



# Components

Forterro Deutschland Abas GmbH

Date: 17.03.2025

# Table of Contents

1. Requirements	2
2. Basic installation of the components	3
2.1. Optional: Download and install components-v2-latest.tgz	3
2.2. ToDo: Unpack the components-<version>.tgz in the standard release of the Abas version	3
2.3. Structure of the components configuration	3
2.4. ToDo: Check of the COMPONENTS_VERSION used	5
2.5. ToDo: Create or adapt the configuration file	5
2.6. Requirements for the component configuration	6
2.7. ToDo: Create/customize .server.conf	7
2.8. ToDo: Install components (as the s3 user)	8
3. Encryption via SSL certificates	9
3.1. Requirements	9
3.2. Use of certificate chains	10
3.3. Configuration of the certificates in the components	11
4. Upgrade of components	12
4.1. Upgrade of an existing component installation	12
4.2. Upgrade the Abas installer modules to the components	13
5. Advanced configuration	17
5.1. Grafana for visualizing the component logs	17
5.2. Set up a dashboard user with credentials in Keycloak	17
5.3. Setting up an LDAP(S) configuration (optionally with Kerberos authentication) in Keycloak	21
5.4. Installation on distributed systems (from COMPONENTS_VERSION 1.0.1)	27
5.5. Installation of multiple REST API instances	28
5.6. SSL encryption of the full text search	31
5.7. Creating additional proxy rules	33
6. Temporary configuration docker containers	35
7. Additional components	36
7.1. BPM (with-bpm)	36
7.2. DMS-Rest-API and RabbitMQ for DMS (with-dms)	36
7.3. Prodaso Abas Connector (with-pac)	37
8. FAQ	39
8.1. How is the configuration structured?	39
8.2. When should which configuration file be adjusted?	39

Refers to component versions: 2.x.x

# 1. Requirements

- To download the components, there must be an internet connection available for the Abas server.
- You will need your Extranet credentials for the download.
- You will need a "Tenant ID" to install.
- Docker Compose standalone version 2.20 (and higher) and Docker Engine version 20.10 (and higher) are required for the installation.
- For the installation, a "docker login" to the artifact repository used must have been successful beforehand.
- From components.tgz version 1.0.1 an SSH key for the connection from the component host to the Abas host must be set up (even if this is the same host).
- Certificates must be available for the installation that must comply with some known [prerequisites](#).
- The use of IPv4 is a mandatory requirement for the components.

## 2. Basic installation of the components

### 2.1. Optional: Download and install components-v2-latest.tgz

Use your Extranet credentials for authentication.

Laden Sie das neueste Archiv herunter und entpacken Sie es.

```
cd $(dirname $HOMEDIR)

wget --user <user> --ask-password https://abasartifactory.jfrog.io/artifactory/abas.downloads-releases/erp/components/components-v2-latest.tgz

tar -xzf $(dirname $HOMEDIR)/components-v2-latest.tgz
mv components $(stat --printf='%Ucomponents' $HOMEDIR)
```

### 2.2. ToDo: Unpack the components-<version>.tgz in the standard release of the Abas version



By default, the components.tgz that matches the Abas version is located in the \$HOMEDIR/archives directory. The COMPONENTS\_VERSION entry in templates/common-default.env indicates the version.

Entpacken Sie das Archiv z.B. parallel zur Abas Installation, z.B. mit dem Benutzernamen als Präfix dem das \$HOMEDIR gehört.

```
cd $(dirname $HOMEDIR)
tar -xzf $HOMEDIR/archives/components-<version>.tgz
mv components $(stat --printf='%Ucomponents' $HOMEDIR)
```

Während der Installation wird ein SSH-Zugriff vom Ort des components Verzeichnis auf das abas-HOMEDIR benötigt.

```
|---<abas-HOMEDIR> (gehört hier s3)
|      |-- <Mandant>
|      |-- <Mandant>
|---<s3components>
|      |-- bin
|      |-- ...
```

### 2.3. Structure of the components configuration

```
|---<s3components>
|      |-- components-installer-default.env
|      |-- components-installer-custom.env
```

```

|-- config
|   |-- rest-api-ssl-NAME.env.template
|   |-- rest-api-NAME.env.template
|   |-- vts-ssl-proxy-NAME.env.template
|-- templates
|   |-- common-default.env
|   |-- common-custom.env
|   |-- docker-compose.yml
|   |-- .env
|   |-- core
|       |-- <Verzeichnis je standard Komponente>
|           |-- service-default.env
|           |-- service-custom.env
|           |-- docker-compose.yml
|           |-- ...
|-- dynamic
|   |-- <Verzeichnis je individueller Komponente>
|       |-- service-default.env
|       |-- service-custom.env
|       |-- docker-compose.yml
|       |-- ...
|-- bin
|   |-- components_installer.sh
|-- installation (wird durch components_installer.sh generiert)

```

### s3components/components-installer-default.env

Defines values for environment variables of the component configuration used by components\_installer.sh.

### s3components/components-installer-custom.env

Allows values of 'components-installer-default.env' to be overwritten, is taken into account by components\_installer.sh.



These files will be your personal entry point for the configuration. View the 'default' values and configure in 'custom'. Because only 'custom' files are not overwritten when upgrading to a new component version and your customizations are retained.

### s3components/config

Contains templates for creating additional REST-API instances (see [Installing multiple REST-API instances](#)) and VTS-SSL proxies (see [SSL encryption of the full-text search \(VTS\)](#)).

### s3components/templates

Contains templates of the component configuration to be filled by components\_installer.sh.

### s3components/templates/common-default.env

Defines environment variables that are relevant for all components.

### s3components/templates/common-custom.env

Empty, it allows values of the 'common-default.env' to be overwritten.

### **s3components/templates/.env**

Defines environment variables that are used by docker-compose to validate the docker-compose.yml.

### **s3components/templates/core**

Contains templates for creating the standard components.

### **s3components/templates/dynamic**

Contains templates for creating individual components.

### **s3components/templates/(core|dynamic)/<component>/service-default.env**

One for each component with individually required environment variables.

### **s3components/templates/(core|dynamic)/<component>/service-custom.env**

One for each component, which is provided empty and allows values of the 'service-default.env' to be overwritten.

