

VisionLabs LUNA Access

Administrator manual

2.17.0

Contents

1	Glossary	4
2	Introduction	5
3	System requirements	6
4	Access architecture	7
4.1	Access scheme	7
4.2	Sequence diagrams	8
4.2.1	Interaction of internal components of Access	8
4.2.2	Creating a component	10
4.2.3	Automatic monitoring	11
5	Licensing	13
6	Deploying the Access	14
6.1	Preparing for installation	14
6.2	Docker and Docker Compose installation	14
6.3	Preparing and unpacking the distribution	14
6.3.1	Setting up Access	15
6.4	Launching the Access	18
7	Accounts Management	19
7.1	Adding an administrator account	19
7.2	Viewing Account List	19
7.3	Deleting Account	19
8	Scripts	21
8.1	Image loading script	21
9	Access update	22
10	Deleting Access	24
11	Logging system	25
11.1	Logging guide	25
11.1.1	Settings file	25
11.1.2	Container logs	26
11.1.3	Env file	26
11.1.4	Information about the working environment	27
11.1.5	Screenshots of UI	27

11.2	Tracking passages using trace_id	28
11.2.1	Tracking Luna Platform Event Passes	28
11.2.2	How to find trace_id by person's full name	28
11.2.3	Search for an event in Luna Platform by trace_id	29

1. Glossary

Term	Description
Docker	An open platform for developing, shipping, and running applications. Docker provides the ability to package and run an application in a loosely isolated environment called a container
Docker Compose	A tool for defining and running multi-container Docker applications
Identification	Search for the most suitable biometric face template by comparing the feature vectors of a photo image of a face with a list of similar biometric templates in the database (one to many)
Container	A runnable instance of an image. A container is defined by its image as well as any configuration options provided to it when it was created or started. When a container is removed, any changes to its state that are not stored in persistent storage disappear. The isolation and security allow to run many containers simultaneously on a given host
Docker image	A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform. It provides a convenient way to package up applications and preconfigured server environments
Software	A program or set of programs used to control a computer
Physical access control system (PACS)	A set of hardware and software tools aimed at controlling the entrance and exit in order to ensure safety and regulate visits to a particular facility

2. Introduction

The document describes the process of installing and configuring the VisionLabs LUNA Access service version 2.17.0 (hereinafter — Access), and also contains hardware and software requirements for the software.

The configuration and installation process must be performed under a superuser account (with root rights).

3. System requirements

Software Requirements (Table 1).

Table 1. Software Requirements

Resource	Recommended
Operating system (OS)	Docker supported OS (CentOS, RedOS and etc.)
Docker	v.20 and up
Docker-compose	v.1.21 and up
unzip	v.6.0-47

Workstation hardware requirements (Table 2).

Table 2. Hardware Requirements

Resource	Recommended
CPU	Intel/AMD x64 2,0 hHz
RAM	4 GB or more
Free space on disk (HDD/SSD)	20 GB or more

4. Access architecture

4.1. Access scheme

The Access scheme is shown below (Figure 1) and (Table 3).

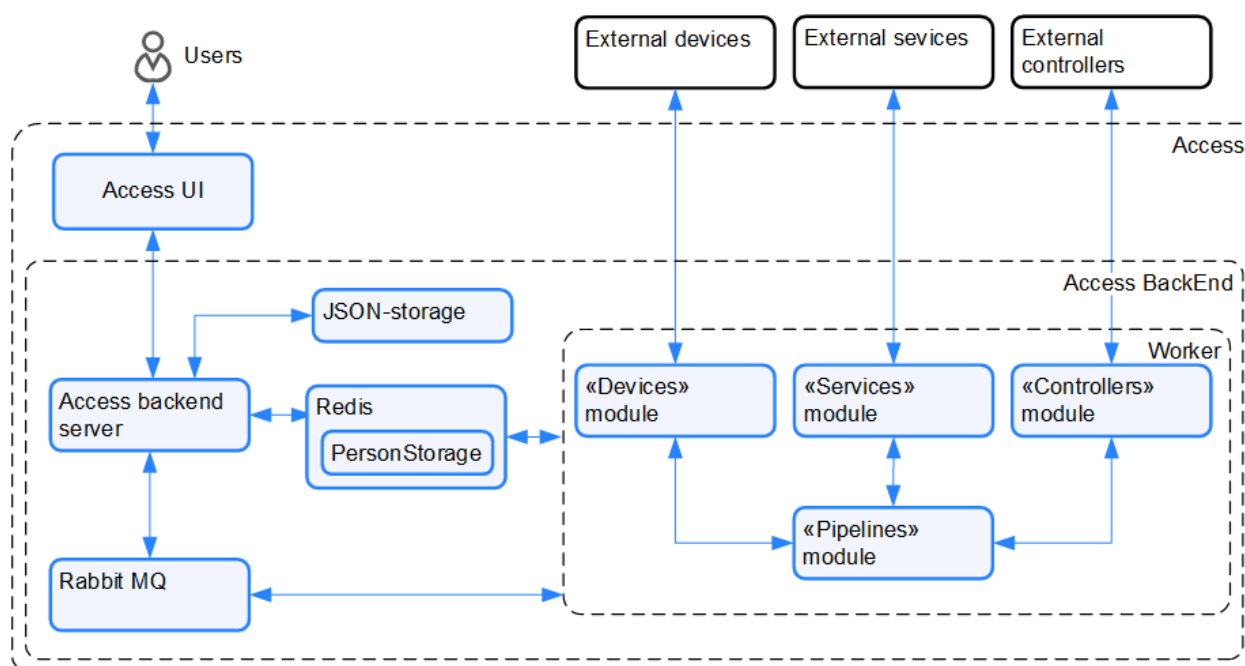


Figure 1: Scheme Access

Table 3. Scheme description.

