Centralized Logging System

Major Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by

Kritika Deora: (19UCSE4028)

Purvansh Parmar: (19UCSE4014)

Under the Supervision of

Simran Chaudhary
Assistant Professor

Department of Computer Science and Engineering
MBM University, Jodhpur
June,2022

Department of Computer Science & Engineering

M.B.M University
Ratanada, Jodhpur, Rajasthan, India –342011

CERTIFICATE

This is to certify that the work contained in this report entitled **"Centralized Logging System**" is submitted by the group members Ms. Kritika Deora (Roll. No: 19UCSE4028) and Mr. Purvansh Parmar (Roll No: 19UCSE4014) to the Department of Computer Science & Engineering, M.B.M. University, Jodhpur, for the partial fulfilment of the requirements for the degree of **Bachelor of Engineering** in **Computer Science and Engineering**.

They have carried out their work under my supervision. This work has not been submitted else-where for the award of any other degree or diploma.

The project work in our opinion, has reached the standard fulfilling of the requirements for the degree of Bachelor of Engineering in Computer Science in accordance with the regulations of the Institute.

Simran Chaudhary
Assistant Professor
(Guide)
Dept. of Computer Science & Engg.
M.B.M. University, Jodhpur

Prof. N.C. Barwar
(Head)
Dept. of Computer Science & Engg.
M.B.M. University, Jodhpur

This page was intentionally left blank.

DECLARATION

We, ***Kritika Deora and Purvansh Parmar,*** hereby declare that this project titled **“Centralized Logging System”** is a record of original work done by us under the supervision and guidance of ***Simran Chaudhary***.

We, further certify that this work has not formed the basis for the award of the Degree/Diploma/Associateship/Fellowship or similar recognition to any candidate of any university and no part of this report is reproduced as it is from any other source without appropriate reference and permission.

SIGNATURE OF STUDENT

(Purvansh Parmar)
8th Semester, CSE
Enroll. - 19R/18606
Roll No. - 19UCSE4014

SIGNATURE OF STUDENT

(Kritika Deora)
8th Semester, CSE
Enroll. - 18R/06210
Roll No. - 19UCSE4028

This page was intentionally left blank.

ACKNOWLEDGEMENT

We wish to extend our sincere gratitude to our project guide Simran Chaudhary, Assistant Professor, Dept. of Computer Science and Engineering, for her valuable guidance and encouragement which has been absolutely helpful in successful completion of this project.

We express our sincere gratitude to our project mentor Anil Gupta, Professor, Dept. of Computer Science and Engineering, for his support, cooperation, and motivation provided to us during the project.

We are indebted to Dr. Nemi Chand Barwar, Head, Dept. of CSE for his valuable support.

At last we must express our sincere heartfelt gratitude to all the staff members of Computer Science and Engineering Department who helped us directly or indirectly during this course of work.

This page was intentionally left blank.

ABSTRACT

Due to the rise in microservices architecture there is an increasing demand for implementing a centralized logging system, to structurize how the logs are stored and retrieved to improve time required to resolve these issues. These microservices are distributed over various cloud providers / infrastructures and have their own logging system, this makes the problem discerning, this in turn makes transaction errors more tedious to debug.

Centralized logging system is a dependency which uses Spring AOP to provide modularized code. We are using Log4j to standardize all the logs. To explore, analyse, visualize, and monitor all the activities generated by the services we will use AWS CloudWatch.

The technologies used in this centralized logging are Spring AOP, Log4J and CloudWatch.

This page was intentionally left blank.

Contents

List of Figures

Chapter 1

INTRODUCTION

Almost every application that runs in a server environment generates logs automatically. These logs are a vital part of any system because they provide essential information about how a system is presently operating and also, how it operated in the past. By searching through log data, you're able to pinpoint issues, errors, and trends. However, it can be extremely time consuming and frustrating to manually look up one particular error on hundreds, or even thousands of servers, across thousands of log files.

Centralized Logging System is a type of logging solution system that consolidates all of your log data and pushes it to one central, accessible, and easy-to-use interface. Centralized logging is designed to make your life easier. Not only does CLS provide multiple features that allow you to easily collect log information, but it also helps you consolidate, analyze, and view that information quickly and clearly.

The purpose of a centralized logging system is to streamline and automate the process of manual log management, reducing the time and effort often involved in maintaining a massive repository of log data.

Services :

- Storing log data from multiple sources in a central location
- Easily searching inside the logs for important information
- Low costs and increased storage and backup for historical data
- Accessing different level of logs in different files.

- Applying pointcuts using custom annotation as well as matching signature patterns.

1.1 Problem Statement

Due to the rise in microservices architecture there is an increasing demand for implementing a centralized logging system, to structurize how the logs are stored and retrieved to improve time required to resolve these issues. These microservices are distributed over various cloud providers / infrastructures and have their own logging system, this makes the problem discerning, this in turn makes transaction errors more tedious to debug.

1.2 Product Scope

- It is an utility, so can be implemented with any microservice by using it as a maven dependency.

- It makes storing and searching for logs easy and convenient.

- By implementing CLS, the extra work related to distributed logging is reduced, that is all the logs are stored at one place only.

- It is useful for debuggers, as they can trace the error logs easily on CloudWatch or file.

- By using CLS, it makes the microservice more dynamic, profitable, and secure.

- It is very useful in transaction based microservice as we can easily trace all the transactions.

Chapter 2

TECHNOLOGY USED

2.1 Spring AOP

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects.

Aspect : A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement.

Joinpoint : This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.

Advice : This is the actual action to be taken either before or after the method execution. This is the actual piece of code that is invoked during program execution by Spring AOP framework.

Pointcut : This is a set of one or more joinpoints where an advice should be executed. You can specify PointCuts using expressions or patterns as we will see in our AOP examples.

Types of Advice :

● **Before advice**: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).

● **After returning advice:** Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.

● **After throwing advice:** Advice to be executed if a method exits by throwing an exception.

● **After (finally) advice:** Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).

● **Around advice:** Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behaviour before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

Custom Annotation : By using annotation meta-data, our core business logic isn't polluted with our transaction code. This makes it easier to reason about, refactor, and to test in isolation. This can be executed using pointcuts.

2.2 Log4J

Log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License. log4j has three main components:

● loggers: Responsible for capturing logging information.

● appenders: Responsible for publishing logging information to various preferred destinations.
● layouts: Responsible for formatting logging information in different styles.

