Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатики и систем управления»

КАФЕДРА «Информационные системы и телекоммуникации»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Разработка OSGi сервиса получения логов от системы Suricata»

Студент группы ИУ3-71 _____ М.Ю. Агапов
(подпись, дата)

Руководитель курсовой работы _____ А.М. Иванов
(подпись, дата)

2017 г.

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатики и систем управления»

КАФЕДРА «Информационные системы и телекоммуникации»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ: «Разработка OSGi сервиса получения логов от системы Suricata»

Студент группы ИУ3-71 _____ М.Ю. Агапов
(подпись, дата)

Руководитель курсового проекта _____ А.М. Иванов
(подпись, дата)

2017 г.

Содержание

1 Техническое задание 3

2 Теоретическая часть 4

2.1 Выявление заинтересованных сторон и их интересов 4

2.2 Suricata IDS 5

2.3 Apache Hadoop 5

3 Конструкторская часть 7

3.1 Выбор технических решений, удовлетворяющих интересам заинтересованных сторон 7

3.2 Структура проекта 8

3.2.1 Класс LogParser 9

3.2.2 Класс FileWorker 9

3.3 Диаграмма компонентов 10

3.4 Диаграмма классов 11

4 Технологическая часть 12

4.1 Запуск анализа данных 12

4.2 Создание бинарной сборки системы 12

4.3 Анализ исходного кода с помощью метрик качества 13

4.4 Анализ зависимостей в коде системы 15

4.5 Тестирование на корректность работы 16

4.6 Тестирование производительности 17

5 Выводы 19

6 Список источников 20

7 Приложение 21

1 Техническое задание

Разработка OSGi сервиса получения логов от системы Suricata:

- изучить соответствующие системы;
- спроектировать интерфейс компонента;
- реализовать компонент;
- спроектировать JUnit тесты, провести тестирование;
- описать требования, конструкцию, особенности сборки и запуска в документации.

2 Теоретическая часть

2.1 Выявление заинтересованных сторон и их интересов

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 1. Заинтересованные стороны и их интересы по отношению к системе

Заинтересованные стороны	**Интересы заинтересованных сторон**
Пользователь	П1 Удобство предварительной настройки и запуска; П2 Высокая скорость обработки логов; П3 Близкая к линейной зависимость времени обработки логов от их количества.
Специалист по анализу данных	Р1 Удобный для постобработки выходной формат представления данных; Р2 Доступ к новым данным в режиме реального времени;

	P3 Использование распространенных систем хранения данных.
Владелец опенсорсного проекта (project owner)	B1 Быстрая и полная передача исходного кода, настроек, документов. B2 Возможность в дальнейшем усложнять реализацию системы.

2.2 Suricata IDS

Suricata – система обнаружения вторжений (англ. Intruder Detection System, IDS), разрабатываемая организацией Open Information Security Foundation (OISF). Исходный код распространяется по лицензии GPLv2.

Suricata изначально разрабатывалась как многопоточная IDS. На текущий момент система эффективно работает с 24 и более процессорами. Кроме того, Suricata может использовать вычисления на стороне GPU.

В системе Suricata реализован функционал предотвращения вторжений (англ. Intrusion Prevention System, IPS) для быстрого запрета нежелательных действий без дополнительных программных продуктов. В Suricata используется два режима IPS: NFQ и AF_PACKET

NFQ IPS режим работает следующим образом:

- Пакет попадает в iptables
- Правило iptables направляет его в очередь NFQUEUE, например iptables -I INPUT -p tcp -j NFQUEUE
- Из очереди NFQUEUE пакеты могут обрабатываться на уровне пользователя, что и делает Suricata
- Suricata прогоняет пакеты по настроенным правилам (rules) и в зависимости от них может вынести один из трех вердиктов: NF_ACCEPT, NF_DROP и самое интересное — NF_REPEAT.
- Пакеты, попадающие в NF_REPEAT, могут быть промаркированы в системе, и направлены обратно в начало текущей таблицы

iptables, что дает огромный потенциал для влияния на дальнейшую судьбу пакетов с помощью правил iptables [1].

2.3 Apache Hadoop

Hadoop – проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из большого количества узлов. Используется для реализации поисковых и контекстных механизмов многих высоконагруженных веб-сайтов. Разработан на Java в рамках вычислительной парадигмы MapReduce, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполняемых на узлах кластера и естественным образом сводимых в конечный результат.

Проект состоит из четырёх модулей – Hadoop Common (связующее программное обеспечение – набор инфраструктурных программных библиотек и утилит, используемых для других модулей и родственных проектов), HDFS (распределенная файловая система), YARN (система для планирования заданий и управления кластером) и Hadoop MapReduce (платформа программирования и выполнения распределённых MapReduce-вычислений).

HDFS (Hadoop Distributed File System) – файловая система, предназначенная для хранения файлов больших размеров, поблочно распределённых между узлами вычислительного кластера. Все блоки в HDFS (кроме последнего блока файла) имеют одинаковый размер, и каждый блок может быть размещен на нескольких узлах, размер блока и коэффициент репликации (количество узлов, на которых должен быть размещён каждый блок) определяются в настройках на уровне файла. Благодаря репликации обеспечивается устойчивость распределенной системы к отказам отдельных узлов. Файлы в HDFS могут быть записаны лишь однажды (модификация не

поддерживается), а запись в файл в одно время может вести только один процесс. Организация файлов в пространстве имён – традиционная иерархическая: есть корневой каталог, поддерживается вложение каталогов, в одном каталоге могут располагаться и файлы, и другие каталоги.

Развёртывание экземпляра HDFS предусматривает наличие центрального узла имён (англ. name node), хранящего метаданные файловой системы и метаинформацию о распределении блоков, и серии узлов данных (англ. data node), непосредственно хранящих блоки файлов. Узел имён отвечает за обработку операций уровня файлов и каталогов – открытие и закрытие файлов, манипуляция с каталогами, узлы данных непосредственно отрабатывают операции по записи и чтению данных. Узел имён и узлы данных снабжаются веб-серверами, отображающими текущий статус узлов и позволяющими просматривать содержимое файловой системы. Административные функции доступны из интерфейса командной строки [2].

3 Конструкторская часть

3.1 Выбор технических решений, удовлетворяющих интересам заинтересованных сторон

В таблице ниже представлены результаты выбора технических решений, позволяющие удовлетворить интересы заинтересованных сторон по отношению к системе.

Таблица 2. Технические решения, удовлетворяющие интересам ЗС

Интересы заинтересованных сторон	**Технические решения**
П1 Удобство предварительной настройки и запуска;	Запуск будет осуществляться со стандартными настройками, либо с указанием файла логов и сервера HDFS.