Component	Description
Users	The system administrator configures and maintains Access.
Access	A set of management software tools that allow for the collaboration of VisionLabs products and various access control and management systems (ACS).
Access UI	Access graphical user interface.
Access BackEnd	BackEnd Access, responsible for working with external components, interaction with the UI.
Access backend server	A set of libraries that combines Access and UI modules.
Rabbit MQ	Message queue broker.
Redis	A system for storing data in the form of structures to ensure the performance of Access.

Component	Description
PersonStorage	Local storage storing the connection between ACS personalities and their biometric data. Used for data synchronization between access control systems and biometric systems (LP, CBS) or external services.
JSON-storage	settings.json file to store integration settings and user data.
Worker	A set of modules for interacting with external components.
Pipeline module	Access module, which provides the basic logic and interaction of modules.
Devices Module	Access module that contains libraries for connecting external devices to Access
Services Module	Access module that contains libraries for connecting external services to Access
Controllers Module	Access module that contains libraries for working with external controllers
External devices	Connectable cameras, terminals, and thermal imaging cameras that transmit a video stream for further processing. For a complete list of available devices, see the User Manual
External services	PACS or VisionLabs products that can be used in integration. For a complete list of available PACS or VisionLabs products, see the User Manual
External controllers	External controllers (for example, converters for ACS controllers) used in integrations.

4.2. Sequence diagrams

4.2.1. Interaction of internal components of Access

Interaction of Access components on the example of typical integration of video signal source + LP5 + Sigur (Figure 2) and (Table 4).

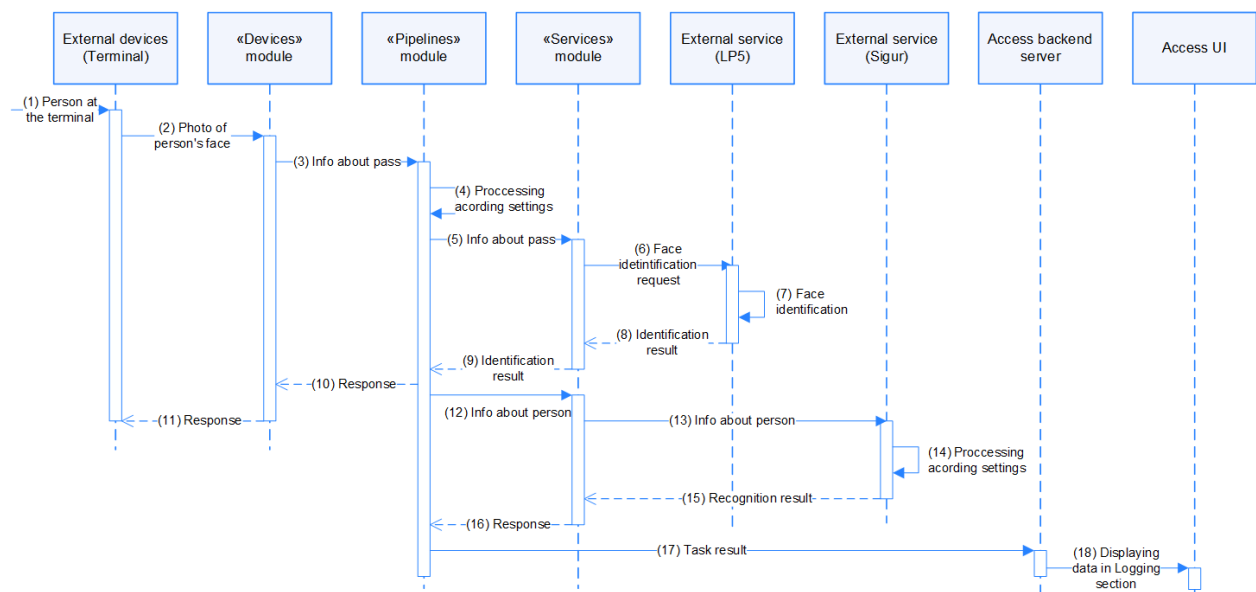


Figure 2: Process scheme

Table 4. Sequence diagram description

Step	Description
(1)	A person approached the turnstile intending to pass, and the terminal captured the person's face.
(2)	The terminal transmitted the person's photo to the Devices module.
(3)	The Devices module sends the face data to the Pipeline module for further processing.
(4)	The Pipeline module processes the photo according to its internal settings.
(5)	The Pipeline module transmits the passage data to the Service module for further processing.
(6)	The Service module sends a request to the external service (LP5) for face identification.
(7)	The external service (LP5) performs the identification.
(8)	The external service (LP5) returns a response with the results of the face recognition and identification.
(9)	The Services module transmits the recognition data to the Pipeline module.
(10)	The Pipeline module returns the recognition result to the Devices module.
(11)	The Devices module sends the person's name data to the terminal for displaying the name and a message about successful identification on the terminal screen.
(12)	The Pipeline module transmits the recognition data to the Services module.