Logging Levels :

ALL - All levels including custom levels.
DEBUG - Designates fine-grained informational events that are most useful to debug an application.
INFO - Designates informational messages that highlight the progress of the application at coarse-grained level.
WARN - Designates potentially harmful situations.
ERROR - Designates error events that might still allow the application to continue running.
FATAL - Designates very severe error events that will presumably lead the application to abort.
OFF - The highest possible rank and is intended to turn off logging.
TRACE - Designates finer-grained informational events than the DEBUG.

For the standard levels, we have ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF

2.3 CloudWatch

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. Amazon CloudWatch is basically a metrics repository. An AWS service—such as Amazon EC2—puts metrics into the repository, and you retrieve statistics based on those metrics. If you put your own custom metrics into the repository, you can retrieve statistics on these metrics as well.

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization).

A **namespace** is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch.

Each metric data point must be associated with a **time stamp**. The time stamp can be up to two weeks in the past and up to two hours into the future. If you do not provide a time stamp, CloudWatch creates a time stamp for you based on the time the data point was received.

A **dimension** is a name/value pair that is part of the identity of a metric. You can assign up to 10 dimensions to a metric

Log Stream is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream.

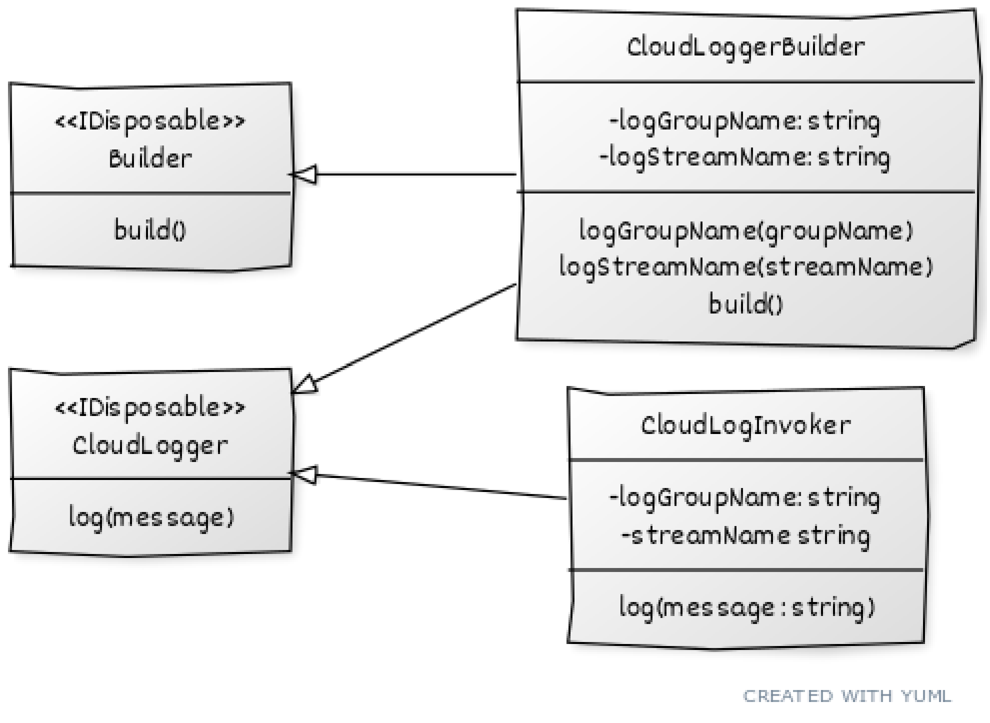
Log Group is a group of log streams that share the same retention, monitoring, and access control settings. You can define log groups and specify which streams to put into each group. There is no limit on the number of log streams that can belong to one log group.

You can search your log data using the Filter and pattern syntax. You can search all the log streams within a log group, or by using the AWS CLI you can also search specific log streams. When each search runs, it returns up to the first page of data found and a token to retrieve the next page of data or to continue searching. If no results are returned, you can continue searching.

