UDV Orchestrator

Руководство администратора

Версия 4.1

udv.group

support@udv.group

Содержание

1. Архитектура решения ... 3

2. Требования для установки ... 7

3. Установка и настройка сервисов ... 9

3.1. Установка UDV Orchestrator ... 9

3.1.1. Быстрая установка ... 9

3.1.2. Установка при наличии доступа к сети Интернет ... 10

3.1.3. Установка без доступа к сети Интернет ... 11

3.1.4. Действия после установки ... 12

3.2. Установка агентов Lambda Executor ... 13

3.2.1. Установка на машинах под управлением ОС Windows ... 13

3.2.2. Установка на машинах под управлением ОС Linux ... 15

3.3. Удаление агентов Lambda Executor ... 16

3.3.1. Удаление с машин под управлением ОС Windows ... 16

3.3.2. Удаление с машин под управлением ОС Linux ... 17

4. Загрузка скриптов, плейбуков и сопутствующих метаданных ... 18

4.1. Требования для запуска утилиты ... 18

4.1.1. Требования к программному обеспечению ... 18

4.1.2. Требования к размещению файлов ... 18

4.2. Подготовка к запуску утилиты ... 20

4.3. Запуск утилиты ... 21

4.3.1. Запуск с помощью Docker ... 21

4.3.2. Запуск без использования контейнеров ... 21

4.3.3. Автоматизация установки с помощью make ... 21

5. Управление сервисами ... 23

5.1. Запуск и приостановка сервисов ... 23

5.2. Обновление сервисов ... 23

5.3. Резервное копирование и восстановление ... 23

5.4. Логирование работы сервисов ... 24

5.4.1. Просмотр лог-файлов в MinIO ... 25

5.4.2. Просмотр лог-файлов в Grafana ... 25

6. Параметры ENV-файла ... 27

7. Адреса сервисов по умолчанию ... 34

1. Архитектура решения

UDV Orchestrator входит в систему UDV SOAR — программное решение, включающее следующие подсистемы и компоненты:

- подсистема оркестрации, предназначенная для автоматического выполнения части действий на всех этапах обработки инцидентов ИБ (локализация, расследование, реагирование) — UDV Orchestrator;
- подсистема реагирования на инциденты ИБ, предназначенная для сбора инцидентов ИБ из различных источников, их классификации, формирования плана мероприятий по реагированию и отслеживанию этапов и метрик реализации данного плана;
- служебные компоненты авторизации, хранения данных и интеграции между подсистемами.

Прим.:

SOAR (Security Orchestration, Automation and Response) — класс программных продуктов, предназначенных для координации и управления различными средствами защиты информации, сбора событий информационной безопасности, автоматизации сценариев реагирования на них.

