



ООО «ПОЛЕТ»

ИНН 7730582675 КПП 773301001 ОГРН 1087746610896

125373, Москва, Походный проезд д. 14, цок.эт, пом.І, ком .3, тел: 8 (967) 104-01-09

Retail Suite Global.Loyalty&CRM

Инструкция по установке

1. Назначение продукта

Типовое отраслевое решение, программа для ЭВМ (далее программное обеспечение или ПО) RS Global.Loyalty&CRM является уникальной разработкой ООО «Полет», предназначена для автоматизации управления взаимоотношениями с покупателями в розничных сетях и одиночных розничных магазинах любого формата и размера. С помощью ПО RS Global.Loyalty&CRM можно, организовывать программы поощрения любой сложности, выстраивать коммуникации с покупателями и анализировать эффективность предоставленных сервисов. Программное обеспечение состоит из нескольких лицензируемых компонент и использует запатентованные сервисы собственной разработки ООО «Полет». Программное обеспечение RS Global.Loyalty&CRM состоит из следующих лицензируемых компонент:

Retail Suite.Global Loyalty&CRM - основной компонент, является обязательным для работы остальных компонент программного обеспечения. Предназначен для управления лояльностью розничных покупателей, позволяет создавать и управлять различными программами поощрения, содержит транзакционные сервисы, взаимодействующие с кассовым ПО, ERP и системами управления торговлей на розничном предприятии.

Retail Suite.Global Loyalty&CRM Personal Account – личный кабинет пользователя, позволяет зарегистрировать карты лояльности, получать информацию о общих и персональных предложениях, просматривать покупки и бонусные транзакции.

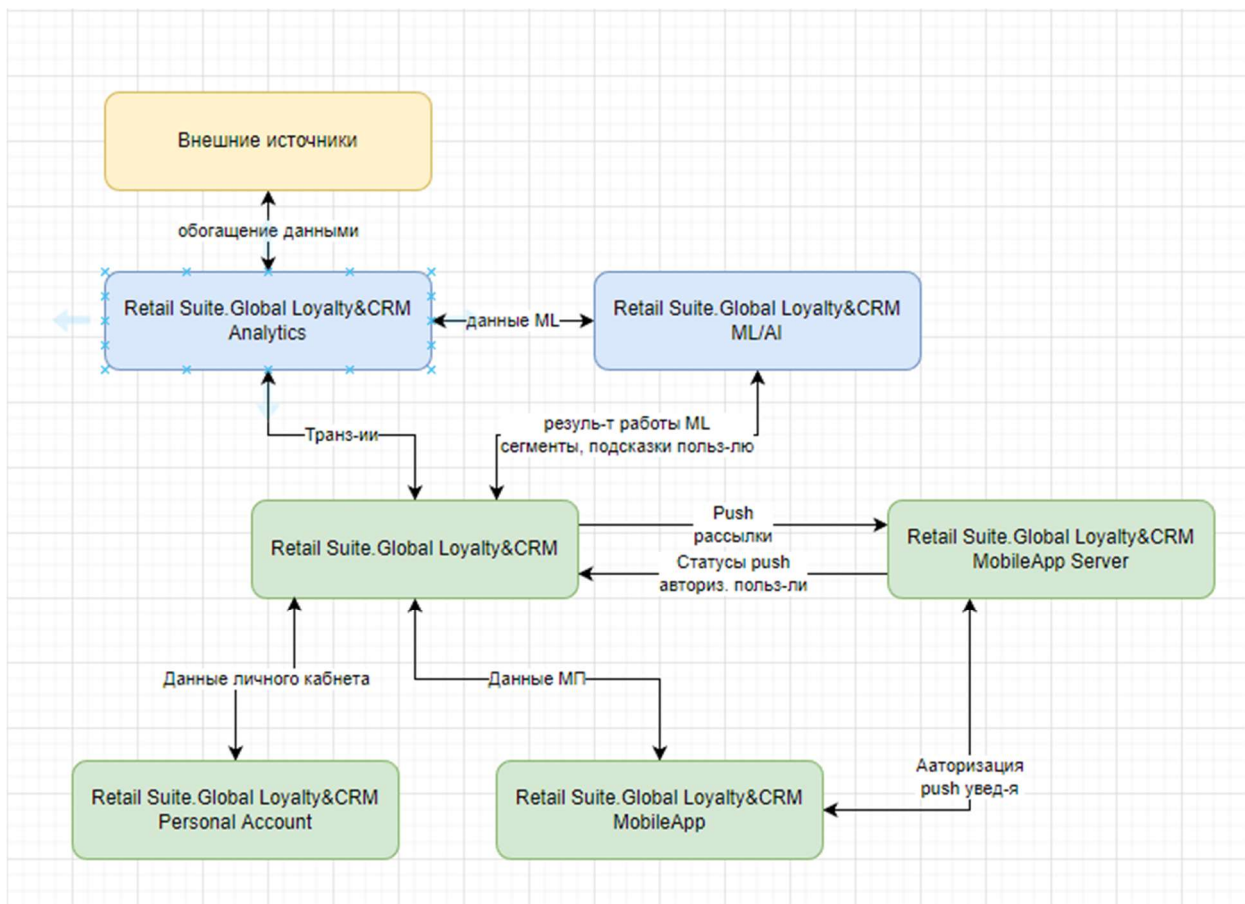
Retail Suite.Global Loyalty&CRM MobileApp – мобильное приложение Android и iOS, позволяет получать и регистрировать карты лояльности, получать информацию о общих и персональных предложениях, просматривать покупки и бонусные транзакции, оплачивать бонусами.

Retail Suite.Global Loyalty&CRM MobileApp Server - компонент регистрации пользователей мобильных устройств и рассылок push уведомлений.

Retail Suite.Global Loyalty&CRM Analytics – аналитический компонент программного обеспечения, который расширяет возможности управления лояльностью покупателей, предоставляя ритейлеру знания, полученные методами машинного обучения на массивах данных Big Data. Позволяет осуществлять сегментацию клиентской базы, анализ программ поощрений и проведенных коммуникаций с покупателями.

Retail Suite.Global Loyalty&CRM ML/AI – компонент содержащий ML\AI процедуры, позволяющие выявлять товарные ассоциации и предпочтения клиентов, прогнозировать отток, давать персональные предложения для покупателей и помогать ритейлеру глубже изучить жизненный цикл клиента. Работая с большими данными, алгоритмы искусственного интеллекта создают многомерный профиль покупателя, который дает ритейлеру возможность лучше понимать желания и потребности своей клиентской базы, предсказывать рыночные тенденции и принимать более разумные решения в управлении лояльностью. Положительный опыт взаимодействия покупателя с ритейлером значительно повышает лояльность клиента, узнаваемость бренда и конверсионность предложений. Работа данного компонента требует лицензии на компонент Retail Suite.Global Loyalty&CRM Analytics

2. Архитектурная схема программных компонент Retail Suite.Global Loyalty&CRM



Инструкция по установке программы для ЭВМ Retail Suite Global.Loyalty&CRM

Программное обеспечение состоит из следующих компонент:

- Retail Suite.Global Loyalty&CRM
- Retail Suite.Global Loyalty&CRM Personal Account
- Retail Suite.Global Loyalty&CRM MobileApp
- Retail Suite.Global Loyalty&CRM MobileApp Server
- Retail Suite.Global Loyalty&CRM Analytics
- Retail Suite.Global Loyalty&CRM ML/AI

Компоненты устанавливаются отдельно.