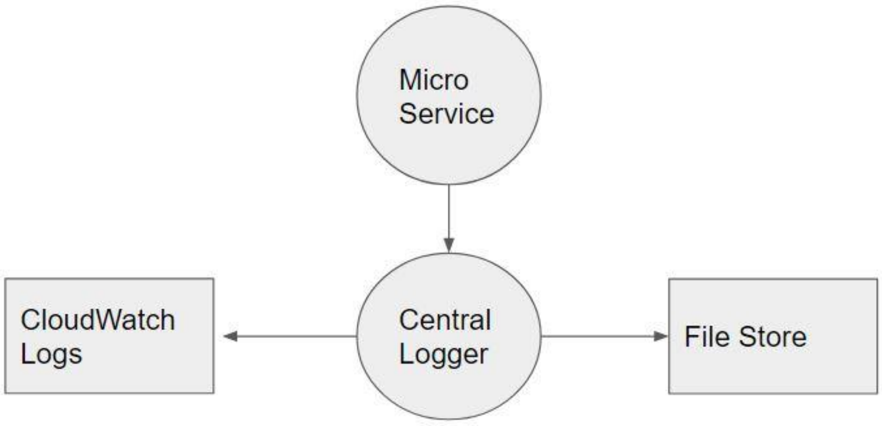
Chapter 3

PROJECT DETAILS

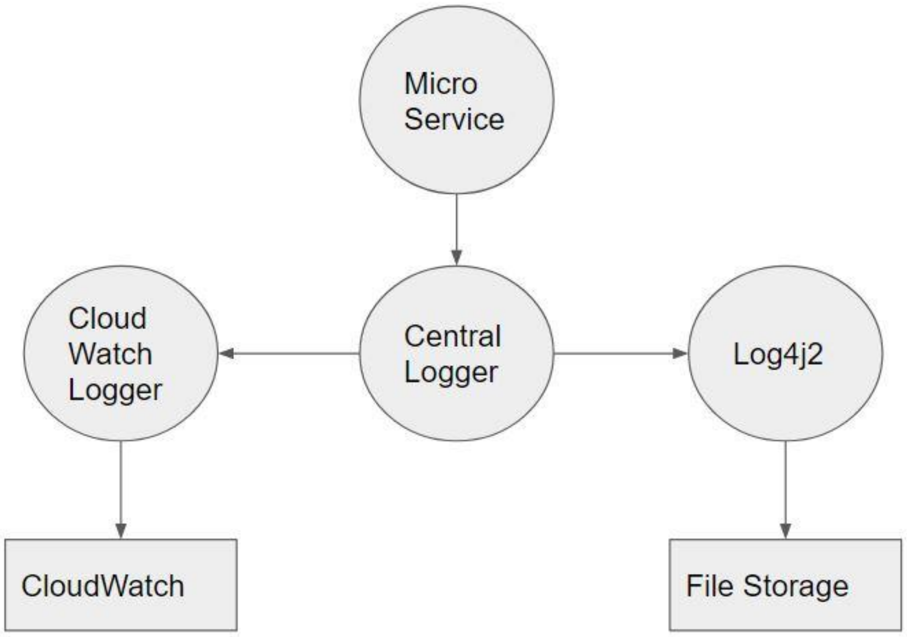
3.1 MODULAR DECOMPOSITION



3.1 Class Diagram for builder class

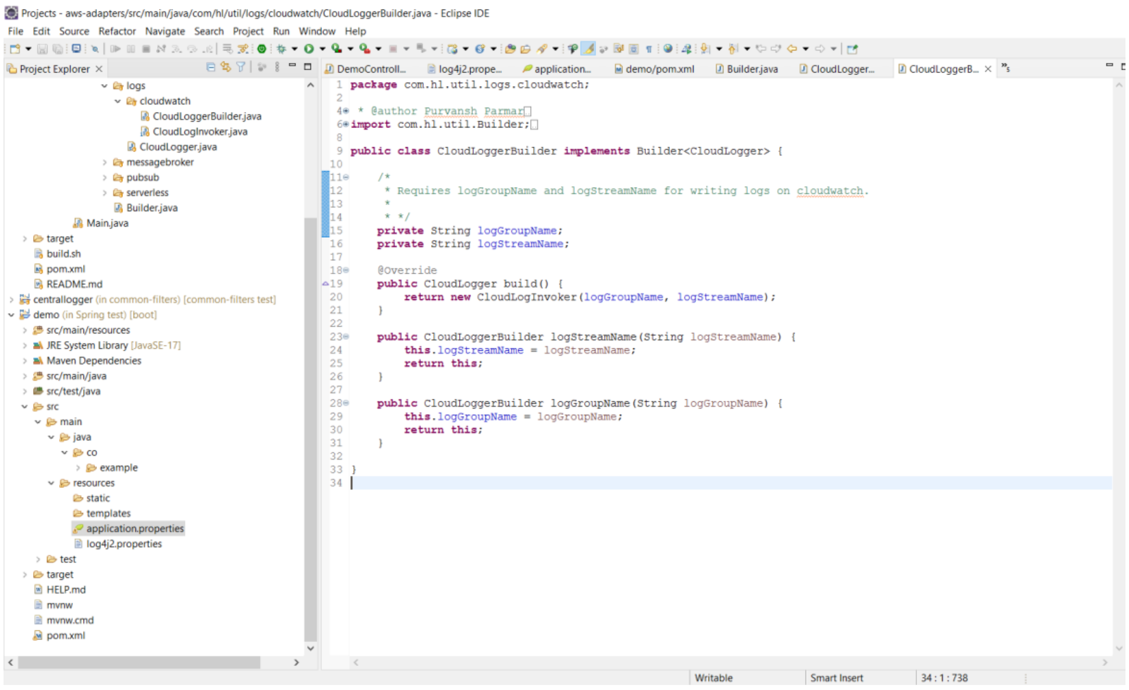


3.2 Context Diagram



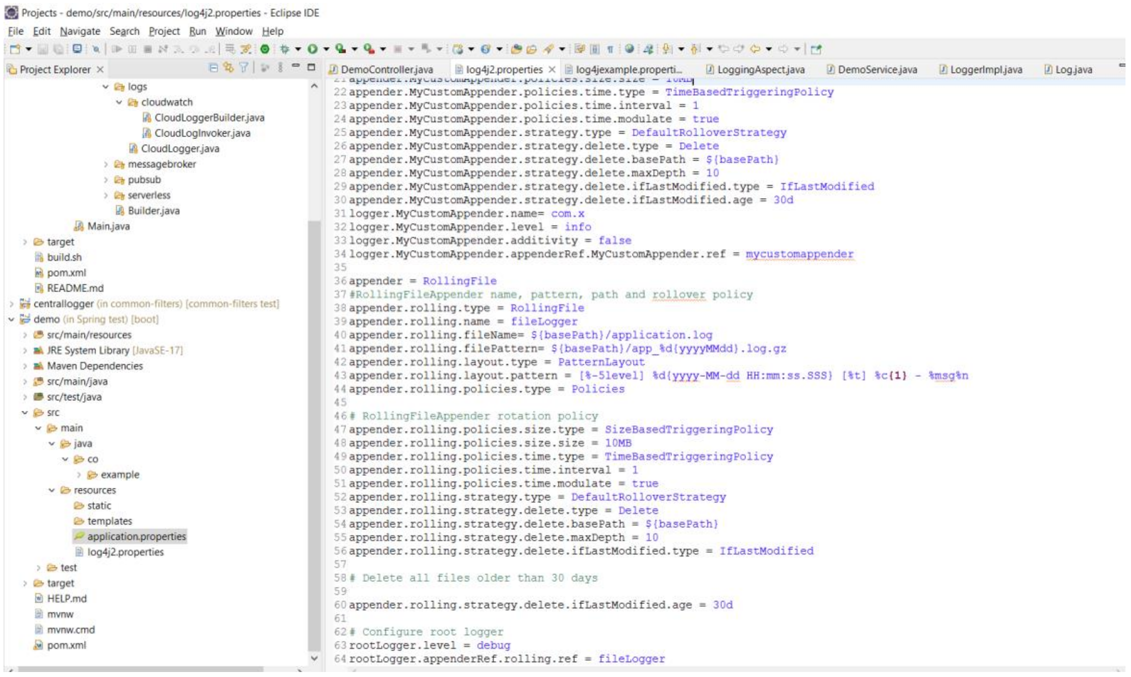
3.3 1-level DFD

3.2 CODE SNAPSHOTS


Projects - aws-adapters/src/main/java/com/hl/util/logs/cloudwatch/CloudLoggerBuilder.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
DemoControll... log4j2.prope... application... demo/pom.xml Builder.java CloudLogger... CloudLoggerB...
Writable Smart Insert 34 : 1 : 738