Step	Description
(13)	The Services module transmits the recognition data to the external service (Sigur).
(14)	The external service (Sigur) processes the recognition result according to its internal instructions. In the case of successful recognition, it sends a signal to open the turnstile.
(15)	The external service (Sigur) returns the processing result to the Services module.
(16)	The Services module transmits the processing data to the Pipeline module.
(17)	The Pipeline module transmits the data to the Access backend server using RabbitMQ.
(18)	The Access backend server sends data to the Access UI for displaying the passage result in the Logging section.

4.2.2. Creating a component

Creating a component in Access UI (Figure 3) and (Table 5).

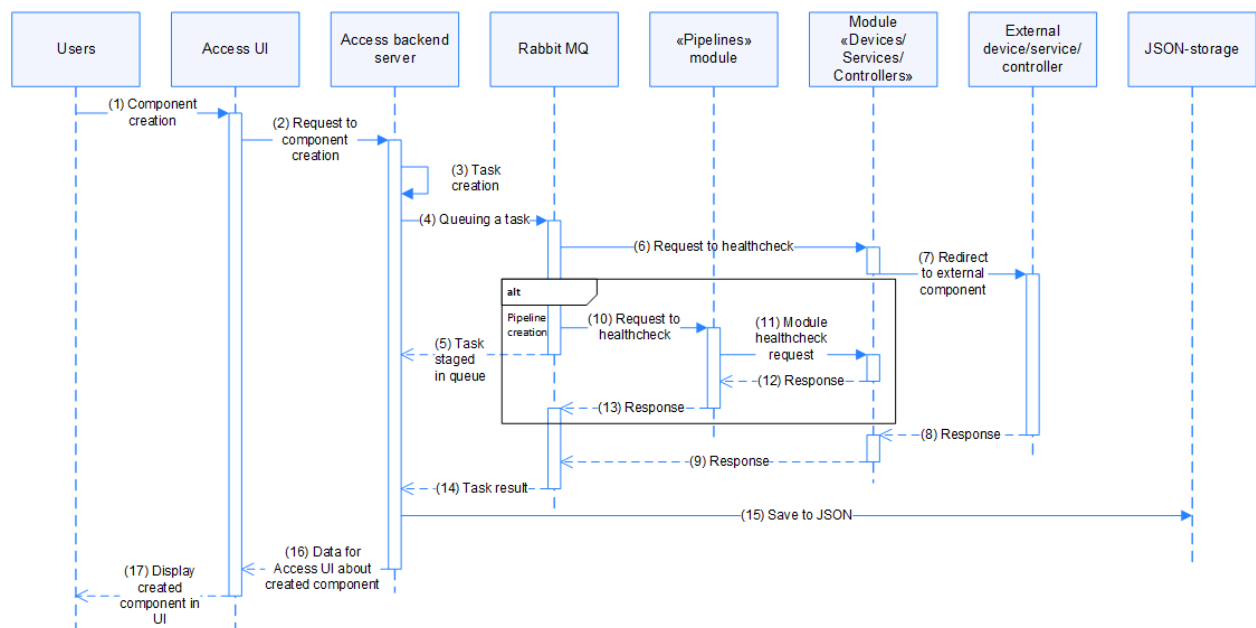


Figure 3: Process scheme

Table 5. Sequence diagram description

Step	Description
(1)	The user fills in the data of a new object (device, service, pipeline, or controller) in the Access UI.

Step	Description
(2)	Access UI sends a request to the Access backend server to create a component.
(3)	Access backend server creates the component and generates a task to check the status of the connected component.
(4)	Access backend server sends a task to Rabbit MQ to enqueue the task.
(5)	Rabbit MQ returns the response that the task has been queued.
(6)	Rabbit MQ sends a request to check the status of the component to the appropriate module (Devices/Services/Controllers) specified in the component's settings.
(7)	The Devices/Services/Controllers module redirects the status check request to an external device/service/controller.
(8)	External device/service/controller returns an activity status response.
(9)	The Devices/Services/Controllers module returns a response about the state of the external device/service/controller.
(10)	Alt pipeline creation. Rabbit MQ sends a request to the Pipeline Module to check the status of the components specified in the pipeline settings.
(11)	Alt pipeline creation. The Pipeline module redirects the status check request to the Devices/Services/Controllers modules specified in the pipeline settings.
(12)	Alt pipeline creation. The Devices/Services/Controllers modules return a response about the availability of components.
(13)	Alt pipeline creation. The Pipeline module returns a status response.
(14)	Rabbit MQ returns the result of the task to the Access backend server.
(15)	Access backend server sends data to the JSON-storage to save information about the pass.
(17)	Access backend server sends data to the Access UI to display the result of creating the component.
(18)	Access UI displays the created components with an active status for the user.

4.2.3. Automatic monitoring

Automatic monitoring of the state of components (Figure 4) and (Table 6).