Установка компонента Retail Suite.Global Loyalty&CRM

Требования к аппаратному обеспечению

Для центрального сервера RS.Loyalty предъявляются следующие требования к аппаратному обеспечению:

Общее количество касс	Максимальное количество чеков в секунду	CPU	RAM, Gb	HDD, Gb	LAN
1 - 15	1	4 ядра (8 потоков) 2.6GHz	8	110	100 Mbps
16 - 50	5	4 ядра (8 потоков) 3.3GHz	8	200	100 Mbps
51 - 200	20	4 ядра (8 потоков) 3.5GHz	16	800	100 Mbps
201 - 500	50	8 ядер (16 потоков) 3.2GHz	32	2000	100 Mbps
501 - 1000	100	8 ядер (16 потоков) 3.4GHz	32	4000	100 Mbps
1001 - 2000	200	16 ядер (32 потока) 2.2GHz	64	8000	100 Mbps
2001 - 4000	400	24 ядра (44 потока) 2.2GHz	64	16000	100 Mbps

Требования к ОС и платформам для RS Global.Loyalty&CRM:

Операционная система	GNU/Linux - DEB/RPM Based РЕД ОС Astra Linux Debian
Фреймворк ОС	dotnet-sdk-6.0 aspnetcore-runtime-6.0
Сервер баз данных	PostgresPRO Postgresql
Брокер сообщений	Брокер сообщений RabbitMQ версии 3.11.0 и выше
Сервер http	Сервер http Nginx версии 1.18.0 и выше
Браузер	Браузер Google Chrome версии 114.0 и выше Браузер Яндекс

Совместимость с кассовым ПО

Наименование	Версия
Frontol 6	6.14.0
Frontol 5	5.26.5
Frontol xPOS	3.7.3
Set Retail 10	10.3.11
Artix	4.6

Для установки системы RS Global.Loyalty&CRM необходимо наличие следующих предустановленных сервисов:

Сервер баз данных postgresql (или PostgresPRO) версии не ниже 14

RabbitMQ

SSL сертификат

В системе RS Global.Loyalty&CRM есть следующие собственные сервисы:

RS.Loyalty.Modules.Analytics.Service

RS.Loyalty.ClientPortal.ApiService

RS.Loyalty.Modules.Communications.Service

RS.Loyalty.Modules.CRM.Service

RS.Loyalty.Integration.ApiService

RS.Loyalty.Modules.Locations.Service

RS.Loyalty.Modules.Loyalty.Service

RS.Loyalty.Modules.ProductCatalog.Service

RS.Loyalty.Modules.Security.Service

RS.Loyalty.Service

RS.Loyalty.UI.ApiService

У каждого сервиса есть свой конфигурационный файл appsettings.json

В них прописываются:

Параметры подключения к базе данных

Параметры подключения к брокеру сообщений RabbitMQ,

Параметры логирования

Параметры, необходимые для работы каждого модуля

Перед запуском необходимо настроить эти конфигурационные файлы.

Для запуска модулей с использованием https, необходимо выполнить команды

```
dotnet dev-certs https -ep ${HOME}/.aspnet/https/aspnetapp.pfx -p  
<CREDENTIAL_PLACEHOLDER>
```

```
dotnet dev-certs https --trust
```

При выполнении этих команд, будет создан самоподписанный сертификат и добавлен в доверительные.

Если есть сертификат в формате pfx, то для его использования необходимо в конфигурационные файлы прописать блок

```
"Kestrel": {  
  "Certificates": {  
    "Default": {
```

```
"Path": "nginx-selfsigned.pfx", # путь, где размещен сертификат
"Password": "loyalty" # пароль, прописанный при создании сертификата
}
}
}
}
9
```

Для OS Linux

Сборки всех модулей, UI, Личного кабинета и файл с версией находятся в личном кабинете пользователя по адресу lk.supsoft.ru

У модуля должно быть разрешение на запуск.

Для того чтобы запустить модуль необходимо в терминале, находясь в каталоге с модулем выполнить команду

`module --urls="https://*:port"` , где `module` - название запускаемого модуля (например `RS.Loyalty.UI.ApiService`), `port` - номер свободного порта

Пример запуска модуля `RS.Loyalty.UI.ApiService` с использованием `systemd`

```
[Unit]
Description=UI.ApiService
[Service]
User=user
Group=user
```

```
WorkingDirectory=/home/user/Work/RSL6/UIApiService
```

```
ExecStart=/home/user/Work/RSL6/UIApiService/RS.Loyalty.UI.ApiService --
urls="https://:7000*"
```

```
Environment="DOTNET_gcServer=0"
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Для OS Windows

Сборки всех модулей, UI, Личного кабинета и файл с версией находятся в `\\build.corp.local\RSL6\develop`

Для того чтобы запустить модуль необходимо в командной строке необходимо, находясь в каталоге с модулем выполнить команду

`module.exe --urls="https://*:port"` , где `module.exe` - название запускаемого модуля (например `RS.Loyalty.UI.ApiService.exe`),
`port` - номер свободного порта

Есть отдельный модуль, который не должен работать постоянно.

CLI - это модуль, с помощью которого можно создать или обновить базу.

UI (Админка)

Работает через модуль `RS.Loyalty.UI.ApiService`

Пример настройки Nginx:

```
server {
listen 8082;
location / {
try_files $uri $uri/ /index.html =404;
root /var/www/html/wwwroot; # каталог, где размещен UI
index index.html index.htm;
}
location ~ (/api/auth/signal/upload) {
```

```
proxy_pass <https://localhost:7000;> # url модуля RS.Loyalty.UI.ApiService
proxy_set_header Host $host;
10
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}
}
```

ClientPortal (Личный кабинет клиента)

Работает через модуль RS.Loyalty.ClientPortal.ApiService

Пример настройки Nginx:

```
server {
listen 8092;
location / {
try_files $uri $uri /index.html =404;
root /var/www/html/rs/loyalty; # каталог, где размещен ClientPortal
index index.html index.htm;
}
location ~ (/api) {
proxy_pass <https://localhost:7009;> # url модуля RS.Loyalty.ClientPortal.ApiService
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}
}
11
```

Создание и обновление базы данных

Создание новой базы возможно двумя способами:

1. в CLI вызвать команду `rs-loyalty-cli data create-db`. При выполнении данной команды, будет создана база (название базы будет взято из `appsettings.json` самого CLI) и выполнена миграция.
2. Создать базу вручную, прописать название в `appsettings.json` самого CLI и выполнить команду `rs-loyalty-cli data migrate`. При выполнении команды будут выполнены и зарегистрированы миграционные скрипты

После этого запускаем сервисы.

Обновление базы, для которой уже применялась миграция

В CLI вызвать команду `rs-loyalty-cli data migrate`.

При выполнении этой команды, будут выполнены и зарегистрированы новые миграционные скрипты.

При выполнении этой команды, может изменяться структура таблиц, а так же могут быть очищены данные. В `readmodels_state` должны быть сброшены счетчики, для таблиц у которых было изменение структуры.

После запуска сервисов, данные должны будут перестроиться.

Обновление базы, к которой еще не применялась миграция

Поскольку при миграции, выполняются скрипты, в которых есть create таблиц, индексов, без проверки if not exist, выполнить команду rs-loyalty-cli data migrate без ошибок не получится.

Поэтому следует сделать следующие шаги:

1. Выполнить команду в CLI rs-loyalty-cli data register-migrations. При выполнении этой команды, будет создана таблица schemaversions и в ней будут зарегистрированы все миграционные скрипты.
2. Создать новую базу и выполнить для нее команду в CLI rs-loyalty-cli data migrate. Тем самым мы получим базу, с актуальной структурой.
3. Затем получить сравнить эти две базы и получить diff скрипт.
4. Поскольку в миграционных скриптах могут быть изменения структуры таблиц, то необходимо будет найти все строки в миграционных скриптах, в которых будет update таблицы readmodels_state и добавить эти строки в конец diff скрипта.
5. Запустить все сервисы.
6. Проверить, что в таблице readmodels_state нет записей, у которых в колонке globalsequencenumber значение 0.

Настройка параметров сервиса RS.Loyalty.Modules.Analytics.Service

Формат файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Serilog - настройка параметров логирования

AllowedHosts - настройка фильтрации адресов

MassTransitSettings - настройка параметров подключения к сервису MassTransit

Настройка параметров сервиса RS.Loyalty.ClientPortal.ApiService

Общая структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

AllowedHosts - настройка фильтрации адресов

Serilog - настройка параметров логирования

ClientSettings - настройка параметров ЛК и МП

JWTSettings - настройка параметров получения JWT токена

CacheSettings - настройка параметров кэширования

PrivacySettings - настройка параметров ЛК и МП

MassTransitSettings - настройка параметров подключения к сервису MassTransit

MProxySettings - настройка параметров подключения к сервису MProxy

Настройка параметров кэширования

Формат блока настроек:

StoreRefreshTimeOut - периодичность обновления кэша магазинов

TerritorialDivisionRefreshTimeOut - периодичность обновления кэша территориального деления

FeedBackReasonsTimeOut - периодичность обновления причин обращения обратной связи

Настройка параметров подключения к сервису MProxy

Формат блока настроек:

"Serilog": {},

"AllowedHosts": string,