П2 Высокая скорость обработки логов; П3 Близкая к линейной зависимость времени обработки логов от их количества.	Различные потоки могут обрабатывать различные файлы логов одновременно. Корректность работы будет проверяться в процессе разработки с помощью юнит тестов.
Р1 Удобный для постобработки выходной формат представления данных; Р2 Доступ к новым данным в режиме реального времени; Р3 Использование распространенных систем хранения данных.	В качестве формата выходных данных будет использоваться CSV. Приложение будет ожидать поступления новых логов. Приложение будет записывать преобразованные логи в HDFS.
В1 Быстрая и полная передача исходного кода, настроек, документов. В2 Возможность в дальнейшем усложнять реализацию системы.	Самодокументируемый код с использованием Google Code Style. С системой будет поставляться пример и ReadMe с инструкциями по сборке и запуску примера.

3.2 Структура проекта

Разберем работу написанного проекта. Для начала опишем основные классы: LogParser, FileWorker. На рисунке 1 показано расположение классов в пакетах.

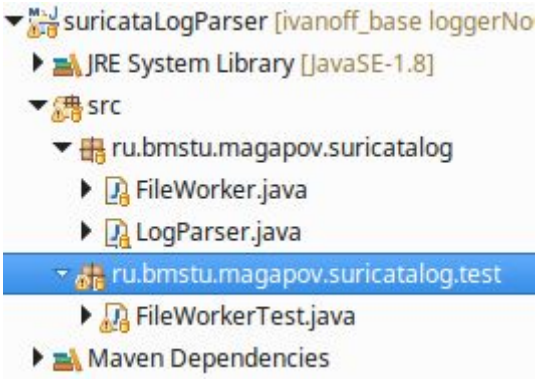

Рисунок 1 – Расположение классов в пакетах.

3.2.1 Класс LogParser

Класс LogParser является классом, который настраивает параметры обработчика логов и запускает его в отдельном потоке. Единственным методом данного класса является публичный статический метод main(String args[]), являющийся точкой входа в программу.

3.2.2 Класс FileWorker

Класс FileWorker является основным классом приложения. Он реализует интерфейс java.lang.Runnable для возможности запускать его в отдельном потоке.

Класс имеет приватные поля: String logFileName – имя файла логов, String hdfsURI – ссылка на HDFS ноду, static final Logger LOG – статическое поле логгера для вывода служебной информации при тестировании.

В классе реализованы 3 конструктора: конструктор по умолчанию, запускающий обработчик логов со стандартными параметрами (файл логов – /var/log/suricata/eve.json; ссылка на HDFS – hdfs://localhost:9000), конструктор, принимающий в качестве параметров имя файла логов и ссылку на HDFS и конструктор, принимающий в качестве параметров только ссылку на HDFS. В последнем случае файл логов остается стандартным.

Метод read() – основной метод данного класса. В нем происходит открытие файла логов, подключение к серверу HDFS, обработка логов и записывание их на сервер HDFS. Метод обрабатывает логи построчно до тех

пор, пока не будет получен сигнал interrupt. В этом случае обработка останавливается.

Вспомогательный метод parseDepth() производит рекурсивный обход лога в глубину, формируя массив значений логов и пополняя заголовочный массив логов новыми ключами.

3.3 Диаграмма компонентов

Предполагается, что данный проект будет использован в рамках более сложной системы, состоящей из IDS/IPS Suricata, Apache Hadoop и алгоритмов анализа преобразованных логов. Изначально Suricata анализирует трафик и формирует логи на основании своих правил. Затем полученные логи поступают на вход разрабатываемого в рамках данного проекта модуля. Модуль преобразует их в формат CSV и записывает их в файловое хранилище Hadoop HDFS. На последнем этапе преобразованные логи обрабатываются пользовательскими алгоритмами обработки больших данных для интерпретации данных в необходимый вид. На рисунке 2 показана диаграмма компонентов системы обработки логов.

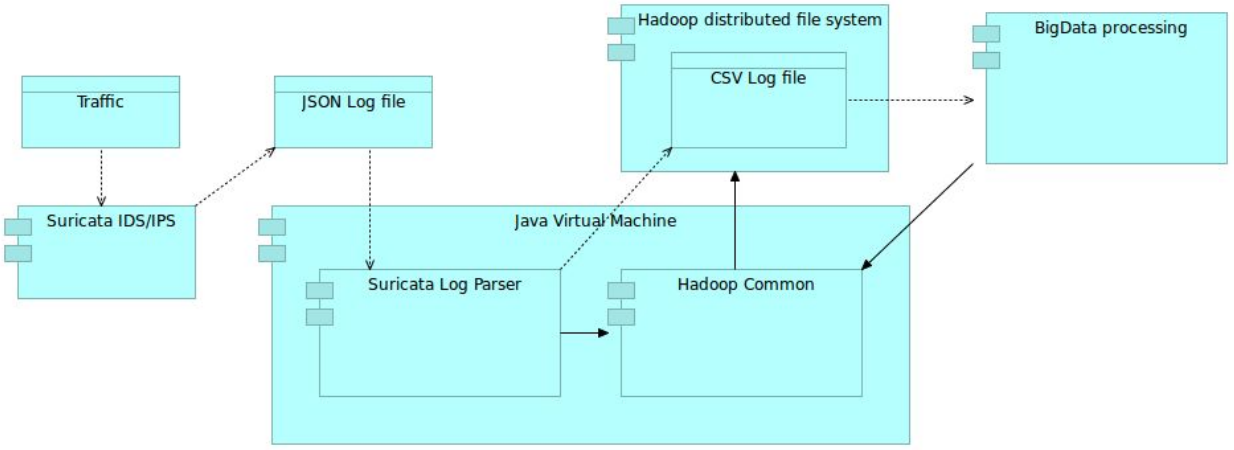

Рисунок 2 – Диаграмма компонентов системы обработки логов

На рисунке 3 представлена диаграмма компонентов нашего проекта. Класс FileWorker реализует интерфейс Runnable, использует средства Hadoop

File System и систему логирования org.slf4j, отправляет результаты в HDFS, запускается из LogParser.

