\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Project workflow**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Channel (Upstream)->POL->(GPPSP,EPIC,ACI Gateway)->Downstream SBJ APIS->Channel

EPIC/ACI/GPPSP(Gateway)->IBM MQ->POL/RTPIL->CBIL/HUB/MTE/SA-> EPIC/ACI/GPPSP(Gateway)

Channel->HSBC Net, Connect, Heritage Connect, XAPI, WPB,GLCM,GPI,RBWM,

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**USRTP-(US real time Payments)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GPI/RBWM-(WPB/PPE)NET(Membership check): channel->POL->GPPSP

GLCM channels(Pani001v03)NET payment initiation: NET->POL->GPPSP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SLDD(Srilanka demand draft)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**MYRFP->(MY Real time file processing)-need to cross check**

**HSBC** need to support industry initiative which has proposed significant changes to existing payments schemes such as SWIFT (cross boarder payments system), real time payments(RTP), ACH payments, SWIFT is also enabling for ISO 20022 messagiung from NOV2022 with CBPR+guidelines and other RTGS schemes are also enhancing themselves for ISO 2022/HVPS+guidelines.

HSBC is decide to roll out technology upgrade of its end to end architecture within RTP and HV payments stream. HSBC is aiming to do technology upgrade and migrate the existing Mule based application for various RTP, ACH sites to java/SBJ(Mule to IKP upgrade)

Outgoing->

Channels(CONNECT->SFTP(control+M job, file watcher, schedular), NET)->HUB/ACI->POL

Channel->pol->ACI

ACI\/gateway->POL->Channels

Inward flow:

ACI->POL->Channel

Message format:

1. Pain13v7->pain13- to pain14
2. Pain14v7->Pain13 to CBIL/HUB/Json
3. Camt007->camt007 to Camt025
4. Camt0014->
5. Camt025
6. Admi002->Bespoke json/Camt025

HSBC Connect->Control+M, File watcher, file schedular, linus shell script,SFTP🡪POL(Proxy,router->initator API->gateway/downstream-CBIL/HUB->Channel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Development:
   1. JDK-11,17,21----features related to project. (All features version wise)
   2. Junit, Mockito, power mock—search to updated features.
   3. Settings.xml—for maven—all required activities
   4. Maven—features—commands for run debug etc.
   5. Gradle-features, commands for run, debug etc.
   6. IntelliJ—frequently used shot cut commands, project settings, maven settings, jdk settings, compiler settings etc.
   7. XML accept on controller—all configurations.
   8. Docs to xml, xml to docs, currency and value pojo creation-conversion
   9. Api call—httpclient, 3 times retries httpclient, rest template, use jar to call other api,webcliet etc.—all concept
   10. Sonar fixes—challenges-need to know basics concept, hap pipeline.
   11. Sona type/similar repository—version dependencies of jar –optimization and its waiver off
   12. SAST, DAST, checkmarks—fix and its waiver off
   13. Codigo🡪code overview and analysisi
   14. Proxy API
   15. Code review
   16. Git, GitHub and its command to commit, push,rollback etc.
   17. Git tag
   18. Git merge->one branch to another branch
   19. Java 8, streams api programming-in depth
2. Deployment:
   1. jenkin
   2. ikp portal—
   3. shp portal—for amazon
   4. kubernates
   5. docker
   6. pipeline—hap pipeline—all pipelines
   7. Dev, UAT, Preprod, prod –deployment process
   8. SAST, DAST,checkmarx, sonar—concept to generate report
3. Project Readiness:
   1. Sonar, sonatype, sast,dast, checkmarx—waiver off—process
   2. Control +M, cluster, yellow belt, orange belt
   3. Grafana
   4. Splunk
   5. MDC logger
   6. Way to create jar after the restapi build—dependencies
   7. Partner handshaking
   8. PVT validation
   9. Cluster-making
4. BA-work
   1. confluence page
   2. jira board
   3. user stories
   4. jira cleaning activity
   5. label, version,-pointers, spic, poc, sprint---ba analysis to create jira for userstories, spills over
   6. hdlc design
   7. Pain8, pacs 2, all details format of the payload
5. Scrum master
   1. standup –work related to scrummaster
   2. –need to discuss and add its
6. Mule API
   1. Mule api analysis
   2. Work flow
   3. Connecter
   4. Api call
   5. Dwl
   6. Logger
   7. Print msg
   8. Payload
   9. Working flow
7. IBM MQ
   1. Queue—creation
   2. Q-Configuration
   3. Q-certificates.jks
   4. Q.MQSLSocket factory
   5. JMS listenere
   6. All required dependencies
   7. Challenges in MQ structure
8. Testing activity
   1. SIT1
   2. SIT2
   3. E2e
   4. PT
   5. PVT
   6. Regression
   7. Automation
   8. Gherking->Cucumber
9. Project Architect
10. IT PM
11. Client manager—work
12. Billing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[java - Configure Sonar to exclude files from Maven pom.xml - Stack Overflow](https://stackoverflow.com/questions/21425012/configure-sonar-to-exclude-files-from-maven-pom-xml)