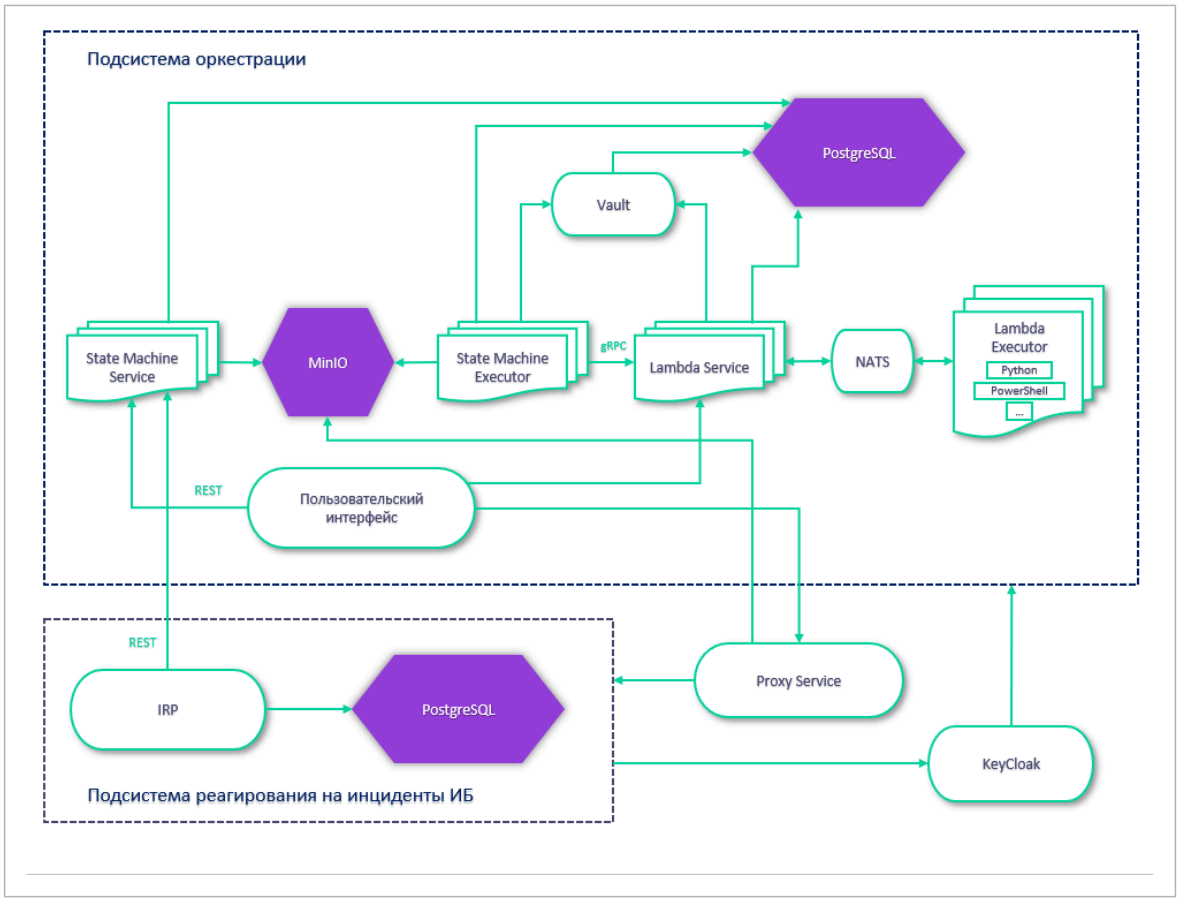


Рис. 1. Схема ПО UDV SOAR

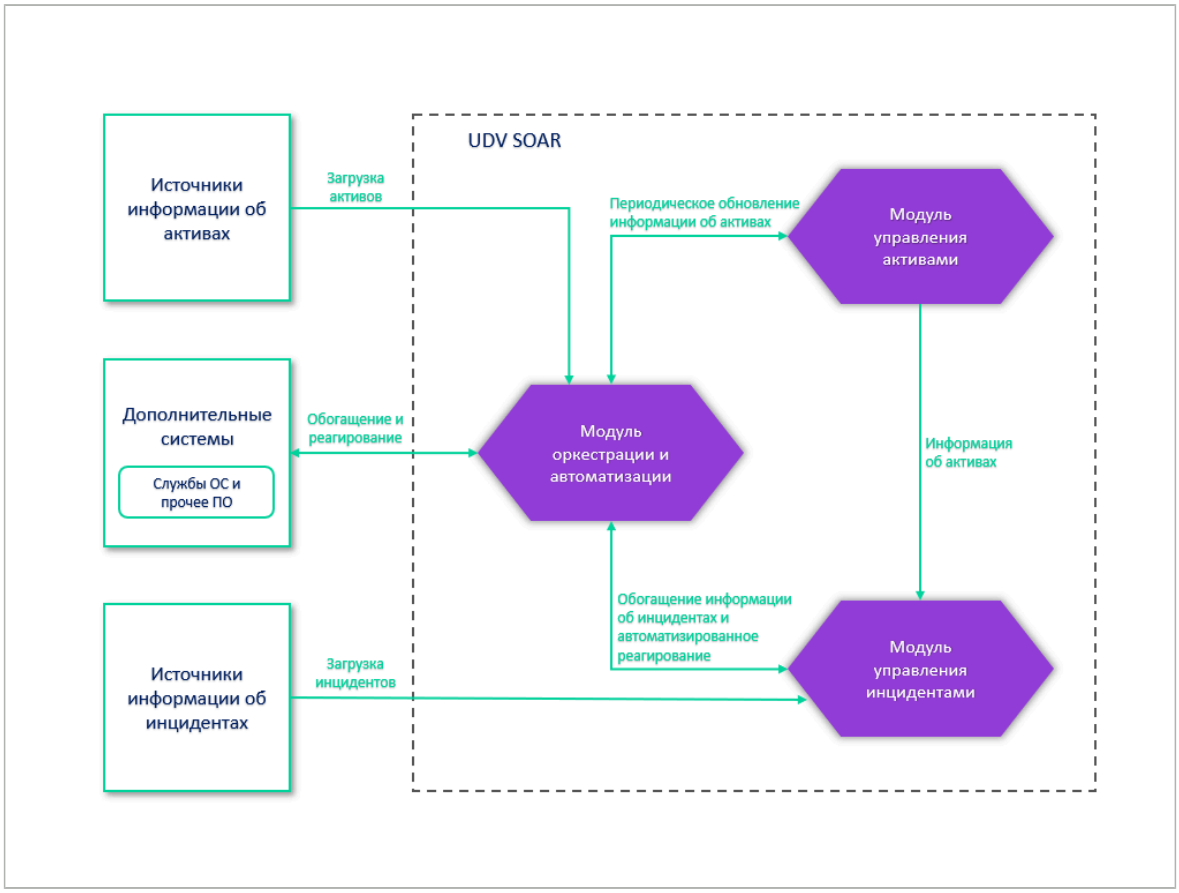


Рис. 2. Схема взаимодействия UDV SOAR с внешними системами

Решение включает компоненты двух типов: приложения собственной разработки и стороннее программное обеспечение с открытым исходным кодом (зависимости).

Табл. 1. Приложения собственной разработки

ПРИЛОЖЕНИЕ	ОПИСАНИЕ
`Lambda Executor`	Агент для удаленного запуска скриптов и плейбуков. По умолчанию устанавливается один внутренний. При необходимости можно установить дополнительные агенты на нужных клиентских машинах (см. Установка агентов Lambda Executor)
`Lambda Service`	Компонент, обеспечивающий запуск скриптов и плейбуков и контролирующий этот процесс через предоставляемый `REST API`
`Loader Service`	Компонент, позволяющий импортировать и экспортировать архивы объектов (системных и обычных) с помощью `REST API`
`Proxy Service`	Компонент, обеспечивающий интеграцию данных между сервисами подсистем
`State Machine Executor`	Компонент, управляющий расписанием и порядком запуска плейбуков
`State Machine Service`	Компонент, управляющий созданием и хранением заданий на запуск плейбуков и предоставляющий `REST API`, а также связь через протокол `WebSocket` для управления плейбуками, их запуском и связанной с ними информацией

Табл. 2. Сторонние зависимости

ПРИЛОЖЕНИЕ	ОПИСАНИЕ	ТИП ЛИЦЕНЗИИ
`Grafana`	Система визуализации метрик работы сервисов	`AGPL-3.0 license`
`Keycloak`	Система авторизации	`Apache-2.0 license`
`MinIO`	Объектное хранилище плейбуков, скриптов, файлов входных данных и контекста выполнения плейбуков, а также временных файлов	`AGPL-3.0 license`

Табл. 2. Сторонние зависимости

ПРИЛОЖЕНИЕ	ОПИСАНИЕ	ТИП ЛИЦЕНЗИИ
`NATS`	Система обмена сообщениями между компонентами `Lambda Service` и `Lambda Executor`	`Apache-2.0 license`
`Pgadmin`	Инструмент для управления базой данных `PostgreSQL`	`PostgreSQL license`
`PostgreSQL`	Базы данных для сервисов `KeyCloak`, `Vault`, подсистем UDV SOAR	`PostgreSQL license`
`Prometheus`	Система сбора и хранения метрик работы сервисов	`Apache-2.0 license`
`Vault`	Система безопасного хранения секретов	`Business Source License`

2. Требования для установки

Для успешной установки решения должны выполняться следующие требования:

Табл. 3. Требования к аппаратному обеспечению

ПАРАМЕТР	МИНИМАЛЬНОЕ ЗНАЧЕНИЕ	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ
Количество ядер процессора	8	16
Объем оперативной памяти	16 ГБ	32 ГБ
Объем свободной памяти накопителя	50 ГБ	100 ГБ

Табл. 4. Требования к программному обеспечению

КОМПОНЕНТ	ЗНАЧЕНИЕ
Рекомендованные ОС	**Серверная часть:** `Debian` 11 и более поздних версий, `Astra Linux` 1.7.1 и более поздних версий, `Ubuntu` 20 и более поздних версий, `РЕД ОС` 7.2 и более поздних версий
	Клиентская часть: `Debian 11` и более поздних версий, `Astra Linux` 1.7.1 и более поздних версий, `Ubuntu` 20 и более поздних версий, `РЕД ОС` 7.2 и более поздних версий, `Windows` 7 и более поздних версий
`Docker`	19 и более поздних версий
`Docker Compose`	2.14 и более поздних версий
`Git`	2.25 и более поздних версий

Если `Docker` и `Docker Compose` ранее уже были установлены на компьютере с дистрибутивом на основе `Debian`, рекомендуется удалить это ПО и установить заново. Для этого используйте следующие команды:

```
sudo apt update && sudo apt install wget -y
wget https://download.docker.com/linux/debian/dists/stretch/pool/stable/amd64/containerd.io_1.4.3-1_amd64.deb
wget https://download.docker.com/linux/debian/dists/stretch/pool/stable/amd64/docker-ce-cli_19.03.9~3-0~debian-stretch_amd64.deb
wget https://download.docker.com/linux/debian/dists/stretch/pool/stable/amd64/docker-ce_19.03.9~3-0~debian-stretch_amd64.deb
sudo dpkg -i containerd.io_1.4.3-1_amd64.deb docker-ce-cli_19.03.9~3-0~debian-stretch_amd64.deb docker-ce_19.03.9~3-0~debian-stretch_amd64.deb
wget https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86_64
```

```
sudo mv ./docker-compose-linux-x86_64 /usr/bin/docker-compose
sudo chmod +x /usr/bin/docker-compose
```

По умолчанию вместе с дистрибутивом предоставляются реквизиты для авторизации в реестре контейнеров разработчика. Для авторизации в реестре контейнеров используйте команду `docker login cr.udv.group`.

