**Logging in Spring Boot:**

**Logging in Spring Boot:**

Local Env: Developers will write the code and will test

DEV Env: Developers will perform integration testing

SIT Env: Testing team will test app functionality

UAT Env: Client side team will test app functionality

PROD Env: Live Deployment

**What is Logging?**

=> The process of storing application execution details to console/ file is called as Logging

=> To understand runtime behavior of our code

=> To identify exceptions in the code

=> To identify root cause of the exception

**Logging Architecture:**

1) Logger

2) Layout

3) Appender

=> Logger is a class which is used to generate log msgs

logger.trace ()

logger.debug ()

logger.info ()

logger.warn ()

logger.error ()

=> Layout represents log msg format

=> Appender represents destination to store log msgs

Ex: ConsoleAppender (will print msgs on console)

Ex: FileAppender (will store msgs in file)

**Logging Levels:**

=> Log Level will decide which msgs should be generated/printed.

TRACE > DEBUG > INFO > WARN > ERROR

=> When we set LOG level, from that level all higher levels will be printed.

Note: In spring boot, the default log level is INFO.

=> We can change log level in boot application like below

logging.level.root = debug

logging.file.name=ashokit.log

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

|  |
| --- |
| **EX:**  **@RestController**  **public** **class** **MsgRestController** {  **private** **static** Logger logger = LoggerFactory.getLogger (MsgRestController.class);  **@GetMapping** ("/welcome")  **public** String **getMsg** () {  logger.info ("getMsg () - execution stated");  String msg = "Welcome to Logging in spring boot";  **try** {  **int** i = **10** / **0**;  } **catch** (Exception e) {  logger.error ("Exception: " + e.getMessage ());  }  logger.info ("getMsg () - execution ended");  **return** msg;  }  } |

=> In real time, multiple users will access our application on daily basis.

=> If we maintain single log file then lot of data will be stored in that file

=> After few days/months we can't open that file because of huge size.

=> To overcome this problem we will use Rolling concept in logging.

**Rolling File Appender:**

2 ways

1) Time Based Rolling

Ex: Create New Log file for every 24 hours (day wise)

2) Size Based Rolling

Ex: Store only 1 GB data in one log file

=> We can configure rolling in 2 ways

1) Properties / yml file

2) Xml configuration (recommended)

ex: logback.xml

**Logging Tools:**

1) Log4J (vulnerability)

2) Log4j2

3) Log back

4) Log stash

5) JCL

Note: We will use SLF4J as a facede for logging.

**Configuring logback.xml**

-> Logback.xml file is used to configure logging information

-> We will keep logback.xml file in src/main/resources folder

-> Below is the sample logback.xml file

|  |
| --- |
| <configuration>  <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">  <encoder>  <pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>  </encoder>  </appender>  <appender name="RollingFile"  class="ch.qos.logback.core.rolling.RollingFileAppender">  <file>MyApp.log</file>  <encoder>  <pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>  </encoder>  <rollingPolicy  class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">  <fileNamePattern>MyApp-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  <maxFileSize>1MB</maxFileSize>  <maxHistory>30</maxHistory>  <totalSizeCap>10MB</totalSizeCap>  </rollingPolicy>  </appender>  <root level="INFO">  <appender-ref ref="Console" />  <appender-ref ref="RollingFile" />  </root>  </configuration> |

**Log Monitoring Tools:**

=> These are used for monitoring logs of our application

1) ELK / EFK - Open source s/w

2) Splunk - Commercial s/w

Splunk: https://youtu.be/BUfOuWLgnMU?si=i9K\_iS7FJ9pISIok

EFK: https://youtu.be/8MLcbbfEL1U?si=OpRu00vghWF5lMNO