Система классификации посетителей

Руководство по установке и первоначальной настройке программного продукта

Оглавление

1. Термины и сокращения

Термин / Аббревиатура	Значение
БД	База данных.
ОС	Операционная система.
API	Программный интерфейс приложения, предназначенный для взаимодействия со сторонними приложениями или пользователями.
Аккаунт	Учетная запись посетителя сети букмекерских клубов, к которой привязаны его персональные данные и все его кошельки. Одному посетителю может принадлежать только один аккаунт.
Баллы классификации	Условные значения, которые присваиваются аккаунту при совершении ставок с денежного баланса и используются при расчете класса посетителя. Количество баллов привязано к аккаунту. Сбрасываются по истечении периода накопления.
Класс посетителя (Класс аккаунта)	Класс, который присваивается аккаунту посетителя в соответствии с количеством баллов классификации аккаунта.
Кошелек	Учетная запись посетителя букмекерского клуба, к которой привязана информация о балансах и ставках посетителя в этом клубе. В одном клубе у одного посетителя может быть только один кошелек.
Период накопления баллов классификации	Период, по окончании которого баллы классификации сбрасываются. Составляет один календарный месяц.
Система классификации посетителей	Программный продукт, предназначенный для классификации посетителей сети букмекерских клубов на основании данных о совершенных ими ставках.
Эндпойнт	Шлюз, соединяющий серверные процессы приложения с внешним интерфейсом.

2. О документе

Настоящий документ содержит полное руководство по установке и первоначальной настройке программного продукта Система классификации посетителей. Также в документе приведены системные требования к оборудованию, предназначенному для установки ПО.

3. Назначение программного продукта

Система классификации посетителей представляет собой набор серверных приложений, предназначенный для интеграции с другими приложениями и подсистемами как часть информационной системы обработки ставок на спортивные события.

Система классификации посетителей осуществляет распределение посетителей сети букмекерских клубов по классам. Каждому аккаунту посетителя присваивается определенный класс на основании данных о ставках на спортивные события, которые совершает этот посетитель в клубах сети.

Данные о классах посетителей, предоставляемые Системой классификации посетителей, необходимы для корректной работы приложений, предназначенных для:

- сбора и анализа статистических данных о совершении ставок посетителями клубов;
- получения метрик работы клубов.

4. Системные требования

В настоящем разделе перечислены аппаратные и программные требования, необходимые для корректной работы Системы классификации посетителей.

4.1. Минимальные аппаратные требования

Для обеспечения стабильного функционирования Системы классификации посетителей аппаратная часть должна обладать следующими характеристиками:

- Операционная система, способная запускать контейнеры. Предпочтительно ОС *Linux*;
- Система управления контейнерной виртуализацией (*Docker*);
- Подключение к серверу очередей RabbitMQ;
- Количество логических ядер процессора: 4;
- Семейство процессоров: x86-64;
- Частота процессора: 3.0. ГГц;
- Объем установленной памяти: 4 Гб.

4.2. Минимальные требования к сторонним компонентам и/или системам

Для развертывания Системы классификации посетителей должны быть предварительно установлены следующие компоненты окружения:

- *Docker* 24.0.2 (open-source community edition);
- *RabbitMQ* 3.11.21 (Open Source license);
- *PostgreSQL* 14.21 (Open Source license).

Для разработки и модернизации Системы классификации посетителей должны быть использованы следующие программные продукты и языки программирования:

- *VSCode* 1.81 (Open Source license);
- *DBeaver Community* 23.1.5 (Open Source license);
- Платформа *Node.js* 12.
- Язык программирования *JavaScript*.

5. Установка и настройка системы

Система классификации посетителей включает в себя следующие компоненты (Рис. 1):

- *gate* – приложение для приема, первичной валидации и передачи на сервер внешних запросов;
- *classification-system-api* – приложение для управления процессами расчета классов посетителей;
- *classification-system* – приложение для расчета классов посетителей;
- база данных для хранения баллов классификации и классов посетителей.

