

**Создание  
интеллектуальной  
системы анализа логов  
с использованием  
машинного обучения  
для кибербезопасности**

# Состав команды

Ильин Владимир

Смоленцев Роман

Калашников Олег

## Задачи

Настроить сбор и обработку логов

Нормализовать логи для последующего анализа

Разработать и обучить модель, основанную на выбранном алгоритме

Проверить работу модели на уникальных данных

# Текущее состояние: Уже решенные задачи

- Anomaly Detection: используются алгоритмы, такие как:
  - Isolation Forest — для поиска аномальных паттернов.
  - Autoencoders — для выявления отклонений от нормального поведения.
- Классификация и предсказание инцидентов:
  - Алгоритмы классификации (SVM, *Random Forest*, Gradient Boosting) применяются для разделения событий на нормальные и потенциальные угрозы.
  - Нейронные сети, в частности рекуррентные (LSTM), используются для работы с временными рядами логов и предсказания событий.
- Кластеризация:
  - Алгоритмы, такие как K-means и DBSCAN, используются для группировки логов по схожести и выделения новых аномалий.

# Текущее состояние: Платформы и инструменты

- ELK Stack (Elasticsearch, Logstash, Kibana): позволяет собирать, хранить и визуализировать логи;
- Splunk: включает возможности машинного обучения для анализа данных;
- Grafana + Prometheus: мониторинг и визуализация с некоторыми возможностями интеграции ML;

# Текущее состояние: Примеры успешного применения

- SOC (Security Operation Centers): используют ML для автоматизации расследования инцидентов.
- SIEM-системы (Security Information and Event Management): уже внедряют AI и ML для корреляции событий.

# Нерешенные проблемы

- Отсутствие структурированности логов, вероятность содержания шума и дублей

Возможное решение: Использование NLP для очистки текстовых данных, автоматические фильтрация и парсинг с использованием Regex и библиотек

- Модели могут не распознавать новые типы атак

Возможное решение: Применение алгоритмов, адаптирующихся к изменениям во времени (онлайн-обучение).  
Использование генеративных моделей.

- Производительность: для точной обработки логов в реальном времени необходимы значительные вычислительные ресурсы.

Возможное решение: Использование распределённых систем и облачных платформ, переход на более легковесные алгоритмы.

- Фальшивые срабатывания

Возможные решения: Построение ансамблей моделей, тончайшая настройка порогов для детекции аномалий, имплементация feedback loops.

# Обработка и нормализация логов

Был выбран инструмент Logstash.

Это конвейер обработки данных, который принимает данные из нескольких источников одновременно, преобразует их и затем отправляет клиенту или в хранилище.

Logstash активно используется в современных DevOps-практиках.

# Логи NGINX

>  train\_logs.log

```
1 169.131.30.123 -- [29/Nov/2024:03:04:40 +0000] "PUT /contact HTTP/1.1" 202 897
2 234.223.138.167 -- [29/Nov/2024:07:59:40 +0000] "DELETE /about HTTP/1.1" 201 1261
3 81.156.91.106 -- [28/Nov/2024:22:02:40 +0000] "GET /api/data HTTP/1.1" 204 605
4 3.115.221.2 -- [28/Nov/2024:10:47:40 +0000] "GET /api/data HTTP/1.1" 503 1138
5 193.109.32.131 -- [29/Nov/2024:02:13:40 +0000] "POST /index.html HTTP/1.1" 500 1633
6 53.35.194.81 -- [28/Nov/2024:19:56:40 +0000] "DELETE /api/data HTTP/1.1" 200 1788
```