3. Установка и настройка сервисов

В этом разделе:

1. **Установка UDV Orchestrator**
2. **Установка агентов Lambda Executor**
3. **Удаление агентов Lambda Executor**

3.1. Установка UDV Orchestrator

3.1.1. Быстрая установка

При использовании данного варианта развертывания сервисы конфигурируются на основе IP-адреса системы — все обращения к интерфейсам происходят через IP-адрес и порт, а также автоматически создается самоподписанный сертификат.

Чтобы быстро развернуть готовое к работе решение в выбранной конфигурации, воспользуйтесь установочным скриптом. Он предоставляется разработчиком в отдельном файле.

Другие варианты развертывания описаны в разделах «Установка без доступа к сети Интернет» и «Установка при наличии доступа к сети Интернет».

3.1.2. Установка при наличии доступа к сети Интернет

> Прим.:
>
> Данный вариант развертывания подразумевает использование wildcard-сертификата и доменных имен для обращения к сервисам.

Чтобы установить ПО при наличии доступа к сети Интернет:

1. Распакуйте полученный архив в любую удобную директорию и перейдите в нее.
2. Для использования своего самоподписанного wildcard-сертификата, который автоматически генерируется при первом запуске:

    а. Откройте `ENV`-файл.

    б. В основных параметрах замените значение параметра `COMPOSE_PROXY_DOMAIN` со стандартного на доменное имя вашего wildcard-сертификата.

    в. Скопируйте сертификат `nginx.crt` и ключ к нему (`nginx.key`) в папку `https`.

    г. Для корректной работы дополнительно выполните следующие действия:
    - добавьте выбранные DNS-имена на DNS-сервер или на локальные машины, на которых будут использоваться сервисы;
    - добавьте сгенерированный самоподписанный CA-сертификат из папки `https` на все системы, с которых планируется взаимодействие с сервисами.

3. Запустите сервисы:

```
docker-compose up -d
```

3.1.3. Установка без доступа к сети Интернет

Чтобы установить ПО без доступа к сети Интернет:

1. Запустите дистрибутив для установки через Интернет на любой системе с доступом к Интернету и установленным `Docker`:

   ```
   docker-compose up -d
   ```

2. После успешного запуска сохраните все скачанные образы в единый архив:

   ```
   docker_images=$(docker inspect $(docker-compose ps -aq) | jq -r '.[] | .Config.Image')
   docker_tag=$(docker inspect $(docker-compose ps soar-lambda-service -q) | jq -r '.[]
    | .Config.Image' | awk -F: '{ print $2 }')
   docker save ${docker_images[@]} -o docker-images.tar &&
   tar czvf docker-images-${docker_tag}.tar.gz docker-images.tar && rm docker-images.tar
   ```

3. Скопируйте полученный архив `docker-images.tar.gz` на сервер без доступа к Интернету.
4. На сервере без доступа к Интернету загрузите образы в систему из архива:

   ```
   tar vfzx docker-images.tar.gz &&
   docker load -i docker-images.tar
   ```

5. Далее установка производится аналогично процессу установки при наличии доступа в сеть Интернет.

3.1.4. Действия после установки

Перед запуском убедитесь, что у текущего пользователя имеются полные права (на чтение, запись и запуск содержимого папки дистрибутива). Если необходимых прав нет, смените пользователя, от имени которого которого запускаются сервисы, или расширьте права текущего пользователя.

> Важное замечание:
>
> Убедитесь, что расширение прав не противоречит принятым политикам безопасности.

Список названий запускаемых образов, который можно изменить, доступен в параметрах `ENV`-файла.

Проверить корректность работы запущенных сервисов можно двумя способами:

- в веб-интерфейсе `Grafana`;
- путем входа на адрес каждого сервиса.

> Важное замечание:
>
> Данные для входа в сервисы, устанавливаемые по умолчанию, предоставляются по запросу. После входа и первоначальной настройки смените пароли.

Чтобы проверить версию установленной подсистемы, откройте пользовательский интерфейс по адресу или `https://soar.test.local` по умолчанию и раскройте боковое меню. Версия будет указана в его нижней части.

3.2. Установка агентов Lambda Executor

Чтобы автоматизировать развертывание агентов `Lambda Executor` на устройствах, можно использовать установочные скрипты. Далее приведены инструкции по их использованию.

> Прим.:
>
> В версии подсистемы 3.8 для пользователей с ролью `Администратор` была добавлена возможность устанавливать и удалять агенты в веб-интерфейсе. Подробнее об этом см. в документе **UDV Orchestrator. Руководство пользователя. Раздел «Работа с агентами»**.

3.2.1. Установка на машинах под управлением ОС Windows

Состав инсталлятора:

1. `full_installer.bat` — главный установщик, с помощью которого можно развернуть агент и все необходимые зависимости;
2. `agent_installer.py` — установщик агента;
3. `nssm-2.24` — утилита для установки службы Windows.

   > Внимание:
   >
   > Антивирусные программы могут определять данную утилиту как потенциальную угрозу. Игнорируйте эти предупреждения.

4. `certs` — папка с сертификатами безопасности (требуемый формат имен — `*.crt`).

   > Прим.:
   >
   > Для корректной установки сертификатов необходимо наличие установленной утилиты `certutil`.

Для работы главного установщика `full_installer.bat` поместите в одну директорию с ним:

- файлы `lambdaExecutor.exe` и `lambdaExecutorConfig.json`;
- папку `venv` с виртуальным окружением `Python`;
- папку `certs` с `CRT`-сертификатами (необязательный элемент).

Чтобы запустить установку, введите команду:

```
.\full_installer.bat le_agent nats.local:4222
```

Здесь:

- `le_agent` — это задаваемое имя агента `Lambda Executor` (необязательный параметр — если его не указать, имя будет сгенерировано автоматически);
- `nats.local:4222` — это адрес `NATS` (необязательный параметр).

Установщик `full_installer.bat` выполняет следующие операции:

1. Создает директорию `%SYSTEMDRIVE%/soar/python` и копирует в нее содержимое папки `venv`.
2. Устанавливает `PATH` на `%SYSTEMDRIVE%/soar/python/Scripts`.
3. Запускает скрипт-установщик агента `agent_installer.py`.
4. Копирует сертификаты, если они были размещены в папке `certs`.

Требования для работы установщика агента `agent_installer.py`:

1. Состав файлов — в директорию со скриптом поместите:
   - папку `nssm-2.24`;
   - готовый бинарный файл — `lambdaExecutor.exe`;
   - конфигурационный файл — `lambdaExecutorConfig.json`.
2. Пути к нужным для установки файлам не должны содержать пробелы.
3. Для установки и запуска службы запустите терминал **с правами администратора**.