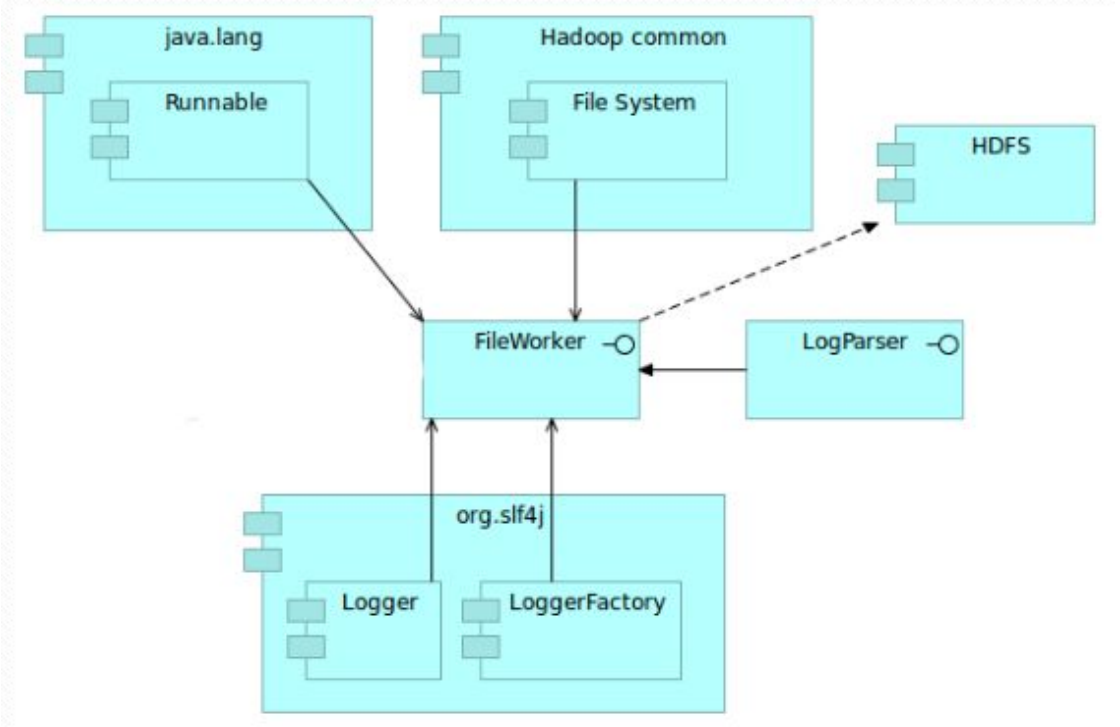

Рисунок 3 – Диаграмма компонентов программы обработки логов

3.4 Диаграмма классов

На рисунке 4 представлена диаграмма классов нашего проекта. Весь проект состоит из трех классов (см. раздел 3.2 Структура проекта), поэтому мы их отобразили на данной диаграмме.

Класс LogParser использует в качестве зависимости класс FileWorker.

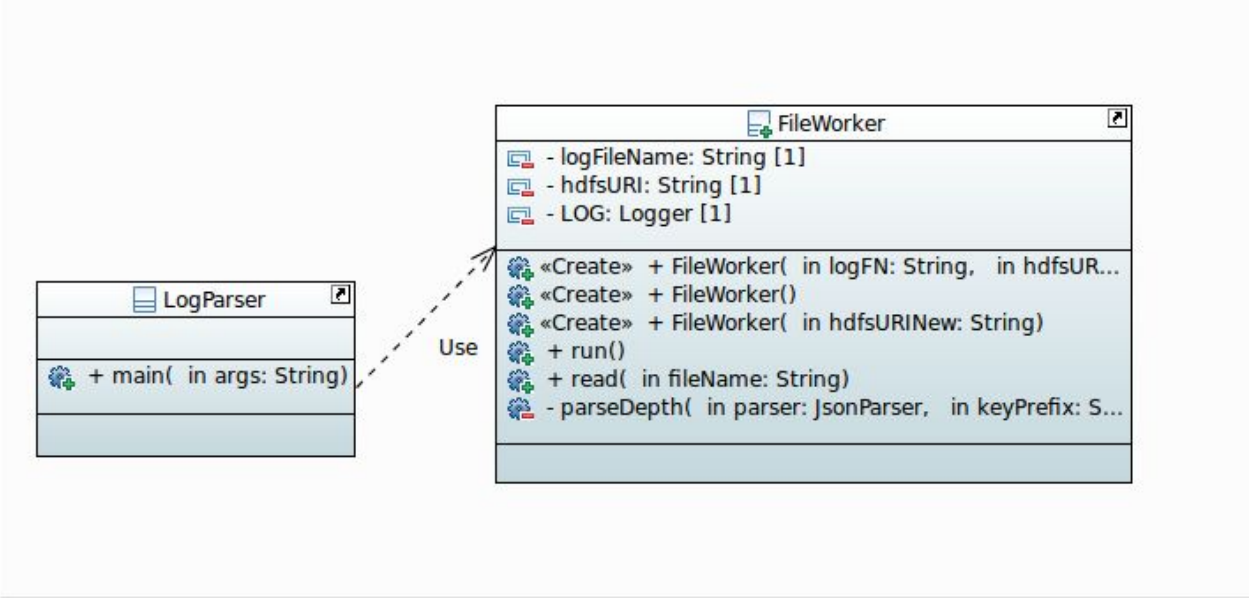

Рисунок 4 – Диаграмма классов

4 Технологическая часть

4.1 Запуск анализа данных

Исходный код проекта доступен в репозитории GitHub (ветка loggerNoOSGI) [3].

Запускающим классом является класс LogParser. В методе main объявляется и инициализируется объект класса FileWorker fw со стандартными параметрами. Данный объект передается в качестве параметра конструктора при создании объекта класса Thread myThread. Затем вызывается метод myThread.start().

4.2 Создание бинарной сборки системы

Для сборки проекта используется фреймворк для автоматизации сборки Maven. В проекте в корневой папке находится файл pom.xml, который содержит в себе все необходимые зависимости, поэтому нет необходимости подгружать другие библиотеки.

4.3 Анализ исходного кода с помощью метрик качества

На рисунке 5 показано соотношение пакетов проекта по их размеру, на рисунке 6 – соотношение классов в пакете ru.bmstu.magapov.suricatalog по их размеру. По данным рисункам можно сделать вывод, что класс FileWorker имеет самый большой размер.

Далее на рисунке 7 отображен список всех метрик по разделам. Всего имеется четыре раздела:

- метрики количества (Count);
- метрики сложности (Complexity);
- метрики Роберта Мартина (Robert C. Martin);
- метрики Чидамбера-Кемерера (Chidamber & Kermerer).

Первый раздел с метриками количества (Count) содержит следующие метрики:

- количество классов верхнего уровня (Unit);
- среднее число внутренних классов на класс (Classes / Class);
- среднее число методов в классе (Methods / Class);
- среднее число полей в классе (Fields / Class);
- число строчек кода (ELOC);
- число строчек кода на модуль (ELOC / Unit).