In the log4j2 **PatternLayout** pattern:

* **%d{HH:mm:ss.SSS}**: Prints the date and time in the specified format (**HH:mm:ss.SSS** indicates hours, minutes, seconds, and milliseconds).
* **[%t]**: Prints the thread name.
* **%-5level**: Prints the log level left-aligned with a width of 5 characters. The **-** ensures left alignment.
* **%logger{36}**: Prints the logger name with a maximum width of 36 characters. This can help in formatting the output and maintaining a consistent width for readability.
* **%X{id}**: Prints the value associated with the MDC (Mapped Diagnostic Context) key "id". The MDC allows you to enrich log messages with contextual information, and **%X{key}** retrieves the value of the specified key.
* **%msg**: Prints the log message.
* **%n**: Prints a newline character, indicating the end of the log entry.

So, in the context of **%msg**, it represents the actual log message that you have provided when logging a statement. For example:

javaCopy code

logger.info("This is your log message");

To implement custom logging with a specific key using log4j in a Spring Boot application, you can follow these steps:

**Step 1: Add Dependencies**

Include the necessary dependencies in your **pom.xml** file:

xmlCopy code

<!-- log4j2 dependencies --> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-log4j2</artifactId> </dependency>

**Step 2: Configure log4j2**

Create a **log4j2.xml** file in the **src/main/resources** directory. Customize the configuration to include your specific key.

Example **log4j2.xml**:

xmlCopy code

<?xml version="1.0" encoding="UTF-8"?> <Configuration status="WARN"> <Appenders> <Console name="Console" target="SYSTEM\_OUT"> <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %X{yourKey} - %msg%n"/> </Console> </Appenders> <Loggers> <Root level="info"> <AppenderRef ref="Console"/> </Root> </Loggers> </Configuration>

Replace **yourKey** in **%X{yourKey}** with the key you want to use in the MDC.

**Step 3: Use MDC in Your Code**

In your Spring Boot service or component, you can set the value for your specific key in the MDC before logging:

javaCopy code

import org.apache.logging.log4j.LogManager; import org.apache.logging.log4j.Logger; import org.apache.logging.log4j.ThreadContext; import org.springframework.stereotype.Service; @Service public class YourService { private static final Logger logger = LogManager.getLogger(YourService.class); public void yourMethod() { // Set your specific key in the MDC ThreadContext.put("yourKey", "your-custom-value"); // Log your message logger.info("Your log message with custom key"); // Clear the MDC context ThreadContext.clearMap(); } }

**Step 4: Run Your Application**

When you run your Spring Boot application, log messages will be outputted to the console with the specified key included.

This custom log configuration allows you to include specific contextual information associated with your key in the log messages. Adjust the configuration and code according to your specific needs and use ca

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

Handshakee-with channels

Handshake with Cibil/hub

Handshake with Imps

Handshake with sanctions

OAT deployment

PT-support

Scans/Perform sonar, cyberflows and fix issues

SIT reports=based on the code create code tags release

API security implementation

Splunk Onboarding

Grafana/observability

Pipeline/configuration related changes

Control setup

E2e with TDS

No units related deployment issues

Get signed off from -centrl security team

Pipeline—Merge configs and secrets