgbvBARnpeGaKpivAYd&index=11>**)**

**Ques. How Spring Boot Application works internally?**

Ans. The application starts using the "main method" which calls the "run" method (static method of SpringApplication class). From the run method, the main application context kicks off which searches for the classes annotated with @Configuration and initializes all the declared beans in those configuration classes. Based on the scope of those beans, stores those beans in JVM, specifically in a space inside JVM which is known as IOC container. After the creation of all the beans, automatically configures the dispatcher servlet and registers the default handler mappings, messageConverts, etc.

**Spring Boot:**

Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.

**Spring Boot features:**

project, maven project, starter project wizard, spring Initializer, cli, application, annotations, dm, properties, starters, actuator, JPA, JDBC etc.

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

Note: To create a new Spring Boot application, we will use the following in our example:

* [Java 1.8+](http://www.oracle.com/)
* [Spring STS](https://www.spring.io/tools/sts)
* [Maven](https://maven.apache.org/)

==============================================================

Spring Boot Project

There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.

* Spring Maven Project
* Spring Starter Project Wizard
* Spring Initializer
* Spring Boot CLI

**Note:** We need to configure it in order to make it a Spring Boot project. Here, we are adding parent to our Maven project. It is used to declare that our project is a child to this parent project.

Pom.xml file **<parent>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-parent**</artifactId>**

**<version>**1.5.4.RELEASE**</version>**

**</parent>**

After that add the following dependency to the pom.xml file. Here, we are adding web dependency by adding spring-boot-starter-web.

**Note - Maven project will add web dependency to the project by downloading the jar.**

**<dependencies>**

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-web**</artifactId>**

**</dependency>**

**</dependencies>**

<properties>

<java-version>1.8</java-version>

</properties>

**Spring Boot – Runners (TUTORIAL POINT)**

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.

**1.Application Runner:** Application Runner is an interface used to execute the code after the Spring Boot application started.

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(String... arg0) throws Exception {

System.out.println("Hello world from Command Line Runner");

}

}

**2. Command Line Runner:** Command Line Runner is an interface. It is used to execute the code after the Spring Boot application started.

package com.tutorialspoint.demo;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(String... arg0) throws Exception {

System.out.println("Hello world from Command Line Runner");

}

}

# Spring Boot Starter:

Starters are a set of convenient dependency descriptors which we can include in our application.

Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the **spring-boot-starter-data-jpa** dependency in your project.

Starter should follow a naming pattern like: **spring-boot-starter**-\*, where \* is a particular type of application. This naming structure is intended to help when you need to find a starter.

# The following application starters are provided by Spring Boot under the org.springframework.boot group:

# spring-boot-starter-thymeleaf

# spring-boot-starter-data-couchbase

# spring-boot-starter-web-services

# spring-boot-starter-mail

# spring-boot-starter-data-redis

# spring-boot-starter-integration

# spring-boot-starter-jersey

# spring-boot-starter-hateoas

# spring-boot-starter-validation

## Spring Boot technical starters

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-undertow | It is used for Undertow as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-jetty | It is used for Jetty as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-logging | It is used for logging using Logback. Default logging starter. |
| spring-boot-starter-tomcat | It is used for Tomcat as the embedded servlet container. Default servlet container starter used by spring-boot-starter-web. |
| spring-boot-starter-log4j2 | It is used for Log4j2 for logging. An alternative to spring-boot-starter-logging. |

# ------------------

------------------------------------------------

# Spring Boot Actuator

Spring Boot provides an actuator to monitor and manage our application. Actuator is a tool which has HTTP endpoints. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

We can use HTTP and JMX endpoints to manage and monitor the Spring Boot application.

Spring Boot actuator kind of monitoring webservices like what are the property you exposes , rest services, bean configured,

To get production-ready features, we should use the spring-boot-actuator module. We can enable this feature by adding it to the pom.xml file.

**/actuator/health**.

There are three main features of Spring Boot Actuator:

* Endpoints
* Metrics
* Audit

**<dependencies>**

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-actuator**</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>**org.springframework.data**</groupId>**

**<artifactId>**spring-data-rest-hal-browser**</artifactId>**

**</dependency>**  **</dependencies>**

Note: Spring boot Actuater exposes meta data services

==================================================================================

# Spring Boot Thymeleaf View(<https://www.youtube.com/watch?v=67VTXf-Dt5w>)

It is a server side Java template engine for web application. It's main goal is to bring elegant natural templates to your web application.

It can be integrate with Spring Framework and ideal for HTML5 Java web applications.

