Operations

Production deployment

Didmos consists of different modules (see didmos 2.0). Some of these modules are based on existing open source software with configuration and extensions provided by DAASI (e. g. the LDAP server in didmos Core or didmos Auth). Other modules are developed and shipped by DAASI (e.g. didmos LUI) . Therefore the deployment and operations model for the components is quite heterogenous.

Deployment is supported as either Docker containers or as a VM based deployment for most components with some exceptions as per the following list:

	Docker	VM based (Ubuntu)
Core	yes	yes
LUI	yes	yes
Authenticator	yes	yes
Provisioner	yes	no
ETL	yes	no
Pwd Synchronizer	no	yes

For VM based deployment only Ubuntu 18/20 is currently supported.

Docker deployment

Docker images are provided for all components (except Pwd Synchronizer) and this is the preferred deployment model.

In order to run didmos as docker containers the following requirements must be met:

- docker: https://docs.docker.com/compose/install/, version 20.10.0 or later
- docker-compose: https://docs.docker.com/compose/install/, version 1.25.0 or later

A docker-compose.yml file describes the system as a whole. See the following documentation and example for didmos2-demo:

https://gitlab.daasi.de/didmos2-demo/didmos2-demo-compose/-/tree/master/deploy

The docker-compose.yml file for individual projects might deviate from this example, as more or less components are included and configuration might be different. Furthermore a .env file must be located in the same directory which contains deployment specific variables.

On the docker host these files are usually located in either /root/docker or /opt/didmos.

The following commands might be useful for operations:

```
# Start
docker-compose up -d

# Stop
docker-compose down

# Display Status
docker-compose ps

# Show logs of individual container
docker logs {container-name}

# Restart individual container
docker restart {container-name}
```

VM based deployment

For the VM based deployment project specific Ansible roles are provided for initial setup. The general setup is documented here: https://gitlab.daasi.de/didmos2/didmos2-compose/-/tree/master/ansible

Please note that most didmos projects are extended by project specific roles for setup of extensions and project specific components. Generally these roles are also required for a full setup.

After running the initial setup via Ansible please refer to the following chapters for details on operations for each of the didmos modules:

Module specific details

didmos Core

Docker

didmos Core consists of two Docker containers:

Container	Description
{project-name}-core	API
{project-name}-openldap	LDAP Metadirectory

The logs of each component can be accessed via docker logs.

```
docker logs {container-name}
```

Configuration is possible via docker environment variables (for supported parameters). An example for this configuration is described in section Provisioner - Docker.

VM based

LDAP Metadirectory

The LDAP Metadirectory is installed via the Ubuntu distribution during the initial Ansible setup (i.e. apt install `slapd`).

It can be administered using the following commands:

```
systemctl {start|stop|restart|status} slapd
```

slapd logs using rsyslog. The logging is configured in the file **rsyslog.conf.** The most recent logged messages can be read using this command:

```
journalctl -r -t openldap
```

The configuration of the log level for slapd uses the attribute **olcLogLevel** in the **cn=config** subtree in the LDAP Metadirectory. The value can be changed using a LDAP client (e. g. Apache Directory Studio) using cn=config as Base-DN.

These log levels are supported:

Table 7.1: Debugging Levels

Level	Keyword	Description
-1	any	enable all debugging
0		no debugging
1	(0x1 trace)	trace function calls
2	(0x2 packets)	debug packet handling
4	(0x4 args)	heavy trace debugging
8	(0x8 conns)	connection management
16	(0x10 BER)	print out packets sent and received
32	(0x20 filter)	search filter processing
64	(0x40 config)	configuration processing
128	(0x80 ACL)	access control list processing
256	(0x100 stats)	stats log connections/operations/results
512	(0x200 stats2)	stats log entries sent
1024	(0x400 shell)	print communication with shell backends
2048	(0x800 parse)	print entry parsing debugging
16384	(0x4000 sync)	syncrepl consumer processing
32768	(0x8000 none)	only messages that get logged whatever log level is set

Numeric values and keywords for log levels are equivalent.