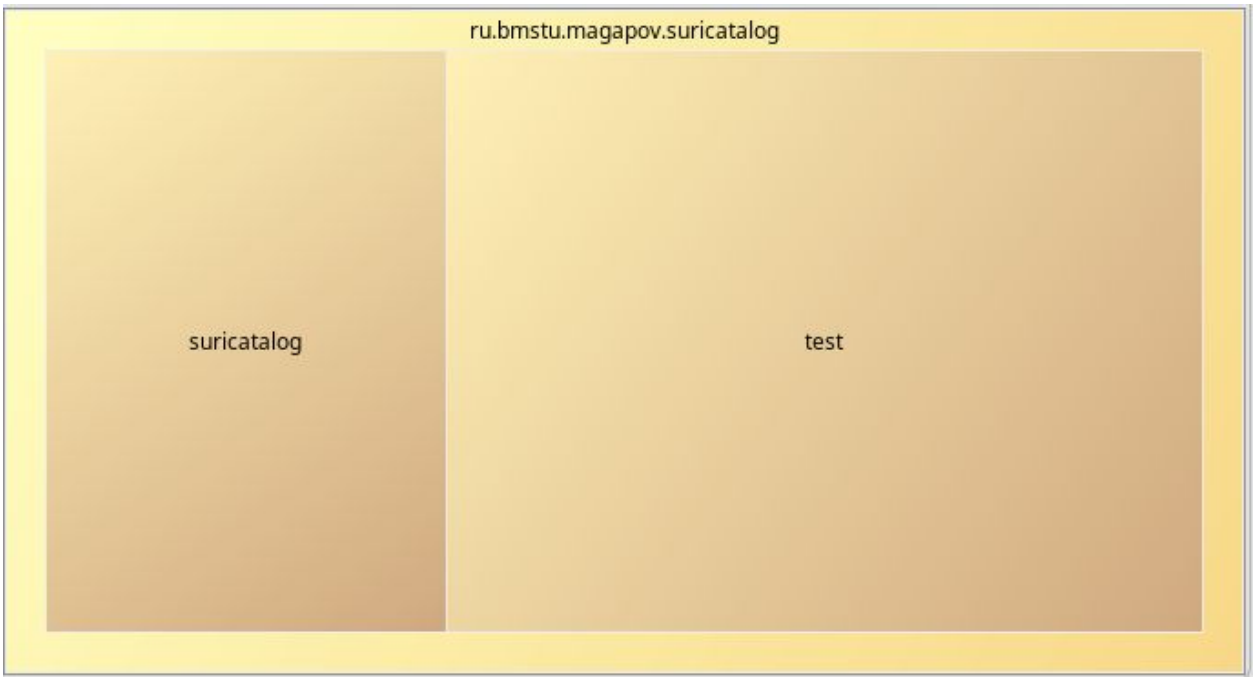

Рисунок 5 – Соотношение пакетов по размеру

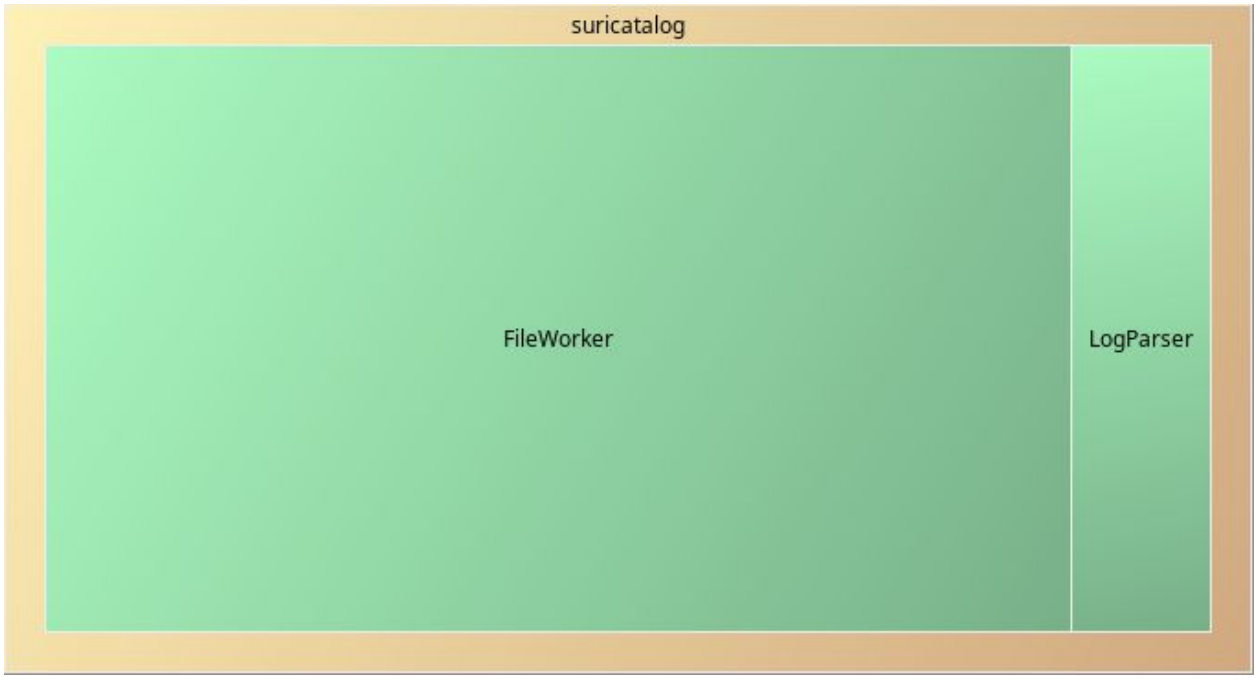

Рисунок 6 – Соотношение классов по размеру

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

- средняя циклическая сложность (CC);
- метрика Fat (Fat);
- средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

- нормализованное расстояние от основной последовательности (D);
- абстрактность (A);
- нестабильность (I);
- число афферентных соединений (Ca);
- число эфферентных соединений (Ce).

Последний раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

- средняя длина метода на класс (WMC);
- средняя глубина наследования (DIT);
- среднее количество классов-наследников (NOC);
- среднее число соединений класса (CBO);
- среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
- отсутствие единства методов (LCOM).