In the following example, we are using Thymeleaf as our HTML template and rendering it from controller.

In order to use Thymeleaf we must add it into our pom.xml file like:

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-thymeleaf**</artifactId>**

**</dependency>**

after that mentioning it in our template that we are using this library. Like:

1. **<html** xmlns:th="https://thymeleaf.org"**>**

============================================

# Spring Boot JPA

Spring Boot provides **spring-boot-starter-data-jpa** starter to connect Spring application with relational database efficiently. You can use it into project POM (Project Object Model) file.

Here, we are creating a spring-boot application which uses JPA to connect to the database.

Note: Spring Boot provides default server and apache derby database. So, we will use that in our application.

# Spring Boot JDBC

Spring Boot provides starter and libraries for connecting to our application with JDBC. Here, we are creating an application which connects with Mysql database. It includes the following steps to create and setup JDBC with Spring Boot.

* Create a database : create database springbootdb
* Create a table in to mysql:

Create table user(id **int** UNSIGNED primary key not **null** auto\_increment, name varchar(100), email varchar(100));

* Creating a Spring Boot Pproject

-----------------------------------------------------------------

# Spring Annotations

#### Note - The @RestController and @RequestMapping annotations are Spring MVC annotations. They are not specific to Spring Boot

# Package org.springframework.stereotype

Annotations denoting the roles of types or methods in the overall architecture (at a conceptual, rather than implementation, level).

[Description](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/package-summary.html#package.description)

|  |  |
| --- | --- |
| **Annotation Types Summary** | |
| **Annotation Type** | **Description** |
| [**Component**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Component.html) | Indicates that an annotated class is a "component". |
| [**Controller**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Controller.html) | Indicates that an annotated class is a "Controller" (e.g. |
| [**Indexed**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Indexed.html) | Indicate that the annotated element represents a stereotype for the index. |
| [**Repository**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Repository.html) | Indicates that an annotated class is a "Repository", originally defined by Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects". |
| [**Service**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Service.html) | Indicates that an annotated class is a "Service", originally defined by Domain-Driven Design (Evans, 2003) as "an operation offered as an interface that stands alone in the model, with no encapsulated state." |

The **@RestController** annotation informs to the Spring to render the result back to the caller.

The **@RequestMapping** annotation is used to provide routing information. It tells to the Spring that any HTTP request should map to the corresponding method. We need to import org.springframework.web.annotation package in our file.

================================================

# Spring Boot Dependency Management

Spring Boot manages dependencies and configuration automatically. You don't need to specify version for any of that dependencies.

Spring Boot upgrades all dependencies automatically when you upgrade Spring Boot.

#### Note - We can also specify a version by overriding Spring Boot recommendations if we think that's necessary.

#### Note - Each release of Spring Boot is associated with a base version of the Spring Framework so we highly recommend you to not specify its version on your own.

## Maven Dependency Management System

The Maven project inherits the following feature from the parent project.

* Java version 1.6 as the default compiler.
* UTF-8 source encoding.
* A Dependency Management section which allows you to exclude <version> tags for common.
* Dependencies, inherited from the spring-boot-dependencies POM.
* Sensible resource filtering.
* Sensible plugin configuration.

## Inheriting the starter parent

We can configure our project to inherit from the **spring-boot-starter-parent** by simply setting as below.

**<parent>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-parent**</artifactId>**

**<version>**2.0.0.BUILD-SNAPSHOT**</version>**

**</parent>**

#### Note - You should only need to specify the Spring Boot version number on this dependency. If you import additional starters, you can safely exclude the version number.

### Changing the Java version

we can easily set Java version for our project in the properties section as given below.

1. **<properties>**
2. **<java.version>**1.8**</java.version>**
3. **</properties>**

Adding Spring Boot Maven Plugin

We can include Maven plugin in our pom.xml file. It is used to package the project as an executable jar. We are adding it here.