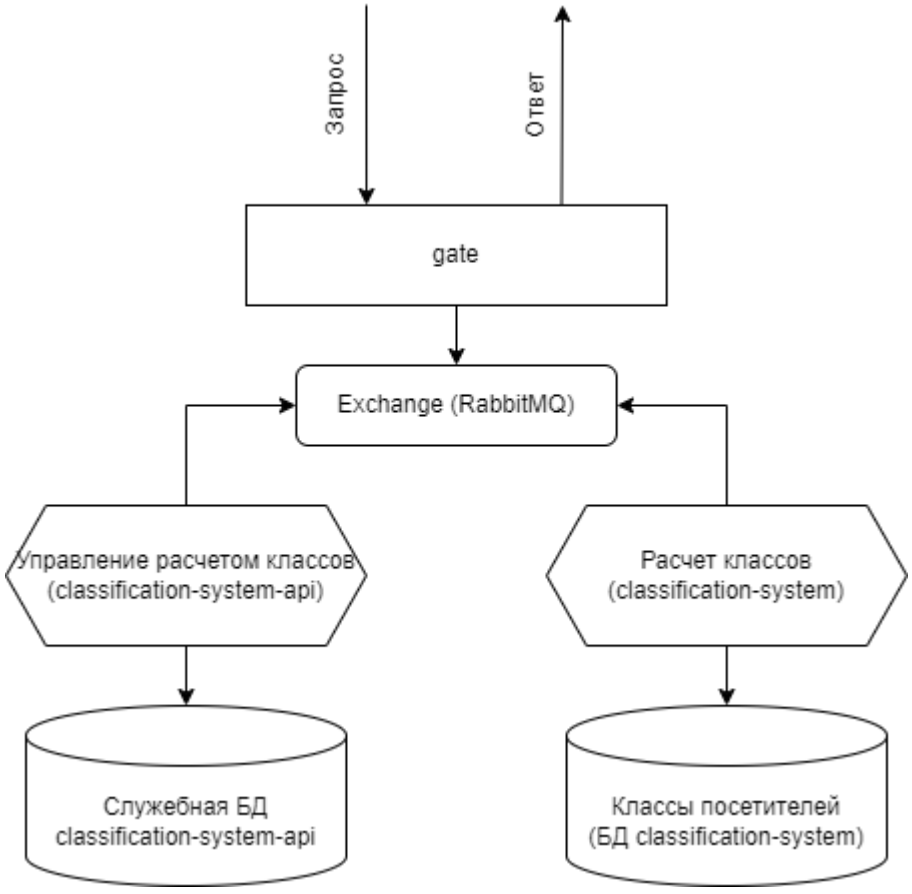

Рис. 1 Компоненты Системы классификации посетителей

Обмен сообщениями между сервисами выполняется по стандарту AMQP, в качестве брокера сообщений используется Rabbit MQ.

База данных сервиса *classification-system-api* служит для хранения служебной информации.

Перед запуском программного продукта необходимо:

1. Установить и настроить компоненты внешнего окружения (5.1);
2. Подготовить файлы конфигурации (0);
3. Выполнить сборку образов приложений. Каждое приложение, входящее в программный продукт, собирается в отдельном Docker-контейнере (5.3);
4. Запустить сервисы Системы классификации посетителей (5.4);
5. Создать и настроить токен доступа (5.5).

5.1. Подготовка окружения

В настоящем разделе приведен порядок действий для предварительной подготовки каждого компонента внешнего окружения:

1. PostgreSQL (5.1.1);
2. Docker (5.1.2);
3. RabbitMQ (5.1.3).

.

5.1.1 Подготовка PostgreSQL

PostgreSQL может быть установлена как из репозитория *Debian 11*, так и из официального репозитория СУБД.