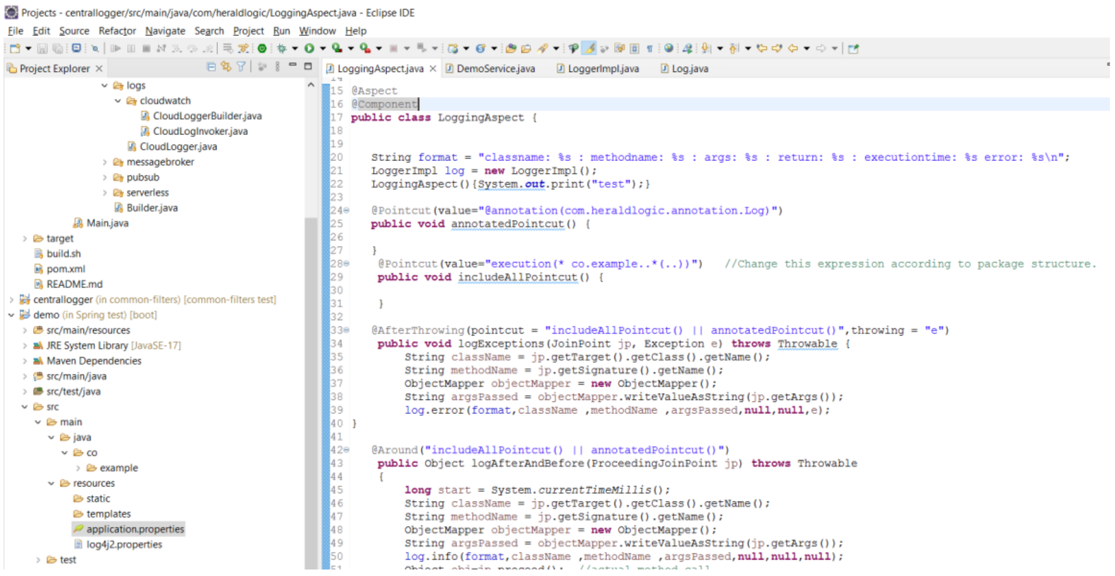
```java
package com.hl.util.logs.cloudwatch;

 * @author Purvansh Parmar
import com.hl.util.Builder;

public class CloudLoggerBuilder implements Builder<CloudLogger> {

    /*
     * Requires logGroupName and logStreamName for writing logs on cloudwatch.
     *
     * */
    private String logGroupName;
    private String logStreamName;

    @Override
    public CloudLogger build() {
        return new CloudLogInvoker(logGroupName, logStreamName);
    }

    public CloudLoggerBuilder logStreamName(String logStreamName) {
        this.logStreamName = logStreamName;
        return this;
    }

    public CloudLoggerBuilder logGroupName(String logGroupName) {
        this.logGroupName = logGroupName;
        return this;
    }

}
```

3.4 CloudLoggerBuilder.java Class to set group name and stream name for Cloudwatch Logs



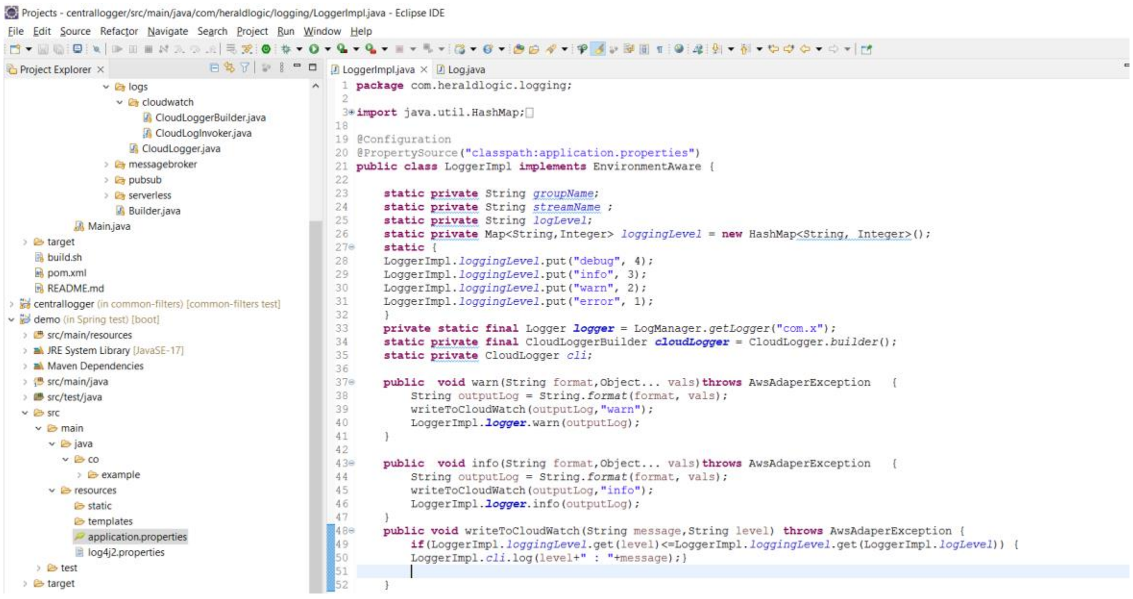
```
appender.MyCustomAppender.policies.time.type = TimeBasedTriggeringPolicy
appender.MyCustomAppender.policies.time.interval = 1
appender.MyCustomAppender.policies.time.modulate = true
appender.MyCustomAppender.strategy.type = DefaultRolloverStrategy
appender.MyCustomAppender.strategy.delete.type = Delete
appender.MyCustomAppender.strategy.delete.basePath = ${basePath}
appender.MyCustomAppender.strategy.delete.maxDepth = 10
appender.MyCustomAppender.strategy.delete.ifLastModified.type = IfLastModified
appender.MyCustomAppender.strategy.delete.ifLastModified.age = 30d
logger.MyCustomAppender.name= com.x
logger.MyCustomAppender.level = info
logger.MyCustomAppender.additivity = false
logger.MyCustomAppender.appenderRef.MyCustomAppender.ref = mycustomappender

appender = RollingFile
#RollingFileAppender name, pattern, path and rollover policy
appender.rolling.type = RollingFile
appender.rolling.name = fileLogger
appender.rolling.fileName= ${basePath}/application.log
appender.rolling.filePattern= ${basePath}/app_%d{yyyyMMdd}.log.gz
appender.rolling.layout.type = PatternLayout
appender.rolling.layout.pattern = [%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n
appender.rolling.policies.type = Policies

# RollingFileAppender rotation policy
appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
appender.rolling.policies.size.size = 10MB
appender.rolling.policies.time.type = TimeBasedTriggeringPolicy
appender.rolling.policies.time.interval = 1
appender.rolling.policies.time.modulate = true
appender.rolling.strategy.type = DefaultRolloverStrategy
appender.rolling.strategy.delete.type = Delete
appender.rolling.strategy.delete.basePath = ${basePath}
appender.rolling.strategy.delete.maxDepth = 10
appender.rolling.strategy.delete.ifLastModified.type = IfLastModified

# Delete all files older than 30 days

appender.rolling.strategy.delete.ifLastModified.age = 30d

# Configure root logger
rootLogger.level = debug
rootLogger.appenderRef.rolling.ref = fileLogger
```