More information about logging and debugging OpenLDAP can be found in the documentation:

- https://www.openldap.org/doc/admin24/slapdconfig.html
- https://www.openldap.org/doc/admin24/tuning.html#Logging

didmos Core API Server

The didmos Core API server is installed as a python virtual environment and deployed as a mod_wsgi app in Apache webserver. The following locations on the VM are used:

Component	Location in file system
Python Virtual environment	/opt/didmos2coreEnv
Python application	/opt/didmos2core
Configuration (Templates and default config)	/opt/didmos2core/general /opt/didmos2core/customer/customer_config
Configuration (Overrides)	/etc/didmos/core
Logs	/var/log/didmos
Apache config (mod_wsgi integration)	/etc/apache2/sites-available/api-ssl.conf
Apache logs	/var/log/apache2

Restarting the didmos Core API server is possible via the Apache webserver:

```
systemctl {start|stop|restart|status} apache2
```

Backups of LDAP database

The core data of a didmos system is stored in the LDAP server and therefore backups of the entire LDAP server should be done regularly.

This can be done in different ways:

1. Full VM snapshot
2. Backup of data folders
    a. /var/lib/ldap (mdb database)
    b. /etc/openldap/slapd.d (config)
    c. /MIGRATIONS (state of migrations)
3. LDIF export

Note that in case of a docker deployment the folders are stored in docker volumes with the following names, which must be backed up:

- {project-name}-openldap-db
- {project-name}-openldap-config
- {project-name}-openldap-mig

didmos LUI

Docker

didmos LUI consists of the following Docker container:

- {project-name}-frontend

In this container, the compiled frontend (Angular JavaScript app with assets like images, CSS-files etc.) is shipped using an nginx webserver.

The logs can be accessed via docker logs (see list of general commands). Since the application itself runs as Java Script in the web browser, for debugging purposes the browser console might be more useful than the server side logs.

Configuration is possible via docker environment variables (for supported parameters). An example for this configuration is described in section Provisioner - Docker.

VM based

The compiled frontend (Angular JavaScript app with assets like images, CSS-files etc.) is located in /var/www/didmos2lui and then shipped as static files using an Apache webserver. The following locations on the VM are used:

Component	Location in file system
Frontend files	/var/www/didmos2lui
Configuration file	/var/www/didmos2lui/assets/config/environment.json
Apache config	/etc/apache2/sites-available/lui-ssl.conf
Apache logs	/var/log/apache2

In general, changes to the functionality always require recompiling the static files from source and then redeploying the compiled application on the VM.

didmos Auth

Docker

didmos Auth consists of the following Docker container:

Container	Description
{project-name}-auth	Application
{project-name}-mongo	MongoDB Database

In the -auth container Auth is running as a mod_wsgi application inside an Apache webserver.

The -mongo container is running a MongoDB for storage of the OIDC OP (i.e. registered clients, tokens).

The logs can be accessed via docker logs (see list of general commands).

Configuration is possible via docker environment variables (for supported parameters). For a list of general environment variables refer to didmos2 Authenticator. An example for this configuration is described in section Provisioner - Docker.

VM based

didmos Auth is installed as a python virtual environment and deployed as a mod_wsgi app in Apache webserver. The following locations of the VM are used:

Component	Location in file system
Virtual environment (Python)	/opt/didmos2auth
Configuration	/etc/satosa
Logs	/var/log/satosa
Apache config (mod_wsgi integration)	/etc/apache2/sites-available/auth.conf
Apache logs	/var/log/apache2
MongoDB logs	/var/log/mongodb

Restarting didmos Auth is possible via the Apache webserver:

```
systemctl {start|stop|restart|status} apache2
```

The application is based on Satosa and most of the configuration in /etc/satosa follows the default Satosa configuration (see https://github.com/IdentityPython/SATOSA).

MongoDB is installed via the Ubuntu distribution during the initial Ansible setup (i.e. apt install `mongodb`).