Установка из репозитория Debian 11

1. Сначала обновите список пакетов. Запустите терминал и выполните команду:

```
sudo apt update
```

2. Пакет PostgreSQL находится в репозитории *Debian*, поэтому вы можете установить его с помощью утилиты *apt*. Для этого выполните:

```
sudo apt-get install postgresql=11*
```

3. После завершения установки проверьте статус соответствующей службы с помощью команды:

```
sudo systemctl status postgresql
```

4. Если служба не запустилась автоматически, вы можете запустить её вручную. Для этого выполните:

```
sudo systemctl start postgresql
```

Установка из официального репозитория СУБД

1. Прежде всего необходимо добавить ключ подписи GPG:

```
curl -fsSL https://www.postgresql.org/media/keys/ACCC4CF8.asc |
sudo gpg --dearmor -o /usr/share/keyrings/postgresql-keyring.gpg
```

2. Теперь вы готовы добавить репозиторий *Postgres*:

```
echo "deb [signed-by=/usr/share/keyrings/postgresql-keyring.gpg]
http://apt.postgresql.org/pub/repos/apt/  "$(. /etc/os-release &&
echo "$VERSION_CODENAME")"-pgdg main" |
sudo tee /etc/apt/sources.list.d/postgresql.list
```

3. Обновите репозиторий системы с помощью команды:

```
sudo apt update
```

4. После обновления выполните PostgreSQL install на *Debian*:

```
sudo apt install postgresql
```

5. После установки PostgreSQL необходимо подготовить базы данных и пользователей:

```
sudo -u postgres psql
CREATE DATABASE classification-system-api--l2-a8ce1057;
CREATE DATABASE classification-system--l3-99aac3ca;
CREATE USER gate--l1-d789e231 WITH PASSWORD 'password';
CREATE USER classification-system-api--l2-a8ce1057 WITH PASSWORD
'password';
CREATE USER classification-system--l3-99aac3ca WITH PASSWORD 'password';
```

```
ALTER DATABASE classification-system-api--l2-a8ce1057 OWNER TO
classification-system-api--l2-a8ce1057_user;
ALTER DATABASE classification-system--l3-99aac3ca OWNER TO
classification-system--l3-99aac3ca_user;
GRANT ALL PRIVILEGES ON DATABASE classification-system-api--l2-a8ce1057
TO classification-system-api--l2-a8ce1057_user;
GRANT ALL PRIVILEGES ON DATABASE classification-system--l3-99aac3ca TO
classification-system--l3-99aac3ca_user;
```

6. Для возможности подключения сервисов вставьте в конец файла */etc/postgresql/11/main/pg_hba.conf* следующую строку:

```
host all all 0.0.0.0/0 md5
```

5.1.2 Подготовка Docker

1. Установите зависимости с помощью команды:

```
sudo apt-get install ca-certificates curl gnupg
```

2. Добавьте ключ подписи GPG, выполнив команды:

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg |
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

3. Добавьте репозиторий, выполнив команды:

```
echo \
"deb [arch="$(dpkg --print-architecture)"
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Теперь установите *Docker*:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Примечание: вместо Docker может использоваться любая другая платформа контейнеризации.

5.1.3 Подготовка RabbitMQ

Рассмотрим процесс с настройкой официального репозитория для установки RabbitMQ https://www.rabbitmq.com/changelog.html.

1. Для установки curl необходимо выполнить команду:

```
apt install curl
```

2. После этого необходимо выполнить две команды:

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.deb.sh | sudo bash
```

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/erlang/script.deb.sh | sudo bash
```

В результате в систему будут добавлены репозитории для установки RabbitMQ и erlang.

3. Выполните установку брокера:

```
apt install rabbitmq-server
```

После установки брокера на Ubuntu он автоматически будет запущен и настроен для автозапуска. Проверить можно командой:

```
systemctl status rabbitmq-server
```

4. Выполните создание виртуального хоста для изоляции рабочего пространства, а также создайте пользователей, которым будет выдан полный доступ на созданный виртуальный хост.

   4.1. Создание виртуального хоста.

   Название виртуального хоста должно соответствовать хосту сервисов *classification-system-api* и *classification-system* – postgresql.

