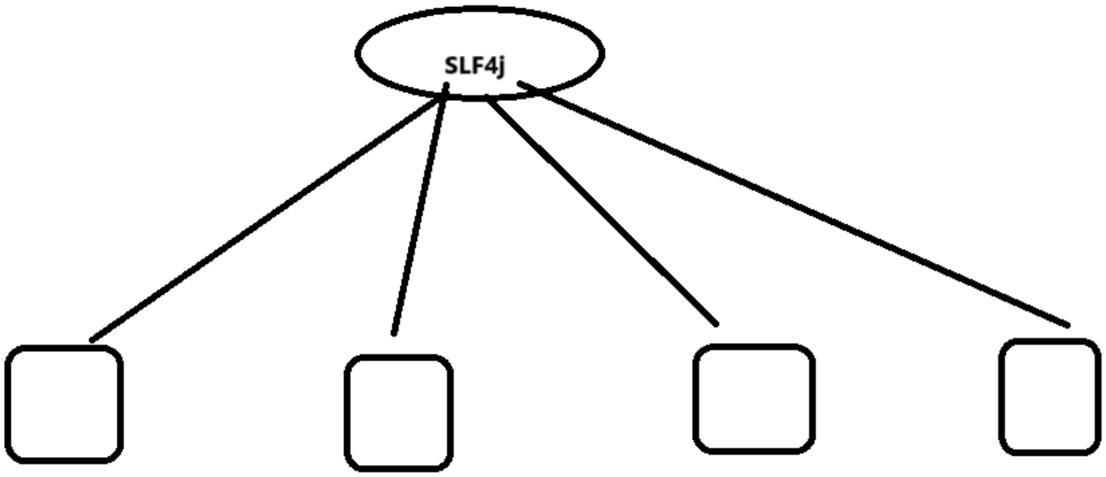
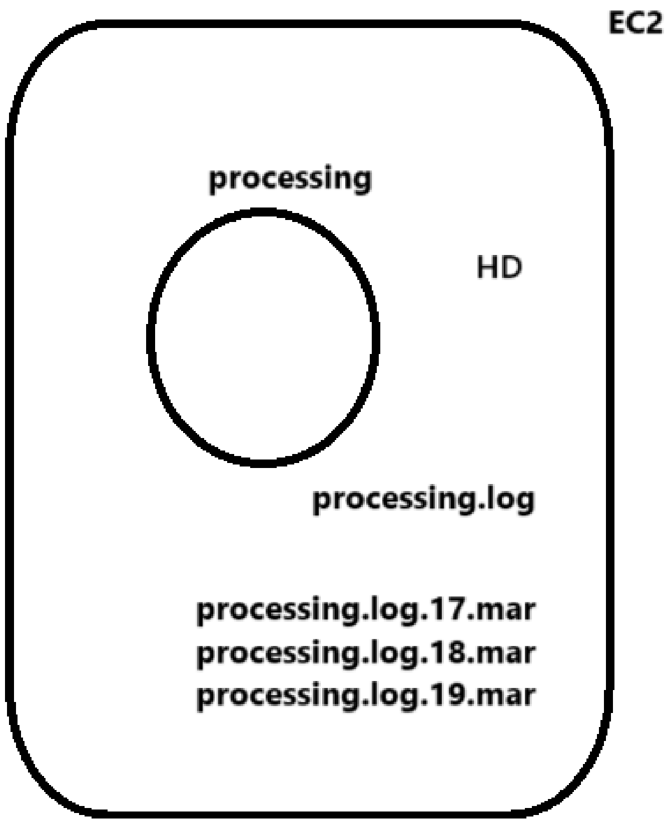