```
"ConnectionString": string,  
"ClientSettings": {},  
"JWTSettings": {},  
"CacheSettings": {},  
"PrivacySettings": {},  
"MassTransitSettings": {},  
"MProxySettings": {}  
"CacheSettings": {  
"StoreRefreshTimeOut": 60,  
"TerritorialDivisionRefreshTimeOut": 60,  
"FeedBackReasonsTimeOut": 60  
}
```

```
"MProxySettings": {  
"Url": "https://m.proxy.root.url",  
"DisableSSL": true,
```

15

Url - адрес используемого сервиса

DisableSSL - отключение проверок сертификата

Timeout - таймаут выполнения операций

CompanyUID - UID компании из MProху

Token - Token компании из MProху

```
"Timeout": 60,
```

```
"CompanyUID" : "",
```

```
"Token" :
```

```
}
```

16

Настройка сервиса RS.Loyalty.Modules.Communications.Service

Файл настройки appsettings.json содержит следующие элементы:

ConnectionString - параметры подключения к базе данных

Serilog - настройка параметров логирования

Drivers - настройка параметров отправки коммуникаций

SendApi - настройка параметров отправки коммуникаций

ValidationSettings - настройка параметров отправки коммуникаций

ValidationServiceSettings - настройка параметров отправки коммуникаций

MassTransitSettings - настройка параметров подключения к сервису MassTransit

```
{
```

```
ConnectionString: "",
```

```
Serilog: {},
```

```
Drivers: {},
```

```
SendApi: {},
```

```
ValidationSettings: {},
```

```
ValidationServiceSettings: {},
```

```
MassTransitSettings: {}
```

```
}
```

Настройка параметров сервиса RS.Loyalty.Modules.CRM.Service

Структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования
Profiles - настройка параметров личного кабинета/мобильного приложения
MassTransitSettings - настройка параметров подключения к сервису MassTransit
LoyaltyCardSettings - настройка параметров карт

```
{  
  "ConnectionString": string,  
  "Logging": {},  
  "Serilog": {},  
  "Profiles": {},  
  "MassTransitSettings": {},  
  "LoyaltyCardSettings": {}  
}
```

Настройка параметров сервиса RS.Loyalty.Integration.ApiService

Структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования

EventsBusSettings - настройка параметров выгрузки во внешние системы

AllowedHosts - настройка списка фильтрации

MassTransitSettings - настройка параметров подключения к сервису MassTransit

```
{  
  "ConnectionString": string,  
  "Logging": {},  
  "Serilog": {},  
  "EventsBusSettings": {},  
  "AllowedHosts": string,  
  "MassTransitSettings": {}  
}
```

Настройка параметров сервиса RS.Loyalty.Modules.Locations.Service

Сервис интеграции отвечает за локализацию системы. Настройка работы сервиса осуществляется с помощью изменения набора

параметров в файле appsettings.json . Структура файла:

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования

CountyCodes - настройка списка часовых поясов

EventsBusSettings - настройка параметров выгрузки во внешние системы

AllowedHosts - настройка списка фильтрации

MassTransitSettings - настройка параметров подключения к сервису MassTransit

Настройка списка часовых зон

Осуществляется с помощью элемента:

RU - международное обозначение страны использования системы.

```
{  
  "ConnectionString": string,  
  "Serilog": {},  
  "Logging": {},  
}
```

```
"CountyCodes": {},
"AllowedHosts": string,
"MassTransitSettings": {}
}
1 "CountyCodes": ["RU"]
```

Настройка параметров сервиса RS.Loyalty.Modules.Loyalty.Service

Формат файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования

AllowedHosts - настройка списка фильтрации

CouponServiceSettings - настройка параметров генерации купонов

PromoCodeServiceSettings - настройка параметров генерации промо-кодов

Profiles - настройка параметров личного кабинета/мобильного приложения

MassTransitSettings - настройка параметров подключения к сервису MassTransit

Настройка параметров генерации купонов

PoolSize - размер пакета генерации купонов

GenerationNumberOfAttemptsByCycle - количество попыток генерации в цикле

GenerationNumberOfAttemptsByNumberGeneration - количество попыток генерации по номеру

GenerationNumberOfAttemptsByLifeCycle - количество попыток генерации по жизненному циклу

GenerationCycleTimeout - пауза между генерацией пакетов в секундах

Настройка параметров промо-кодов

```
{
"ConnectionString": string,
"Logging": {},
"Serilog": {},
"AllowedHosts": string,
"CouponServiceSettings": {},
"PromoCodeServiceSettings": {},
"MassTransitSettings": {}
}
"CouponServiceSettings": {
"PoolSize": 10000,
"GenerationNumberOfAttemptsByCycle": 100,
"GenerationNumberOfAttemptsByNumberGeneration ": 1000,
"GenerationNumberOfAttemptsByLifeCycle": 100,
"GenerationCycleTimeout": 600
}
"PromoCodeServiceSettings": {
"PoolSize": 10000,
"GenerationNumberOfAttemptsByCycle": 100,
"GenerationNumberOfAttemptsByLifeCycle": 100,
21
```

PoolSize - размер пакета генерации промо-кодов

GenerationNumberOfAttemptsByCycle - количество попыток генерации в цикле

GenerationNumberOfAttemptsByLifeCycle - количество попыток генерации по жизненному циклу

GenerationCycleTimeout - пауза между генерацией пакетов в секундах

5

6

"GenerationCycleTimeout": 600

22

Настройка параметров сервиса RS.Loyalty.Modules.ProductCatalog.Service

Структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования

AllowedHosts - настройка фильтрации адресов

MassTransitSettings - настройка параметров подключения к сервису MassTransit

{

"ConnectionString": string,

"Logging": {},

"Serilog": {},

"AllowedHosts": string,

"MassTransitSettings": {}

}

Настройка параметров сервиса RS.Loyalty.Modules.Security.Service

Структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Logging - настройка уровня логирования

Serilog - настройка параметров логирования

SecuritySettings - настройка параметров безопасности

AllowedHosts - настройка фильтрации адресов

MassTransitSettings - настройка параметров подключения к сервису MassTransit

Настройка параметров безопасности

Формат блока настроек:

SecurityKey - секретный ключ сервиса

ValidIssuer - сервис - эмитент

ValidAudience - сервис - получатель

ExpiryInMinutes - время действия токена

RefreshExpiryInMinutes - время обновления токена

{

"ConnectionString": string,

"Logging": {},

"Serilog": {},

"SecuritySettings": {},

"AllowedHosts": string,

"MassTransitSettings": {}

}

"SecuritySettings": {

"SecurityKey": "imSoseCretKeyYEAHyeahIWILLkEEpAllSEcreTsIHAVEn0Rem0rse",