Чтобы запустить установку в директорию по умолчанию с именем на основе сетевого, введите команду:

```
python.exe .\agent_installer.py
```

Чтобы запустить установку в произвольную директорию, введите команду:

```
python.exe .\agent_installer.py --install_dir=D:\Agent
```

Чтобы запустить установку в произвольную директорию с произвольным именем, введите команду:

```
python.exe .\agent_installer.py --install_dir=D:\Agent --executor_name=Agent6457
```

Установщик `agent_installer.py` выполняет следующие операции:

1. Копирует исполняемый файл `lambdaExecutor.exe` и файл конфигурации `lambdaExecutorConfig.json` в установочную директорию.

   > Прим.:
   >
   > Установочная директория задается параметром `--install_dir`. Если не задать ее, используется директория по умолчанию — `%SYSTEMDRIVE%/LambdaExecutorAgent`. Если установочной директории не существует, она будет создана с учетом правил создания в Windows.

2. Редактирует конфигурацию в установочной директории, чтобы задать уникальное имя агента, передаваемое в параметре `--executor_name`.

   > Прим.:
   >
   > Если имя не указать, оно будет сгенерировано автоматически на основе сетевого имени машины. Если в имени машины есть символы, отличные от букв и цифр, они будут заменены на `0`. Это требуется для корректной работы `NATS`.

3. Устанавливает `runtimes.json`, исходя из переданного параметра `--python`.
4. Устанавливает `lambdaExecutor.exe` в качестве службы и запускает ее.
5. Выводит имя агента в консоль и файл `./executor_name.txt` в директории с установочным скриптом.

3.2.2. Установка на машинах под управлением ОС Linux

Состав инсталлятора:

1. `full_installer.sh` — главный установщик, с помощью которого можно развернуть агент и все необходимые зависимости;
2. `agent_installer.py` — установщик агента.

Для работы главного установщика `full_installer.sh` поместите в одну директорию с ним:

- готовый бинарный файл `lambdaExecutor`;
- конфигурационный файл `lambdaExecutorConfig.json`;
- скрипт `agent_installer.py`;
- папку `venv` с виртуальным окружением `Python`.

Чтобы запустить установку, введите команду:

```
sudo ./full_installer.sh 3.8 le_agent nats.local:4222
```

Здесь:

- `3.8` — это указанная в `venv` версия `Python`;
- `le_agent` — это задаваемое имя агента `Lambda Executor` (необязательный параметр — если его не указать, имя будет сгенерировано автоматически);
- `nats.local:4222` — это адрес `NATS` (необязательный параметр).

Установщик выполняет следующие операции:

1. Копирует `Python` и требуемые библиотеки в директорию агента.
2. Устанавливает `PATH` и `PYTHONPATH` для использования интерпретатора и библиотек.
3. Запускает `agent_installer.py`:
    - копирует бинарный и конфигурационный файлы;
    - создает и запускает `systemd`-демон `soar_agent`.

3.3. Удаление агентов Lambda Executor

Чтобы автоматизировать удаление агентов `Lambda Executor` с устройств, можно использовать специально разработанные скрипты. Далее приведены инструкции по их использованию.

> Прим.:
>
> В версии подсистемы 3.8 для пользователей с ролью `Администратор` была добавлена возможность устанавливать и удалять агенты в веб-интерфейсе. Подробнее об этом см. в документе **UDV Orchestrator. Руководство пользователя. Раздел «Работа с агентами»**.

3.3.1. Удаление с машин под управлением ОС Windows

Состав деинсталлятора:

1. `uninstaller.bat` — главный деинсталлятор, с помощью которого можно удалить службу агента и все связанные с ним файлы;
2. `agent_uninstaller.py` — деинсталлятор агента;
3. `nssm-2.24` — утилита для удаления Windows-службы.

> Прим.:
>
> Для корректной работы рекомендуется не менять состав файлов и запускать деинсталлятор из директории, в которой расположен главный `BAT`-файл.

Чтобы запустить деинсталлятор, введите команду:

```
.\uninstaller.bat
```

Деинсталлятор выполняет следующие операции:

1. Останавливает и удаляет службу агента `Lambda Executor`.
2. Удаляет файлы приложения `soar_agent`.
3. Удаляет `Python` с машины, на которой был установлен агент.

3.3.2. Удаление с машин под управлением ОС Linux

Состав деинсталлятора:

1. `uninstaller.sh` — главный деинсталлятор, с помощью которого можно удалить службу агента и все связанные с ним файлы;
2. `agent_uninstaller.py` — деинсталлятор агента.

Чтобы запустить деинсталлятор, введите команду:

```
sudo ./uninstaller.sh
```

Деинсталлятор выполняет следующие операции:

1. Останавливает `systemd`-сервис `soar_agent` и удаляет `soar_agent.service` из директории `/etc/systemd/system`.
2. Удаляет файлы агента.
3. Удаляет `Python` и переменные среды для поиска интерпретатора.

4. Загрузка скриптов, плейбуков и сопутствующих метаданных

В этом разделе:

1. **Требования для запуска утилиты**
2. **Подготовка к запуску утилиты**
3. **Запуск утилиты**

Чтобы автоматизировать загрузку скриптов, плейбуков и сопутствующих метаданных на устройствах, можно использовать дополнительную утилиту. Далее приведены инструкции по ее использованию.

> Прим.:
>
> В версии подсистемы 4.1 была добавлена возможность использования массовых операций по импорту и экспорту объектов в веб-интерфейсе. Подробнее об этом см. в документе «UDV Orchestrator. Руководство пользователя».

4.1. Требования для запуска утилиты

4.1.1. **Требования к программному обеспечению**

Для запуска утилиты потребуются следующие компоненты:

1. `Python` 3.8 и более поздних версий;
2. `Docker Compose`;
3. утилита `make`.

4.1.2. Требования к размещению файлов

Все загружаемые файлы должны быть размещены в корневой директории проекта в папке `data`. Плейбуки в формате `JSON` должны располагаться в директории `/data/playbooks`, скрипты — в директории `/data/scripts`.