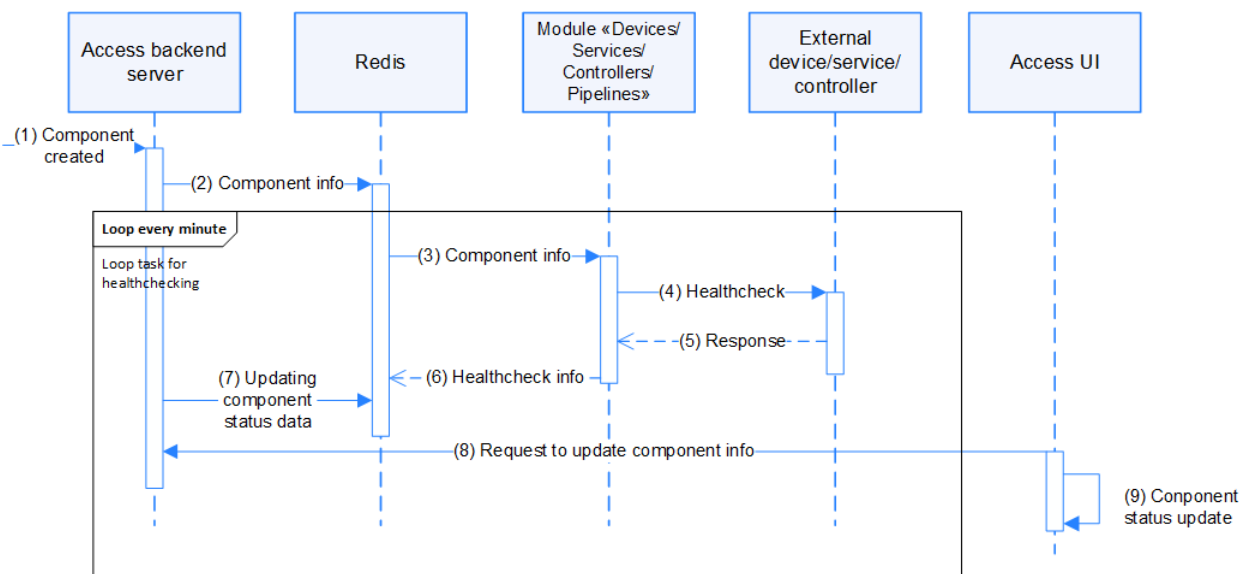


Figure 4: Process scheme

Table 6. Sequence diagram description

Step	Description
(1)	The entry point for this process is the creation of any component.
(2)	The Access backend server transmits the data of the created component to Redis.
(3)	Redis sends the data of the created component to a Worker, which every minute sends a request to the connected Modules to check the status of external devices.
(4)	The Modules forward the connection check request to the external components.
(5)	The external components return a response regarding the connection status; if no response is received, the module considers the connected device to be offline.
(6)	The Module records the component status data in Redis.
(7)	The Access backend server retrieves the component status data from Redis.
(8)	The Access UI requests the component's activity status from the Access backend server.
(9)	The Access UI updates the component activity status indicator.

5. Licensing

Access does not require a license to work.

Systems and services that are used in the integration may require license. In this case, the license is purchased separately from the copyright holder.

6. Deploying the Access

This section describes how to install and use Docker and Docker Compose to deploy Access.

Docker and Docker Compose are not included in the Service distribution package.

6.1. Preparing for installation

Create a main directory where all versions of the product will be located in the future and go to it.:

```
sudo mkdir /var/lib/vl-access-2/  
sudo chown $(whoami) /var/lib/vl-access-2/  
cd /var/lib/vl-access-2/
```

6.2. Docker and Docker Compose installation

Use the official instructions to install Docker Engine and Docker Compose for the OS you are using.

Before starting Access, you need to make sure that Docker is running and active.

1. Start Docker:

```
systemctl start docker  
systemctl enable docker
```

2. Check Docker status:

```
systemctl status docker
```

Response must contain Active (running) status.

6.3. Preparing and unpacking the distribution

The distribution package is an archive of the form «vl-access-2-v2.17.0».

A link to download the distribution kit must be requested from a VisionLabs representative.

The archive contains the components necessary for the installation and operation of Access.

The archive does not include dependencies that are included in the standard distribution package of OS and can be downloaded from open sources.

The Service is launched from a Docker image.

Follow these steps:

1. Move vl-access-2-v2.17.0.zip to /var/lib/vl-access-2.
2. Install the unzip archiver if it is not installed. The command specifies version 6.0-47, later versions of the utility have not been tested.:

```
yum install unzip-6.0-47.el8_10
```

3. Unpack files:

```
unzip vl-access-2-v2.17.0.zip -d vl-access-2-v2.17.0
```

4. The directory contains:

- the docs directory with documentation in pdf/html format;
- README_FOR_ENGINEERS.md file with description of quick start;
- .env configuration file.
- docker-compose.yml configuration file;
- conf.yml configuration;
- CHANGELOG.md changelog;
- archive with the image vl_access_2_v2.17.0.tar.gz (if available).