```
"ValidIssuer": "RS.Loyalty.UI.ApiService",  
"ValidAudience": "RS.Loyalty.UI.ApiService",  
"ExpiryInMinutes": 600,  
"RefreshExpiryInMinutes": 6000  
}
```

Настройка параметров сервиса RS.Loyalty.Service

Структура файла appsettings.json :

ConnectionString - параметры подключения к базе данных

Serilog - настройка параметров логирования

JWTSettings - настройка параметров получения JWT токена

Licence - номер лицензии

AllowedHosts - настройка фильтрации адресов

MassTransitSettings - настройка параметров подключения к сервису MassTransit

```
{  
"ConnectionString" : string,  
"Serilog": {},  
"JWTSettings": {},  
"Licence": string,  
"AllowedHosts": string,  
"MassTransitSettings": {}  
}
```

Настройка параметров логирования

Параметры логирования задаются элементом вида:

Using - указывает на используемые для логирования каналы вывода. Доступно два элемента: Serilog.Sinks.Console - вывод на консоль, Serilog.Sinks.File - запись в файл.

MinimumLevel - минимальный уровень логирования. Доступны уровни Info, Debug, Error

WriteTo - определяет устройства вывода сообщений лога. Console - вывод сообщений на экран, File - запись в файл. Параметр path определяет расположение и имя файла лога, rollingInterval - интервал записи лога в файл.

```
"Serilog": {  
"Using": [  
"Serilog.Sinks.Console",  
"Serilog.Sinks.File"  
],  
"MinimumLevel": "Debug",  
"WriteTo": [  
{  
"Name": "Console"  
},  
{  
"Name": "File",  
"Args": {
```

```
"path": "/home/user/Work/RSL6/Logs/CommunicationsService/module-communications-.log",
"rollingInterval": "Day"
}
}
]
}
```

Настройка подключения к базе данных

Подключение к БД задается строкой подключения вида:

Host - адрес SQL сервиса

Port - порт подключения к БД. Порт используемый по умолчанию - 5432

Database - имя базы данных

Username - имя пользователя для подключения к БД

Password - пароль пользователя для подключения к БД

Maximum Pool Size - максимальное количество подключений к БД. Значение по умолчанию - 1024.

```
1 "Host=localhost;Port=5432;Database=rsl6_2;Username=rsl;Password=rsl;Maximum Pool Size=1024;"
```

Настройка белого списка хостов

С помощью белого списка ограничивается внешний доступ фильтрацией ip адресов. Формат элемента:

AllowedHosts - список хостов, которым будет предоставлен доступ

* - доступ будет предоставлен любому хосту

192.168.0.1 - запросы будут обрабатываться только от хоста с указанным адресом

```
1 "AllowedHosts": "*"
```

Настройка подключения к сервису MassTransit

Для реализации сервиса используется RabbitMQ. Настройка подключения осуществляется с помощью блока:

Host - адрес узла, на котором расположен сервис

Port - порт подключения к сервису. Порт по умолчанию - 5672

UserName - логин для подключения к сервису

Password - пароль для подключения к сервису.

```
"MassTransitSettings": {
  "Host": "localhost",
  "Port": 5672,
  "UserName": "user",
  "Password": "user"
}
```

Настройка уровня логирования

Для тех сервисов, в которых предусмотрена настройка уровня логирования, используется элемент следующего формата:

LogLevel - уровень логирования

Default - уровень логирования сервиса по умолчанию

Microsoft.AspNetCore - уровень логирования фреймворка ASP.NET Core

```
"Logging": {  
  "LogLevel": {  
    "Default": "Error",  
    "Microsoft.AspNetCore": "Warning"  
  }  
}
```

Настройка параметров получения JWT токена

Формат блока настроек:

securityKey - секретный ключ

ValidIssuer - сервис - эмитент

ValidAudience - сервис - получатель

ExpiryInMinutes - время действия токена

```
"JWTSettings": {  
  "securityKey": "CodeMazeSecretKey",  
  "validIssuer": "CodeMazeAPI",  
  "validAudience": "https://localhost:6001",  
  "expiryInMinutes": 600  
}
```

Установка аналитических компонент Retail Suite.Global Loyalty&CRM Analytics и Retail Suite.Global Loyalty&CRM ML/AI

Для работы компонент Retail Suite.Global Loyalty&CRM Analytics и Retail Suite.Global Loyalty&CRM ML/AI необходимо предварительно установить сервисы Retail.Suite business Intelligence, docker (по официальной инструкции к используемому дистрибутиву <https://docs.docker.com/engine/install/>)

2. Авторизоваться в корпоративном docker-registry

```
echo "${REGISTRY_PASSWORD}" | docker login "${REGISTRY_HOST}" -u  
"${REGISTRY_USER}" --password-stdin
```

3. Загрузить на сервер, где будет развернут стенд, архив с необходимой версией релиза. Релиз доступен для пользователей в личном кабинете lk.supsoft.ru
Архив дистрибутива

```
distribution-${RETAIL_SUITE_GLOBAL.LOYALTY&CRM_VERSION}.zip
```

4. Переместить полученный архив в /opt, распаковать его и переименовать директорию, полученную в результате распаковки архива

```
mv distribution-${RETAIL_SUITE_GLOBAL.LOYALTY&CRM_VERSION} /opt && \  
cd /opt && \  
unzip distribution-${RETAIL_SUITE_GLOBAL.LOYALTY&CRM_VERSION} && \  
mv distribution Retail Suite Global.Loyalty&CRM-micro
```

5. Переименовать .env_template в .env

```
cd /opt/Retail Suite Global.Loyalty&CRM-micro && mv .env_template .env
```

6. Внести соответствующие изменения в .env-файл. Ниже перечислены переменные, значения которых необходимо установить

POSTGRES_PASSWORD - пароль пользователя postgres. Все микросервисы используют пользователя postgres, при работе с базой данных

KEYCLOAK_ADMIN_PASSWORD - пароль admin-пользователя для доступа в консоль keycloak

KEYCLOAK_HOSTNAME - хостнейм или ip-адрес хоста, который будет ассоциирован с консолью keycloak.

API_GATEWAY_EXTERNAL_URL - URL по которому будет доступен портал.

SHELFSPACE_ADMIN_EMAIL - email admin-пользователя портала

METATRON_DB_USER - пользователь СУБД postgres

METATRON_DB_PASSWORD - пароль пользователя postgres

DISCOVERY_DATASOURCE_USERNAME - пользователь СУБД postgres

DISCOVERY_DATASOURCE_PASSWORD - пароль пользователя postgres

MDX_DATASOURCE_USERNAME -пользователь СУБД postgres

MDX_DATASOURCE_PASSWORD - пароль пользователя postgres

7. Пример .env-файл, в котором заполнены все обязательные переменные