# Пример настройки Logstash

```
infra > logstash > config > logstash.yml
```

```
1 http.host: "0.0.0.0"
2 xpack.monitoring.enabled: false
3
```

```
infra > logstash > config > pipelines.yml
```

```
1 - pipeline.id: nginx_logs
2   path.config: "/usr/share/logstash/pipeline/nginx_logs.conf"
3
```

```
infra > logstash > pipeline > nginx_logs.conf
```

```
1 input {
2   file {
3     path => "/var/log/nginx/access.log"
4     start_position => "beginning" # Use only on the first run
5     sincedb_path => "/usr/share/logstash/data/sincedb" # Persistent state tr
6     ignore_older => 0
7   }
8 }
9
10 filter {
11   # Parse the log message using the COMBINEDAPACHELOG pattern
12   grok {
13     match => {
14       "message" => '%{COMBINEDAPACHELOG}'
15     }
16   }
17
18   # Rename 'clientip' to 'ip' (if applicable)
19   mutate {
20     rename => { "clientip" => "ip" }
21   }
22
23   # Drop logs from a specific IP address
24   if [source][address] == "172.18.0.1" or [url][original] == "/requests" { #
25     drop { }
26   }
27 }
28
29 output {
30   # Send processed logs to an HTTP endpoint
31   http {
32     url => "http://api.thesmolentsev.ru/analyze"
33     http_method => "post"
34     format => "json"
35   }
36 }
```

# Выбор алгоритма для построения модели

Для задачи был выбран алгоритм RandomForest.

В качестве аналога рассматривался алгоритм LR (Logistic Regression), но выбор был сделан именно в пользу первого.

# Особенности данных

Логи веб-сервера, как правило, имеют нелинейные зависимости между параметрами. Например, взаимодействие между IP-адресом, типом HTTP-запроса и статусом ответа может указывать на аномалии. LogisticRegression плохо работает с такими сложными зависимостями.

# Особенности данных

Random Forest способен учитывать нелинейные взаимосвязи между признаками, так как он строит множество деревьев решений, каждое из которых рассматривает разные аспекты данных.

## Устойчивость к шуму

Random Forest устойчив к выбросам и шуму в данных. В логах веб-сервера могут быть случайные или малозначимые запросы, которые LogisticRegression может воспринять как важные.

# Многоклассовая классификация

Логи NGINX содержат не только бинарные состояния (нормально/нет), но и классы.

- 200 (успешно) - нормальный запрос к серверу;
- 4xx - подозрительные запросы;
- 5xx (ошибки сервера) - потенциально важный сигнал. Logistic Regression сложнее адаптировать к задачам многоклассовой классификации.