5. Change directory:

```
cd vl-access-2-v2.17.0
```

When working with a distribution without an image, you must download the image at startup, see [Launching](#)

6.3.1. Setting up Access

Make the configuration using the .env file (Table 7):

```
nano .env
```

Table 7. Description of env parameters

Parameter	Description	Default values
FastAPI Settings		

Parameter	Description	Default values
DEBUG	Access debug mode - output in the OS logs and in the interface of information like Debug about the work of Access <ul style="list-style-type: none"> • 1 – debugging. • 0 – without debugging. 	0
VL_-ACCESS_-TAG	The Access tag is taken from the internal settings. It is not recommended to change this parameter.	2.17.0
LOG_DB_-HOST	The host name for storing logs.	log-storage
LOG_DB_-PORT	The port for connecting to log-storage.	27017
LOG_DB_-NAME	Name of the log database.	logs
LOG_DB_-USER	DB user name.	username
LOG_DB_-PASSWORD	DB user's password.	password
C_FORCE_-ROOT	Force Celery to run as root user.	true
WORKER_-CONCURRENCY	The maximum number of processes in the Worker component that can be processed in parallel. It is set based on the load on the system.	16
WORKERS_-AMOUNT	Number of worker instances (required to increase the number of processed events, used on highly loaded objects).	1
Redis Settings		
REDIS_-HOST	Redis host name.	redis
REDIS_-PORT	Redis port.	6379
REDIS_DB_-BASE	The number of the primary Redis database.	0

Parameter	Description	Default values
REDIS_DB_- PERSONS	The number of the Redis database that stores information about people.	1
REDIS_DB_- CELERY_- BEAT	Redis database number for storing information about periodic worker-beat service tasks.	2
Rabbit Settings		
RABBITMQ	Name of the message queue broker. Access only supports Rabbit MQ.	rabbitmq
RABBITMQ_- USER	User login to connect to Rabbit MQ	guest
RABBITMQ_- DEFAULT_- PASSWORD	Password to connect to Rabbit MQ	guest
RABBITMQ_- PROTOCOL	RabbitMQ protocol type. Only AMQP is supported	amqp
RABBITMQ_- URL	Address to connect to Rabbit	<pre> \${ RABBITMQ_PROTOCOL }://\${ RABBITMQ_DEFAULT_USER }: \${ RABBITMQ_DEFAULT_PASS }@\${RABBITMQ }:5672/ </pre>
CELERY_- BROKER_- URL	Address to connect to Celery	<pre> \${ RABBITMQ_PROTOCOL }://\${ RABBITMQ_DEFAULT_USER }: \${ RABBITMQ_DEFAULT_PASS }@\${RABBITMQ }:5672/ </pre>
FrontEnd Settings		

Parameter	Description	Default values
BACKEND_HOST	Host of fastapi service. Specify IP address if frontend and fastapi are running on different machines	fastapi
BACKEND_PORT	Port to connect to the Access backend	9091

6.4. Launching the Access

1. Import the image:

1.1. If there is an image in the archive:

```
docker load -i vl-access-2-images-v2.17.0.tar.gz
```

1.2. Without image:

```
docker login dockerhub.visionlabs.ru
docker-compose pull
docker logout dockerhub.visionlabs.ru
```

2. Add symlink to the directory /var/lib/vl-access-2/, which links to the latest version of the product:

```
ln -s /var/lib/vl-access-2-v2.17.0 /var/lib/vl-access-2/current
```

3. Run the Service:

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

Access containers come with pre-installed utilities needed to work with the image.

4. Check the availability of the GUI at: `http://<IP_address>:9092/`.

To access the Access interface, create an administrator (see [Account Management](#)).

7. Accounts Management

7.1. Adding an administrator account

Access supports creating an account with the Administrator role. For more information about roles and accesses, see the Roles in the Service section in the User's Guide.

1. Run the administrator creation script:

```
docker-compose exec fastapi python backend/manage.py createadmin
```

2. Follow the console instructions:

```
(venv) [root@localhost vl-access-2]# docker-compose exec fastapi python
backend/manage.py createadmin
Admin creation:
Enter your login: admin
Enter your password:
Confirm your password
Admin user created successfully
```

The administrator's login must be unique

3. Check if the administrator was created correctly. Log in to the created account: `http://<IP_address>:9092/`.

7.2. Viewing Account List

1. Run the command to view the list of created administrators:

```
docker-compose exec fastapi python backend/manage.py listadmin
```

A list of administrators will be displayed in the console:

```
(venv) [root@localhost vl-access-2]# docker-compose exec fastapi python
backend/manage.py listadmin
admin
admin2
```

7.3. Deleting Account

Get the login information for the account you need to delete using the [view](#) command.

1. Run the command to remove the administrator:

```
docker-compose exec fastapi python backend/manage.py deleteadmin
```

2. Enter the administrator login.

In response to the request, the console will display a message about successful deletion:

```
(venv) [root@localhost vl-access-2]# docker-compose exec fastapi python
    backend/manage.py deleteadmin
Admin deletion:
Enter your login: admin2
Admin was deleted successfully.
```

8. Scripts

8.1. Image loading script

The script `scripts/save_images.sh` allows you to upload docker images using the paths specified in `docker-compose.yml` format `tar.gz` (Table 8).

Uploading may be necessary to save images to storage media and transfer them to an object in the absence/poor Internet connection.

Running the script:

```
./save_images.sh -f <path_to/docker-compose.yml> -d <local_path_to/
docker_images> -u <dockerhub_username> -p <dockerhub_password>
```

Table 8. Available Commands:

Key	Description
--	-----
-h	Command Help
-f	Path to and name docker-compose.yml. Required
-d	Path to and directory for saving the image. Required
-o	Image name.
-u	Login to connect to dockerhub.visionlabs.ru. Required
-p	Password for connecting to dockerhub.visionlabs.ru. Required

The final archive must be placed in the Access directory and launched according to [instructions](#).