```
DOCKER_REGISTRY_HOST=registry.supsoft.ru
```

```
BI_VERSION=97.1.0
```

```
BI_UI_SOURCE=rs
```

```
API_GATEWAY_EXTERNAL_URL=test-docker
```

```
# SSL
```

```
SSL_ENABLED=true
```

```
SSL_TRUST_STORE_PASSWORD=password
```

```
# Portainer
```

```
PORTAINER_EXTERNAL_PORT=9100
```

```
# Postgres
```

```
POSTGRES_EXTERNAL_PORT=5432
```

```
POSTGRES_PASSWORD=password
```

```
# Keycloak
```

```
KEYCLOAK_HOSTNAME=test-docker
```

```
KEYCLOAK_EXTERNAL_PORT=9000
```

```
KEYCLOAK_ADMIN_PASSWORD=admin
```

```
KEYCLOAK_IMPORT_FILE=rs-bi-realm.json
```

```
# Microservices common
```

```
EUREKA_INSTANCE_PREFER_IP_ADDRESS=false
```

```
EUREKA_REGISTRY_EXTERNAL_PORT=8761
```

```
CONFIG_SERVER_EXTERNAL_PORT=8888
```

```
GATEWAY_EXTERNAL_PORT=8765
```

```
# OAuth
```

```
OAUTH_SERVER_URL=http://keycloak:8080
```

```
OAUTH_REALM_NAME=rs-bi
```

```
OAUTH_CLIENT_ID=oauth
```

```
OAUTH_CLIENT_SECRET=PeLRGb4QatijQjKi7DiicJrN6Xb1TWEs
```

```
OAUTH_RESOURCE_ACCESS_RESOURCE_NAME=oauth
```



```

# Discovery Legacy
DISCOVERY_LEGACY_VERSION=development-latest
DISCOVERY_LEGACY_SERVER_URL=http://discovery-legacy:8180
METATRON_JAVA_OPTS=$JAVA_OPTS -Xms2g -Xmx4g -
XX:MaxMetaspaceSize=512m
METATRON_ENV_PROFILES=local,postgres-default-db,logging-
file,scheduling,microservice,managements
METATRON_CACHE_PATH=cache
METATRON_SMTP_HOST=<SMTP-узел>
METATRON_SMTP_PORT=<SMTP-порт>
METATRON_SMTP_USERNAME=<SMTP-пользователь>
METATRON_SMTP_PASSWORD=<SMTP-пароль>

METATRON_DB_TYPE=postgres
METATRON_DB_SCHEMA=discovery_legacy
METATRON_DB_URL=jdbc:postgresql://db:5432/rs_bi?currentSchema=$METATRO
N_DB_SCHEMA
METATRON_DB_USER=postgres
METATRON_DB_PASSWORD=${POSTGRES_PASSWORD}

METATRON_MAIL_BASE_URL=test-docker
HADOOP_HDFS_BASE_URL=hdfs://<HDFS_URL>:8020
HADOOP_SPARK_LIVY_URL=http://<SPARK_LIVY_URL>:8999
HADOOP_SPARK_HISTORY_URL=http://<SPARK_HISTORY_URL>:18081
HADOOP_HIVE_HOSTNAME=<HIVE_URL>
HADOOP_HIVE_PORT=10500
HADOOP_HIVE_USER=hive
HADOOP_HIVE_PASSWORD=hive
HADOOP_THRIFT_HIVE_URL=thrift://<THRIFT_HIVE_URL>:9083

# Discovery API
DISCOVERY_DATASOURCE_URL=jdbc:postgresql://db:5432/rs_bi
DISCOVERY_DATASOURCE_SCHEMA_NAME=discovery_api
DISCOVERY_DATASOURCE_USERNAME=postgres
DISCOVERY_DATASOURCE_PASSWORD=${POSTGRES_PASSWORD}

# MDX API
MDX_DATASOURCE_URL=jdbc:postgresql://db:5432/rs_bi
MDX_DATASOURCE_SCHEMA_NAME=mdx_api
MDX_DATASOURCE_USERNAME=postgres
MDX_DATASOURCE_PASSWORD=${POSTGRES_PASSWORD}

# Logging
LOGGING_LOGSTASH_ENABLED=false
LOGGING_LOGSTASH_DESTINATION=localhost:5000

```

8. После подготовки .env-файла, необходимо запустить все микросервисы.

Запуск выполняется с использованием docker-compose

```
cd /opt/Retail Suite Global.Loyalty&CRM-micro && docker compose --env-file .env up -d
```

9. Статус запущенных контейнеров можно посмотреть через docker

```
cd /opt/Retail Suite Global.Loyalty&CRM-micro && docker compose ps
```

Либо через portainer, доступный по адресу

```
http://${SERVER_IP}:9000
```

Во время первого доступа в portainer, необходимо будет создать admin-пользователя.

10. Для доступа к portalу удобно использовать nginx в качестве прокси.

Пример виртуального хоста для портала:

```
server {
```

```
    listen 80;
```

```
    server_name test-docker;
```

```
    charset utf-8;
```

```
    set $gateway 'http://127.0.0.1:7070';
```

```
    proxy_http_version 1.1;
```

```
    proxy_set_header Host $host;
```

```
    proxy_redirect .* $host;
```

```
    proxy_cookie_domain .* $host;
```

```
    proxy_connect_timeout 3600;
```

```
    proxy_send_timeout 3600;
```

```
    proxy_read_timeout 3600;
```

```
    default_type text/html;
```

```
    client_max_body_size 0;
```

```
    error_log /var/log/nginx/test-docker/error.log error;
```

```
    access_log /var/log/nginx/test-docker/access.log;
```

```
    location ^~/ {
```

```
        proxy_pass $gateway$request_uri;
```

```
    }
```

```
    location ^~/stomp/ {
```

```
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection "upgrade";
```

```
        proxy_pass $gateway$request_uri;
```

```
    }
```

```
}
```

11. Просмотр логов контейнера

```
cd /opt/Retail Suite Global.Loyalty&CRM-micro && docker logs {ID_CONTAINER}
```

12. Остановка стека докер контейнеров

```
cd /opt/Retail Suite Global.Loyalty&CRM-micro && docker compose down --volume
```

Обновление стенда на следующий релиз

1. Остановить контейнеры

```
docker compose down
```

2. В файле .env в параметре BI_VERSION прописать новое значение версии, и запустить повторно стек контейнеров. Произойдет скачивание новых образов docker содержащих более свежую версию Retail Suite Global.Loyalty&CRM.

```
docker compose up -d
```

Далее необходимо установить кластер HDP 3.0

Установка всех компонент кластера описана в документации [HDP3 по ссылке](#)

Подготовка окружений кластера, установка сертификатов

Зайти под пользователем root на сервер для установки сервиса Ambari

```
ssh-keygen  
.ssh/id_rsa  
.ssh/id_rsa.pub  
cat id_rsa.pub >> authorized_keys
```

.ssh папку скопировать в /root и в home директорию user'a, под которым будем ставить (/home/aloha).

Изменить права доступа на:

```
chmod 700 ~/.ssh  
chmod 600 ~/.ssh/authorized_keys
```

На каждом сервере проверить подключение к другим серверам под обоими пользователями:

```
[root@apachai1 aloha]# ssh root@apachai4.apm.local ls  
The authenticity of host 'apachai4.apm.local (10.10.8.149)' can't be established.  
ECDSA key fingerprint is SHA256:HPaVr/HxhhjbrCCrflY5Hz50hMRp5pEsl9jbsxrM0Gc.  
ECDSA key fingerprint is MD5:92:ee:a2:cd:07:a1:fd:98:83:93:b2:20:1b:fc:8f:29.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'apachai4.apm.local,10.10.8.149' (ECDSA) to the list of known hosts.  
anaconda-ks.cfg  
[root@apachai1 aloha]# ssh root@apachai4.apm.local ls  
anaconda-ks.cfg
```

Проверить на каждом сервере, что пользователь получает root привилегии без введения пароля:

```
[aloha@apachai1 ~]$ sudo su -
Последний вход в систему:Пт янв 25 11:28:45 MSK 2019с 10.14.3.89на pts/0
[root@apachai1 ~]#
```

Синхронизировать время на серверах, например утилитой ntp

```
[root@apachai1 aloha]# yum install -y ntp
...
Установлено:
ntp.x86_64 0:4.2.6p5-28.el7.centos
Установлены зависимости:
autogen-libopts.x86_64 0:5.18-5.el7 ntpdate.x86_64 0:4.2.6p5-28.el7.centos
Выполнено!
[root@apachai1 aloha]# systemctl enable ntpd
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.
[root@apachai1 aloha]#
```

Создать БД PostgresPRO\PostgreSQL

user= postgres