The administration of MongoDB can be done using these commands:

```
systemctl {start|stop|restart|status} mongodb
```

Backups of MongoDB database

Persistent data from MongoDB should be backed up frequently, especially for the registered clients in the OIDC OP. Otherwise they have to be registered again in case of data loss.

Refer to https://docs.mongodb.com/manual/core/backups/ for general backup strategies.

Provisioner

Docker

didmos Provisioner consists of the following Docker containers:

Container	Description
didmos2-rabbitmq	RabbitMQ queue
{project-name}-ra	Requesting authority
{project-name}-xyz-worker	Worker nodes, possible multiple containers for each target system

The logs can be accessed via docker logs (see list of general commands).

The composition of the container and basic configuration is done in the file docker-compose.yml.

docker-compose.yml

```
ra:
    image: ${CUSTOMER_RA_IMAGE}:${CUSTOMER_RA_TAG}
    container_name: ${CUSTOMER}-ra
    depends_on:
      - rabbitmq
    volumes:
      - didmos2-ra-config:/opt/daasi/didmos2/ra/config/
    environment:
      LDAP_URL: ${LDAP_URL}
      RA_LDAP_ACCOUNT_PW: ${RA_LDAP_ACCOUNT_PW}
      RA_SENDER_EMAIL: ${RA_SENDER_EMAIL}
      RA_DEFAULT_RECEIVER_EMAIL: ${RA_DEFAULT_RECEIVER_EMAIL}
      INIT_PARAM: -i 2
      RABBITMQ_URL: ${RABBITMQ_URL}
      RABBITMQ_USER: ${RABBITMQ_USER}
      RABBITMQ_PW: ${RABBITMQ_PW}
      LDAP_ACCESSLOG_PW: ${LDAP_ACCESSLOG_PW}
      LDAP_MANAGER_PW: ${LDAP_MANAGER_PW}
      SMTP_HOST: ${SMTP_HOST}
      SMTP_USER: ${SMTP_USER}
      SMTP_PASSWORD: ${SMTP_PASSWORD}
      SMTP_PORT: ${SMTP_PORT}
    extra_hosts:
     - "host.docker.internal:host-gateway"
```

Additional configuration is possible via docker environment variables (for supported parameters). These variables are referenced in the docker-compose.yml and set in the .env file:

.env file

```
...
LDAP_ACCESSLOG_PW=...
LDAP_MANAGER_PW=...
LDAP_URL=ldap://host.docker.internal:389
LDAP_BIND_DN="cn=manager,dc=didmos,dc=de"

RA_LDAP_ACCOUNT_PW=...
RA_SENDER_EMAIL=noreply@daasi.de
RA_DEFAULT_RECEIVER_EMAIL=noreply@example.com

RABBITMQ_URL=rabbitmq
RABBITMQ_USER=admin
RABBITMQ_PW=...
RABBITMQ_PORT=5672
...
```

For a list of general environment variables refer to didmos2 Provisioner.

After changing one or more values in the .env file a recompose has to be performed by using this command:

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

A simple restart of the affected containers does not suffice.

The basic principle of docker-compose and docker environment variables applies to all docker deployments for didmos 2.

ETL

Docker

didmos ETL consists of the following Docker container:

- {project-name}-etl

The data and config is mounted as docker volumes from the host system like so (the variables are defined in .env):

```
    volumes:
      - /${ETL_DATA_DIR}/:/var/didmos/:rw
      - /${ETL_CONF_DIR}/:/etc/didmos/etl/:rw
```

Typically the data and config directories are set as following on the host system:

- ETL_DATA_DIR=/var/didmos/etl/etl-data
- ETL_CONF_DIR=/var/didmos/etl/etl-conf

The logs can be accessed via docker logs (see list of general commands).

Configuration is possible via docker environment variables (for supported parameters) but generally via files in the volume. An example for the configuration using docker environment variables is described in section Provisioner - Docker.