9. Access update

1. Export the settings file of the old version of the product by clicking on the ▼ to the right of the user's avatar and click on the **Export settings** button (Figure 5).

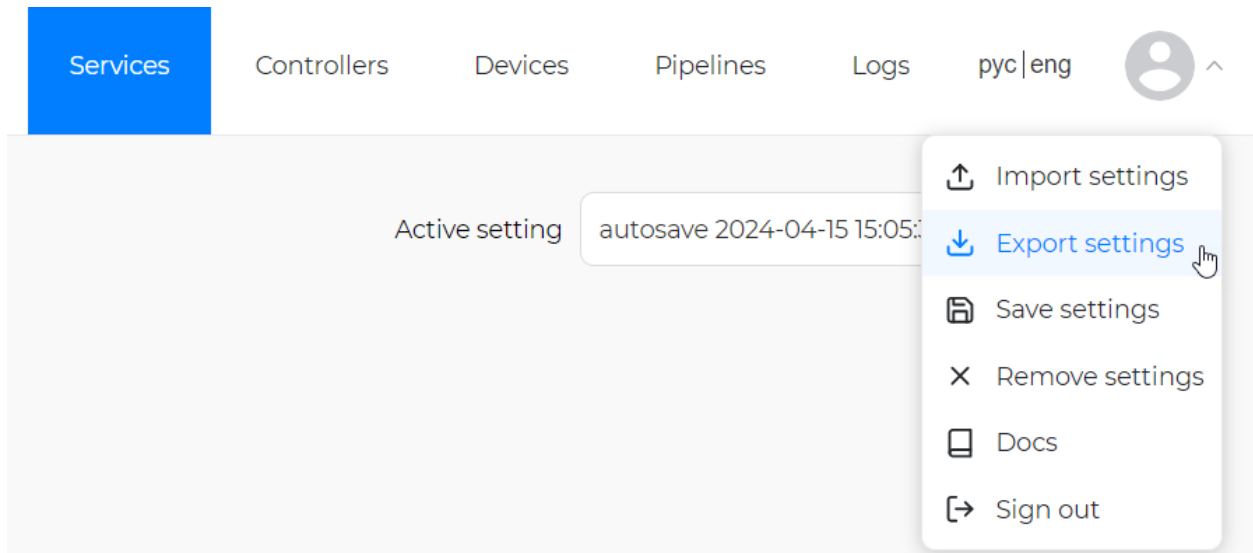
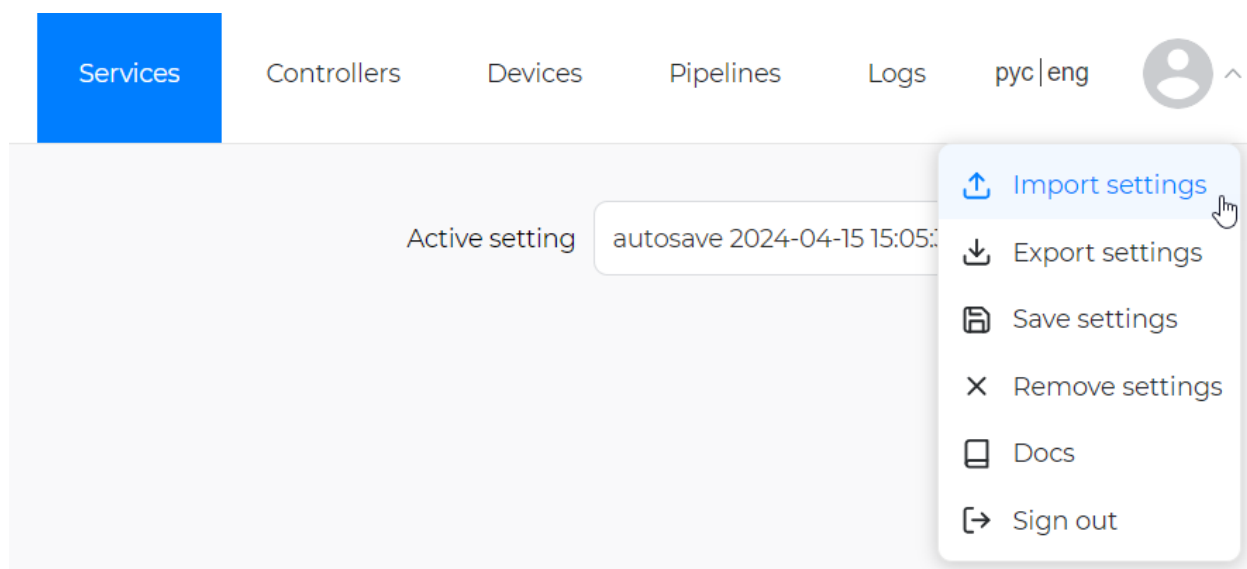