# Масштабируемость

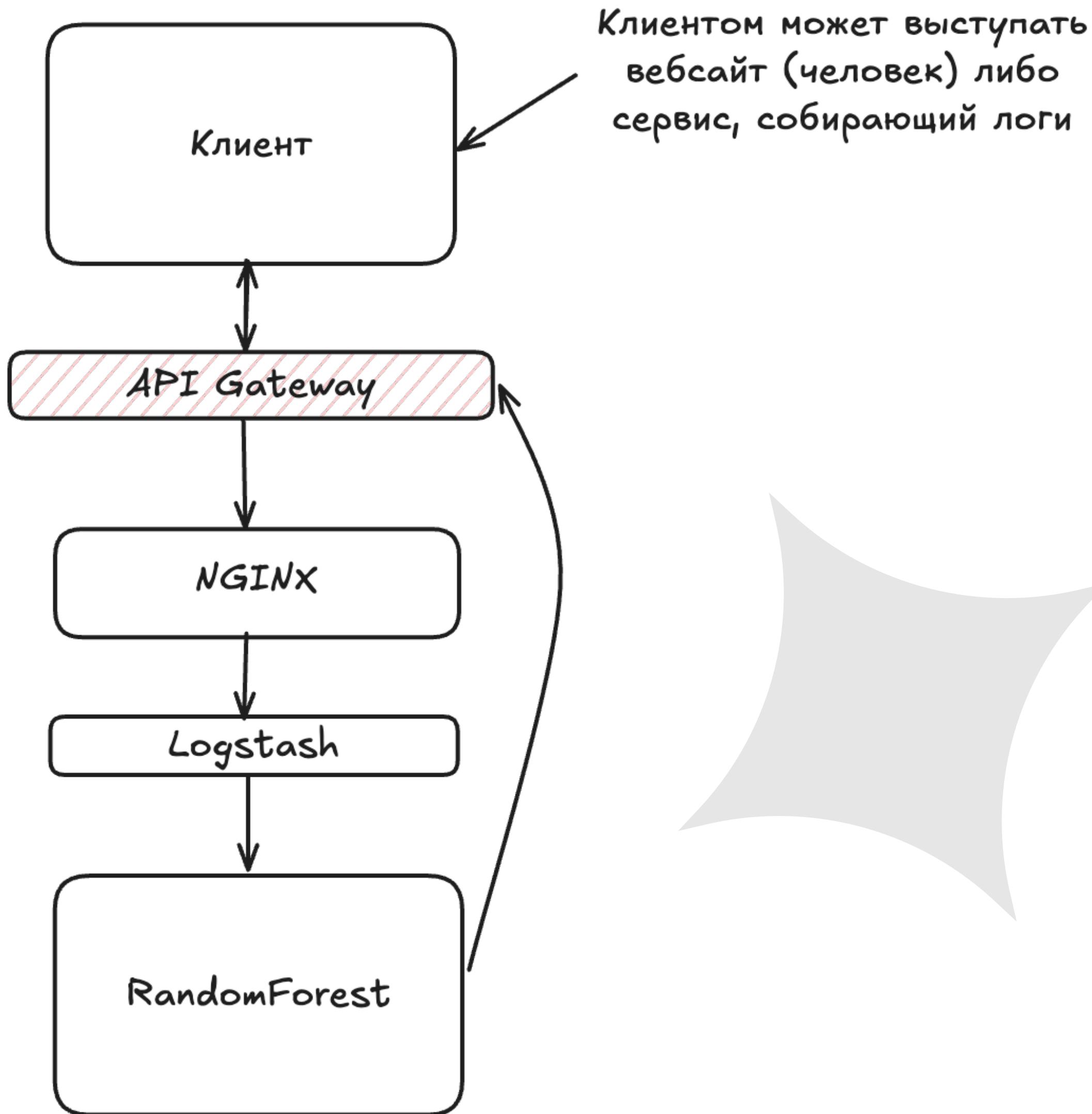
При желании полученную систему можно собрать в Docker-образ и запускать его с настоящими сервисами, работающими на базе NGINX.

Это делает систему полезной для инженеров, специализирующихся на сетевых технологиях.

Именно это мы и сделали.