```
rabbitmqctl add_vhost postgresql
```

   4.2. Создание пользователей. На данном этапе необходимо выполнить создание трех пользователей, их имена должны совпадать с именами пользователей для сервисов *gate*, *classification-system-api*, *classification-system* – *gate--l1-d789e231*, *classification-system-api--l2-a8ce1057*, и *classification-system--l3-99aac3ca* соответственно.

```
rabbitmqctl add_user gate--l1-d789e231
rabbitmqctl add_user classification-system-api--l2-a8ce1057
rabbitmqctl add_user classification-system--l3-99aac3ca
```

   По каждому созданному пользователю необходимо придумать и ввести пароль. Поле для ввода пароля появится автоматически.

   *При необходимости пароль может быть изменен командой:

```
rabbitmqctl change_password classification-system-api--l2-a8ce1057
```

   4.3. Чтобы выдать пользователям права к виртуальному хосту – выполните команду по каждому пользователю:

```
rabbitmqctl set_permissions gate--l1-d789e231 ".*" ".*" ".*"
rabbitmqctl set_permissions classification-system-api--l2-a8ce1057 ".*"
".*" ".*"
rabbitmqctl set_permissions classification-system--l3-99aac3ca ".*"
".*" ".*"
```

4.4. Чтобы сделать пользователя администратором, выполните команду:

```
rabbitmqctl set_user_tags classification-system-api--l2-a8ce1057
administrator
```

После этого выполните аналогичную команду по остальным пользователям.

5.2. Подготовка файлов конфигурации

Перед первым запуском системы необходимо установить конфигурационные параметры сервисов *gate*, *classification-system-api* и *classification-system*. Файлы конфигурации для каждого сервиса расположены в папке */<название сервиса>/helm/values/prod.conf.d*.

Также в директории */<название сервиса>/helm/values/* находится файл конфигурации *prod.values.yaml*.

Файлы имеют идентичные названия для каждого из сервисов.

5.2.1 Конфигурация gate

1. Создайте файлы конфигурации для сервиса *gate* в папке */helm/values/prod.conf.d*:
    1.1. Информацию об используемых доменах в файле domains.json;

```
{
  "domains": ["bet", "classification", "report"]
}
```

    1.2. Параметры для токена jwt в файле jwt.json:

```
{
  "jwt": {
    "keyValue": "",
    "keyFile": "/app/config/conf.d/jwt.pub"
  }
}
```

    1.3. Параметры логирования в файле log.json:

```
{
  "log": {
    "fileLevel": "silly",
    "consoleLevel": "silly"
  }
}
```

1.4. Параметры для rmq в файле rmq.json:

```
{
  "rmq": {
    "connection": {
      "hostname": "rabbitmq",
      "username": "gate--l1-d789e231"
    }
  }
}
```

2. Добавьте в файл *prod.values.yaml* содержимое:

```
---
environment: production

replicaCount: 1

resources:
  requests:
    memory: 128Mi
  limits:
    memory: 128Mi

useVariables:
  NODE_OPTIONS: "--max-old-space-size=98"

labels:
  prometheus: "scrape"
```

```
ports:
  - protocol: TCP
    port: 8080
    name: metrics
  - port: 3003
    name: ws
    protocol: TCP
  - port: 7777
    name: http
    protocol: TCP

healthcheck:
  livenessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  startupProbe:
    httpGet:
      path: /metrics
      port: 8080
    failureThreshold: 60
    periodSeconds: 5