```
CREATE DATABASE rangerkms;
CREATE DATABASE rangeradmin;
CREATE DATABASE oozie;
CREATE DATABASE hive;
CREATE USER rangerkms WITH PASSWORD 'rangerkms';
CREATE USER rangeradmin WITH PASSWORD 'rangeradmin';
CREATE USER hive WITH PASSWORD 'hive';
CREATE USER oozie WITH PASSWORD 'oozie';
GRANT ALL PRIVILEGES ON DATABASE rangerkms TO rangerkms;
GRANT ALL PRIVILEGES ON DATABASE rangeradmin TO rangeradmin;
GRANT ALL PRIVILEGES ON DATABASE hive TO hive;
GRANT ALL PRIVILEGES ON DATABASE oozie TO oozie;
```

Даем доступ к подключению с соседних машин (в этом случае вообще всем внешним машинам)

```
-bash-4.2$ vim /var/lib/pgsql/data/postgresql.conf
listen_addresses = '*'
-bash-4.2$ vim /var/lib/pgsql/data/pg_hba.conf
host all all 0.0.0.0/0 md5
```

```
-bash-4.2$ psql
psql (9.2.24)
Введите "help", чтобы получить справку.
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 строка)
postgres=#
```

Добавляем драйвер postgresql-jdbc

```
find / -name postgresql-jdbc.jar 2>/dev/null
```

Если не нашли, ставим:

```
yum install postgresql-jdbc*  
ls /usr/share/java/postgresql-jdbc.jar  
chmod 644 /usr/share/java/postgresql-jdbc.jar
```

Регистрируем на сервере

```
ambari-server setup --jdbc-db=postgres --jdbc-driver=/usr/share/java/postgresql-jdbc.jar
```

Задаем предварительные настройки Ambari

Hive: hive.strict.managed.tables = false

Hdfs: hadoop.proxyuser.airflow.hosts=*, hadoop.proxyuser.airflow.hosts=*

YARN Queue Manager: Создаем очереди: aloha, etl.

Ranger. Создаем пользователя для схем баз данных.

Создать пользователя и группу: group=rsl, user=rsl
Добавить соответствующие права.

HDFS: Создать каталог user'a на hdfs:

```
hdfs dfs -mkdir /user/rsl  
sudo -u hdfs hadoop fs -chown -R nch /user/rsl
```

Также нужно проверить права необходимых пользователей: aloha, airflow, hive, spark3

Spark. Создаем структуру проекта

GIT: Репозиторий проекта:

<https://git.polet-it.ru/rs.analytics/analytics/-/tree/development/RSL/>

Развернуть объекты базы данных с помощью скрипта \scripts\CreateDBObjects.scala,
из папки

<https://git.polet-it.ru/rs.analytics/analytics/-/tree/development/RSL/scripts/ddl/rsl>.

Для этого на стенде rsl6reg (10.14.8.33)

- создать папку /home/aloha/scripts,
- скопировать содержимое папки RSL/scripts/ из репозитория,
- запустить spark-shell (spark v 3.4)

```
spark-shell --master local --packages io.delta:delta-core_2.12:2.4.0,com.github.scopt:scopt_2.12:4.0.1 --conf  
"spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension" --conf  
"spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
```

- запустить CreateDBObjects.scala с соответствующими параметрами:

```
"-c", "demo", "-p", "/home/aloha/scripts/ddl/rsl/stage", "-a", "rsl", "-s", "stage", "-t", "external"
```

(аналогично для tmp, dmart),

где demo – имя клиента,
/home/aloha/scripts/ddl/rsl/stage – путь к каталогу со скриптами создания объектов, rsl – название проекта,
stage – схема БД (stage, tmp, dmart)

Spark. Создаем и заполняем служебные объекты БД

Информацию по служебным объектам скопировать в виде csv-файлов в соответствующие нужным таблицам папки в схеме stage: hdfs:///user/aloha/stage/demo (valid_periods, time_periods, calendar, events, и др).

Затем загрузить их в tmp, dmart соответствующими скриптами.

Postgres. Создаем объекты БД

GIT: <https://git.polet-it.ru/rs.analytics/analytics/-/tree/development/RSL/source>

Создать необходимые объекты БД Postgres с помощью соответствующих скриптов.

Установка интеграционных компонент

Необходимо установить на сервер утилиты java 11 и karaf.

```
wget https://archive.apache.org/dist/karaf/4.4.1/apache-karaf-4.4.1.tar.gz
```

```
tar -xvzf ./apache-karaf-4.4.1.tar.gz
```

```
wget https://download.java.net/java/GA/jdk11/9/GPL/openjdk-11.0.2_linux-x64_bin.tar.gz
```

```
tar -xvzf ./openjdk-11.0.2_linux-x64_bin.tar.gz
```

edit bash_profile

```
# .bash_profile
```

```
# Get the aliases and functions
```

```
if [ -f ~/.bashrc ]; then
```

```
<----->. ~/.bashrc
```

```
fi
```

```
# User specific environment and startup programs
```

```
export JAVA_HOME=/home/karaf/jdk-11
```

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin
```

```
export PATH
```

Запустить karaf в качестве system сервиса

Запустить karaf с поддержкой консоли

```
$KARAF_HOME/bin/karaf
```

Установить feature Service Wrapper

```
karaf@root(>) feature:install service-wrapper
```

Подготовить Apache Karaf для регистрации в качестве сервиса/демона в операционной системе

```
karaf@root(>) wrapper:install
```

Выключить karaf

```
karaf@root(>) system:shutdown
```

Переключиться на root пользователя

```
sudo su -
```

Отредактировать karaf.service для запуска экземпляра karaf под пользователем karaf

```
vi $KARAF_HOME/bin/karaf.service
```

```
# Добавить в раздел [Service]
```

```
User=karaf
```

```
Group=karaf
```

```
# Сохранить
```

Зарегистрировать Apache Karaf в качестве сервиса/демона

```
systemctl enable $KARAF_HOME/bin/karaf.service
```

Запустить karaf

```
systemctl start karaf
```

Настроить репозитории в системе

```
$KARAF_HOME/etc/org.ops4j.pax.url.mvn.cfg
org.ops4j.pax.url.mvn.repositories= \
  https://archiva.polet-it.ru/repository/internal/@id=internal, \
  https://archiva.polet-it.ru/repository/snapshots/@id=snapshots@snapshots@noreleases, \
  https://archiva.polet-it.ru/repository/mirror2/@id=mirror2,
feature:install maven
feature:repo-add mvn:org.apache.activemq/artemis-features/2.20.0/xml/features
feature:repo-add mvn:org.ops4j.pax.jms/pax-jms-features/1.1.1/xml/features
feature:install artemis-jms-client artemis-core-client
feature:install pax-jms-config pax-jms-artemis pax-jms-pool-narayana
feature:repo-add mvn:io.hawt/hawtio-karaf/2.14.3/xml/features
feature:install hawtio
```

CXF Нужно добавить перед camel, для исправления неправильной зависимости

```
feature:repo-add mvn:org.apache.cxf.karaf/apache-cxf/3.4.5/xml/features
feature:repo-add mvn:org.apache.camel.karaf/apache-camel/3.14.0/xml/features
// Camel и его модули ставить разными командами
feature:install camel
```

feature:install camel-jms camel-cxf camel-kafka

feature:repo-add mvn:com.retail/rsl-integration-features/1.1.0-SNAPSHOT/xml/features

Repository:

\$KARAF_HOME/etc/org.ops4j.pax.url.mvn.cfg

Необходимо установить флаг:

org.ops4j.pax.url.mvn.defaultLocalRepoAsRemote = true

Доступ в Hawtio:

\$KARAF_HOME/etc/users.properties

Необходимо раскомментировать:

karaf = karaf,_g_:admingroup

g:admingroup = group,admin,manager,viewer,systembundles,ssh

Для корректного решения зависимостей.