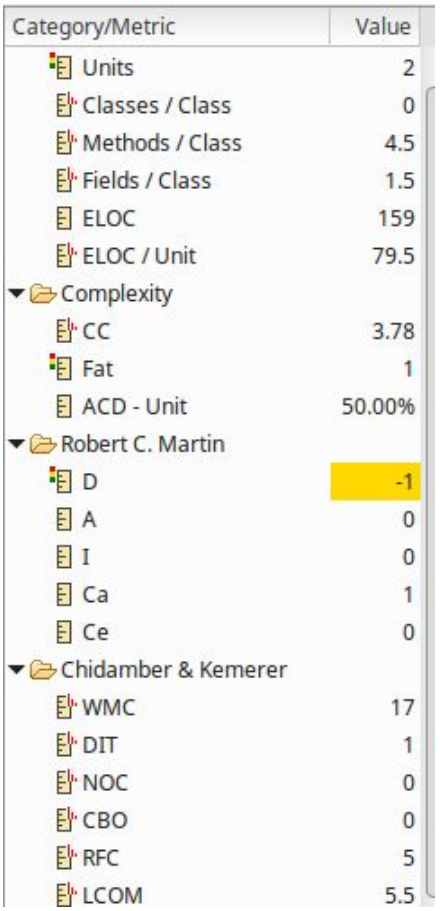

Рисунок 7 – Значения метрик

4.4 Анализ зависимостей в коде системы

Рисунки 8 и 9 характеризуют зависимости между пакетами проекта и внешними пакетами.

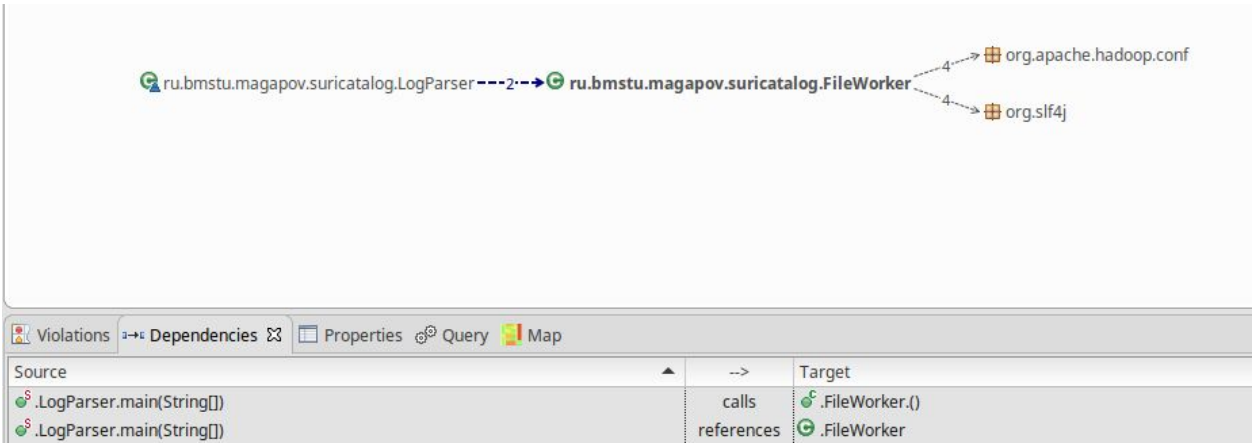

Рисунок 8 – Зависимости внутри пакета ru.bmstu.magapov.suricatalog

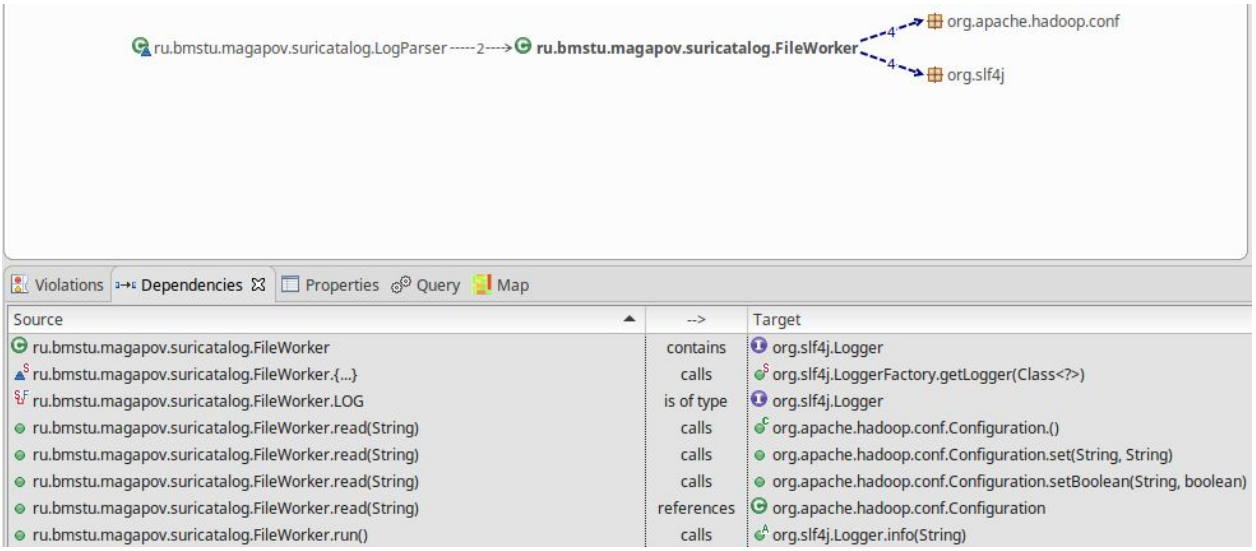

Рисунок 9 – Зависимости между пакетами ru.bmstu.magapov.suricatalog и пакетами org.apache.hadoop.conf, org.slf4j

4.5 Тестирование на корректность работы

Для проверки корректности работы написанного кода были написаны три теста. Первый тест expectedWrongFile() проверяет корректность работы приложения в случае, если для обработки был выбран несуществующий файл. Для этого в тесте создается объект класса FileWorker с несуществующим входным файлом. Затем данный объект запускается в потоке и через mockAppender и captorLoggingEvent ожидается появление сообщения на уровне INFO в логгере LOG класса FileWorker о некорректности файла.

Второй тест expectedWrongConnect() проверяет работу приложения в случае невозможности подключения к серверу Hadoop. Проверка осуществляется аналогично expectedWrongFile().

Третий тест writeAndRead() проверяет корректность записи логов в HDFS. Для этого создается файл с одним тестовым логом, который подается на вход объекта класса FileWorker, который, в свою очередь, запускается в отдельном потоке. После обработки файла инициализируется новое подключение к серверу Hadoop, на котором проверяется наличие только что

записанного файла логов. Данный файл считывается и сравнивается с файлом, который должен был получиться после обработки. В случае успешного сравнения тест пройден.