3.5 Setting Properties such as file type, file pattern and log level for the rolling file appender in Log4J.properties file



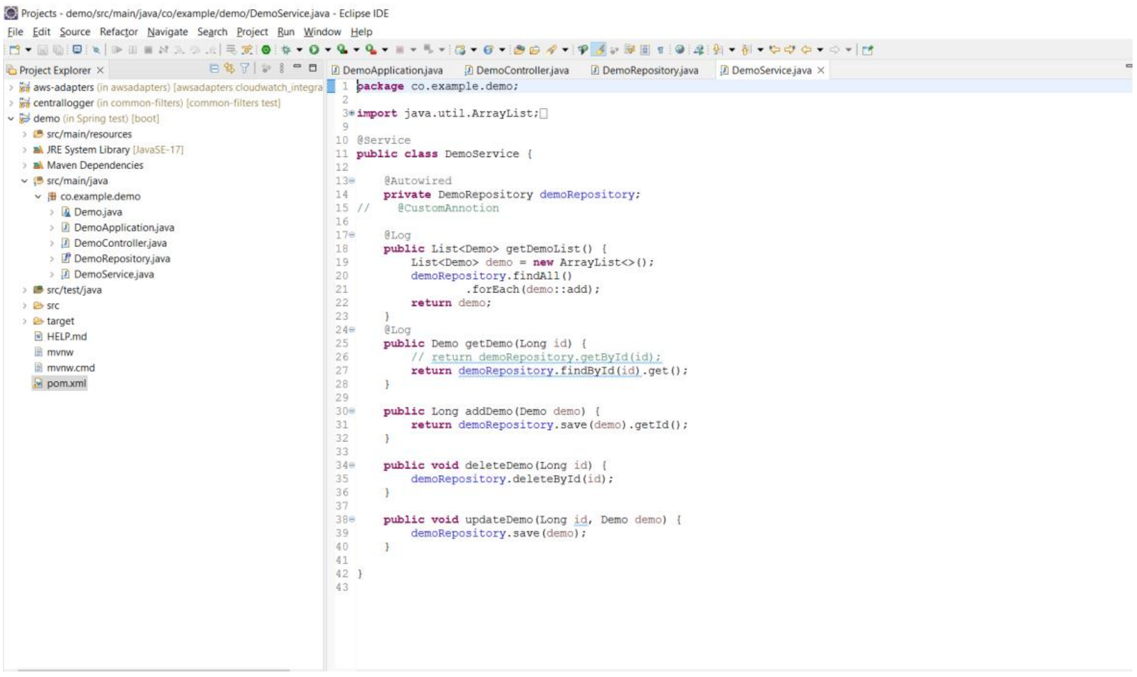
```
@Aspect
@Component
public class LoggingAspect {


   String format = "classname: %s : methodname: %s : args: %s : return: %s : executiontime: %s error: %s\n";
   LoggerImpl log = new LoggerImpl();
   LoggingAspect(){System.out.print("test");}

   @Pointcut(value="@annotation(com.heraldlogic.annotation.Log)")
   public void annotatedPointcut() {

   }
    @Pointcut(value="execution(* co.example..*(..))")   //Change this expression according to package structure.
    public void includeAllPointcut() {

    }

   @AfterThrowing(pointcut = "includeAllPointcut() || annotatedPointcut()",throwing = "e")
    public void logExceptions(JoinPoint jp, Exception e) throws Throwable {
        String className = jp.getTarget().getClass().getName();
        String methodName = jp.getSignature().getName();
        ObjectMapper objectMapper = new ObjectMapper();
        String argsPassed = objectMapper.writeValueAsString(jp.getArgs());
        log.error(format,className ,methodName ,argsPassed,null,null,e);
}

   @Around("includeAllPointcut() || annotatedPointcut()")
    public Object logAfterAndBefore(ProceedingJoinPoint jp) throws Throwable
    {
        long start = System.currentTimeMillis();
        String className = jp.getTarget().getClass().getName();
        String methodName = jp.getSignature().getName();
        ObjectMapper objectMapper = new ObjectMapper();
        String argsPassed = objectMapper.writeValueAsString(jp.getArgs());
        log.info(format,className ,methodName ,argsPassed,null,null,null);
```

3.6 LoggingAspect.java class to execute pointcuts and set the format for logs.



```java
package com.heraldlogic.logging;

import java.util.HashMap;

@Configuration
@PropertySource("classpath:application.properties")
public class LoggerImpl implements EnvironmentAware {

    static private String groupName;
    static private String streamName ;
    static private String logLevel;
    static private Map<String,Integer> loggingLevel = new HashMap<String, Integer>();
    static {
    LoggerImpl.loggingLevel.put("debug", 4);
    LoggerImpl.loggingLevel.put("info", 3);
    LoggerImpl.loggingLevel.put("warn", 2);
    LoggerImpl.loggingLevel.put("error", 1);
    }
    private static final Logger logger = LogManager.getLogger("com.x");
    static private final CloudLoggerBuilder cloudLogger = CloudLogger.builder();
    static private CloudLogger cli;

    public  void warn(String format,Object... vals)throws AwsAdaperException   {
        String outputLog = String.format(format, vals);
        writeToCloudWatch(outputLog,"warn");
        LoggerImpl.logger.warn(outputLog);
    }

    public  void info(String format,Object... vals)throws AwsAdaperException   {
        String outputLog = String.format(format, vals);
        writeToCloudWatch(outputLog,"info");
        LoggerImpl.logger.info(outputLog);
    }
    public void writeToCloudWatch(String message,String level) throws AwsAdaperException {
        if(LoggerImpl.loggingLevel.get(level)<=LoggerImpl.loggingLevel.get(LoggerImpl.logLevel)) {
        LoggerImpl.cli.log(level+" : "+message);}

    }
```

3.7 LoggerImpl.java class to initialize file logger and CloudWatch logger and file the logs using log.info(), log.warn() functions.