.m2/repository/org/apache/camel/karaf/apache-camel/apache-camel-3.14.0-features.xml

Заменить:

1.

<bundle>mvn:org.codehaus.woodstox/stax2-api/3.1.4</bundle>

<bundle>mvn:org.codehaus.woodstox/woodstox-core-asl/4.4.1</bundle>

на

<bundle dependency="true">mvn:org.codehaus.woodstox/stax2-api/3.1.4</bundle>

<bundle dependency="true">mvn:org.codehaus.woodstox/woodstox-core-asl/4.4.1</bundle>

2.

На версии apache-cxf 4.3.5 не актуально:

<repository>mvn:org.apache.cxf.karaf/apache-cxf/(3,4)/xml/features</repository>

на

<repository>mvn:org.apache.cxf.karaf/apache-cxf/3.4.5/xml/features</repository>

Настройка:

\$KARAF_HOME/etc/ com.retail.rsl6.cfg

Необходимо заполнить следующим содержимым:

kafka.bootstrap-servers=vmi250453.contaboserver.net:6667, \
vmi250624.contaboserver.net:6667, \
vmi250627.contaboserver.net:6667, \
vmi250631.contaboserver.net:6667

kafka.topic=rsl6_test

kafka.autoOffsetReset=latest

kafka.groupId=


```
kafka.repositoryFile=tmp/repo.dat  
kafka.offsetRepository=Y
```

```
$KARAF_HOME/etc/ com.retail.rsl6.cfg
```

Необходимо заполнить следующим содержимым:

```
osgi.jdbc.driver.name=PostgreSQL JDBC Driver  
dataSourceName=rsl6DS  
databaseName=rsl6_stage  
pool=hikari  
user=?  
password=?  
url=jdbc:postgresql://rsl6reg.corp.local:5432/rsl6_stage?currentSchema=rsl6_test  
hikari.maximumPoolSize=20
```

Установить и настроить Apache Airflow

Подготовка:

Установить Python 3.10 с зависимостями

Установить PostgreSQL\ PostgersPRO

Создать базу данных airflow

Установить RabbitMQ

Настроить кластерный режим RabbitMQ

Убедиться, что RabbitMQ демоны не запущены

Определить, какой узел будет MASTER.

На non-MASTER узлах бэкапим .erlang.cookie файл

```
mv /var/lib/rabbitmq/.erlang.cookie /var/lib/rabbitmq/.erlang.cookie.backup
```

Копируем файл “/var/lib/rabbitmq/.erlang.cookie” с MASTER узла на другие узлы и сохраняем по тому же пути.

Устанавливаем права на .erlang.cookie файл

```
chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie  
chmod 600 /var/lib/rabbitmq/.erlang.cookie
```

Запускаем MASTER RabbitMQ Daemon

```
systemctl start rabbitmq-server.service
```

На каждом non-MASTER узле добавляем их в кластер и запускаем

```

# Запускаем сервис
systemctl start rabbitmq-server.service
# Останавливаем приложение
rabbitmqctl stop_app
# Добавляем текущую машину в кластер
rabbitmqctl join_cluster rabbit@{MASTER_HOSTNAME}
# например, где MASTER_HOSTNAME = vmi471363, т.е.
rabbitmqctl join_cluster rabbit@vmi471363
# Запускаем приложение
rabbitmqctl start_app
# Проверяем статус
rabbitmqctl cluster_status
# Проверка статуса должна вернуть нечто следующее:
# Cluster status of node rabbit@{NODE_HOSTNAME} ...
#
[{nodes,[[disc,[rabbit@{MASTER_HOSTNAME},rabbit@{NODE_HOSTNAME}]]],
#
{running_nodes,[rabbit@{MASTER_HOSTNAME},rabbit@{NODE_HOSTNAME}]}]}

```

TODO: Setup a load balancer to balance requests between the the Nodes

Port Forwarding

Port 5672 (TCP) → Port 5672 (TCP)

Port 15672 (HTTP) → Port 15672 (HTTP)

Health Check

Protocol: HTTP

Ping Port: 15672

Ping Path: /

TODO: Point all processes to that LB

Установка Apache Airflow

MASTER (airflow-webserver, airflow-scheduler, airflow-worker)

SLAVE (airflow-webserver, airflow-worker)

На каждом узле (MASTER, SLAVE):

Залогиниться под airflow.

Активируем виртуальную среду

```
source ~/software/env3/bin/activate
```

Устанавливаем [из PyPI](#) с extras модулями:

```
Pip install "apache-  
airflow[async,celery,sqlite,postgres,microsoft.mssql,odbc,oracle,rabbitmq,apache  
.hdfs,apache.hive,apache.livy,apache.spark,apache.sqoop,common.sql,ftp,http,i  
map,jdbc,ssh]==2.7.1" --constraint  
"https://raw.githubusercontent.com/apache/airflow/constraints-2.7.1/constraints-  
3.10.txt"
```

Необходимые модули всегда можно установить по мере надобности

Создаем переменную среды AIRFLOW_HOME

```
echo 'export AIRFLOW_HOME=/opt/airflow' >> ~/.bash_profile  
source ~/.bash_profile
```

Создать директорию \${AIRFLOW_HOME}

```
sudo mkdir /opt/airflow  
sudo chown -R airflow:airflow /opt/airflow
```

Запускаем команду "airflow", чтобы создавался конфигурационный файл в
\${AIRFLOW_HOME} .

```
airflow config list --defaults
```

Меняем владельца директории \${AIRFLOW_HOME} на airflow

```
chown -R airflow:airflow /opt/airflow
```

Изменяем настройки {AIRFLOW_HOME}/airflow.cfg

```
cd /opt/airflow  
vi airflow.cfg
```

на следующие:

```
[core]  
# Для работы в кластерном режиме необходим CeleryExecutor  
executor = CeleryExecutor
```

```
# Не загружаем примеры  
load_examples = False
```

```
[database]  
# Подключение к БД, созданной ранее  
# sql_alchemy_conn = postgresql+psycopg2://<user>:<password>@<host>/<db>  
sql_alchemy_conn =  
postgresql+psycopg2://airflow:airflow@vmi648694.contaboserver.net:5432/airflo  
w
```

Пул подключений к БД

```
sql_alchemy_pool_size = 50
# Дополнительное количество подключений
sql_alchemy_max_overflow = 25
```

```
[celery]
# Устанавливаем URL брокера RabbitMQ (если используем CeleryExecutor)
# broker_url = amqp://guest:guest@{RABBITMQ_HOST}:5672/
broker_url = amqp://admin:E6fCKIfCDprG@vmi648694.contaboserver.net:5672/

# Устанавливаем ссылку на БД для сохранения результатов работы
result_backend
db+postgresql://airflow:airflow@vmi648694.contaboserver.net:5432/airflow
```

```
[smtp]
По необходимости.
```

Создать директорию для DAG:

```
mkdir /opt/airflow/dags
chown -R airflow:airflow /opt/airflow
```

На MASTER узле:

Инициализируем БД

```
airflow db init
```

Создаем пользователя интерфейса admin:

```
airflow users create \
    --username admin \
    --password <password> \
    --firstname Admin \
    --lastname Adminoff \
    --role Admin \
    --email info-test@hexone.ru
```