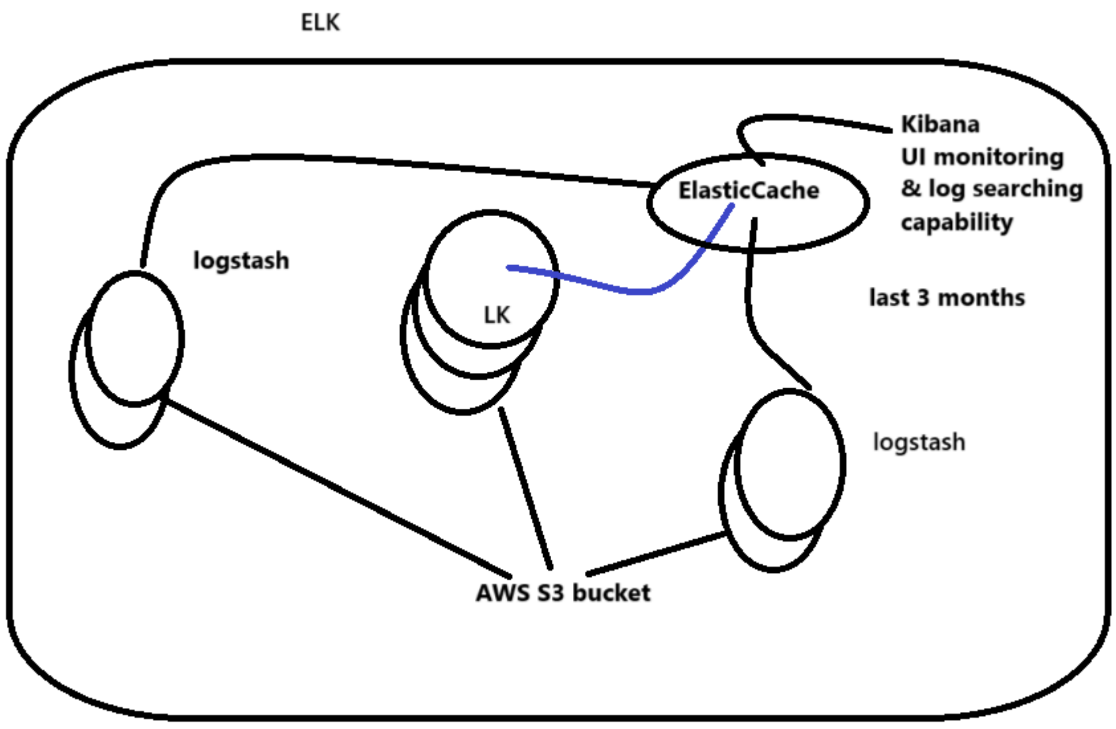
DIAGRAM