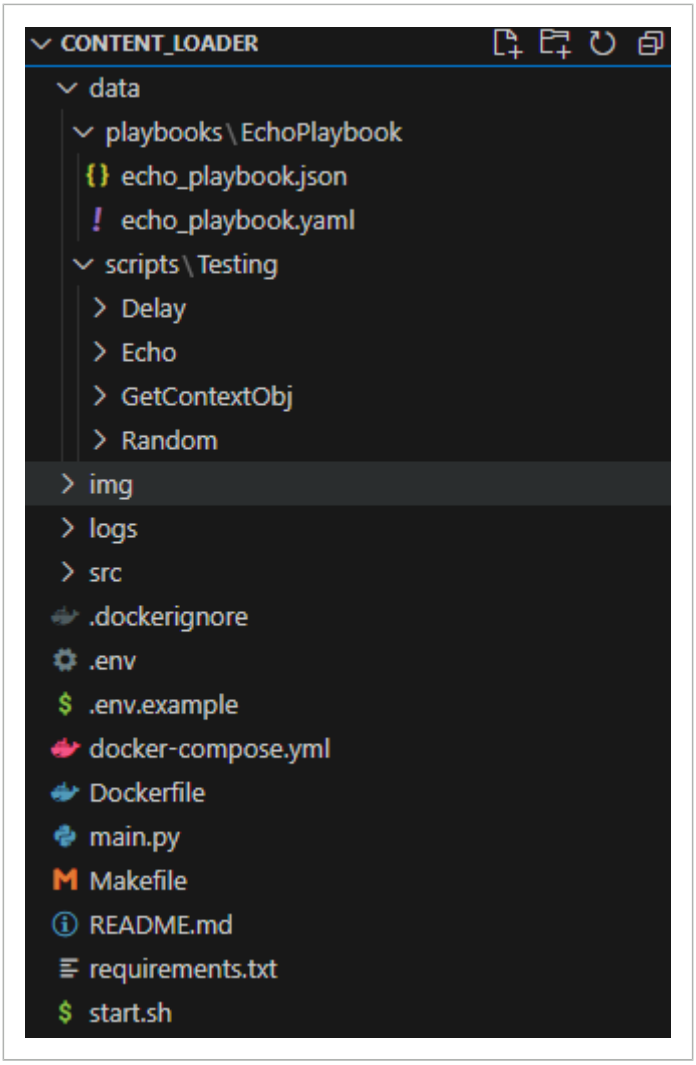


Рис. 3. Корневая директория проекта с папкой data

> Прим.:
>
> Для успешной загрузки скриптов каждый из них должен находится в отдельной папке и содержать файл с телом скрипта и `YAML`-файл с метаданными (в частности, секретами, которые также будут загружены в подсистему).

Количество поддиректорий не имеет значения — утилита найдет скрипты и плейбуки вне зависимости от того, насколько «глубоко» они расположены. Например, скрипт может располагаться следующим образом:

```
└── data
    ├── playbooks
    └── scripts
        └── subdir1
            └── subdir2
                └── subdir3
                    └── EchoScript
                        ├── echo.py
                        └── echo.yaml
```

В таком случае утилита найдет и загрузит `EchoScript` в подсистему.

То же самое касается плейбуков. Ниже приведен пример возможного расположения скриптов и плейбуков перед загрузкой:

```
└── data
    ├── playbooks
    │   ├── playbook_1.json
    │   └── playbook_2.json
    └── scripts
        ├── Script1
        │   ├── script_1.py
        │   └── script_1.yaml
        ├── Script2
        │   ├── script_2.py
        │   └── script_2.yaml
        └── Script3
            ├── script_3.py
            └── script_3.yaml
```

4.2. Подготовка к запуску утилиты

Чтобы подготовиться к запуску, в корневой директории (рядом с файлом `docker-compose.yaml`) создайте `ENV`-файл со следующими переменными:

Табл. 5. Переменные `ENV`-файла для запуска утилиты

ПАРАМЕТР	ОПИСАНИЕ
`KEYCLOAK_CLIENT_ID=client_id`	`client_id` для `Keycloak`
`KEYCLOAK_CLIENT_SECRET=`*`secret`*	`client_secret` для `Keycloak`
`KEYCLOAK_SCOPE=profile`	`scope` для `Keycloak`
`KEYCLOAK_GRANT_TYPE=password`	`grant_type` для `Keycloak`
`KEYCLOAK_USERNAME=`*`username`*	имя пользователя `Keycloak`
`KEYCLOAK_PASSWORD=`*`password`*	пароль пользователя `Keycloak`
`KEYCLOAK_URL=https://HOST_IP:8080`	адрес сервера `Keycloak`
`LAMBDA_SERVICE_URL=https://HOST_IP:8089`	адрес `Lambda Service`
`STATE_MACHINE_SERVICE_URL=https://HOST_IP:8088`	адрес `State Machine Service`
`SOAR_API_VERSION =v4.0`	текущая версия `API`

4.3. Запуск утилиты

Запустить утилиту можно тремя способами:

- с помощью `Docker`;
- без использования контейнеров;
- автоматически с помощью утилиты `make`.

В случае успешного завершения результаты работы выводятся в консоль в виде следующих сообщений:

```
Всего загружено плейбуков: 5
Всего загружено скриптов: 14
Всего загружено секретов: 6
Имена и id доступны в лог файле
```

Критические ошибки также появляются в консоли. Логи располагаются в папке `logs`. При запуске с использованием `Docker` соответствующий том также называется `logs`.

4.3.1. Запуск с помощью Docker

Чтобы запустить утилиту с помощью `Docker`, в корневой папке введите следующую команду:

```
docker-compose up --build
```

4.3.2. Запуск без использования контейнеров

Чтобы запустить утилиту без использования контейнеров, в корневой папке введите следующие команды:

1. Установите виртуальное окружение:

   ```
   python3.8 -m venv venv
   ```

2. Активируйте виртуальное окружение:

   ```
   source ./venv/bin/activate
   ```

3. Установите зависимости:

   ```
   pip install -r requirements.txt
   ```

4. Запустите скрипт:

   ```
   python3.8 main.py
   ```

4.3.3. Автоматизация установки с помощью make

Утилита `make` позволяет автоматизировать установку. Для этого используйте следующие команды:

1. Создайте контейнер со скриптом и контентом:

   ```
   build-container
   ```

2. Запустите контейнер:

   ```
   start-container
   ```

3. Подготовьте `ENV`-файл:

   ```
   env-prepare
   ```

4. Запустите скрипт без использования контейнера:

```
run
```

5. Управление сервисами

В этом разделе:

1. **Запуск и приостановка сервисов**
2. **Обновление сервисов**
3. **Резервное копирование и восстановление**
4. **Логирование работы сервисов**

5.1. Запуск и приостановка сервисов

Запустить и приостановить работу сервисов можно в директории, в которой был разархивирован предоставленный дистрибутив.

1. Чтобы запустить работу, выполните команду:

   ```
   docker-compose up -d
   ```

2. Чтобы приостановить работу, выполните команду:

   ```
   docker-compose down
   ```

5.2. Обновление сервисов

Чтобы обновить сервисы, укажите их новые версии в `ENV`-файле в следующих параметрах:

- `SOAR_BACKEND_LD_IMAGE`
- `SOAR_BACKEND_LS_IMAGE`
- `SOAR_BACKEND_SM_IMAGE`
- `SOAR_BACKEND_SME_IMAGE`
- `SOAR_BACKEND_PS_IMAGE`
- `SOAR_BACKEND_LE_IMAGE`
- `SOAR_FRONTEND_ASL_IMAGE`
- `ORCHESTRA_IMAGE`

Описания этих параметров и примеры их заполнения см. в таблице «Пути к образам».

Изменив значения, запустите обновление сервисов. Для этого выполните следующую команду:

```
docker-compose up -d
```

> Важное замечание:
>
> Если обновление производится без доступа к сети Интернет, необходимо предварительно перенести обновленный дистрибутив на машину, где развернуто решение.

5.3. Резервное копирование и восстановление

Чтобы создать резервную копию подсистемы, выполните следующую команду:

```
docker-compose exec backup-client bash backup-client-entrypoint.sh backup all
```

Архив с резервной копией появится в папке `./docker-compose-backups/`.