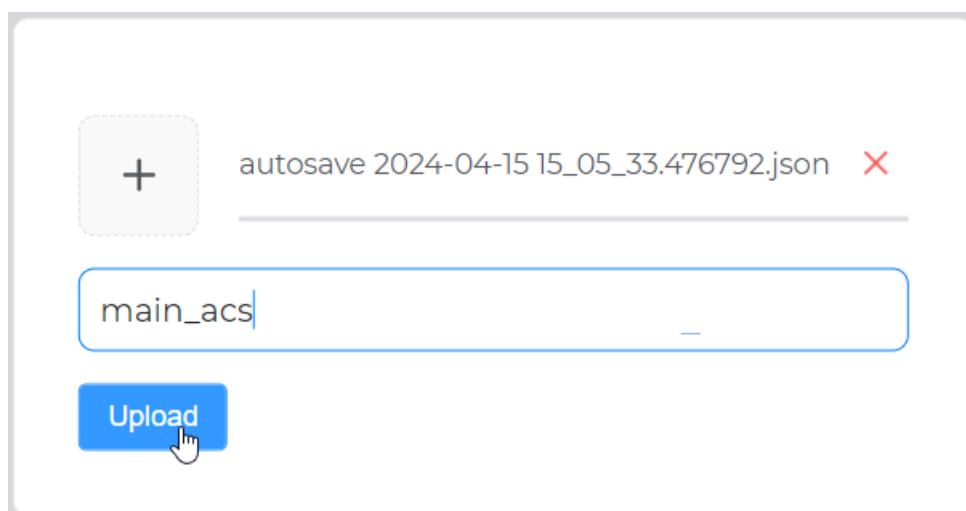
Figure 5: Export settings

A file with all component settings in json format will be loaded.

2. Save the configuration file.the env of the old version, located along the path `/var/lib/vl-access-2/vl-access-2-v2.*.*/.env`, where `v2.*.*` is the version number of the product.
3. To update Access, perform [Deletion](#) → [Unpacking the new distribution version](#) → [Configuring env parameters](#) (transfer the necessary variable values from the saved configuration file.env the old version to a new file) and repeat [Installing and launching Access](#).
4. After installing the new version, go to UI Access, click on the arrow ▼ to the right of the user's avatar and the **Import Settings** button (Figure 6).

**Figure 6:** Import settings

In the window that opens, select the previously saved json file, enter the name of the setting and click the **Upload** button (Figure 7).

**Figure 7:** Load settings file

When trying to import a settings file from an earlier version of Access to a new one, backward compatibility errors may occur. In case of problems, contact VisionLabs technical support.

10. Deleting Access

Deleting Access is done by stopping and deleting Docker containers.

Follow these steps:

1. Switch to the super user:

```
sudo su
```

2. Delete the Access directory:

```
rm -rf vl-access-2-v2.*.*
```


11. Logging system

11.1. Logging guide

Collecting logs is necessary for:

- Providing VisionLabs technical support information for filing a ticket to find a problem;
- Independent search for errors.

To receive complete information about an emergency situation when using Access, you must prepare and submit information to a VisionLabs representative about:

- [Settings file](#);
- [Container logs](#);
- [File .env](#);
- [Information about the working environment](#);
- [UI Screenshots](#) (only in cases of UI errors).

11.1.1. Settings file

Settings file is a JSON file that contains information about the components used in Access.

The settings file becomes available for export after creating any component in Access.

1. After creating any of the 4 types of components, click on ▼ to the right of the user's avatar and click the "Export Settings" button (Figure 8).

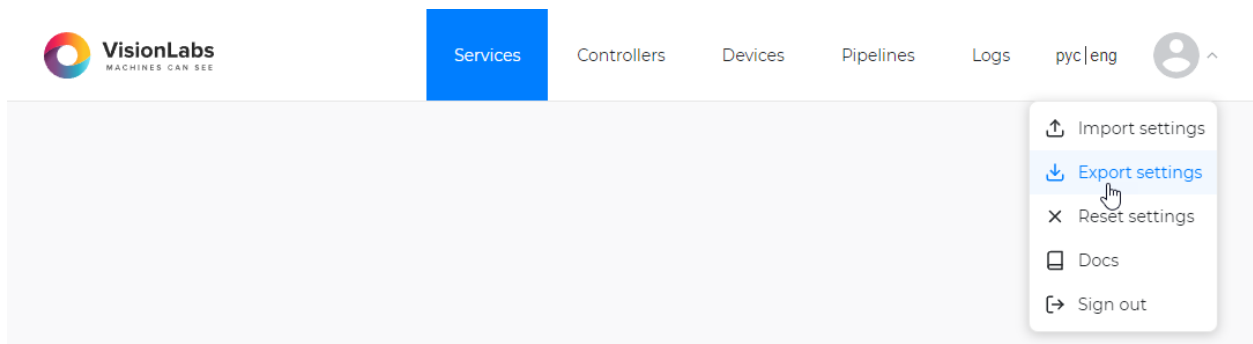


Figure 8: Export settings

This will download a JSON file called vl-access_settings.json.

2. Find the JSON on your local machine.

For Linux systems, the default is /home/<username>/Downloads.

3. Rename the settings file depending on the main services and devices used, for example, bolid+gate+fast.json.

11.1.2. Container logs

Container log files contain all information about the operation of Access from the moment of startup (docker compose up) to the creation of logs, provided that the containers are running.

1. Open the Access directory in the console.

For self-checking, make sure that this directory contains /db and docker-compose.yml (Figure 9).