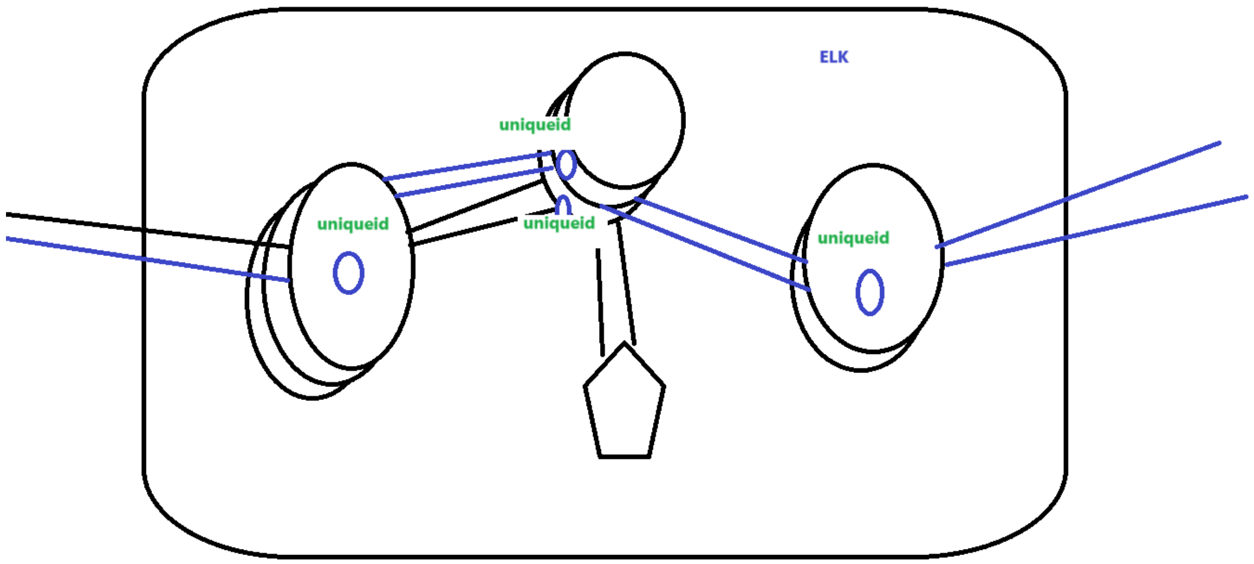


Log rolling





TRACEId & SPANID - dritributed tracking



LIVE NOTES

One important aspect which is used in every project.

Logging!

Week7:
- Logging
- UnitTesting
- AWS
  - SecretsManager
- ActiveMQ

Logging!

We use below object for logging purpose
org.slf4j.Logger

S.o.p using Logger.
3 Major:
- in s.o.p focus is only on message. However we need mandatory additional information when logging. timestamp, thread, which class
-

```
---() {

    log-statement - (1)
    log.info()

    for() - 10times {
        log-statement - (10)
        log.debug()

        for() - 10time {
            log-statement - (100)
            log.trace(-)
        }
    }


}
```

we tell developer,log enough information, so you know what is happening..
developers started detailed logging..
in PROD don't log too much - excess information. Only required information should be in PROD

=> we need this information for debugging
=> In non-prod env, have the 100 line logging, but not in prod

=> We need to write log statements, however some statements should be printed in PROD, and some should be only till non-prod. With s.o.p its not possible.

----

3. It logs to console

    all logs should be written in files

    controller.log (PController)
    error.log all errors
    http.log HttpServiceEngine
    application.log

=====

if not s.o.p, then what we use.

Multiple logging frameworks, logging libraries...

log4j
log4j2 Logger
    org.log4j2.sdafas
logback
    org.logback.sdssd
-
-
-

- Whatever library you use for logging, it would be impossible to change in future, since you would need to update, each & every java class.
- learn how to log with each library.

Spring Boot:
    SLF4J with Logback

How to do logging using Slf4J

```
System.out.println()

System {
        PrintStream out
}

PrintStream {
        public void println(String x) {
        if (getClass() == PrintStream.class) {
            writeln(String.valueOf(x));
        } else {
            synchronized (this) {
                print(x);
                newLine();
            }
        }
    }
}
```

object call some method.

org.slf4j.Logger => java class
    info(message)

How to work with Logger object?

1. Either you manually crate logger object
    org.slf4j
    private Logger logger = LoggerFactory.getLogger(PaymentController.class);

2. Use Lombok @Slf4j
    lombok.extern.slf4j.Slf4j;
    internally Lombok creates an object of Logger & gives you variable reference as "log"

log levels - there are 5 log levels to work with (in SLF4J)
    TRACE
    DEBUG
    INFO
    WARN

ERROR

In PROD, some external configuration set to INFO level
all log statements written using info & above will be printed.

there are several methods (log levels) we can use in project.
however whether to print a statement or not depends on some external configuration.

Non-PROD - DEBUG level
DEBUG, INFO, WARN, ERROR

PROD - INFO level
INFO, WARN, ERROR

==> logger object provides multiple methods for logging. However whether to print those statements depends on external configuration.

the default external config (log level) is set to INFO, so all the statements with info(), warn(), error() will be printed.

==> how to change the external configuration
- "root" configuration
generic configuration applicable to entire code
to your project code, & also all the jars which are there in class path.
default root config is INFO. All "info" & above(warn, error) statements in your code & in jar files will be printed.
if you change root config to TRACE, then trace & above from your code & all jar files will be printed.
logging.level.root=TRACE

We don't want to know internal details about libraries so keep root config as INFO only.
logging.level.root=INFO

- "package level" configuration
application specific configuration
logging.level.com.hulkhiretech=DEBUG

keep root to INFO, & for your application logger, setup based on your package.
non-prod: DEBUG

prod: INFO

==> When to use which log level

TRACE
- too much granular information
- even in non-prod, default DEBUG will run, so TRACE will not be visible.
- which you are testing in non-prod, if you want to look for trace statements, then explicitly configure your package level configuration to TRACE