**<build>**

**<plugins>**

**<plugin>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-maven-plugin**</artifactId>**

**</plugin>**

**</plugins>**

**</build>**

## Developer Tools

Spring Boot includes an additional set of tools that can make the application development experience a little more pleasant. The spring-boot-devtools module can be included in any project to provide additional development-time features. To include devtools support, add the module dependency to your build, as shown in the following listings for Maven and Gradle:

**Maven**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencie

**Spring Boot Logging(**<https://www.tutorialspoint.com/spring_boot/spring_boot_logging.htm>**)**

# # Spring boot support default logging but we can configure custome loggin in spring boot by logback configuration

# #:Logback only support xml based config it exist in classpath resources folder

<?xml version = "1.0" encoding = "UTF-8"?>

<configuration>

<appender name = "STDOUT" class = "ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>

</encoder>

</appender>

<appender name = "FILE" class = "ch.qos.logback.core.FileAppender">

<File>/var/tmp/mylog.log</File>

<encoder>

<pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>

</encoder>

</appender>

<root level = "INFO">

<appender-ref ref = "FILE"/>

<appender-ref ref = "STDOUT"/>

</root>

</configuration>

**Spring Boot Interview Questions**

* Differnce b/w Spring and Sprin Boot?
* Deploy Spring Boot Project (<https://medium.com/swlh/deploying-spring-boot-applications-15e14db25ff0>).

**Q: How to secure rest api by spring boot security?(Example use Project WebAppSprintBootApp)**

**Or check how to use role based security and url based security**

[**https://www.youtube.com/watch?v=J8I3s0sSP0c**](https://www.youtube.com/watch?v=J8I3s0sSP0c)**)**

<https://www.baeldung.com/spring-boot-security-autoconfiguration>

## 1. Simplest way to add all required jars is add **spring-boot-starter-security** dependency.

## 2. Configure WebSecurityConfigurerAdapter

## To enable authentication and authorization support in spring boot rest apis, we can configure a utility class ****WebSecurityConfigurerAdapter****. It helps in requiring the user to be authenticated prior to accessing any configured URL (or all urls) within our application.

@SuppressWarnings("deprecation")

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable().authorizeRequests().anyRequest().fullyAuthenticated().and().httpBasic();

}

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().withUser("admin").password("admin123").roles("ADMIN");

}

@Bean

public static NoOpPasswordEncoder passwordEncoder() {

return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();

}

}

**Q:Enable HTTPS/SSL in Spring Boot/HTTPS using Self-Signed Certificate in Spring Boot? (Example use Project WebAppSprintBootApp)**

SSL: Secure Socket Layer.

**SSL** – stands for *Secure Sockets Layer*. It is the industry standard protocol for keeping an internet connection secure by safeguarding all sensitive data that is being sent between two systems, preventing hackers from reading and modifying any information transferred.

**TLS** – (Transport Layer Security) is an updated, more secure, version of SSL. It adds more features. Today, certificates provided by certificate authorities are based on TLS only. But regarding secured communication over network, the term SSL is still common as it is the old and just become popular among community.

**HTTPS** – (Hyper Text Transfer Protocol Secure) appears in the URL when a website is secured by an SSL certificate. It is the secured version of HTTP protocol.

**Truststore and Keystore** – Those are used to store SSL certificates in Java but there is little difference between them. truststore is used to store public certificates while keystore is used to store private certificates of client or server.

## 1.Create your own self signed SSL certificate(<https://howtodoinjava.com/spring-boot/spring-boot-ssl-https-example/>)

To get SSL digital certificate for our application we have two options –

1. to create a self-signed certificate
2. to obtain SSL certificate from certification authority(CA) we call it CA certificate.

For today’s demo purpose we will create self-signed certificate generated by java keytool command. We need to run the keytool -genkey command from command prompt.

Here is the exact command we will use –

keytool -genkey -alias selfsigned\_localhost\_sslserver -keyalg RSA -keysize 2048 -validity 700 -keypass changeit -storepass changeit -keystore ssl-server.jks

#### Spring boot HTTPS Config./Spring boot SSL Configuration

First we need to copy the generated keystore file (ssl-server.jks) into the resources folder and then open the application.properties and add the below entries.

server.port=8443

server.ssl.key-alias=selfsigned\_localhost\_sslserver

server.ssl.key-password=changeit

server.ssl.key-store=classpath:ssl-server.jks

server.ssl.key-store-provider=SUN

server.ssl.key-store-type=JKS

#### Redirect from HTTP to HTTPS: This is an optional step in case you want to redirect your HTTP traffic to HTTPS, so that the full site becomes secured. To do that in spring boot, we need to add HTTP connector at 8080 port and then we need to set redirect port 8443. So that any request in 8080 through http, it would be automatically redirected to 8443 and https.

private Connector redirectConnector() {

  Connector connector = new Connector("org.apache.coyote.http11.Http11NioProtocol");

  connector.setScheme("http");

  connector.setPort(8080);

  connector.setSecure(false);

  connector.setRedirectPort(8443);

  return connector;

}

# Spring Boot AOP

The application is generally developed with multiple layers. A typical Java application has the following layers:

* **Web Layer:** It exposes the **services** using the REST or web application.
* **Business Layer:** It implements the **business logic** of an application.
* **Data Layer:** It implements the **persistence logic** of the application.