Projects - demo/src/main/java/co/example/demo/DemoService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
aws-adapters (in awsadapters) [awsadapters cloudwatch_integra
centrallogger (in common-filters) [common-filters test]
demo (in Spring test) [boot]
src/main/resources
JRE System Library [JavaSE-17]
Maven Dependencies
src/main/java
co.example.demo
Demo.java
DemoApplication.java
DemoController.java
DemoRepository.java
DemoService.java
src/test/java
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
DemoApplication.java
DemoController.java
DemoRepository.java
DemoService.java

```java
package co.example.demo;

import java.util.ArrayList;

@Service
public class DemoService {

    @Autowired
    private DemoRepository demoRepository;
//    @CustomAnnotion

    @Log
    public List<Demo> getDemoList() {
        List<Demo> demo = new ArrayList<>();
        demoRepository.findAll()
                .forEach(demo::add);
        return demo;
    }
    @Log
    public Demo getDemo(Long id) {
        // return demoRepository.getById(id);
        return demoRepository.findById(id).get();
    }

    public Long addDemo(Demo demo) {
        return demoRepository.save(demo).getId();
    }

    public void deleteDemo(Long id) {
        demoRepository.deleteById(id);
    }

    public void updateDemo(Long id, Demo demo) {
        demoRepository.save(demo);
    }

}
```

3.8 Demo Application implementing the utility.

3.3 ARCHITECTURE

- **Builder.java class :** It is a template class that builds an object of the specified type. In our project, we are using it for the CloudLogger class object.
- **CloudLogger.java class** : It contains a builder function which returns a new CloudLoggerBuilder class object and also instantiates a log function that takes a message as an output and is used to log on the CloudWatch.
- **CloudLoggerBuilder.java class :** It sets the value of logGroupName and logStreamName as provided by the user, And also creates a CloudLoggerInvoker class object with groupname and streamname as parameters.
- **CloudLoggerInvoker.java Class :** This class implements cloudlogger. This has actual implementation of log method. This method allows us to send logs to cloudwatch, with particular Stream Name and Log group name.Log group name allows us to setup a environment for all related services whereas stream name helps us to divide logs generated by each microservice. This architecture allows to modularize logs, but maintain relationship.
- **LoggingAspect.java class** : In this class, we declare all the pointcuts. We have used custom annotation as well as implemented pointcuts on the package functions. The expression can be changed as per the package structure. As per the requirements of the project, we have used two advices, Around and AfterThrowing. “Around” advice can perform custom behaviour before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception. “AfterThrowing” advice to be executed if a method exits by throwing an exception.
- **LoggerImpl.java class :** It initializes file logger and CloudWatch logger. It consists of different functions for different log levels, info, warn, error, debug. To write to CloudWatch, WriteToCloudWatch() function is used which uses Log() function with cloudLogger class object. The logs from the file logger are saved in a Rolling file, whose properties are set in Log4J.properties file. The order of logging levels is followed in CloudWatch logs too.
- **Log.java class :** It is a custom annotation created, and set as a pointcut in LoggingAspect class, it can be implemented on any method with this annotation.

- In **Application.properties**, we set the group name, stream name and logging level for cloudwatch logs. From there we retrieve them in LoggerImpl class, and use the log() function accordingly.

Annotated Pointcut :

```
@Pointcut(value="@annotation(com.heraldlogic.annotation.Log)")
  public void annotatedPointcut() { }

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Log {
//Add time required to run
}
```

Description : Pointcut set to custom annotation, named LoggingAnnotation.
The @Target annotation tells us where our annotation will be applicable. Here we are using ElementType.Method, which means it will only work on methods.
And @Retention just states whether the annotation will be available to the JVM at runtime or not.

IncludeAll Pointcut :

```
@Pointcut(value="execution(* com.example..*(..))")   //Change this expression
according to package structure.
   public void includeAllPointcut() { }
```

Description : Pointcut applied to all the methods from the package inside com.example…. Irrespective of the return type and the arguments.

METHODS -

Log Function :

```
@Around("includeAllPointcut() || annotatedPointcut()")
   public Object logAfterAndBefore(ProceedingJoinPoint jp) throws Throwable
   {
```

Description : Around advice executed. It is applied before and after calling the actual method.
It logs the arguments before and after the function executes. It uses Proceeding joinpoint object, which is used to get class name, method name, and the arguments, mapped using object mapper.
Before the execution of the method on which pointcut is applied, these all are logged.
After the method execution, along with these three the time taken to execute the method is also logged.
When an exception is thrown, it goes to the original method that throws the exception.
//objectmapper

```
Object obj=jp.proceed();  //actual method call
```

Description : proceed executes the method on which the pointcut is applied. And after the execution of the method, it stores the returning object in the obj.

Parameters :

jp : Proceeding JoinPoint Object
System.currentTimeMillis() : It returns the current time in milliseconds.

Log Exception Function :

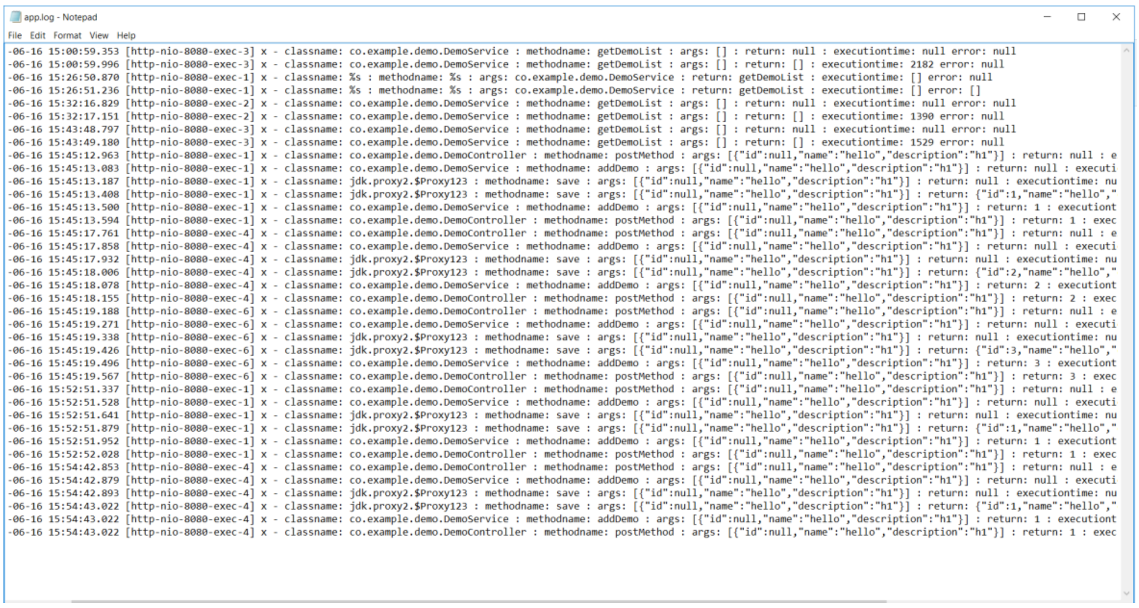
```
@AfterThrowing(pointcut = "includeAllPointcut() || annotatedPointcut()",throwing = "e")
    public void logExceptions(JoinPoint jp, Exception e) throws Throwable { }
```

Description : AfterThrowing declares the throws advice. It is applied if the actual method throws an exception.
When the exception is thrown, this method logs the error.