### **s3components/bin/components\_installer.sh**

Copies all files from 's3components/templates' to 's3components/installation'.

Sets values for environment variables in component configuration.

The values come from 'components-installer-default.env' and 'components-installer-custom.env'.



Existing 'custom.env' files in 'installation' are not overwritten.



Do not configure in the templates provided by abas, but either in the 'components-installer-custom.env' or in the 'service-custom.env' or 'common-custom.env' **files generated after 'installation'**. The latter also offer you the option of adding environment variables for components that have not yet become standard components.

## **2.4. ToDo: Check of the COMPONENTS\_VERSION used**

After you have unpacked the components.tgz, first check which component version is concerned. To do this, open the **common-default.env** file in the s3components/templates directory and search for the entry COMPONENTS\_VERSION.

## **2.5. ToDo: Create or adapt the configuration file**

As of COMPONENTS\_VERSION 2.x, the configuration files are located in the **s3components/templates** directory. These configuration files are templates that are filled with information from s3components/components-installer-default.env and s3components/components-installer-custom.env.

Please check the configuration of the **s3components/components-installer-default.env** file and customize your configuration in the **s3components/components-installer-custom.env** file, as it will not be overwritten during an upgrade.

You can find out which information is absolutely necessary step by step by running the installation script (bin/components\_installer.sh) and by reading the following documentation.

## 2.6. Requirements for the component configuration

The variables listed are **mandatory variables** and must be set before starting.

*The following is a description of the mandatory variables:*

### HOST\_NAME (and HOST\_DOMAIN)

Host name or IP address of the host - the certificates must be issued for these, i.e. the combination of HOST\_NAME + HOST\_DOMAIN must be entered in the certificate as the Subject Alternative Name. Optionally, the **HOST\_DOMAIN** variable can be defined as a suffix for the HOST\_NAME variable. This means, for example, that system\_a.my-domain will be used from HOST\_NAME=system\_a and HOST\_DOMAIN=.my-domain for all components when starting.

### CUSTOMER\_ROOT\_CA\_CERTIFICATE

Full path of your root CA certificate or an intermediate certificate.

**Important:** The components require read access to the file! Some of the components will trust this certificate and therefore implicitly also the CUSTOMER\_ISSUED\_CERTIFICATE, as it is stored in their trust store.

### CUSTOMER\_ISSUED\_CERTIFICATE

Full path to a certificate signed with your root CA certificate. Keycloak and nginx (web server) will authenticate themselves with this certificate. All other components must be able to validate this using their trust store (in which the CUSTOMER\_ROOT\_CA\_CERTIFICATE is imported) or the CUSTOMER\_CA\_BUNDLE\_PEM.

**Important:** The components require read access to the file!

### CUSTOMER\_ISSUED\_CERTIFICATE\_PRIVATE\_KEY

Full path to the private key of your signed certificate.

**Important:** The components require read access to the file!

### CUSTOMER\_CA\_BUNDLE\_PEM

Full path to a file that contains a **complete** certificate chain of Base64 certificates. This chain must consist, from top to bottom, of the **signed certificate, 0 to n intermediate certificates and finally the root CA certificate**

So copy the contents of all relevant certificates into this file and read [the notes on CUSTOMER\\_CA\\_BUNDLE\\_PEM](#) again if you are unsure.

**Important:** The components require read access to the file!

### B64\_TENANT

Your tenant ID.

### ABAS\_PASSWORD

A password of an admin user in Abas using which the JWT component can login

### ABAS\_SSH\_USER

SSH user on the Abas host. Can alternatively be passed to the installation script as --abas-ssh-user parameter.

### ABAS\_SSH\_HOST

Hostname of the Abas host. Can alternatively be passed to the installation script as --abas-ssh-host parameter.



## ABAS\_SSH\_PORT

Port of the Abas host. Can alternatively be passed to the installation script as `--abas-ssh-port` parameter.

The listed variables are **optional variables** and can be set before starting.

You can find a description of important optional variables below:

## COMPOSE\_PROFILES

Docker Compose standalone standard environment variable through which you can specify profile names separated by a comma.

The profiles for services are predefined in `docker-compose.yml`.

## ABAS\_HOMEDIR

HOMEDIR on the Abas server.

If the variable is not set, the attempt is made to determine the HOMEDIR when starting the component.

Should you have problems determining the HOMEDIR, you can also pass the path using the `-h` option when calling the installer.

## HELP\_HOSTNAME

Host name of the server on which the Online Help is installed.

Is required for the configuration of the web server container if you want to use the Online Help.

## 2.7. ToDo: Create/customize `.server.conf`

To access a client via REST API, you must create a `.server.conf` file either for each client or globally in `$HOMEDIR`. In addition to connection information for the `license_controller` and `jwt_auth_userinfo` components, this configuration file also contains your installation number.

*Inhalt der `.server.conf`*

```
[LicenseController]
url = http://<host_fqdn>:<license_controller_port>
installation = <installation_number>

[SSO]
server = http://<host_fqdn>:<jwt_port>
verify_peer = false
```



The placeholders `<host_fqdn>`, `<license_controller_port>`, `<installation_number>` and `<jwt_port>` must be replaced with your values. The standard ports of the components can be found in the file `s3components/components-installer-default.env`.

*Nutzung des Installations-Skriptes*

```
cd s3components
bin/components_installer.sh -r -h <abas-homedir>
```

With `-r`, queries for generating configuration files are activated in the `installation/rest-api/abasconfig` directory. This makes a client available via the Rest API.

With `-ra` (or `-r -a`) you can make all clients available via the Rest-API without manual queries.

With `-h` you can specify the Abas homedir if it cannot be determined automatically.

If you do not want to check the contents of the `.server.conf` files in the `ABAS_HOMEDIR` or the mandates, you can use the `--skip-server-conf` option.

Use `--help` to get an overview of all options.

## 2.8. ToDo: Install components (as the s3 user)

Um die Docker-Images der Komponenten herunterzuladen, führen Sie zunächst einen Docker-Login an der Abas Docker-Registry aus:

```
docker login abasartifactory.jfrog.io
```

After successful installation (execution of `bin/components_installer.sh` to generate the configuration under `s3components/installation` based on `s3components/templates`), you can use the components with `docker-compose` commands.