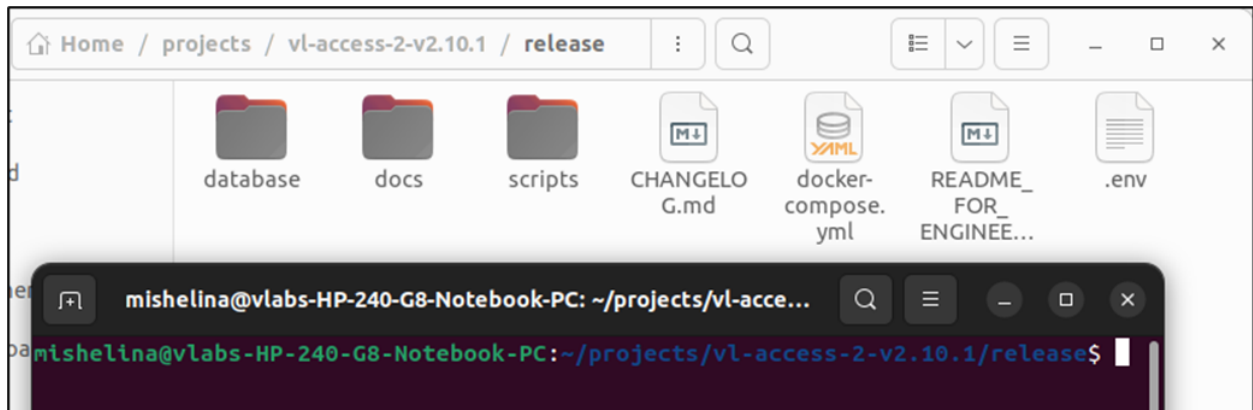


Figure 9: Correct directory

2. Activate debug mode: in the .env file, set the value of the variable DEBUG=1.
3. Restart Access:

```
docker-compose down  
docker-compose up
```

4. Run the commands to record worker and fastapi containers logs:

The name of the log file must contain the names of the main components.

```
docker-compose logs worker &> worker_имена< сервисовустройств/, bolid+gate+  
fast>.log  
docker-compose logs fastapi &> fastapi_имена< сервисовустройств/, bolid+gate  
+fast>.log
```

5. Check for the generated .log files in the same directory.

11.1.3. Env file

The .env file is located in the root folder of the distribution (in the same place as docker-compose.yml), but may not appear in the default UI.

1. Locate the .env file to provide to your VisionLabs representative.

11.1.4. Information about the working environment

Check the hardware and software properties of the working machine with the minimum ones (see [Requirements](#)).

To check the requirements, run the view commands:

1. OS version of the local machine:

```
hostnamectl
```

2. Docker/docker-compose version:

```
docker --version  
docker-compose --version
```

3. Information about hardware:

```
cat /proc/cpuinfo | grep "model name"
```

4. Amount of free RAM:

```
free -h
```

5. Amount of free physical memory:

```
df -h
```

11.1.5. Screenshots of UI

Screenshots are only needed to find a problem with the UI:

- the `is_alive` status of the component is displayed inadequately,
- a component is displayed that should not exist (it was removed, but it “resurrected”),
- an unreadable error appeared,
- there are duplicates in the list in LUNA Clementine - the screenshot should include the dates of creation of the persons in the list,
- in the events in LUNA Clementine something doesn’t match correctly, the event tag is incorrect,
- other problems with the UI.

11.2. Tracking passages using trace_id

A unique identifier, `trace_id`, is generated for each passage through the ACS. This identifier links all events related to one passage of a specific person: from the moment of detection to the provision of access through the turnstile.

Using `trace_id` allows you to quickly and accurately track the entire passage process, analyze delays at different stages and identify possible problems.

Using trace_id, you can track:

- The time of receiving and processing events in the system;
- Information about the recognition result: full name, card number, identifiers in the ACS and biometric system;
- The time of execution of key operations: detection, recognition, sending data to the terminal and controller;
- Other technical details related to processing the passage.

11.2.1. Tracking Luna Platform Event Passes

1. Open the **Luna CLEMENTINE / Luna Platform** interface and go to the **Latest Events** section.
2. Find the oldest successful recognition event.
3. Go to the card of this event by clicking on the arrow in the upper right corner next to the photo (Figure 10).

Successful recognition event

4. Copy the eight-digit `trace_id` from the **Tags** field (Figure 11).

Location of `trace_id` in the event card

5. Make sure that debug mode (debug) is enabled in LUNA Access 2 and DEBUG-level messages are present in the worker log.
6. Find all logs related to this pass by running the command:

```
docker-compose logs fastapi worker | grep "<trace_id>"
```

Example (Figure 12).

Example log

11.2.2. How to find trace_id by person's full name

1. Make sure debug mode (debug) is enabled for LUNA Access 2 and there are DEBUG logs in the worker.

2. Find the required log by full name:

```
docker-compose logs worker | grep -i "<name>"
```

3. The found log will contain `trace_id` in the format `trace <trace_id>` (Figure 13).

Display `trace_id` in log

4. Use this `trace_id` to search all related logs:

```
docker-compose logs fastapi worker | grep "<trace_id>"
```

11.2.3. Search for an event in Luna Platform by `trace_id`

The steps to display an event by `trace_id` in the UI are as follows (Figure 14).

Display `trace_id` in log

1. In **Luna CLEMENTINE / Luna Platform**, go to **Recent Events**.
2. Click the filter icon on the right.
3. Enter the `trace_id` value in the **Tags** field:
4. Click the **Filter** button.