The responsibility of each layer is different, but there are a few common aspects that apply to all layers are **Logging, Security, validation, caching,** etc. These common aspects are called **cross-cutting concerns.**

If we implement these concerns in each layer separately, the code becomes more difficult to maintain. To overcome this problem, **Aspect-Oriented Programming** (AOP) provides a solution to implement cross-cutting concerns.

* Implement the cross-cutting concern as an aspect.
* Define pointcuts to indicate where the aspect has to be applied.

## Types of AOP Advices

There are five types of AOP advices are as follows:

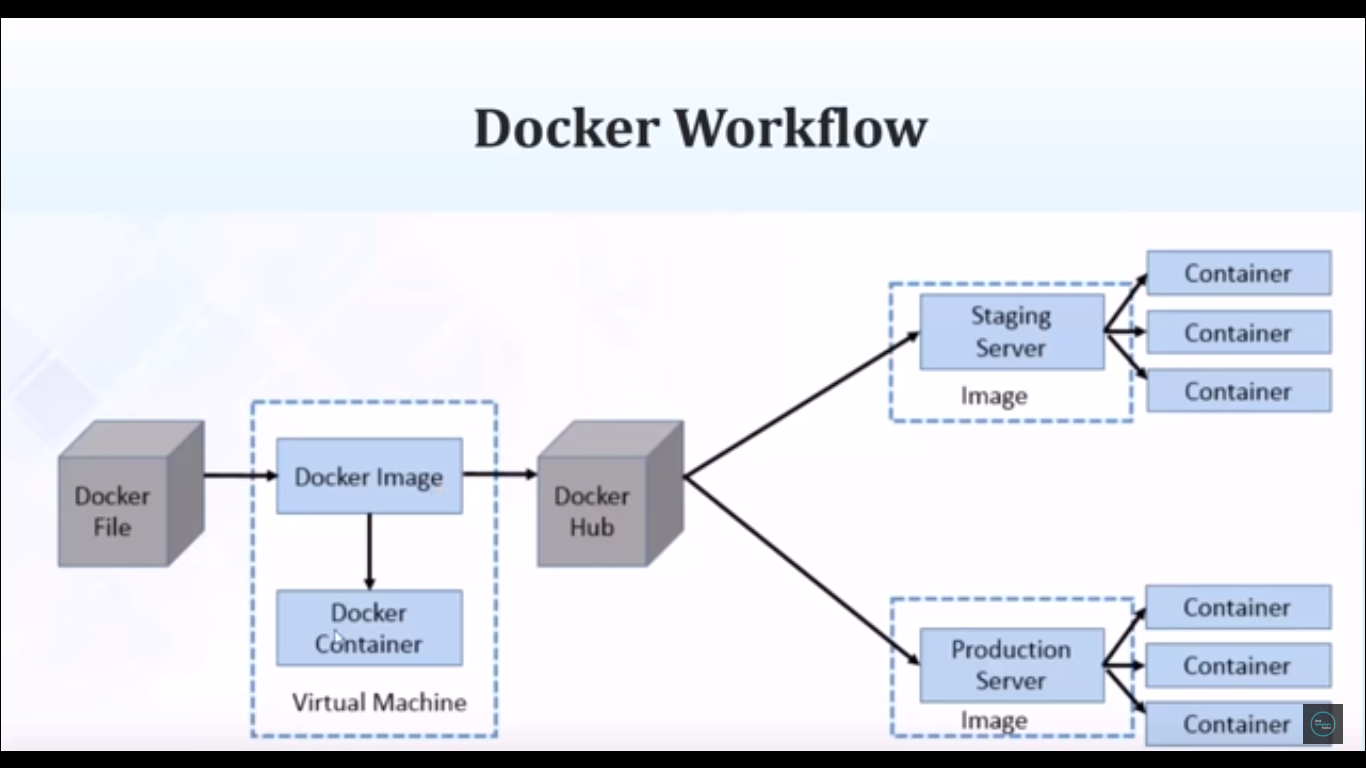
* Before Advice
* After Advice
* Around Advice
* After Throwing
* After Returning

There are **two** benefits of aspects:

* First, the logic for each concern is now in one place instead of scattered all over the codebase.
* Second, the business modules only contain code for their primary concern. The secondary concern has been moved to the **aspect**.