Wechseln Sie in das generierte Installationsverzeichnis:

```
cd $HOMEDIR/s3components/installation
```

Starten Sie die Komponenten z.B. mit folgendem Befehl:

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

Starten Sie nur eine Komponenten gezielt (hierbei gibt es definierte Abhängigkeiten in der `s3components/installation/docker-compose.yml`) z.B. mit folgendem Befehl:

```
docker-compose up -d <docker-service-name>
```

Stoppen Sie die Komponenten und löschen dabei named Volumes z.B. mit folgendem Befehl:

```
docker-compose down -v
```

For more information on using `docker-compose`, read the official documentation: <https://docs.docker.com/compose/reference/overview/>

## 3. Encryption via SSL certificates

One aim when implementing the components was to encrypt the communication. SSL certificates are used for this purpose.

This chapter describes what is required for this and what to look out for in the certificates.



Some components are programmed in Java and use the standard Java validation for SSL certificates. In some cases, this is stricter than validation in a web browser or from `openssl`, for example.

It is therefore important to ensure that the certificates are issued correctly, as otherwise connection errors may occur between the components internally and other systems.

### 3.1. Requirements

The following table describes which files are required and which requirements apply to them:

Certificate/Key	Requirements
Root or Intermediate CA certificate	<ul style="list-style-type: none"> <li>X509v3 Basic Constraints extension must contain <code>CA:TRUE</code> (see <a href="https://www.rfc-editor.org/rfc/rfc5280#section-4.2.1.9">https://www.rfc-editor.org/rfc/rfc5280#section-4.2.1.9</a>)</li> <li>File must be readable for all users</li> </ul>
Signed certificate	<ul style="list-style-type: none"> <li>The CN or the SAN must contain the <code>HOST_NAME</code> and the <code>HOST_DOMAIN</code> from the components configuration</li> <li>File must be readable for all users</li> </ul>
Key for the signed certificate	<ul style="list-style-type: none"> <li>File must be readable for all users</li> </ul>
Intermediate certificate(s) (optional)	<ul style="list-style-type: none"> <li>If the intermediate certificates are to be used, a certificate bundle and/or a PEM file must be created</li> </ul>
Certificate bundle (optional)	<ul style="list-style-type: none"> <li>Should contain all certificates in the following order: <ul style="list-style-type: none"> <li>Signed certificate</li> <li>If available all Intermediate certificates</li> <li>Intermediate CA certificate if available</li> <li>Root CA certificate</li> </ul> </li> <li>File must be readable for all users</li> </ul>
PEM file (optional)	<ul style="list-style-type: none"> <li>Should contain all certificates in the following order: <ul style="list-style-type: none"> <li>Signed certificate</li> <li>If available all Intermediate certificates</li> <li>Intermediate CA certificate if available</li> <li>Root CA certificate</li> </ul> </li> <li>File must be readable for all users</li> </ul>

### 3.1.1. FQDN as CN or SAN in the signed certificate

When establishing an SSL connection, the client checks whether the server name in the signed certificate matches the server name to which it wants to connect. The connection is only established if this is the case.

The `HOST_NAME` and the `HOST_DOMAIN` of the server are defined in the configuration of the components. The components communicate with each other via the Fully Qualified Domain Name (FQDN for short), i.e. the combination of host name and domain.

The signed certificate must therefore contain the FQDN. This can be done either via the **Common Name (CN)** or via the **Subject Alternative Names (SAN)**.

Sample content of the SAN property of a signed certificate:

```
*Auszug Komponenten Konfiguration*
HOST_NAME=customer
HOST_DOMAIN=.customer-domain

*Auszug openssl x509 -in <signiertes Zertifikat> --text*
X509v3 Subject Alternative Name:
    DNS:customer, DNS:customer.customer-domain
```

Entries in the SAN property of an SSL certificate can be defined as a comma-separated list `DNS:<hostname>`, `IP:<ip-address>` and definitions can start with `<email:>`, `<URI:>`, `<DNS:>`, `<RID:>`, `<IP:>`, `<dirName:>`, `<otherName:>`.



If you use SAN entries in your certificate, make sure that they are valid (see <https://datatracker.ietf.org/doc/html/rfc5280#section-4.2.1.6>).

If the SAN extension contains the following entries, for example: `DNS:test.com,URI:test.com`, the entry `URI:test.com` is considered invalid because the schema (e.g., `https://`) is missing (according to RFC5280, a URI entry **must** begin with a schema).

In Java applications, this means that the entire SAN extension (i.e. **all** SAN entries) is considered invalid and the connection to other systems may fail.

## 3.2. Use of certificate chains

In order for the applications to be able to verify themselves across the entire certificate chain, all certificates must be combined in one file.

The order of the certificates in the certificate chain should be as follows:

1. Signed certificate
2. If available, all Intermediate certificates
3. If available Intermediate CA certificate
4. Root CA certificate

A certificate chain can have the file extensions `.crt` or `.pem`, for example.

In the following example, a certificate bundle and a PEM file are created:

```
# Erstellung eines Zertifikats-Bundles
cat <signiertes Zertifikat>.crt <intermediate Zertifikate>.crt <Root CA Zertifikat>.crt >> cert-
bundle.crt

# Erstellung einer PEM-Datei
cat <signiertes Zertifikat>.crt <intermediate Zertifikate>.crt <Root CA Zertifikat>.crt >> cert-
bundle.pem
```



In this example, the certificate bundle and the PEM file have the same content. However, the PEM file may also contain private keys, e.g., which are not typically contained in CRT files.

### 3.3. Configuration of the certificates in the components

The component configuration expects 4 certificate files [compare configuration parameters](#):

```
CUSTOMER_ROOT_CA_CERTIFICATE=<Pfad zum Root oder Intermediate CA Zertifikat>
CUSTOMER_ISSUED_CERTIFICATE=<Pfad zum signierten Zertifikat, zum Zertifikats-Bundle oder zur PEM-
Datei>
CUSTOMER_CA_BUNDLE_PEM=<Pfad zur PEM-Datei>
CUSTOMER_ISSUED_CERTIFICATE_PRIVATE_KEY=<Pfad zum Key des signierten Zertifikats>
```

The variable **CUSTOMER\_ISSUED\_CERTIFICATE** can contain only the signed certificate as well as a certificate chain.

