Log4j2 can be configurd in many ways:

1. Through a configuration file written in XML, JSON, YAML, or properties format.
2. Programmatically, by creating a ConfigurationFactory and Configuration implementation.
3. Programmatically, by calling the APIs exposed in the Configuration interface to add components to the default configuration.
4. Programmatically, by calling methods on the internal Logger class.

Here, it is configured using xml:

Now, this document will try to explain the meaning of the configuration lines written in log4j.xml

Now, it can be seen that some properties are mentioned under <Properties> tag.

**<Property name="LOG\_DIR">NewMongoDBAccessor.log**

**</Property>**

**<Property name="ARCHIVE">logs/app-%d{MM-dd-yyyy}-%i.log**

**</Property>**

**<Property name="PATTERN">%p %d{dd-MM-yyyy ,HH:mm:ss,SSS} [%t- %F-%L] %m%n**

**</Property>**

For simplicity, those can be considered as user defined variables. Which will be used later.

However, the details about <Properties> tag is given below:

Log4j 2 supports the ability to specify tokens in the configuration as references to properties defined elsewhere. Some of these properties will be resolved when the configuration file is interpreted while others may be passed to components where they will be evaluated at runtime. To accomplish this, Log4j uses variations of [Apache Commons Lang](https://commons.apache.org/proper/commons-lang/)'s [StrSubstitutor](https://logging.apache.org/log4j/2.x/log4j-core/apidocs/org/apache/logging/log4j/core/lookup/StrSubstitutor.html) and [StrLookup](https://logging.apache.org/log4j/2.x/log4j-core/apidocs/org/apache/logging/log4j/core/lookup/StrLookup.html) classes. In a manner similar to Ant or Maven, this allows variables declared as **$LOG\_DIR, $ARCHIVE, $PATTERN** to be resolved using properties declared in the configuration itself.

**<Appenders>**

**<RollingFile name="RollingFile" fileName="${LOG\_DIR}"**

**filePattern="${ARCHIVE}" append="true">**

**<PatternLayout pattern="${PATTERN}" />**

**<Policies>**

**<SizeBasedTriggeringPolicy size="4 KB" />**

**</Policies>**

**<DefaultRolloverStrategy max="2" />**

**</RollingFile>**

**<Console name="Console" target="SYSTEM\_OUT">**

**<PatternLayout pattern="${PATTERN}" />**

**</Console>**

**</Appenders>**

Now, Appenders tag define different types of appenders, which are going to be used. (the configuration statements with in <Appenders> </Appenders> )

Now, within **<Appenders> opening tag and </Appenders> closing tag,**

It can be seen that a RollingFile is defined. RollingFile actually defines

A RollingFileAppender. RollingFileAppender is used to print the log in a file.

Now, it can be seen, that the variables which are defined previously using **property tag**, are used here. (or, more clearly, their values are used here)

Now, **filename** defines the filename of the current log file.

**filePattern="${ARCHIVE}" append="true"**

**Now, replace $ARCHIVE with logs/app-%d{MM-dd-yyyy}-%i.log**

Now, what does %d{MM-dd-yyyy} print?

It prints the date in the specified format.

What does %i print?

It prints the counter value. The counter is associated with **DefaultRolloverStrategy** value.If DefaultRolloverStrategy is 2, the counter value will be 0 and 1.

**append=true**

It actually does a great thing.

It can be noticed that,

**<SizeBasedTriggeringPolicy size="4 KB" />**

**SizeBasedTriggeringPolicy is used here.** So, if main log file’s size goes beyond 4kb, the old logs will be maintained in archive files. (Complete depended upon append=true. If append is set false, old logs will be gone forever. (I tested it. You can find the behavioral testing of append=true and append=false In the current folder named as behavioraltestingwhenappendtrue.docx and behavioraltestingwhenappendfalse.docx )

So, the great thing does by SizeBasedTriggeringPolicy is that when sizeBasedTriggeringPolicy is applied and the main log file goes beyond that size, the old logs are maintained in archive.

Now, <console> and </console> these tags are used to define a consoleAppender. consoleAppender is usually used to print logs in either system.out or system.err. It’s default target is however, system.out.

Now, when consoleAppender is used, a layout is used (either the layout pattern needs to be user-defined or default layout pattern will be used)

To format the logevent.

**<PatternLayout pattern="${PATTERN}" />** defines that pattern.

Now, <Loggers> defines different logger.

**<Loggers>**

**<Root level="DEBUG">**

**<AppenderRef ref="RollingFile" level="DEBUG"/>**

**<AppenderRef ref="Console" level="DEBUG"/>**

**</Root>**

**</Loggers>**

Now, <Root level=”DEBUG”> this statement sets the log level of RootLogger. RootLogger can be accessed or used in java programs by using LogManager.getRootLogger function.

Now, under the <Root> opening and </Root> closing tag, log levels are defined for RollingFile as well as Console.

However, the logger configuration in this case, can be improved or can be more well organised, if the following is done:

**<Loggers>**

**<Root level="DEBUG">**

**<AppenderRef ref="RollingFile"/>**

**<AppenderRef ref="Console"/>**

**</Root>**

**</Loggers>**