На рисунке 10 видно, что все тесты отработали успешно. Это значит, что программа отрабатывает корректно при условиях, описанных выше.

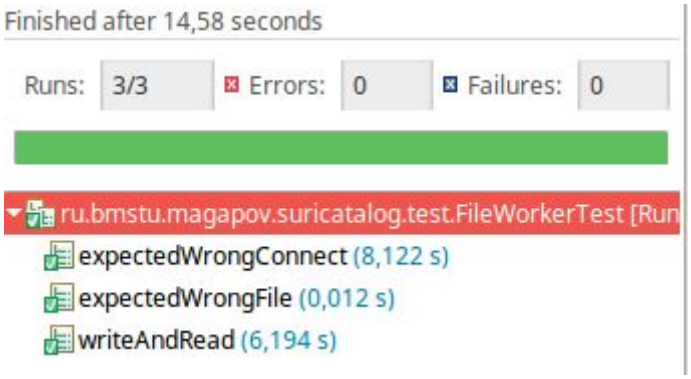

Рисунок 10 – Результат работы тестов для проверки на корректность работы

4.6 Тестирование производительности

Для вычисления зависимости времени работы программы от числа команд, содержащихся в текстовой строке, были написаны еще одиннадцать тестовых метода. Все тесты: write1log(), write10000log(), write20000log(), write30000log(), write40000log(), write50000log(), write60000log(), write70000log(), write80000log(), write90000log(), write100000log() отличаются только количество логов, подаваемых на вход FileWorker. Все файлы обрабатывались в 1 поток. На рисунке 11 показан результат прохождения данных тестов. По результатам прохождения тестов был построен график зависимости времени обработки логов от их количества, изображенный на рисунке 12.

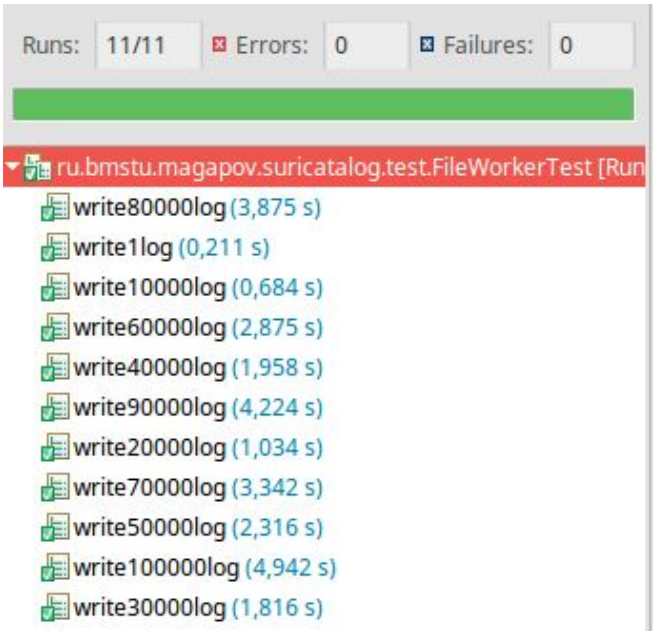

Рисунок 11 – Результат работы тестов для вычисления зависимости времени работы от числа команд

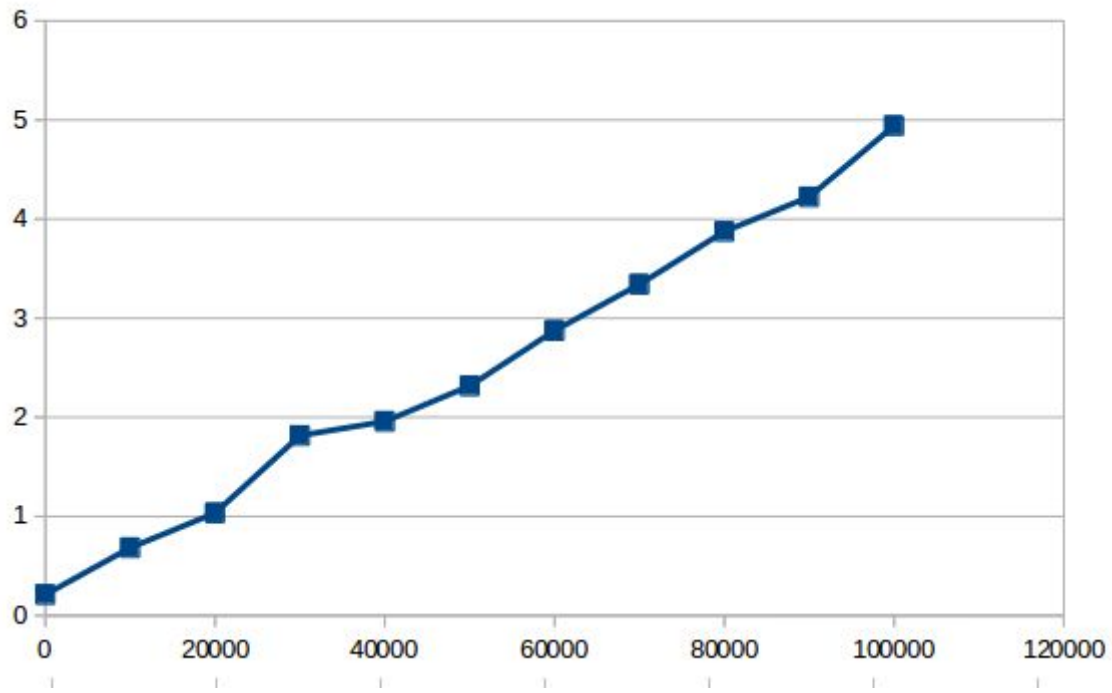

Рисунок 12 – График зависимости времени обработки от количества логов.

По результатам тестирования (рисунок 12) видно, что зависимость времени работы от числа команд можно назвать линейной. Кривизну линии зависимости можно объяснить погрешностью вычисления времени работы программы.

5 Выводы

В ходе курсовой работы были изучены система обнаружения вторжений Suricata и система распределенного хранения больших данных Apache Hadoop. Были получены навыки по настройке и запуску данных систем, а также навыки по работе с ними.

Кроме того, было реализовано приложение, позволяющее обрабатывать логи системы Suricata. В ходе разработки данного приложения были закреплены навыки, полученные в рамках курса «Разработка программного обеспечения».