## 4. Upgrade of components

### 4.1. Upgrade of an existing component installation

If you are still using the Abas Installer for ERP versions prior to Abas 2024.Q3, skip this subchapter and go to [Upgrade the Abas Installer modules to the components](#)



As of COMPONENTS\_VERSION 1.2.5 it's possible to automatically update the components to the latest version.

This feature was deactivated with the release of COMPONENTS\_VERSION 1.4.1 and no longer works.

Instead, you can now explicitly specify the version you wish to upgrade to.



If a COMPONENTS\_VERSION 1.2.9 or lower is currently installed and you would like to upgrade to COMPONENTS\_VERSION 2.0.0 or higher, for example, it is necessary to perform an intermediate upgrade to version 1.3.0.

Should you accidentally start the upgrade to version 2.0.0 before the interim upgrade, the upgrade script will display a corresponding message.

There are two ways to upgrade an existing component installation:

1. Automatic upgrade to a specific version directly from the current installation directory (from COMPONENTS\_VERSION **1.4.1**, see common-default.env)
2. Manual upgrade to any released version from the components directory of the newly downloaded version

In both cases, the script `<components-verzeichnis>/bin/components_upgrade.sh` is used.



When upgrading from COMPONENTS\_VERSION 1.x to COMPONENTS\_VERSION 2.x, the execution location of the components changes from `s3components` to `s3components/installation`. Therefore, data from the `s3components_backup` directory created during the upgrade must be manually transferred to the new installation directory.

#### 4.1.1. Automatic upgrade to a specific version (as of COMPONENTS\_VERSION 1.4.1)

To upgrade to a specific COMPONENTS\_VERSION, you can call the following script directly in the component directory currently in use as of COMPONENTS\_VERSION **1.4.1**:

```
bin/components_upgrade.sh --version <version>
```

Replace `<version>` with the version you want to use. For an overview of the available versions, please visit <https://abasartifactory.jfrog.io/artifactory/abas.downloads-releases/erp/components/> (login required). You only need to specify the version, e.g., `bin/components_upgrade.sh --version 2.0.1`.

In this case, the script downloads the `components-<version>.tgz`, considers the current directory as `--old-components-dir` and overwrites the standard release contents (no custom files).

## 4.1.2. Manual upgrade to any version

To upgrade between component versions to any released version, the following steps are necessary:

1. **Optional:** Download the `components-<version>.tgz`
2. **ToDo:** Unpack the `components-<version>.tgz`
3. Change to the newly unpacked component directory and call the upgrade script, specifying the old component directory:

*COMPONENTS\_VERSION 1.x*

```
bin/components_upgrade --old-components-dir <zu-aktualisierendes-Verzeichnis>
```

*ab COMPONENTS\_VERSION 2.x*

```
bin/components_upgrade --upgrade-dir <zu-aktualisierendes-Verzeichnis>
```

In this case, the script overwrites the "old" directory (no custom files) with the contents from the already downloaded `components-<version>.tgz`.

## 4.2. Upgrade the Abas installer modules to the components

The steps listed in this subchapter are only necessary if:

1. you have an Abas version prior to Abas 2024.Q3.
2. you have installed the Abas installer modules so far.

### 4.2.1. ToDo: Stop the Abas installer modules

Before you install the components, first stop all Abas Installer modules. To do this, you can use, for example, the following command:

```
/usr/local/bin/abas-installer services --stop -m <modul_verzeichnis>
```

### 4.2.2. ToDo: Adopt the data of the Abas Installer modules

To transfer the data of the Abas Installer modules to the components, proceed as follows:

1. **Optional:** Download the `components-<version>.tgz`
2. **ToDo:** Unpack the `components-<version>.tgz`



The data from the Keycloak module cannot be adopted and must be created again in the new Keycloak component after installation.

## Transfer of the REST API configuration files

1. Copy the REST API configuration files from `${BASE_MANDANTDIR}/modules/rest-api/abasconfig/` to `${BASE_MANDANTDIR}/components/installation/rest-api/abasconfig/`.

## Transfer of the dashboards

To transfer your dashboards from the Abas Installer modules to the components, the following requirements must be met:

1. The old MongoDB container of the Abas Installer modules must be started.
2. The new MongoDB container of the components must be started. If this is not yet the case, continue with chapter [ToDo: Create or adapt the configuration file](#).
3. No dashboards have yet been created in the component installation.

## Dump the database and copy it to the local system

First, it is necessary to export the database from the old MongoDB container of the Abas Installer modules:

```
# Dump der Datenbank erstellen und nach /tmp/dashboard speichern
docker exec -it abas_mongodb-abas_mongodb-1 mongodump -u <admin-user> -p <admin-password>
--authenticationDatabase admin -d dashboard -o /tmp/dashboard

# Datenbank-Dump auf lokales Dateisystem kopieren
docker cp abas_mongodb-abas_mongodb-1:/tmp/dashboard /tmp
```

## Rename the collection



Always check whether the name of the exported collection contains the correct tenant. Renaming may also be necessary if the Base64 string has not changed.

In order for the data to be displayed correctly in the new dashboard, the collection may have to be renamed. The collection name is composed as follows: `dashboards.v3.<tenant>`.

To do this, first check your current collection name:

```
# Dateinamen prüfen
cd /tmp/dashboard/dashboard
ls -l
-rw-r--r-- 1 s3 abas 52516 Aug  6 08:09 dashboards.v3.on_premise.bson
-rw-r--r-- 1 s3 abas  191 Aug  6 08:09 dashboards.v3.on_premise.metadata.json
```

## Inhalt der `dashboards.v3.on_premise.metadata.json`

```
{
  "indexes": [
    {
      "v": {
```



```

    "$numberInt": "2"
  },
  "key": {
    "_id": {
      "$numberInt": "1"
    }
  },
  "name": "_id_"
}
],
"uuid": "28a4527c9b494340ad1e53aa51fe0657",
"collectionName": "dashboards.v3.on_premise",
"type": "collection"
}

```

As can be seen here, the tenant name **on\_premise** was used during the module installation (occurs in file names and in line 16 in metadata.json).