Chapter 4

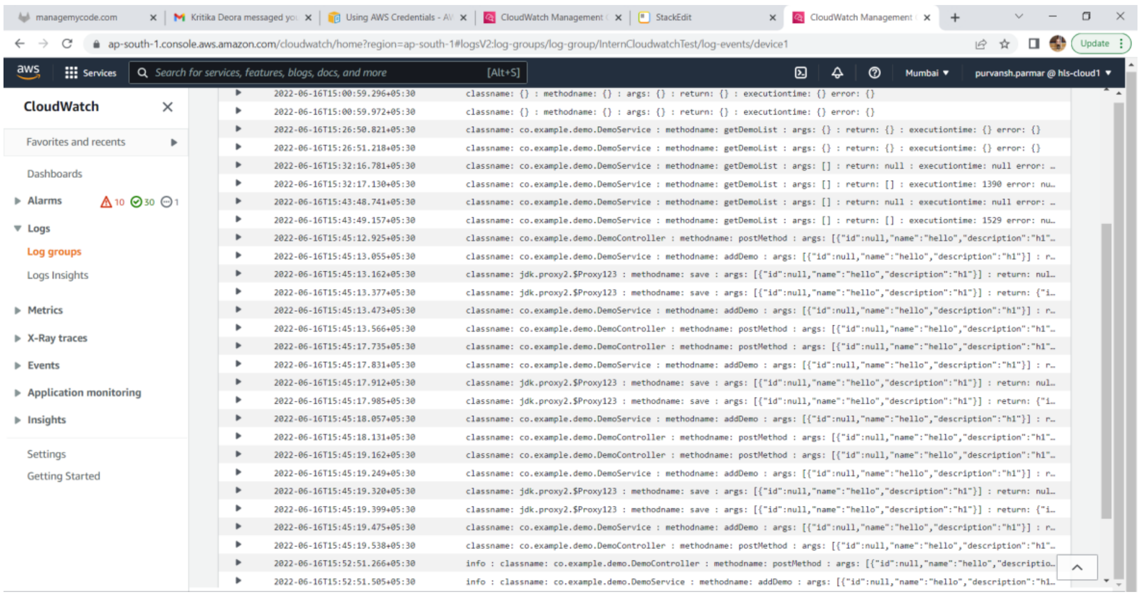
RESULTS

4.1 LOGS CREATED



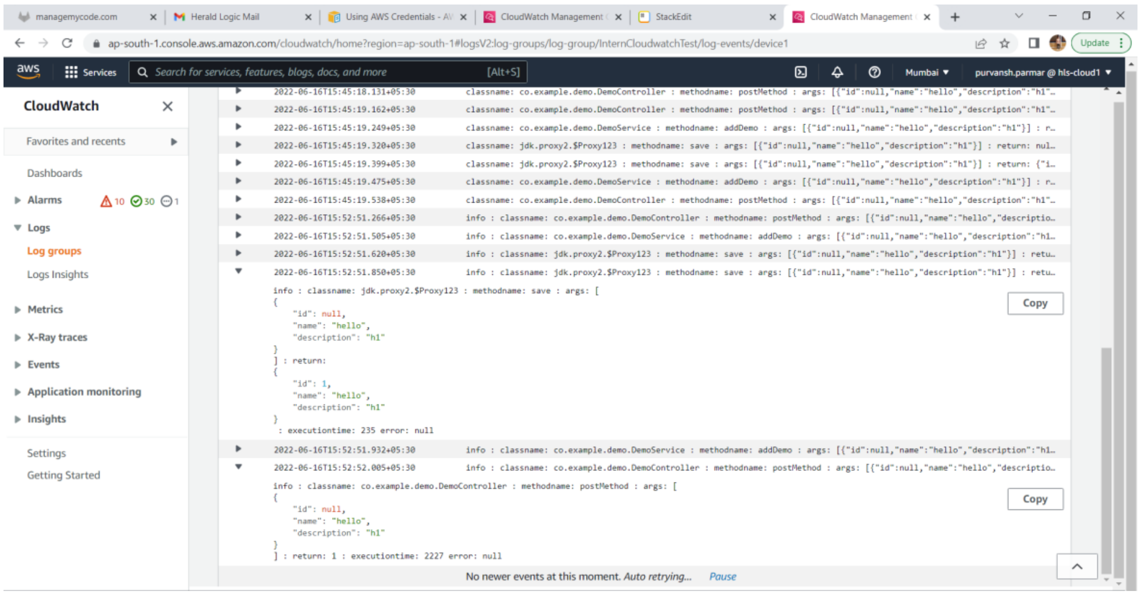
```
-06-16 15:00:59.353 [http-nio-8080-exec-3] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: null : executiontime: null error: null
-06-16 15:00:59.996 [http-nio-8080-exec-3] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: [] : executiontime: 2182 error: null
-06-16 15:26:50.870 [http-nio-8080-exec-1] x - classname: %s : methodname: %s : args: co.example.demo.DemoService : return: getDemoList : executiontime: [] error: null
-06-16 15:26:51.236 [http-nio-8080-exec-1] x - classname: %s : methodname: %s : args: co.example.demo.DemoService : return: getDemoList : executiontime: [] error: []
-06-16 15:32:16.829 [http-nio-8080-exec-2] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: null : executiontime: null error: null
-06-16 15:32:17.151 [http-nio-8080-exec-2] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: [] : executiontime: 1390 error: null
-06-16 15:43:48.797 [http-nio-8080-exec-3] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: null : executiontime: null error: null
-06-16 15:43:49.180 [http-nio-8080-exec-3] x - classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: [] : executiontime: 1529 error: null
-06-16 15:45:12.963 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : e
-06-16 15:45:13.083 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executi
-06-16 15:45:13.187 [http-nio-8080-exec-1] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executiontime: nu
-06-16 15:45:13.408 [http-nio-8080-exec-1] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"id":1,"name":"hello","
-06-16 15:45:13.500 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : executiont
-06-16 15:45:13.594 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : exec
-06-16 15:45:17.761 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : e
-06-16 15:45:17.858 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executi
-06-16 15:45:17.932 [http-nio-8080-exec-4] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executiontime: nu
-06-16 15:45:18.006 [http-nio-8080-exec-4] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"id":2,"name":"hello","
-06-16 15:45:18.078 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: 2 : executiont
-06-16 15:45:18.155 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: 2 : exec
-06-16 15:45:19.188 [http-nio-8080-exec-6] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : e
-06-16 15:45:19.271 [http-nio-8080-exec-6] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executi
-06-16 15:45:19.338 [http-nio-8080-exec-6] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executiontime: nu
-06-16 15:45:19.426 [http-nio-8080-exec-6] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"id":3,"name":"hello","
-06-16 15:45:19.496 [http-nio-8080-exec-6] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: 3 : executiont
-06-16 15:45:19.567 [http-nio-8080-exec-6] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: 3 : exec
-06-16 15:52:51.337 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : e
-06-16 15:52:51.528 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executi
-06-16 15:52:51.641 [http-nio-8080-exec-1] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executiontime: nu
-06-16 15:52:51.879 [http-nio-8080-exec-1] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"id":1,"name":"hello","
-06-16 15:52:51.952 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : executiont
-06-16 15:52:52.028 [http-nio-8080-exec-1] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : exec
-06-16 15:54:42.853 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : e
-06-16 15:54:42.879 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executi
-06-16 15:54:42.893 [http-nio-8080-exec-4] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: null : executiontime: nu
-06-16 15:54:43.022 [http-nio-8080-exec-4] x - classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"id":1,"name":"hello","
-06-16 15:54:43.022 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : executiont
-06-16 15:54:43.022 [http-nio-8080-exec-4] x - classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"}] : return: 1 : exec
```