DEBUG
You want this only in non-prod

INFO
Should give you valuable information(for debugging) about the request being processed
it will be printed in PROD
method entry & response being returned, log in info level

WARN
if you feel, something is warning..
log.warn

ERROR
log.error
error has happened.. exception

more detailed logging configuration you would apply in some xml file.
/src/main/resources
logback-spring.xml

```
#logging.level.root=INFO
#logging.level.com.hulkhiretech=DEBUG
```

```
<root level="INFO">
<logger name="com.hulkhiretech" level="TRACE"
```

define where you want to append (write) the log statements

    CONSOLE
    FILE
define <appender>
and while setting up root or logger, define
    <appender-ref ref="FILE" />

what log pattern should be there.

```
<pattern>[ %level ] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] [${springAppName}] %logger -
%msg%n</pattern>
```

===
how to do env specific logging?

PROD - INFO mode
Non-PROD
    UAT - INFO
    QA - DEBUG
    DEV - DEBUG

in XML you setup to INFO logger.
    for both root & package logger configuration

```
        <logger name="com.hulkhiretech" level="INFO" additivity="false">
      <appender-ref ref="FILE" />
      <appender-ref ref="CONSOLE" />
   </logger>
```

in env specific property file change package logger configuration based on need of your project.

```
logging.level.com.hulkhiretech=DEBUG
```

====

writing to FILE

```
<file>@log4j2.path@/${springAppName}.log</file>
```

@log4j2.path@

spring.profiles.active=@spring.profiles.active@

go to the log file from mobaxterm

=> tail -f <filename>
    gives effect like console

    ctrl + c

since you will work with log files, so you need to know vi editor.

vi filename
Shift + G
    - end of file
:1
    - take to the first line

search for pattern & find previous
    go to last time (shift + g)
    ?<pattern>
    enter
    n (previous)

search for pattern & find next
    go to 1st line (:1)
    /<pattern>
    enter
    n (next)

esc :q!
    quit without saving

```
tail -f payment-processing-service.log | grep 'calling initiatePayment'
```

====

we cannot keep writing logs to the same file for long duration... the size will become so huge, we cannot even open the log files.

we need rolling file appender

====
On daily basis new log file will be created & logs of that day will be saved in that file.
====

D:\ctws\feb25ct\logs\payment-processing-service

17Mar 00:00:00
payment-processing-service.log
    empty data.
that day all logs will be processed & added to this file

17Mar EOD: 23:59:59
payment-processing-service.log.17-mar-25

new file payment-processing-service.log
18Mar 00:00:00

payment-processing-service.log.18-mar-25
payment-processing-service
    data for 19Mar

D3...

D7

D8
D9

we run this system in EC2 with some HD capacity.
    clients might say we need to maintain all logs forever.

Maintain only last 7days of data in your HD. remaining all data move into AWS S3 bucket for maintaing for every.

Daily basis rolling
```
<fileNamePattern>@log4j2.path@/${springAppName}.log.%d{yyyy-MM-dd}</fileNamePattern>
```

====

17Mar - 1

18Mar - 0

19Mar - 3

23Mar - 7

    total 6 files

24Mar

=====

ELK

Elasticsearch
Logstash
Kibana

purpose is to make logs available in one place & make it separable without going to EC2 machine

SringBoot2.x
    => spring cloud sleuth

spring boot 3.x

=> micrometer & brave
=> actuator

traceId(unique for the request across all microservices)
spanId (Unique for the given service)

processing:
[67d7dcec6e3107b369d61d79b3dcc71c] [69d61d79b3dcc71c]

stripe-provider
[67d7dcec6e3107b369d61d79b3dcc71c] [57ec978cd0f3b986]

=====

```
[%X{traceId:-}] [%X{spanId:-}]
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
<dependency>
<groupId>io.micrometer</groupId>
<artifactId>micrometer-tracing</artifactId>
</dependency>
<dependency>
<groupId>io.micrometer</groupId>
<artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
```

```
        management.tracing.enabled=true
management.tracing.sampling.probability=1.0
```

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