Чтобы восстановить систему из резервной копии, выполните следующие команды:

```
docker-compose down
docker-compose up -d postgres minio backup-client
docker-compose exec backup-client bash backup-client-entrypoint.sh restore <.tar.gz name> all
```

> Важное замечание:
>
> `<.tar.gz name>` здесь — это название архива без расширения (например, `2022-12-19_12-00-46`) из директории `./docker-compose-backups/`.

Вы также можете создавать отдельные резервные копии для нескольких категорий данных и восстанавливать данные из них.

1. Чтобы создать резервную копию всех данных сервиса `MinIO`:

   ```
   docker-compose exec backup-client bash backup-client-entrypoint.sh backup minio
   ```

2. Чтобы создать резервную копию всех данных сервиса `PostgreSQL`:

   ```
   docker-compose exec backup-client bash backup-client-entrypoint.sh backup postgres
   ```

3. Чтобы создать резервную копию секретов из тома `Docker`:

   ```
   docker-compose exec backup-client bash backup-client-entrypoint.sh backup secrets
   ```

5.4. Логирование работы сервисов

В ходе работы системы осуществляется логирование событий разного уровня значимости для всех сервисов, входящих в состав решения, и для сервера авторизации `Keycloak`.

Есть следующие уровни логирования:

- `error` — уровень ошибок: в логи попадают только записи об ошибках в приложении;
- `warn` — уровень предупреждений: в логи попадают предупреждения и ошибки;
- `info` — обычный уровень: в логи попадают информационные сообщения, предупреждения и ошибки;

  > Прим.:
  >
  > Уровень `info` устанавливается по умолчанию.

- `debug` — уровень отладки: в логи попадает отладочная информация;
- `trace` — уровень детального логирования: в логи попадает большее количество отладочной информации по сравнению с уровнем `debug`.

> Прим.:
>
> Пункты списка расположены в порядке увеличения количества логируемой информации.

Для эксплуатации системы рекомендуется выбирать уровни `error`, `warn` или `info`.

В таблице ниже указаны лог-файлы сервисов системы и их расположение.

Табл. 6. Расположение лог-файлов

НАИМЕНОВАНИЕ СЕРВИСА	ФАЙЛ
`State Machine Service`	`sm/stateMachineService.log` в бакете `statemachine-service`
`State Machine Executor`	`sm/stateMachineExecutor.log` в бакете `statemachine-service`
`Lambda Service`	`lambdaService.log` в бакете `lambda-service`
`Proxy Service`	`proxyService.log` в бакете `proxy-service`
`Loader Service`	`logs\loaderServce.log` в контейнере `loader-service`

5.4.1. Просмотр лог-файлов в MinIO

Чтобы посмотреть лог-файлы приложений SOAR в `MinIO`:

1. Откройте веб-интерфейс `MinIO` по адресу https://minio.test.local/.
2. Введите логин и пароль и нажмите **Login**.
   - **Результат шага:** откроется страница **Buckets**.
3. Рядом с нужным бакетом нажмите Browse → и откройте файл в соответствии с таблицей Таблица 6. Расположение лог-файлов.

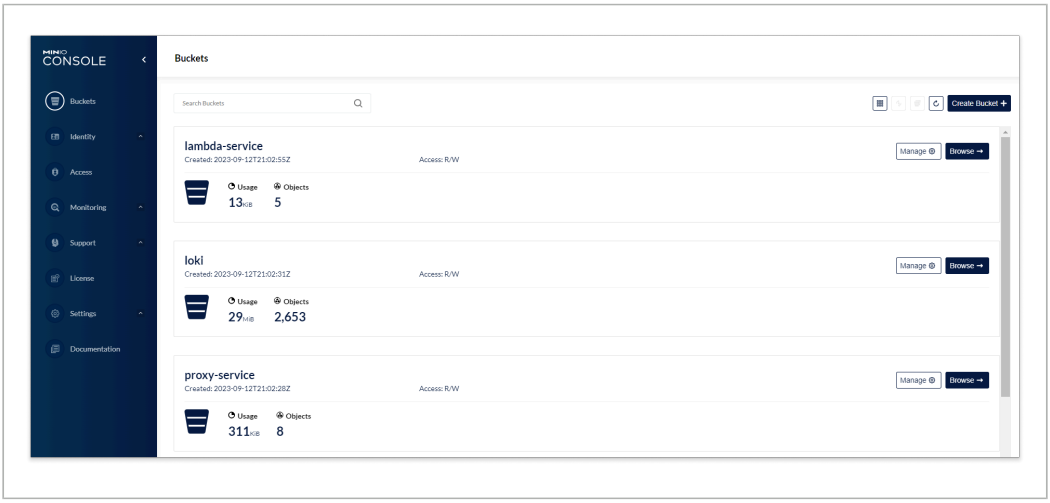


Рис. 4. Веб-интерфейс MinIO. Страница Buckets

5.4.2. Просмотр лог-файлов в Grafana

Чтобы посмотреть лог-файлы других сервисов:

1. Откройте веб-интерфейс `Grafana` по адресу https://grafana.test.local/.
2. Введите логин и пароль и нажмите и нажмите **Log in**.
3. В боковом меню наведите указатель на значок и нажмите **Dashboards**.
4. Откройте раскрывающийся список и выберите Loki.

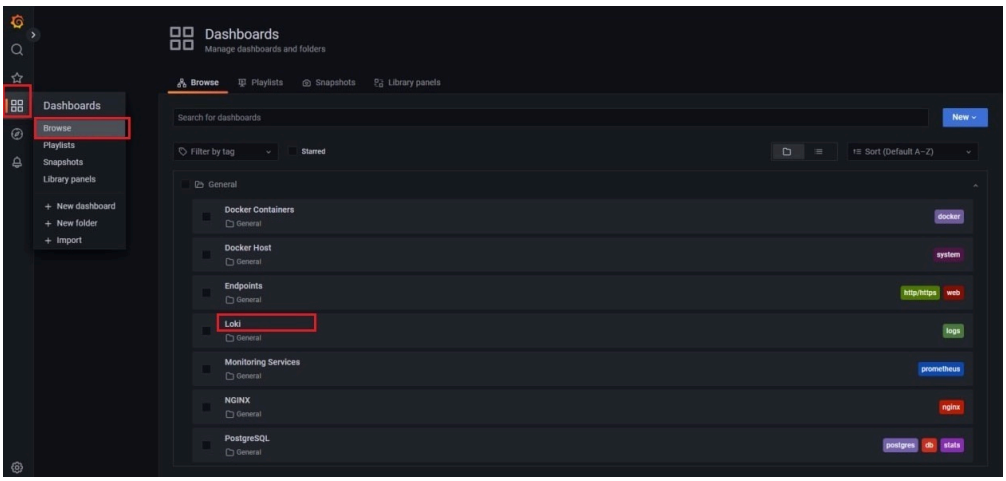


Рис. 5. Веб-интерфейс Grafana. Страница Dashboards

5. В верхней части страницы в поле `container` укажите название нужного сервиса или выберите его из раскрывающегося списка.