Depending on the installation, it is possible that the wrong tenant name was used in the Abas Installer modules. To find out your tenant name, you can decode the B64\_TENANT. To do this, use the command `echo -n "<base64-string>" | base64 -d`. The output should look like this:

#### Dekodierter Inhalt des B64\_TENANT

```

{
  "tenant": "test",
  "tenant_id": "*****_****_****_****_*****",
  "domain": "*****",
  "aws_access_key_id": "*****",
  "aws_secret_access_key": "*****",
  "aws_region": "eu-central-1",
  "stage": "*****"
}

```

In this case, the tenant name would be **test** (line 2). You can now rename the collection using the following commands:

```

# Dateien umbenennen
mv dashboards.v3.on_premise.bson dashboards.v3.test.bson
mv dashboards.v3.on_premise.metadata.json dashboards.v3.test.metadata.json

# collectionName in metadata.json umbenennen
sed -i 's/dashboards.v3.on_premise/dashboards.v3.test/g' dashboards.v3.test.metadata.json

```

#### Transfer data to new MongoDB container

You can now copy the database dump into the new MongoDB container and import it into the database:

```

# Datenbank-Dump in neuen MongoDB-Container kopieren

```

```
docker cp /tmp/dashboard s3components-dashboard-mongodb-1:/tmp
```

```
# Datenbank-Dump in MongoDB importieren
```

```
docker exec -it s3components-dashboard-mongodb-1 mongorestore -u <admin-user> -p <admin-password>  
--authenticationDatabase admin --drop /tmp/dashboard
```

If you now call "My dashboards" in the dashboard web interface, you should be able to see your old dashboards.

## 5. Advanced configuration

### 5.1. Grafana for visualizing the component logs

#### 5.1.1. Installation

components.tgz in the /bin directory contains a script that enables the installation of Grafana, Promtail and Loki.

```
grafana.sh
```

Promtail transmits the components metrics to Loki (centralized logging and log analysis), which in turn are visualized by Grafana.

Call the script and follow the instructions.

First, a configuration is required, which you create using the following caller:

```
grafana.sh --config
```

You can then start the installation:

```
grafana.sh --start
```

The three containers can be stopped with the following command:

```
grafana.sh --stop
```

#### 5.1.2. Setting up a dashboard in Grafana

To do this, call the URL [https://<host\\_fqdn>:3000](https://<host_fqdn>:3000) and log in with the standard credentials (admin:admin). You will be prompted to set a new password.

You can add data sources under "Connections", "Data sources". You should see an existing data source here, of type Loki with URL <http://loki:3100>.

You can create a new dashboard under "Dashboards". Select "Add new panel" and "Query" to create a query. Select the data source "Loki" and enter a query, e.g., `{project="components"}`. You can change the type of panel for visualization, e.g., to "Logs".

Detailed instructions for setting up Grafana can be found in the official [documentation](#).

### 5.2. Set up a dashboard user with credentials in Keycloak

### 5.2.1. Requirement for functionality

Before installing the components, you may have already configured an SSO entry in a `.server.conf` file and set the `ABAS_PASSWORD` in `components/common-custom.env`. If not, you will find details about this in the [Adjust component configuration](#) section.

### 5.2.2. Setup

[Users](#) > `erp_dashboard_user`

## Erp\_dashboard\_user

- Details
- Attributes
- Credentials
- Role Mappings
- Groups
- Consents
- Sessions





ID	47d42f5a-34d5-47d3-b426-173596808eee
Created At	11/29/23 1:50:59 PM
Username	erp_dashboard_user
Email	erp_dashboard_user@abas.de
First Name	
Last Name	
User Enabled 	<input checked="" type="checkbox"/> ON
Email Verified 	<input type="checkbox"/> OFF
Required User Actions 	Select an action...
Impersonate user 	<input type="button" value="Impersonate"/>
	<input type="button" value="Save"/> <input type="button" value="Cancel"/>

Figure 1. Let's set up a user in the Abas Realm in Keycloak with which we can view Abas data in the Dashboard.