# Практическая часть

# Архитектура системы

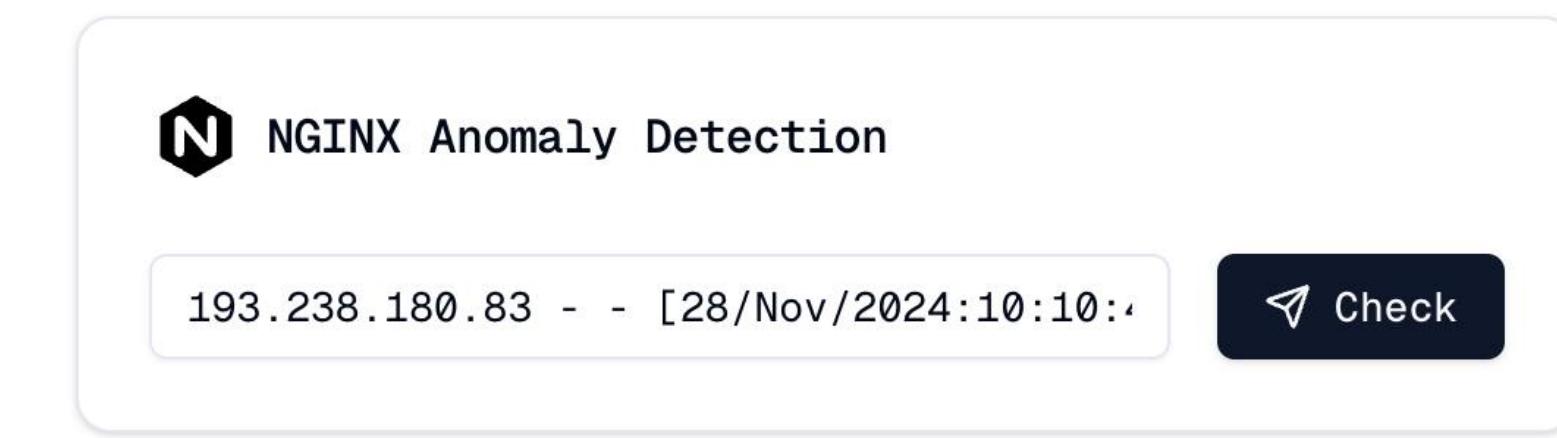


# Клиент

В качестве клиента выступает пользователь для наибольшей  
интерактивности: <http://site.thesmolentsev.ru>

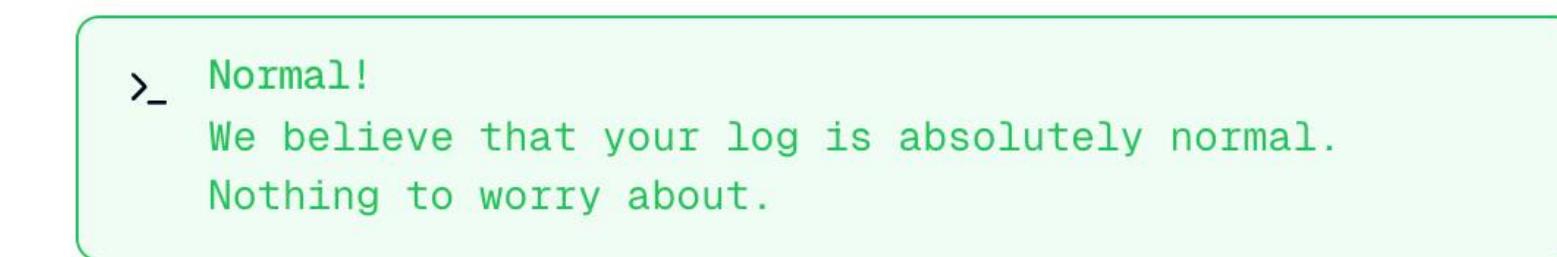
**N** NGINX Anomaly Detection

 Check



Your Input

```
{  
  "timestamp": "28/Nov/2024:10:10:40 +0000",  
  "method": "PUT",  
  "url": "/contact",  
  "statusCode": 200,  
  "bodyBytes": 1722,  
  "host": "193.238.180.83"  
}
```



# API Gateway

Максимально простое демонстративное Flask-приложение с двумя эндпоинтами:

- POST /analyze - проверка лога на аномалию

Принимает данные в формате, отдаваемом Logstash (сл. слайд).

Более удобочитаемые данные (строка NGINX) преобразуются в нужный вид на фронтенд-части

- GET /requests - последние 10 запросов (для дебага)

```
{
  "@timestamp" => 2024-12-01T19:29:15.070557169Z,
  "user_agent" => {
    "original" => "Hello"
  },
  "message" => "62.60.231.155 - - [01/Dec/2024:19:29:15 +0000] \"GET /cgi-bin/luci/;stok=/locale HTTP/1.1\" 404 153 \"-\" \"Hello\"",
  "timestamp" => "01/Dec/2024:19:29:15 +0000",
  "@version" => "1",
  "http" => {
    "version" => "1.1",
    "response" => {
      "status_code" => 404,
      "body" => {
        "bytes" => 153
      }
    },
    "request" => {
      "method" => "GET"
    }
  },
  "event" => {
    "original" => "62.60.231.155 - - [01/Dec/2024:19:29:15 +0000] \"GET /cgi-bin/luci/;stok=/locale HTTP/1.1\" 404 153 \"-\" \"Hello\""
  },
  "url" => {
    "original" => "/cgi-bin/luci/;stok=/locale"
  },
  "host" => {
    "name" => "56d22652e6bc"
  },
  "log" => {
    "file" => {
      "path" => "/var/log/nginx/access.log"
    }
  },
  "source" => {
    "address" => "62.60.231.155"
  }
}
```

```
{  
  "timestamp": "01/Dec/2024:18:21:27 +0000",  
  "http": {  
    "request": {  
      "method": "GET"  
    },  
    "response": {  
      "status_code": 404,  
      "body": {  
        "bytes": 153  
      }  
    }  
  },  
  "url": {  
    "original": "/hello"  
  },  
  "source": {  
    "address": "172.18.0.1"  
  }  
}
```