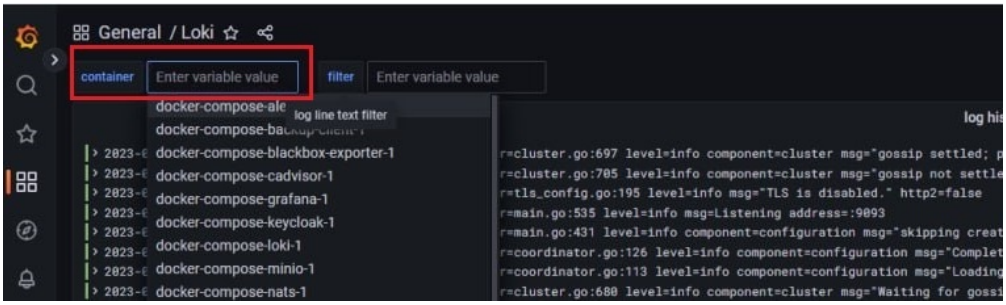


Рис. 6. Веб-интерфейс Grafana. Выбор сервиса на странице Loki

6. Параметры ENV-файла

Основные параметры

Табл. 7. Пути к образам

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_LD_-IMAGE`	`SOAR_BACKEND_LD_IMAGE=cr.udv.group/soar/soar-loader-service:latest`	Путь к образу `Loader Service`
`SOAR_BACKEND_LS_-IMAGE`	`SOAR_BACKEND_LS_IMAGE=cr.udv.group/soar/soar-lambda-service:v4.0`	Путь к образу `Lambda Service`
`SOAR_BACKEND_SM_-IMAGE`	`SOAR_BACKEND_SM_IMAGE=cr.udv.group/soar/soar-state-machine-service:v4.0`	Путь к образу `State Machine Service`
`SOAR_BACKEND_SME_-IMAGE`	`SOAR_BACKEND_SME_IMAGE=cr.udv.group/soar/soar-state-machine-executor:v4.0`	Путь к образу `State Machine Executor`
`SOAR_BACKEND_PS_-IMAGE`	`SOAR_BACKEND_PS_IMAGE=cr.udv.group/soar/soar-proxy-service:v4.0`	Путь к образу `Proxy Service`
`SOAR_BACKEND_LE_-IMAGE`	`SOAR_BACKEND_LE_IMAGE=cr.udv.group/soar/soar-lambda-executor:v4.0`	Путь к образу `Lambda Executor`
`SOAR_FRONTEND_-ASL_IMAGE`	`SOAR_FRONTEND_ASL_IMAGE=cr.udv.group/soar/soar-asleditor:v4.0`	Путь к образу фронтенд-сервиса `SOAR`
`ORCHESTRA_IMAGE`	`ORCHESTRA_IMAGE=cr.udv.group/eplat4m/orchestra:v1.2.0.1`	Путь к образу `Orchestra`

Табл. 8. Домен

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`COMPOSE_PROXY_DOMAIN`	`COMPOSE_PROXY_DOMAIN=.test.local`	Важное замечание: Точка перед началом значения — обязательный элемент

Табл. 9. IP-адрес хоста

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`COMPOSE_PROXY_HOST_IP`		Если не используются доменные имена и wildcard-

Табл. 9. IP-адрес хоста

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
		сертификат, по умолчанию оставить пустым

Табл. 10. Часовой пояс, используемый в сервисах

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`COMPOSE_TIMEZONE`	`COMPOSE_-TIMEZONE=Asia/Yekaterinburg`	

Вспомогательные параметры

Табл. 11. Реквизиты подключения к общей БД `Postgres`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SHARED_POSTGRES_HOST`	`SHARED_POSTGRES_HOST=postgres`	По умолчанию используется внутренняя БД по адресу `Postgres`
`SHARED_POSTGRES_USER`	`SHARED_POSTGRES_USER=developer`	
`SHARED_POSTGRES_PASSWORD`	`SHARED_POSTGRES_PASSWORD=`*password*	

Табл. 12. Путь к публичным образам

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`PUBLIC_IMAGE_PATH`	`PUBLIC_IMAGE_PATH=cr.udv.group/`	Знак «/» в конце обязателен, если значение не пустое. При пустом значении образы скачиваются из публичных реестров

Табл. 13. Ограничения времени выполнения скриптов

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_LAMBDA_-SERVICE_EXECUTION_TIMEOUT`	`SOAR_BACKEND_LAMBDA_SERVICE_-EXECUTION_TIMEOUT=600`	По умолчанию — 10 минут (600 секунд)
`SOAR_BACKEND_LAMBDA_-EXECUTOR_EXECUTION_TIMEOUT`	`SOAR_BACKEND_LAMBDA_EXECUTOR_-EXECUTION_TIMEOUT=600`	

Табл. 14. Уровень логирования сервисов

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_CONSOLE_LOG_-LEVEL`	`SOAR_BACKEND_CONSOLE_LOG_-LEVEL=trace`	
`SOAR_BACKEND_SMS_LOG_LEVEL`	`SOAR_BACKEND_SMS_LOG_LEVEL=error`	

Табл. 15. Реквизиты подключения к `MinIO`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_MINIO_HOST`	`SOAR_BACKEND_MINIO_HOST=minio`	По умолчанию используется внутреннее хранилище по адресу `MinIO` с реквизитами по параметрам `MINIO_`
`SOAR_BACKEND_MINIO_USER`	`SOAR_BACKEND_MINIO_USER=minioadmin`	
`SOAR_BACKEND_MINIO_PASSWORD`	`SOAR_BACKEND_MINIO_-PASSWORD=`*password*	

Табл. 16. Реквизиты подключения к `Keycloak`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_KEYCLOAK_REALM`	`SOAR_BACKEND_KEYCLOAK_REALM=SOAR`	По умолчанию используется внутренний `Keycloak` с реквизитами по параметрам `KEYCLOAK_`
`SOAR_BACKEND_KEYCLOAK_-CLIENT_ID`	`SOAR_BACKEND_KEYCLOAK_CLIENT_-ID=soar_services_app_client`	
`SOAR_BACKEND_KEYCLOAK_-SECRET`	`SOAR_BACKEND_KEYCLOAK_SECRET=`*secret*	

Табл. 17. Реквизиты подключения к `NATS`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_LAMBDA_-SERVICE_EXECUTION_-SECURITY_TOKEN`	`SOAR_BACKEND_LAMBDA_SERVICE_-EXECUTION_SECURITY_TOKEN=`*secret*	
`SOAR_BACKEND_LAMBDA_-EXECUTOR_API_TRANSPORT_-SECURITY_TOKEN`	`SOAR_BACKEND_LAMBDA_EXECUTOR_API_-TRANSPORT_SECURITY_TOKEN=`*secret*	
`SOAR_BACKEND_LAMBDA_-EXECUTOR_SYSTEM_USER_-PASSWORD`	`SOAR_BACKEND_LAMBDA_EXECUTOR_-SYSTEM_USER_PASSWORD=`*password*	

Табл. 18. Реквизиты подключения к `Vault`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_SECRETS_-ENDPOINT`	`SOAR_BACKEND_SECRETS_-ENDPOINT=http://vault:8200`	
`SOAR_BACKEND_SECRETS_TOKEN`	`SOAR_BACKEND_SECRETS_TOKEN=`	По умолчанию используется внутренний по адресу `Vault`. При использовании внутреннего `Vault` токен создается автоматически, а **параметр оставляется пустым**