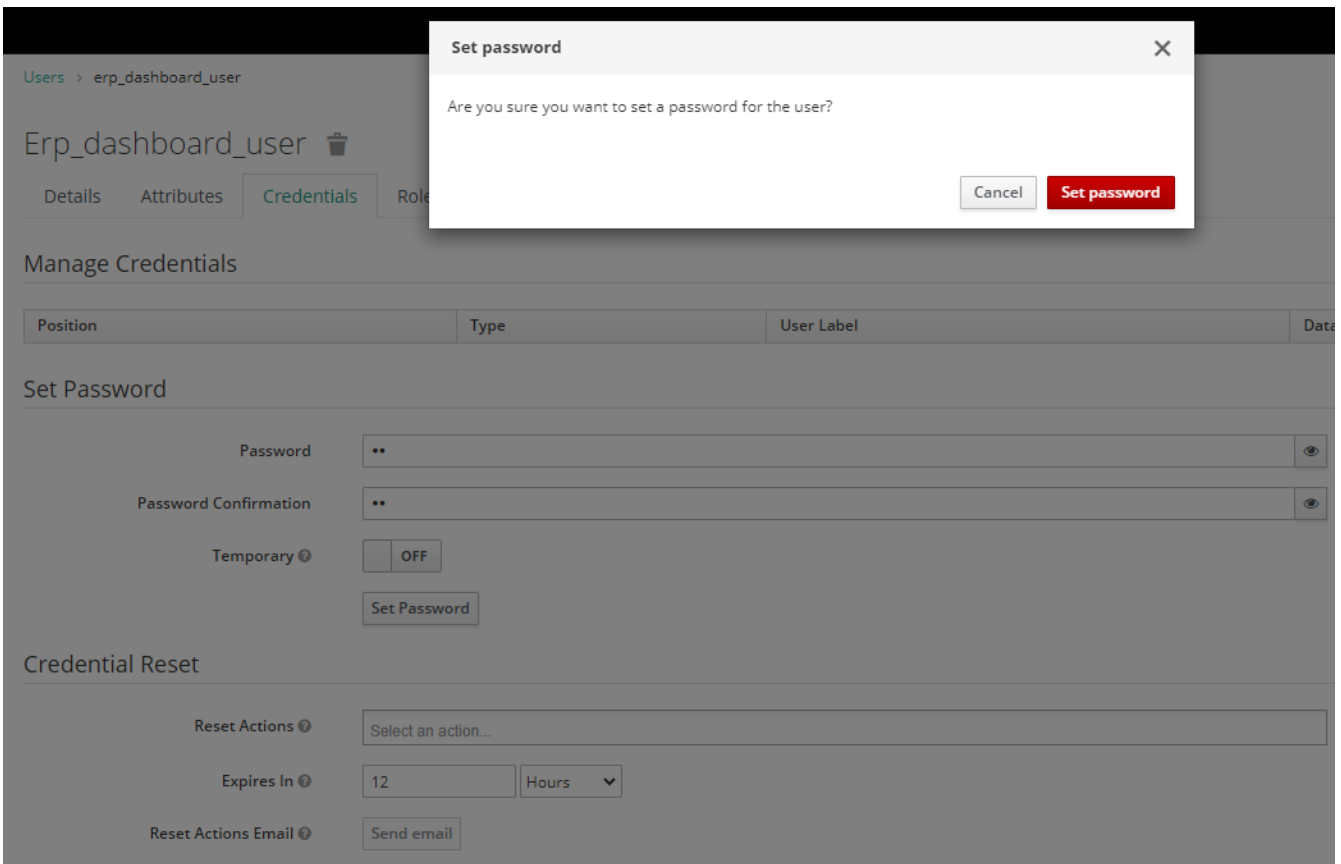


Figure 2. In the next step, assign the credentials (username and password) that you want to use when logging in to the Dashboard.



Pay particular attention to the email address! This must later match the email address of the user authorized for the desired use case in Abas.

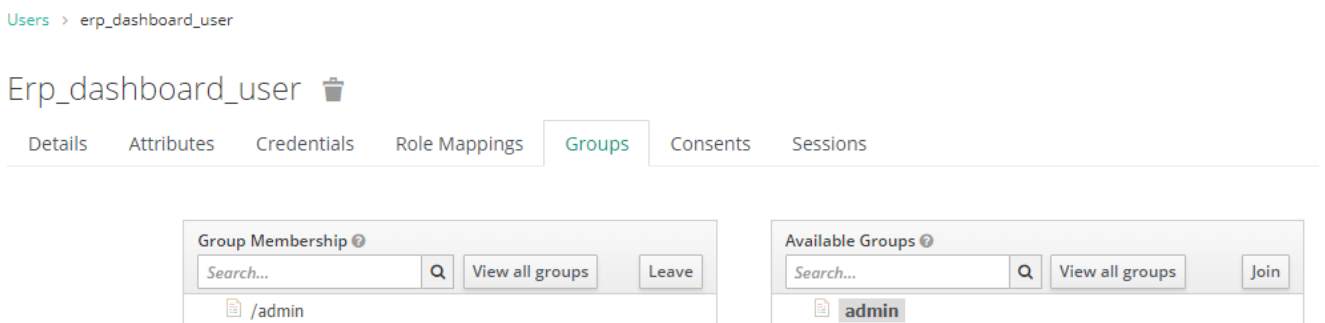


Figure 3. You then assign the user a corresponding group in Keycloak.

Passwortdefinitionen - zeigen [524 PASS erp\_dashboard\_user]

Datei Bearbeiten Ausführen Tabelle Kommando Fenster Info Hilfe

Neues Passwort Erlaubnisse zeigen

Allgemeines Kommandos GUI Zugangsrechte

Identnummer 524 Suchwort PASS

Bearbeiterzeichen erp\_dashbaord Abas ERP Login

DMS-Benutzername Systemlogin

SSO-Login  erp\_dashboard\_user@abas.de

Name erp\_dashboard\_user Editieren

E-Mail erp\_dashboard\_user@abas.de

Externer Username

Abteilung Mitarbeiter

**EINSTELLUNGEN**

Hilfesprache automatisch  Inaktiv

Bediensprache Deutsch  Fremd nutzbar

Druckeinstellungen  Matchcode-Erkennung

JFOP-Server-Konfiguration  Objektauswahl anpassbar

Treffer der Objektauswahl dürfen kopiert werden

Figure 4. In Abas, log in as an admin user and set up a user in a password definition that matches the email address stored in Keycloak, who in this case has the necessary access rights for Dashboard.

### 5.2.3. Optional: addKeycloakUser.sh administration script

After the installation you can create the necessary users for Keycloak and expand the Abas "admin" user.



**These adjustments function after a new installation. For an upgrade there probably isn't an "admin" user, or it has a changed password. The Keycloak users might also have to reference other users for an upgrade. In this case you'll need to use the script with additional options or create the user manually!**

#### Usage vom Skript addKeycloakUser.sh

```
addKeycloakUser.sh -a -k
  -a: Add abas-ERP user (default: admin:admin with ID 27)
  -k: Add Keycloak user (default: admin:admin)

The default users can be overwritten with the following variables.
===== ATTENTION! =====
=== Only change these values if you know exactly what you are doing! ===
ERP_PASSWD_ID: ID of the abas-ERP password dataset (default: 27)
USERNAME:      abas-ERP username of the erp-Dataset
PASSWORD:      abas-ERP password of the erp-Dataset
EMAIL:         abas-ERP EMail of the erp-Dataset

KEYCLOAK_CREDENTIALS_ADMIN_USERNAME: Admin username (default: admin)
KEYCLOAK_CREDENTIALS_ADMIN_PASSWORD: Admin password (default: admin)
KEYCLOAK_PORT_HTTP: The keycloak port (default 8087)
```

#### Keycloak-Benutzer und abas ERP Benutzer anlegen

```
addKeycloakUser.sh -a -k
=== Add keycloak user
Generate token ...
Add user admin ...
Get user ID for admin ...
Get group ID ...
Add group to keycloak ...
Keycloak user successfully added!
=== Change abas ERP admin-user for dashboard
Update admin for client /mnt/abas/erp ... ok
```

## 5.3. Setting up an LDAP(S) configuration (optionally with Kerberos authentication) in Keycloak



The components must already have been installed.

You can also read about the topic in the official [documentation](#).

To set up an LDAP provider in Keycloak, the Keycloak provided by Abas contains a configuration script that can be called manually from elsewhere. The script for the call can be found in the directory of the components under `bin/call_ldap_user_provider.sh`.

Three configurations of LDAP are currently possible:

- [LDAP](#)
- [LDAPS](#)
- [LDAP\(S\) with Kerberos authentication](#)

### 5.3.1. LDAP setup

Perform the following steps to set up LDAP in Keycloak:

1. [Configuration via ENV file](#)
2. [Load configuration to Keycloak](#)

The two steps are described in the following two sub-chapters.