4.1 Logs created using file logger



```
2022-06-16T15:00:59.296+05:30    classname: {} : methodname: {} : args: {} : return: {} : executiontime: {} error: {}
2022-06-16T15:00:59.972+05:30    classname: {} : methodname: {} : args: {} : return: {} : executiontime: {} error: {}
2022-06-16T15:26:50.821+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: {} : return: {} : executiontime: {} error: {}
2022-06-16T15:26:51.218+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: {} : return: {} : executiontime: {} error: {}
2022-06-16T15:32:16.781+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: null : executiontime: null error: …
2022-06-16T15:32:17.130+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: [] : executiontime: 1390 error: nu…
2022-06-16T15:43:48.741+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: null : executiontime: null error: …
2022-06-16T15:43:49.157+05:30    classname: co.example.demo.DemoService : methodname: getDemoList : args: [] : return: [] : executiontime: 1529 error: nu…
2022-06-16T15:45:12.925+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:13.055+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:13.162+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: nul…
2022-06-16T15:45:13.377+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"i…
2022-06-16T15:45:13.473+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:13.566+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:17.735+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:17.831+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:17.912+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: nul…
2022-06-16T15:45:17.985+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"i…
2022-06-16T15:45:18.057+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:18.131+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:19.162+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:19.249+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:19.320+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: nul…
2022-06-16T15:45:19.399+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"i…
2022-06-16T15:45:19.475+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:19.538+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:52:51.266+05:30    info : classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","descriptio…
2022-06-16T15:52:51.505+05:30    info : classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1…
```

4.2 Logs created using CloudWatch Logger



```
2022-06-16T15:45:18.131+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:19.162+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:45:19.249+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:19.320+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: nul…
2022-06-16T15:45:19.399+05:30    classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : return: {"i…
2022-06-16T15:45:19.475+05:30    classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1"}] : r…
2022-06-16T15:45:19.538+05:30    classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","description":"h1"…
2022-06-16T15:52:51.266+05:30    info : classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","descriptio…
2022-06-16T15:52:51.505+05:30    info : classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1…
2022-06-16T15:52:51.620+05:30    info : classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : retu…
2022-06-16T15:52:51.850+05:30    info : classname: jdk.proxy2.$Proxy123 : methodname: save : args: [{"id":null,"name":"hello","description":"h1"}] : retu…
info : classname: jdk.proxy2.$Proxy123 : methodname: save : args: [
{
    "id": null,
    "name": "hello",
    "description": "h1"
}
] : return:
{
    "id": 1,
    "name": "hello",
    "description": "h1"
}
 : executiontime: 235 error: null
2022-06-16T15:52:51.932+05:30    info : classname: co.example.demo.DemoService : methodname: addDemo : args: [{"id":null,"name":"hello","description":"h1…
2022-06-16T15:52:52.005+05:30    info : classname: co.example.demo.DemoController : methodname: postMethod : args: [{"id":null,"name":"hello","descriptio…
info : classname: co.example.demo.DemoController : methodname: postMethod : args: [
{
    "id": null,
    "name": "hello",
    "description": "h1"
}
] : return: 1 : executiontime: 2227 error: null
No newer events at this moment. Auto retrying...    Pause
```

4.3 CloudWatch Logs

Chapter 5

CONCLUSION AND FUTURE SCOPE

The utility centralLogger resolves a lot of issues, like being a maven dependency it can be used with many services. Also, it makes tracing errors easier than in distributed. It is useful in transaction based microservices, by keeping track of all the transactions in one place.

The purpose of a centralized logging system is to streamline and automate the process of manual log management, reducing the time and effort often involved in maintaining a massive repository of log data.

Also it gives us an advantage of using different files for keeping record of different log levels.

We can convert this utility into multi- threaded to optimise and utilise the bandwidth in most cost effective way. We can customise the utility in a manner that the user can work with the database or logging platform of their choice.

References

[1] Centralized logging using aspect oriented programming and spring, https://solocoding.dev/blog/eng_spring_centrlize_logging_with_aop, accessed 07/05/2022.

[2] Configuring Spring Boot to use Gson instead of Jackson, by Rajiv Singh https://www.callicoder.com/configuring-spring-boot-to-use-gson-instead-of-jackson/, accessed 15/05/2022.

[3] Java Logging Framework, https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaLogging.html, accessed 27/05/2022.

[4] Pointcut API in Spring**,** https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch09s02.html, accessed 17/05/2022.

[5] Creating a Custom Log4j2 Appender**,** https://www.baeldung.com/log4j2-custom-appender, accessed 25/05/2022.

[6] How to useinitializemethodinorg.springframework.boot.logging.LoggingSystem, https://www.tabnine.com/code/java/methods/org.springframework.boot.logging.LoggingSystem/initialize, accessed 02/06/2022.

[7] log4j2 properties file for a custom appender**,** https://stackoverflow.com/questions/55011958/log4j2-properties-file-for-a-custom-appender, accessed 07/06/2022.

[8] Search log data using filter patterns, https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/SearchDataFilterPattern.html, accessed 15/06/2022.

[9] Java Logging Best Practices, by Rafal Kuc, https://sematext.com/blog/java-logging-best-practices/, accessed 27/05/2022.

[10] Elements of data-flow diagrams, https://www.cs.uct.ac.za/mit_notes/software/htmls/ch06s05.html, accessed 19/06/2022.