-=========================================================================

**Docker**

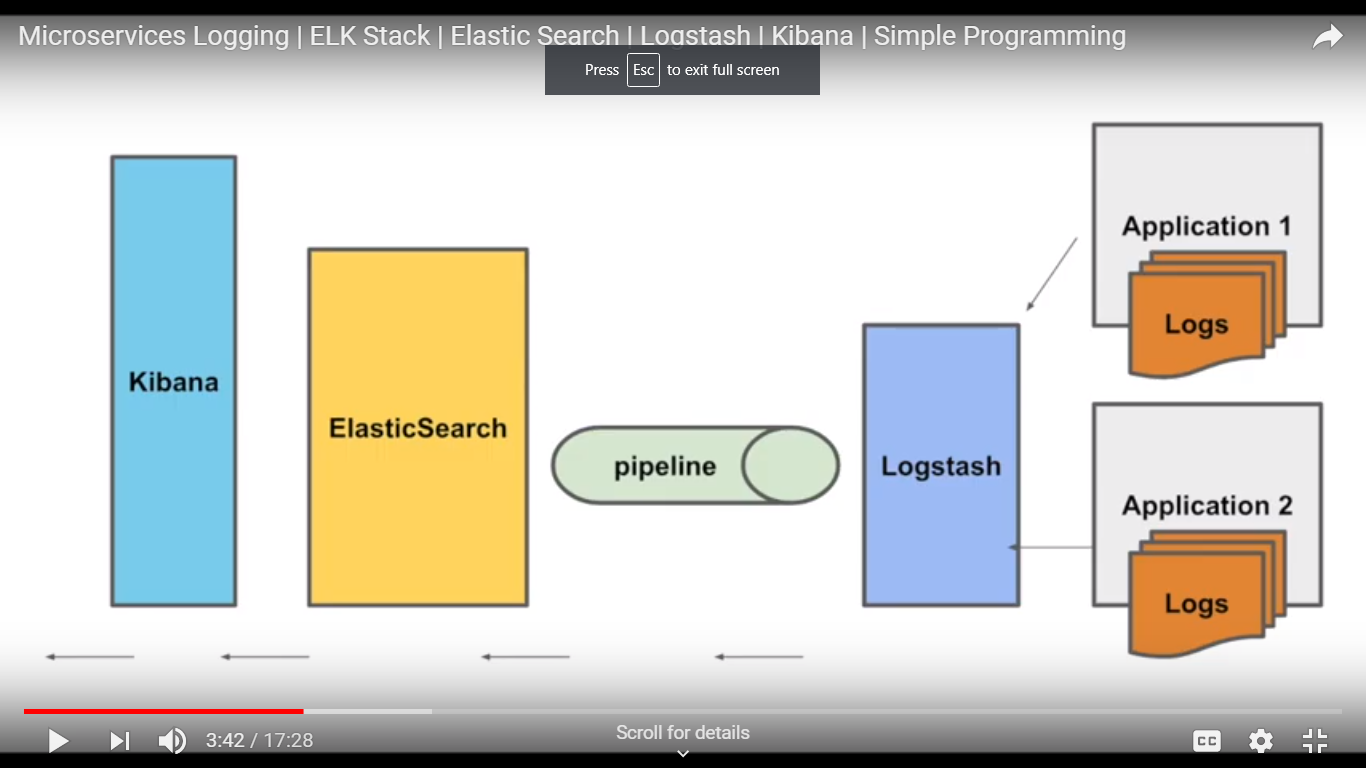


======================================================

**Q. How to configure multiple microservices in multiple services?**

**Q.How to micro services talk with each other?**

**Q: ELK: Elk is a simple and robust platform for logging?** <https://www.javainuse.com/spring/springboot-microservice-elk>



**Spring Boot Interview Questions:**

## What is Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the **RAD (Rapid Application Development)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



**Q: Why should we use spring boot?**

**Ans: stability, Based on jvm , connectivity, open source, cloud native, Flexibility.**

**Q:What is webserver?**

**Webservers are computers which deliver the requested webpages. Every web server has an IP address and a Domain name.**

**There are various webserver in market:**

* **XAMP**
* **Apache**
* **NGNIX**
* **Tornado**
* **Caddy**
* **Microsoft-IIS**

**Q: What is NGnix?**

* **An open source software**
* **Webserver for reverse proxing, caching, and load balancing**
* **Provides http server capability.**
* **Designed for maximum performace and stability.**
* **Functions a proxy server for email(IMAP, POP3, and SMTP)**
* **Uses a non-threaded and event-driven architecture.**