6 Список источников

[1] Suricata. Xakep.ru: [Электронный ресурс]. Режим доступа: https://xakep.ru/2015/06/28/suricata-ids-ips-197/

[2] Hadoop. Википедия: [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Hadoop

[3] GitHub – RPO_Pro [Электронный ресурс]. Режим доступа: https://github.com/magapov/RPO_Pro

7 Приложение

Листинг 1 - класс LogParser

```
package ru.bmstu.magapov.suricatalog;

import java.io.IOException;
import java.lang.InterruptedException;

class LogParser {
	public static void main(String args[]) throws IOException,
InterruptedException {
		FileWorker fw = new FileWorker();
		Thread myThread = new Thread(fw);
		myThread.start();
	}
}
```

Листинг 2 - класс FileWorker

```
package ru.bmstu.magapov.suricatalog;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonToken;

public class FileWorker implements Runnable {
	private String logFileName;
	private String hdfsURI;
	private static final Logger LOG = LoggerFactory.getLogger(FileWorker.class);

	public FileWorker(String logFN, String hdfsURINew) {
		logFileName = logFN;
		hdfsURI = hdfsURINew;
	}

	public FileWorker() {
		this("/var/log/suricata/eve.json", "hdfs://localhost:9000");
```

```
	}

	public FileWorker(String hdfsURINew) {
		this("/var/log/suricata/eve.json", hdfsURINew);
	}

	@Override
	public void run() {
		try {
			this.read(logFileName);
		} catch (FileNotFoundException fileEx) {
			LOG.info("FileNotFoundException throws");
		} catch (IOException ioEx) {
			LOG.info("IOException throws");
		} catch (InterruptedException intEx) {
			LOG.info("InterruptedException throws");
		} catch (Exception e) {
			LOG.info("Exception throws");
		}
	}

	public void read(String fileName) throws Exception {

		JsonFactory factory = new JsonFactory();

		InputStream logStream = new FileInputStream(fileName);

		BufferedReader scan = new BufferedReader(new
InputStreamReader(logStream));

		String logLine = new String();

		String path = "/suricataLog";
		String hdfsFileName = "log.bin";

		Configuration conf = new Configuration();
		conf.set("fs.defaultFS", hdfsURI);

		conf.set("fs.hdfs.impl",
org.apache.hadoop.hdfs.DistributedFileSystem.class.getName());
		conf.set("fs.file.impl",
org.apache.hadoop.fs.LocalFileSystem.class.getName());
		conf.setBoolean("dfs.support.append", true);

		System.setProperty("HADOOP_USER_NAME", "hduser");
		System.setProperty("hadoop.home.dir", "/");

		FileSystem fs = FileSystem.get(URI.create(hdfsURI), conf);

		Path newFolderPath = new Path(path + String.format("%d",
System.currentTimeMillis() / 1000));

		if (!fs.exists(newFolderPath)) {
			fs.mkdirs(newFolderPath);
		}
```

```
            Path hdfswritepath = new Path(newFolderPath + "/" + hdfsFileName);
            Path hdfswritepathHeader = new Path(newFolderPath + "/" + hdfsFileName
+ ".head");
            FSDataOutputStream outputStream = fs.create(hdfswritepath);
            FSDataOutputStream                outputStreamHeader                =
fs.create(hdfswritepathHeader);

            int headerPos = 0;

            ArrayList<String> header = new ArrayList<String>();
            try {
                  while (!Thread.currentThread().isInterrupted()) {
                        if (scan.ready()) {

                              logLine = scan.readLine();

                              JsonParser parser = factory.createParser(logLine);

                              parser.nextToken();

                              ArrayList<String> logs = new ArrayList<String>();

                              parseDepth(parser, "", header, logs);

                              for(; headerPos < header.size(); headerPos++) {

outputStreamHeader.write((header.get(headerPos) + "|").getBytes());
                              }

                              for(int logPos = 0; logPos < logs.size(); logPos++)
{

outputStream.write((logs.get(logPos).isEmpty()          ?          ""          :
logs.get(logPos)).getBytes());
                                    if(logPos != logs.size() - 1)
                                          outputStream.write("|".getBytes());
                                    else
                                          outputStream.write("\n".getBytes());
                              }
                        } else {
                              System.out.println("wait......"                    +
Thread.currentThread().getName());
                              Thread.sleep(1000);
                        }
                  }
            } catch (InterruptedException intEx) {
                  scan.close();
                  outputStream.close();
                  outputStreamHeader.close();
                  fs.close();
            }
      }
```

```
	private void parseDepth(JsonParser parser, String keyPrefix,
ArrayList<String> header, ArrayList<String> logs) throws IOException{
		String parseKey = new String();

		while (!parser.isClosed()) {
			JsonToken token = parser.nextToken();
			if (JsonToken.FIELD_NAME.equals(token)) {
				parseKey = keyPrefix + parser.getCurrentName();
			} else if (JsonToken.VALUE_STRING.equals(token) ||
JsonToken.VALUE_TRUE.equals(token)
					|| JsonToken.VALUE_FALSE.equals(token) ||
JsonToken.VALUE_NUMBER_INT.equals(token)) {
				if(!parseKey.isEmpty() && !header.contains(parseKey))
					header.add(parseKey);
				if (header.indexOf(parseKey) > logs.size() - 1) {
					for (int i = logs.size(); i <=
header.indexOf(parseKey); i++) {
						logs.add("");
					}
				}
				logs.set(header.indexOf(parseKey),
parser.getValueAsString());
				parseKey = "";
			} else if (JsonToken.START_OBJECT.equals(token)) {
				if(!parseKey.isEmpty())
					keyPrefix = parseKey + "-" + keyPrefix;
				parseDepth(parser, keyPrefix, header, logs);
				keyPrefix = keyPrefix.replace(parseKey + "-", "");
				parseKey = "";
			} else if (JsonToken.END_OBJECT.equals(token)) {
				return;
			}
		}
	}
}
```