service:
  type: NodePort

ingress:
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
```

```
    nginx.ingress.kubernetes.io/proxy-read-timeout: "1500"
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/limit-rps: "30"
    kubernetes.io/ingress.global-static-ip-name: "prod-v3-ingress-1"
  letsencrypt: "true"
  letsencryptSecret: "letsencrypt-prod"
  hosts:
  - name: gate-demo.testllc.ru
    paths:
      /api/v1/rest:
        serviceName: gate--0c943e6-service
        servicePort: 7777
      /api/v1/ws:
        serviceName: gate--0c943e6-service
        servicePort: 3003
```

5.2.2 Конфигурация classification-system-api

1. Создайте файлы конфигурации для сервиса *classification-system-api* в папке */helm/values/prod.conf.d*.
   1.1. Параметры для базы данных в файле pg.js:

```
const fs = require("fs")

module.exports = {
  "db": {
    "connections": {
      "pgPool": {
        "connectionType": "pg",
        "host": "postgresql",
        "port": 5432,
        "database": "classification-system-api--l2-a8ce1057",
        "currentSchema": "classification-system-api--l2-a8ce1057",
        "user": "classification-system-api--l2-a8ce1057",
        "dbmCwd": "/app/db",
```

```
      }
    }
  }
}
```

1.2. Параметры для jaeger в файле jaeger.json:

```
{
  "jaeger": {
    "host": "jaeger-agent"
  }
}
```

1.3. Параметры для токена jwt в файле jwt.json:

```
{
  "jwt": {
    "keyValue": "",
    "keyFile": "/app/config/conf.d/jwt.pub"
  }
}
```

1.4. Параметры для rmq в файле rmq.json:

```
{
  "rmq": {
    "connection": {
      "hostname": "rabbitmq",
      "username": "classification-system-api--l2-a8ce1057"
    }
  }
}
```

2. Добавьте в файл *prod.values.yaml* содержимое:

```
---
environment: production
```

```
replicaCount: 1

resources:
  requests:
    memory: 384Mi
  limits:
    memory: 384Mi

useVariables:
  NODE_OPTIONS: "--max-old-space-size=344"

labels:
  prometheus: "scrape"

ports:
  - protocol: TCP
    port: 8080
    name: metrics

healthcheck:
  livenessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
```

5.2.3 Конфигурация classification-system

1. Создайте файлы конфигурации для сервиса *classification-system* в папке */helm/values/prod.conf.d*.

   1.1. Параметры для базы данных в файле pg.js:

```
const fs = require("fs")

module.exports = {
  "db": {
    "connections": {
      "pgPool": {
        "connectionType": "pg",
        "host": "postgresql",
        "port": 5432,
        "database": "classification-system--l3-99aac3ca",
        "currentSchema": "classification-system--l3-99aac3ca",
        "user": "classification-system--l3-99aac3ca",
        "dbmCwd": "/app/db",
      }
    }
  }
}
```

   1.2. Параметры для jaeger в файле jaeger.json:

```
{
  "jaeger": {
    "host": "jaeger-agent"
  }
}
```

   1.3. Параметры для токена jwt в файле jwt.json:

```
{
  "jwt": {
    "keyValue": "",
```

```
    "keyFile": "/app/config/conf.d/jwt.pub"
  }
}
```

1.4. Параметры для rmq в файле rmq.json:

```
{
  "rmq": {
    "connection": {
      "hostname": "rabbitmq",
      "username": "classification-system--l3-99aac3ca"
    }
  }
}
```

2. Добавьте в файл *prod.values.yaml* содержимое:

```
---
environment: production

replicaCount: 1

resources:
  requests:
    memory: 384Mi
  limits:
    memory: 384Mi

useVariables:
  NODE_OPTIONS: "--max-old-space-size=344"

labels:
  prometheus: "scrape"

ports:
  - protocol: TCP
    port: 8080
```

```
    name: metrics

healthcheck:
  livenessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /metrics
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
```

5.3. Сборка Docker-образов

Сценарии сборки для Docker-образа каждого сервиса прописаны в соответствующем *Dockerfile*. Необходимо выполнить сборку Docker-образов для каждого сервиса (*gate*, *classification-system-api*, *classification-system*), а также для RabbitMQ и Базы данных.

Перед сборкой Docker-образа необходимо объявить переменную окружения $NPMRC по примеру:

```
@dm:registry=https://npm.shelopes.com/repository/npm-private/
//npm.shelopes.com/repository/npm-private/:_authToken=NpmToken.THIS-SECRET-TOKEN
```

***Dockerfile*:**

```
FROM node:12-alpine
```

```
ENV NODE_ENV=production
WORKDIR /app
ARG NPMRC
COPY src/package.json src/package-lock.json /app/
RUN [ -n "\$NPMRC" ] && echo "\$NPMRC" > ~/.npmrc && npm ci --production
&& rm -f ~/.npmrc


FROM node:12-alpine
ENV NODE_ENV=production
WORKDIR /app
COPY --from=builder /app/node_modules /app/node_modules
COPY ./src /app
CMD ["index.js"]
```

Каждый Docker-образ необходимо собрать командой:

```
docker build -t <Имя образа>
```

Например, для сервиса *classification-system* команда выглядит так:

```
docker build -t classification-system
```

Внимание!

Dockerfile с содержимым для соответствующего сервиса должен находится в той же директории, что и вы.

5.4. Запуск сервисов

После того, как все Docker-образы собраны, можно запускать сервисы.

Каждый сервис запускается отдельной командой.

Общий вид команды запуска:

```
docker run -d \
  -v /путь/к/локальной/conf.d:/app/conf.d \
  -v /путь/к/локальной/secret.d:/app/secret.d \
```

```
  -p ЛОКАЛЬНЫЙ_ПОРТ:ПОРТ_В_КОНТЕЙНЕРЕ \
  --name ИМЯ_КОНТЕЙНЕРА \
  ИМЯ_И_ВЕРСИЯ_ОБРАЗА
```

Примечание: параметр -p используется только для сервиса *gate*.

5.5. Создание и настройка токена

После запуска сервисов необходимо создать токен для сервиса *classification-system-api*. Для этого можно воспользоваться Open Source сайтом jwt.io или подобными Open Source сервисами для генерации JWT-токена с использованием алгоритма подписания RS256. После выбора данного алгоритма подписания необходимо указать сертификат компании и добавить payload с перечнем всех используемых методов.

Для *classification-system-api*:

```
{
 "data": {
 "permissionId": 0
 },
 "gates": [
   "gate--l1-d789e231",
   "gate--l1-canary"
 ],
 "iat": 1694011224,
 "jti": "0676c6b061ace68a310fdccd7b2ac03c",
 "permissions": {
   "classification": [
   "getClassificationInfo",
   "getClassificationStatusPoints",
   "setClassificationStatusPoints",
   "processClassificationStatuses",
   "supportDataByDateRequested",
   "setClassificationStatus",
   "getClassificationSystemsStatusHistory"
 ],
```

```
 "bet": [
   "betsCreated",
   "sportLossCreated",
   "sportWinsCreated",
   "sportCashOutCreated",
   "sportExpressBonusCreated",
   "sportMultyBetOfTheDayCreated"
  ]
 },
 "ser": "51725348948c7f482b843ce59afb8d6f"
}
```

В результате для сервиса *classification-system-api* будет сгенерирован ID-токен, который нужно будет подставлять во все входящие запросы к программному продукту.

Пример использования:

```
{
    "id": "{{$guid}}",
    "locale": "ru",
    "domain": "classification",
    "event": "getClassificationInfo",
    "data": {
        "accountId": "965123865"
    },
    "tokens": {
        "id": "{{idTokenAP}}",
        "access": "access"
    }
}
```

На месте *idTokenAP* должен содержаться сгенерированный ID-токен для сервиса *classification-system-api*.