# **NGINX, Logstash**

**Базовая конфигурация.**

# Модуль Random Forest

**Принимает:**

{ ip, method, status, size, datetime, country, city }

**Добавляет:**

{ hour, request\_rate }

**Возвращает:**

{ is\_anomaly }

# Модуль Random Forest

Обучен на 1 и 2 тысячах записей логов.

Основные метрики:

- Статус запроса  $\geq 400$ , т.е. ошибочный
- `request_rate > 100`, т.е. слишком частые запросы
- `size > 1.000.000`, т.е. слишком большой размер возвращаемых данных
- `hour < 6`, т.е. наименее вероятные ночные запросы
- `country == 'UNKNOWN'`, т.е. ранее неизвестное местоположение IP-адреса

# Модуль Random Forest

Также пробовали, но не увенчалось успехом:

- Метод запроса DELETE на запрашиваемый ресурс типа index.html

Неверно определял неаномальные запросы, уменьшая точность

- Обработка User-Agent

Увеличивало время работы, при этом давая незначительный результат ( $d_{precision} < 0.01$ )

# Результаты тренировок

```
..adimir:~/study/ml_curs          ..adimir:~/study/ml_curs +  
.  
.venv ~/study/ml_curs git:(main)±11 (3m 4.46s)  
python infra/app/model/train.py  


|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.77      | 0.93   | 0.84     | 96      |
| 1            | 0.92      | 0.74   | 0.82     | 104     |
| accuracy     |           |        | 0.83     | 200     |
| macro avg    | 0.84      | 0.83   | 0.83     | 200     |
| weighted avg | 0.84      | 0.83   | 0.83     | 200     |

  
[[89  7]  
 [27 77]]
```

```
.venv ~/study/ml_curs git:(main)
```

# Результаты тренировок

# Достоинства системы

- Достаточно легковесная (не занимает много места на выделенном сервере)
- Простая в использовании и развёртывании
- Легко масштабируемая
- Позволяет рано диагностировать угрозы
- Адаптивность — Random Forest может обучаться на новых данных
- Полностью автоматизированная

# Пути развития

- Минимизировать вероятность ошибки (в данный момент точность около 85%, что всё равно недостаточно в контексте широкой эксплуатации)
- Повысить отказоустойчивость (систему можно перегрузить и вызвать нехватку ресурсов)
- Добавить систему мониторинга и подробную визуализацию (пользователю будет легче определить, откуда пришла проблема, основываясь на дашбордах)

# Выводы

В результате выполнения работы удалось:

- Создать целостную архитектуру системы
- Настроить автоматический сбор логов и их преобразование в формат, нужный для обучения и тестирования
- Привести собираемые системой логи к единому формату
- Реализовать имплементацию алгоритма машинного обучения, принимающую решения на основании более 5 метрик
- Виртуализировать всю инфраструктуру — систему можно запускать на любой машине
- Протестировать работу системы на уникальных данных и убедиться в её корректности
- Настроить мгновенную отчётность для пользователя о работе системы с помощью фронтенд-интерфейса

# Ссылки на файлы

Исходный код всей инфраструктуры (+датасет и pre-trained модель):

[https://github.com/s1ckoleg/ml\\_curs](https://github.com/s1ckoleg/ml_curs)

Сайт:

<http://site.thesmolentsev.ru>

# Использованные источники

<https://flask.palletsprojects.com/en/stable/>

<https://scikit-learn.org/stable/>

<https://www.elastic.co/guide/en/logstash/current/setup-logstash.html>

<https://inlibrary.uz/index.php/tajet/article/view/43511>

<https://ieeexplore.ieee.org/abstract/document/10579970>

<https://allacademicresearch.com/index.php/AJSTEME/article/view/105>