Листинг 3 - класс FileWorkerTest

```
package ru.bmstu.magapov.suricatalog.test;

import static org.junit.Assert.*;

import java.io.PrintWriter;
import java.net.URI;

import org.apache.commons.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
```

```
import org.junit.runner.RunWith;

import static org.hamcrest.CoreMatchers.is;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.atLeast;
import org.mockito.ArgumentCaptor;
import org.mockito.Captor;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import org.slf4j.LoggerFactory;

import ch.qos.logback.classic.Level;
import ch.qos.logback.classic.Logger;
import ch.qos.logback.classic.spi.LoggingEvent;
import ch.qos.logback.core.Appender;
import ru.bmstu.magapov.suricatalog.FileWorker;


@RunWith(MockitoJUnitRunner.class)
public class FileWorkerTest {
      @Mock
      private Appender mockAppender;

      @Captor
      private ArgumentCaptor<LoggingEvent> captorLoggingEvent;

      @Before
      public void setup() throws InterruptedException {
                                    final     Logger     logger     =     (Logger)
LoggerFactory.getLogger(Logger.ROOT_LOGGER_NAME);
        logger.setLevel(Level.INFO);
        logger.addAppender(mockAppender);
        captorLoggingEvent = ArgumentCaptor.forClass(LoggingEvent.class);
    }

      @After
    public void teardown() throws InterruptedException {
                                    final     Logger     logger     =     (Logger)
LoggerFactory.getLogger(Logger.ROOT_LOGGER_NAME);
        logger.detachAppender(mockAppender);
    }

      @Test
      public void write1log() {
            FileWorker    fw    =    new    FileWorker("/var/log/suricata/1.json",
"hdfs://localhost:9000");
            Thread th = new Thread(fw);
            th.setName("Write1LogThread");
            th.start();
            while(th.getState() != Thread.State.TIMED_WAITING);
            th.interrupt();
            while(th.isAlive());

            assertTrue(true);
      }
```

```
	@Test
	public void write10000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/10000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write10000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write20000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/20000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write20000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write30000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/30000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write30000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write40000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/40000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write40000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
```

```
	}

	@Test
	public void write50000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/50000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write50000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write60000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/60000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write60000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write70000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/70000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write70000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());

		assertTrue(true);
	}

	@Test
	public void write80000log() {
		FileWorker fw = new FileWorker("/var/log/suricata/80000.json",
"hdfs://localhost:9000");
		Thread th = new Thread(fw);
		th.setName("Write80000LogThread");
		th.start();
		while(th.getState() != Thread.State.TIMED_WAITING);
		th.interrupt();
		while(th.isAlive());
```

```
            assertTrue(true);
      }

      @Test
      public void write90000log() {
            FileWorker   fw   =   new   FileWorker("/var/log/suricata/90000.json",
"hdfs://localhost:9000");
            Thread th = new Thread(fw);
            th.setName("Write90000LogThread");
            th.start();
            while(th.getState() != Thread.State.TIMED_WAITING);
            th.interrupt();
            while(th.isAlive());

            assertTrue(true);
      }

      @Test
      public void write100000log() {
            FileWorker   fw   =   new   FileWorker("/var/log/suricata/100000.json",
"hdfs://localhost:9000");
            Thread th = new Thread(fw);
            th.setName("Write100000LogThread");
            th.start();
            while(th.getState() != Thread.State.TIMED_WAITING);
            th.interrupt();
            while(th.isAlive());

            assertTrue(true);
      }

      @Test
      public void expectedWrongFile() throws Exception{
            FileWorker    fw    =    new    FileWorker("/home/dts/123123123.dat",
"hdfs://localhost:9000");
            Thread th = new Thread(fw);
            th.setName("WrongFileThread");
            th.start();

            Thread.currentThread().sleep(10);
            verify(mockAppender,
atLeast(1)).doAppend(captorLoggingEvent.capture());
        final LoggingEvent loggingEvent = captorLoggingEvent.getValue();
        assertThat(loggingEvent.getLevel(), is(Level.INFO));
        assertThat(loggingEvent.getMessage(), is("FileNotFoundException throws"));

        th.interrupt();
        while(th.isAlive());
        assertTrue(true);
      }

      @Test
      public void expectedWrongConnect() throws Exception {
            Thread.currentThread().sleep(1000);
            FileWorker fw = new FileWorker("hdfs://localhost:1234");
```

```
            Thread th = new Thread(fw);
            th.setName("WrongConnectThread");
            th.start();

            Thread.currentThread().sleep(7000);
            verify(mockAppender,
atLeast(1)).doAppend(captorLoggingEvent.capture());
        final LoggingEvent loggingEvent = captorLoggingEvent.getValue();
        assertThat(loggingEvent.getLevel(), is(Level.INFO));
        assertThat(loggingEvent.getMessage(), is("IOException throws"));

        th.interrupt();
        while(th.isAlive());
      }

      @Test
      public void writeAndRead() throws Exception {
            String writeStr = "{\"FirstField\": \"FirstValue\", \"SecondField\":
\"SecondValue\"}" + "\n" +
                                          "{\"FirstField\":     \"OneMoreValue\",
\"ThirdField\": {\"NewField\": \"NewValue\", \"NewField2\": \"NewValue2\"}}";
            PrintWriter writer = new PrintWriter("test.json", "UTF-8");
            writer.println(writeStr);
            writer.close();

            FileWorker fw = new FileWorker("test.json", "hdfs://localhost:9000");
            Thread th = new Thread(fw);
            th.setName("LogParserThread");
            th.start();

            Thread.currentThread().sleep(5000);
            th.interrupt();
            Thread.currentThread().sleep(1000);

            String hdfsURI = "hdfs://localhost:9000";
            String hdfsFileName = "log.bin";

            Configuration conf = new Configuration();
            conf.set("fs.defaultFS", hdfsURI);

            conf.set("fs.hdfs.impl",
org.apache.hadoop.hdfs.DistributedFileSystem.class.getName());
                                                      conf.set("fs.file.impl",
org.apache.hadoop.fs.LocalFileSystem.class.getName());
          conf.setBoolean("dfs.support.append", true);

          System.setProperty("HADOOP_USER_NAME", "hduser");
          System.setProperty("hadoop.home.dir", "/");

          FileSystem fs = FileSystem.get(URI.create(hdfsURI), conf);
          FileStatus[] fileStatus = fs.listStatus(new Path(hdfsURI + "/"));
          Path[] paths = FileUtil.stat2Paths(fileStatus);

          Path newFolderPath = new Path(paths[paths.length - 1].toString());
```

```
            assertTrue(fs.exists(newFolderPath));

            Path hdfsReadPath = new Path(newFolderPath + "/" + hdfsFileName);
             Path hdfsReadPathHead = new Path(newFolderPath + "/" + hdfsFileName +
".head");

            assertTrue(fs.exists(hdfsReadPath));
            assertTrue(fs.exists(hdfsReadPathHead));

            FSDataInputStream inStream = fs.open(hdfsReadPath);
            FSDataInputStream inStreamHead = fs.open(hdfsReadPathHead);

            String inStr = IOUtils.toString(inStream, "UTF-8");
            String inStrHead = IOUtils.toString(inStreamHead, "UTF-8");

            inStream.close();
            inStreamHead.close();
            fs.close();

                                                                    assertEquals(inStr,
"FirstValue|SecondValue\nOneMoreValue||NewValue|NewValue2\n");
                                                                assertEquals(inStrHead,
"FirstField|SecondField|ThirdField-NewField|ThirdField-NewField2|");
      }

}
```