#### Configuration via ENV file

To configure LDAP, customize the file `s3components/installation/keycloak/service-ldap-custom.env`.

Below you will find a list of the available variables with a description:

#### TEST\_LDAP\_CONNECTION

Should a connection test be performed with the configured LDAP connection data before the action is executed? (Default is `true`)

## TEST\_LDAP\_AUTHENTICATION

Should an authentication test be performed with the configured LDAP connection data before the action is executed? (Default is **true**)



To test the connection and authentication, the variables **CONNECTION\_URL**, **BIND\_DN** and **BIND\_CREDENTIAL** must be set.

## LDAP\_CONFIGURATION\_ACTION

What action should be performed in the configuration script while starting Keycloak? Possible options are **create** (does not change anything if it already exists), **delete** (deletes a configuration found under the **USER\_PROVIDER\_LDAP\_ID** and does not create a new one) and **update** (calls **delete** and **create** in one step)

## TRIGGER\_FULL\_SYNC

Should a full sync be triggered at the end of the LDAP configuration script if there is an LDAP configuration created at the time? (Default is **true**)

## CONNECTION\_URL

Connection URL to your LDAP server. Usually in the following format: **ldaps://LDAP\_HOST:636**

## BIND\_DN

DN of the LDAP administrator used by Keycloak to access the LDAP server.

Example: **CN=Administrator,CN=Benutzer,DC=demo,DC=example,DC=com**

Group example: **OU=test,DC=test,DC=datical,DC=net**

## BIND\_CREDENTIAL

LDAP administrator password

## ENABLED

If you disable this user merging provider, it will not be included in queries and imported users will be disabled and read-only until the provider is re-enabled.

Either **true** or **false**

## USER\_PROVIDER\_LDAP\_ID

ID for the provider to be created/updated. For example: **abas-ldapThe**



variable **USER\_PROVIDER\_LDAP\_ID** must contain a value.

## VENDOR

The Vendor configuration parameter in Keycloak's LDAP component specifies the type of LDAP server used. Aside from the general **other** category, specific provider options may include:

**ad** for Microsoft Active Directory.

**edirectory** for Novell eDirectory.

**rhds** for Red Hat Directory Server.

**opendj** for OpenDJ.

**openldap** for OpenLDAP.



**fedora** for Fedora Directory Server.

**tivoli** for IBM Tivoli Directory Server.

**apacheds** for Apache Directory Server.

These values may vary depending on the specific version of Keycloak and the supported LDAP provider. The exact list of supported providers and their respective values can be found in the official documentation or in the corresponding release notes.

## EDIT\_MODE

Configuring the edit mode on the LDAP configuration page defines the user's LDAP update permissions.

### READ\_ONLY

You cannot change the user name, email, first name, last name and other assigned attributes. Keycloak will display an error if a user attempts to update these fields. Password updates are not supported.

### WRITABLE

You can change the user name, email, first name, last name and other assigned attributes and passwords and synchronize them automatically with the LDAP storage.

### UNSYNCED

Keycloak saves changes to user name, email, first name, last name and passwords in the local Keycloak storage so that the administrator has to synchronize this data with LDAP again. In this mode, Keycloak deployments can update user metadata on read-only LDAP servers. This option also applies to the import of users from LDAP into the local Keycloak user database.

## AUTH\_TYPE

Type of authentication method used in the LDAP binding process. It is used in most requests sent to the LDAP server. Currently, only the mechanisms **none** (anonymous LDAP authentication) or **simple** (authentication with login information and password) are available.

## USERS\_DN

Complete DN of the LDAP tree in which your users are located. This DN is the superordinate DN of the LDAP user. For example, it could be **ou=users,dc=example,dc=com**, assuming that your typical user has a DN like **uid='john',ou=users,dc=example,dc=com**.

## USERNAME\_LDAP\_ATTRIBUTE

Name of the LDAP attribute that is mapped as the keycloak user name. For many LDAP server providers, this can be **uid**. For Active Directory it can be **sAMAccountName** or **cn**. The attribute should be filled in for all LDAP user records that you want to import from LDAP to Keycloak.

## RDN\_LDAP\_ATTRIBUTE

Name of the LDAP attribute that is used as the RDN (top attribute) of the typical user DN. Normally it is the same as the LDAP attribute user name, but it is not required. For example, it is common for Active Directory to use **cn** as the RDN attribute when the username attribute might be **sAMAccountName**.

## UUID\_LDAP\_ATTRIBUTE

Name of the LDAP attribute that is used as a unique object identifier (UUID) for objects in LDAP. For many LDAP server providers this is **entryUUID**, but for some it is different. For Active Directory, for example, it should be **objectGUID**. If your LDAP server does not support the concept of UUID, you can use any other attribute that should be unique for LDAP users in the tree. For example **uid** or **entryDN**.

## USER\_OBJECT\_CLASSES

All values of the LDAP objectClass attribute for users in LDAP, separated by commas. For example:

`inetOrgPerson`, `organizationalPerson`. Newly created Keycloak users are written to LDAP with all these object classes and existing LDAP user entries are only found if they contain all these object classes.

### CUSTOM\_USER\_SEARCH\_FILTER

Used to filter the complete list of users and groups in the "User DN" node to the users and groups you want to import into Keycloak. Leave this field empty if you do not require an additional filter.

Can use a filter like `(mail=*)` to include only users with an email address (excludes users with a work account). You can filter by groups or other criteria

### SEARCH\_SCOPE

If the node listed under "User DN" contains nested nodes with users, select `subtree`. Otherwise select `one level`.

Set `SEARCH_SCOPE=2` for `subtree` and `SEARCH_SCOPE=1` for `one level`.

### FULL\_SYNC\_PERIOD

Period for full synchronization in seconds. Default `604800`

### CHANGED\_SYNC\_PERIOD

Period for the synchronization of changed or newly created LDAP users in seconds. Default `86400`

### BATCH\_SIZE\_FOR\_SYNC

Number of LDAP users imported from LDAP to Keycloak in one transaction. Default `200`

### EXTRA\_LDAP\_CONFIG\_ARGS

Additional arguments can be added in the form:

```
-s 'config.<configName>=["<value>"]'
```

For example, debug options can be activated with:

```
-s 'config.syncRegistrations=["false"]' -s 'config.debug=["true"]'
```

## Load configuration to Keycloak



The keycloak container must already be running.