Табл. 19. Реквизиты для авторизации через `Keycloak`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_FRONTEND_KEYCLOAK_-REALM`	`SOAR_FRONTEND_KEYCLOAK_REALM=SOAR`	
`SOAR_FRONTEND_KEYCLOAK_-SECRET`	`SOAR_FRONTEND_KEYCLOAK_-SECRET=`*secret*	

Табл. 20. Хост для выполнения задач по умолчанию

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_FRONTEND_TASK_-RESOURCE_HOST`	`SOAR_FRONTEND_TASK_RESOURCE_-HOST=localhost`	

Табл. 21. Реквизиты подключения к БД `Postgres`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`ORCHESTRA_POSTGRES_COMMON_-DATABASE`	`ORCHESTRA_POSTGRES_COMMON_-DATABASE=common`	
`ORCHESTRA_POSTGRES_QUARTZ_-DATABASE`	`ORCHESTRA_POSTGRES_QUARTZ_-DATABASE=quartz`	
`ORCHESTRA_UPGRADE_RUN_AT_-STARTUP`	`ORCHESTRA_UPGRADE_RUN_AT_-STARTUP=true`	

Табл. 22. Настройки внутренней БД `Postgres`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`POSTGRES_VERSION`	`POSTGRES_VERSION=11`	
`POSTGRES_USER`	`POSTGRES_USER=developer`	
`POSTGRES_PASSWORD`	`POSTGRES_PASSWORD=`*password*	

Табл. 23. Настройки внутреннего `MinIO`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`MINIO_VERSION`	`MINIO_-VERSION=RELEASE.2022-03-05T06-32-39Z`	
`MINIO_ROOT_USER`	`MINIO_ROOT_USER=minioadmin`	
`MINIO_ROOT_PASSWORD`	`MINIO_ROOT_PASSWORD=`*password*	

Табл. 24. Настройки внутреннего `Keycloak`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`KEYCLOAK_VERSION`	`KEYCLOAK_VERSION=16.1.1`	

Табл. 25. Реквизиты подключения к БД `Postgres`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`KEYCLOAK_POSTGRES_DB`	`KEYCLOAK_POSTGRES_DB=keycloak`	

Табл. 26. Реквизиты по умолчанию для администратора

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`KEYCLOAK_ADMIN`	`KEYCLOAK_ADMIN=admin`	
`KEYCLOAK_ADMIN_PASSWORD`	`KEYCLOAK_ADMIN_PASSWORD=`*password*	

Табл. 27. Реквизиты автоматически создаваемой области `realm`, пользователя и клиента

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`KEYCLOAK_SOAR_REALM`	`KEYCLOAK_SOAR_REALM=SOAR`	
`KEYCLOAK_SOAR_ADMIN`	`KEYCLOAK_SOAR_ADMIN=admin`	

Табл. 27. Реквизиты автоматически создаваемой области `realm`, пользователя и клиента

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`KEYCLOAK_SOAR_ADMIN_-PASSWORD`	`KEYCLOAK_SOAR_ADMIN_-PASSWORD=`*password*	
`KEYCLOAK_SOAR_USER`	`KEYCLOAK_SOAR_USER=demo`	
`KEYCLOAK_SOAR_USER_PASSWORD`	`KEYCLOAK_SOAR_USER_-PASSWORD=`*password*	
`KEYCLOAK_SOAR_CLIENT_ID`	`KEYCLOAK_SOAR_CLIENT_ID=soar_-services_app_client`	
`KEYCLOAK_SOAR_SECRET`	`KEYCLOAK_SOAR_SECRET=`*secret*	

Табл. 28. Реквизиты для сервиса импорта

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`SOAR_BACKEND_KEYCLOAK_-SYSTEM_USER`	`SOAR_BACKEND_KEYCLOAK_-SYSTEM_USER=`*system*	
`SOAR_BACKEND_KEYCLOAK_-SYSTEM_USER_PASSWORD`	`SOAR_BACKEND_KEYCLOAK_-SYSTEM_USER_PASSWORD=`*password*	

Табл. 29. Настройки внутреннего `Vault`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`VAULT_POSTGRES_DB`	`VAULT_POSTGRES_DB=vault`	
`VAULT_USER`	`VAULT_USER=admin`	
`VAULT_PASSWORD`	`VAULT_PASSWORD=`*password*	

Табл. 30. Настройки внутреннего `NATS`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`NATS_AUTH_TOKEN`	`NATS_AUTH_TOKEN=`*secret*	

Табл. 31. Настройки внутреннего `PgAdmin`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`PGADMIN_HOST`	`PGADMIN_HOST=pgadmin`	

Табл. 31. Настройки внутреннего `PgAdmin`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`PGADMIN_DEFAULT_EMAIL`	`PGADMIN_DEFAULT_-EMAIL=pgadmin4@pgadmin.org`	
`PGADMIN_DEFAULT_PASSWORD`	`PGADMIN_DEFAULT_PASSWORD=`*password*	

Табл. 32. Настройки внутреннего `Grafana`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`GRAFANA_SECURITY_ADMIN_USER`	`GRAFANA_SECURITY_ADMIN_USER=admin`	
`GRAFANA_SECURITY_ADMIN_-PASSWORD`	`GRAFANA_SECURITY_ADMIN_-PASSWORD=`*password*	
`GRAFANA_LOKI_VERSION`	`GRAFANA_LOKI_VERSION=2.6.1`	
`GRAFANA_LOKI_MINIO_HOST`	`GRAFANA_LOKI_MINIO_HOST=minio`	
`GRAFANA_LOKI_MINIO_USER`	`GRAFANA_LOKI_MINIO_USER=minioadmin`	
`GRAFANA_LOKI_MINIO_PASSWORD`	`GRAFANA_LOKI_MINIO_-PASSWORD=`*password*	
`GRAFANA_LOKI_MINIO_BUCKET`	`GRAFANA_LOKI_MINIO_BUCKET=loki`	
`GRAFANA_LOKI_RETENTION_-PERIOD`	`GRAFANA_LOKI_RETENTION_PERIOD=336h`	

Табл. 33. Настройки внутреннего `nginx`

ПАРАМЕТР	ПРИМЕР ЗАПОЛНЕНИЯ	ПРИМЕЧАНИЯ
`NGINX_VERSION`	`NGINX_VERSION=alpine3.21`	

7. Адреса сервисов по умолчанию

Табл. 34. Стандартные адреса сервисов

СЕРВИС	АДРЕС
`Grafana`	`https://grafana.test.local/`
`Keycloak`	`https://keycloak.test.local/`
`Lambda Executor`	IP-адрес машины клиента
`Lambda Service`	`https://lambda.test.local /`
`MinIO`	`https://minio.test.local/`
`NATS`	`https://nats.test.local/`
`Pgadmin`	`https://pgadmin.test.local/`
`Postgres`	–
`Prometheus`	`https://prometheus.test.local/`
`Proxy Service`	`https://irp-proxy.test.local/`
`State Machine Executor`	–
`State Machine Service`	`https://state-machine.test.local /`
`Vault`	`https://vault.test.local/`