Настраиваем Systemd для запуска Airflow

```
# Скачиваем архив Airflow
cd /tmp/
wget https://github.com/apache/incubator-airflow/archive/2.7.1.zip

# Разархивируем
unzip 2.7.1.zip
# Переходим в директорию со скриптами
cd airflow-2.7.1/scripts/systemd/
# Обновляем содержимое файлов.
# Устанавливаем AIRFLOW_HOME
```

```
# Устанавливаем VENV_HOME
vi airflow
AIRFLOW_HOME=/opt/airflow
VENV_HOME=/home/airflow/software/env3
JAVA_HOME=/usr/jdk64/jdk1.8.0_112
HADOOP_CONF_DIR=/usr/hdp/current/hadoop-client/conf
HIVE_CONF_DIR=/usr/hdp/current/hive-client/conf

# Копируем airflow property файл в целевую директорию
cp airflow /etc/sysconfig/

# Обновляем содержимое airflow-*.service файлов
# Устанавливаем User и Group значения пользователя и группы под
которым будет запускать сервисы airflow
# В нашем случае оставляем airflow
vi airflow-*.service

# Исправляем в ExecStart airflow-webserver.service
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow webserver'
# Исправляем в ExecStart airflow-scheduler.service
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow scheduler'
# Исправляем в ExecStart airflow-worker.service
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow celery worker -q default'
# Исправляем в ExecStart airflow-flower.service
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow celery flower'
# Исправляем в ExecStart airflow-triggerer.service
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow triggerer'

# Удалить из airflow-*.service "--pid /run/airflow/*.pid"
# pid всех демонов airflow должны создаваться в AIRFLOW_HOME

# Копируем airflow services файлы в целевую директорию
cp airflow-*.service /etc/systemd/system/

Включаем автозапуск во время запуска системы

chkconfig airflow-webserver on
chkconfig airflow-scheduler on
chkconfig airflow-worker on
chkconfig airflow-flower on
chkconfig airflow-triggerer on

Запускаем демоны airflow
```

```
systemctl start airflow-webserver
systemctl start airflow-scheduler
systemctl start airflow-worker
systemctl start airflow-flower
systemctl start airflow-triggerer
```

На SLAVE узле:

Копируем файлы airflow, airflow-*.service в соответствующие целевые директории из MASTER узла.

Включаем автозапуск во время запуска системы

```
chkconfig airflow-webserver on
chkconfig airflow-worker on
```

Запускаем демоны airflow

```
systemctl start airflow-webserver
```

```
systemctl start airflow-worker
```

Установка дополнительных модулей Airflow

На каждом узле Airflow:

Залогиниться под airflow.

Активируем виртуальную среду

```
source ~/software/env3/bin/activate
```

Устанавливаем модуль, на примере apache-airflow-providers-oracle:

```
pip install apache-airflow-providers-oracle
```

На MASTER узле:

Перезагрузить airflow-webserver

```
systemctl restart airflow-webserver
```

Добавление очередей в Worker.

Залогиниться под airflow

Остановить worker

```
sudo systemctl stop airflow-worker
```

Добавить новую очередь в скрипт запуска worker'a

```
cd /etc/systemd/system/  
sudo vi airflow-worker.service  
# Добавляем в список после флага '-q' через запятую без пробелов новые  
очереди,  
которые будет обрабатывать worker: н-р,  
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&  
{VENV_HOME}/bin/airflow celery worker -q default,etl_loy,etl_scm'  
Сохраняем
```

Запускаем worker

```
sudo systemctl daemon-reload  
sudo systemctl start airflow-worker
```

Экспорт/Импорт глобальных переменных в Airflow

Экспорт

```
source ~/software/env3/bin/activate  
airflow variables export -v variables_export.json
```

Импорт

```
source ~/software/env3/bin/activate  
airflow variables import -v variables_export.json
```

RSL. Airflow. Развертывание цепочки загрузок.

Airflow. Проверить/Установить необходимую версию pyspark:

```
Активировать environment,  
e.g. source ~/software/env3/bin/activate  
pip install pyspark==3.4.0
```

Airflow. Установить дополнительные пакеты (если не установлены):
(см Providers packages reference — [apache-airflow-providers Documentation](#))

```
pip install apache-airflow-providers-jdbc  
pip install apache-airflow-providers-postgres
```

Hadoop, Ranger: Проверить/создать необходимых пользователей:

```
airflow:hadoop, aloha:hadoop, rsl, hive
```

Ambari: Настроить hadoop config:

```
hadoop.proxyuser.airflow.groups=*
```

```
hadoop.proxyuser.airflow.hosts=*
```

RabbitMQ. Создать необходимые очереди для работы (etl_loy).

Возможно, добавить/изменить очереди:

Исправляем в ExecStart airflow-worker.service

```
ExecStart=/bin/bash -c 'source ${VENV_HOME}/bin/activate &&
${VENV_HOME}/bin/airflow celery worker -q default,etl_loy'
default оставим для общих задач
Airflow. Создать/импортировать Airflow variables для проекта RSL.
Airflow. Настроить connections типа spark
(Connection Id=spark_etl, Type=Spark, Host=yarn,
Extra={"queue": "etl", "deploy-mode": "cluster"})
```

Настройка ETL.

Репозиторий: <https://git.polet-it.ru/rs.analytics/analytics/-/tree/development/RSL/airflow/rsl>

Airflow server (здесь 10.14.8.33).
В папку DAGS_FOLDER (/opt/airflow/dags) (задается в конфигурации)
скопировать подготовленные Dags для проекта RSL.

Создать папку (user=demo) /opt/airflow/dags/demo/rsl

Скопировать все DAGs и конфигурационный файл из проекта
(Analytics/RSL/airflow/rsl) в выше созданную папку на сервере.

Внести изменения в конфигурационный файл __init__.py:

```
DAG_USER='demo'
```

HDFS: Скопировать необходимые файлы на HDFS.

Создать папки /user/aloha/spark/lib, /user/aloha/spark/lib/json, /user/aloha/spark/share

Файл properties.json содержит настройки подключения.
При необходимости откорректировать и скопировать в папку
/user/aloha/spark/lib/json.

Актуальный jar проекта rsl_2.12-1.0.jar скопировать в /user/aloha/spark/lib

spark3.4.0_jars.zip скопировать в hdfs:///spark3.4.0/spark3.4.0_jars.zip

arima_env.tar.gz скопировать в hdfs:///user/aloha/spark/share/arima_env.tar.gz

Установка компонента Retail Suite.Global Loyalty&CRM MobileApp

Установка компонента происходит на устройстве конечного пользователя из магазинов Android и iOS.

Установка Retail Suite.Global Loyalty&CRM MobileApp Server

Для установки приложения mproxy необходимы следующие сервисы в ОС

- Java 8
- Apache Tomcat 9 (>=9.0.80)
- MySQL/MariaDB
- Nginx/Apache

В MySQL/MariaDB создать базу с именем *mproxy_database_2_0* и пользователя для нее.

В Apache Tomcat/Apache (пример настроек указан для Tomcat) опубликовать приложение (WAR-файл), после чего изменить настройки подключения к БД в файле `<apache-tomcat>/webapps/<mproxy-name>/WEB-INF/classes/application.properties`:

```
[...]  
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mproxy_database_2_0?useSSL=false&useUnicode=yes&characterEncoding=UTF-8&serverTimezone=Europe/Moscow  
# Username and password  
spring.datasource.username=mproxy  
spring.datasource.password=password  
[...]
```

В файле конфигурации Apache Tomcat (`<apache-tomcat>/conf/server.xml`) изменить перенаправление корня на приложение, добавив в конец секции `<Host></Host>`:

```
[...]  
<Context path="" docBase="<mproxy-name>"></Context>  
[...]
```

Пример файла виртуального хоста для nginx:

```
[...]  
server {  
    listen 80 default_server;  
  
    root /var/www/html;  
    index index.html index.htm;  
  
    server_name mproxy;  
  
    location / {  
        proxy_set_header X-Forwarded-Host $host;  
        proxy_set_header X-Forwarded-Server $host;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_pass http://127.0.0.1:8080;  
  
        allow all;  
    }  
  
    location ~* ^/(mobile|messaging|resources|api)([/V]) {
```

```
proxy_pass http://127.0.0.1:8080;

allow all;
}

## tomcat manager (block)
location /manager { deny all; }
location /host-manager { deny all; }
}
[...]
```