After you have made the configuration in the `service-ldap-custom.env`, you can execute the configuration script `bin/call_ldap_user_provider.sh`. This script loads configuration to Keycloak and sets up the LDAP provider.

### 5.3.2. LDAPS setup

If you want to use LDAPS for communication between Keycloak and your Active Directory, you must perform the following steps:

1. [Store certificate for LDAPS \(optional\)](#)
2. [Restart Keycloak \(optional\)](#)
3. [Customize LDAP configuration](#)

### Store certificate for LDAPS (optional)

If your AD server uses a self-signed certificate that differs from the `CUSTOMER_ROOT_CA_CERTIFICATE`, you must import this certificate into the internal trust store of Keycloak.

To do this, store the path to your certificate in the variable `CUSTOMER_LDAP_CERTIFICATE` in `s3components/components-installer-custom.env`. Then run the script `s3components/bin/components_installer.sh` again.

### Restart Keycloak (optional)

If you debited the variable `CUSTOMER_LDAP_CERTIFICATE` in the previous chapter, you must restart the Keycloak container. To do this, execute the following command:

```
cd s3components/installation
docker-compose down keycloak
docker-compose up -d
```



This also restarts the dependent shipping containers `jwt-auth-userinfo`, `user-administration`, `rest-api` and `keycloak-init`.

### Customize LDAP configuration

To use LDAPS, set the log in the `CONNECTION_URL` variable in the `service-ldap-custom.env` to `ldaps://HOST:PORT` and adjust the port number to the LDAPS port of your AD.

Then execute the script from chapter [Load LDAP configuration to Keycloak](#). If you already have an LDAP configuration in Keycloak, this will be replaced by the new configuration.

## 5.3.3. Setting up LDAP(S) with Kerberos authentication

If you want to use LDAP(S) with Kerberos authentication, you must perform the following steps:

1. [Store Kerberos configuration and keytab](#)
2. [Restart Keycloak](#)
3. [Customize LDAP configuration](#)

### Store Kerberos configuration and keytab

The `s3components/components-installer-custom.env` must be extended by the following variables:

#### **CUSTOMER\_KRB5\_CONF**

Complete path of your `krb5.conf` of the Kerberos server.

The `krb5.conf` file defines the configuration of the Kerberos client on a Linux system. It contains information about Kerberos realms, KDCs (Key Distribution Centers) and administrative servers as well as encryption settings and other configuration options. The file enables the Kerberos client to perform

proper authentication on the network and to interact with the Kerberos server.



The keycloak container requires read access to the file!

### **CUSTOMER\_KERBEROS\_KEYTAB**

Complete path of your exported keytab of the Kerberos server.

The keytab file is a security file used by Kerberos to facilitate the exchange of authentication information. It contains encrypted authentication keys for user principals or service principals. These keys allow a user or service to authenticate to a Kerberos system without the need to enter a password. The keytab file is typically stored on the user's or service's system and used by Kerberos services to authenticate users or services without requiring them to reveal their passwords.



The keycloak container requires read access to the file!

### **Restart Keycloak**

Keycloak must be restarted for the files to be mounted in the Keycloak container. To do this, execute the following command:

```
cd s3components/installation
docker-compose down keycloak
docker-compose up -d
```



This also restarts the dependent shipping containers `jwt-auth-userinfo`, `user-administration`, `rest-api` and `keycloak-init`.

### **Customize LDAP configuration**

The LDAP configuration in the `service-ldap-custom.env` must be extended by the following variables:

#### **KERBEROS\_REALM**

Name of the Kerberos realm. For example: FOO.ORG

#### **SERVER\_PRINCIPAL**

Full name of the server principal for the HTTP service including server and domain name. For example: HTTP/host.foo.org@FOO.ORG

#### **CUSTOMER\_KERBEROS\_KEYTAB**

This variable is used to control whether the Kerberos switches should be activated in the LDAP configuration script. Enter any value here to activate the switches.

Then execute the script from chapter [Load LDAP configuration to Keycloak](#). If you already have an LDAP configuration in Keycloak, this will be replaced by the new configuration.

## 5.4. Installation on distributed systems (from COMPONENTS\_VERSION 1.0.1)

If the components are to be installed on a server other than the Abas server, this can be done in only a few steps. The same requirements apply as described in the [Requirements](#) chapter.

### 5.4.1. ToDo: Download the components-<version>.tgz archive

Download the components-<version>.tgz archive as described in the [Optional: Download the components-<version>.tgz archive](#) chapter.

### 5.4.2. ToDo: Extract the archive

Change to the directory in which you want to extract the components. Then you can use the following command to extract the archive:

```
tar -xzf components-<version>.tgz
```

### 5.4.3. ToDo: Adapt the configuration file

To adapt the configuration file, proceed as described in the [ToDo: Create or adapt the configuration file](#) chapter.



It is important to record the SSH connection data for the s3 user of the Abas server in the variables `ABAS_SSH_USER`, `ABAS_SSH_HOST`, and `ABAS_SSH_PORT`. In addition, the component user must be able to authenticate themselves to the Abas server as the s3 user via an SSH key.

### 5.4.4. ToDo: Copy the Abas font types

The free text editor requires the Abas standard font types as well as the customer-specific font types. During a component installation, an attempt is made to automatically load the fonts from the Homedir and store them in the components directory under `freitexteditor/fonts`.

In the `docker-compose.yml` in the components directory, you can see how the fonts of the component are made available (in the 'free text editor' under 'volumes').

Alternatively, the font types can also be copied manually. To do this, the `abas-fonts.jar` must be copied from the Abas server from `$HOMEDIR/java/lib/abas-fonts.jar` to `~/components/freitexteditor/fonts/abas-fonts.jar` on the component server. All custom fonts from the Abas server must also be copied from `$HOMEDIR/print/fonts` to `~/components/freitexteditor/fonts` on the component server.

### 5.4.5. ToDo: Set the host FQDN correctly in .server.conf

As described in the [ToDo: Create/Customize .server.conf](#) chapter, the license server and the SSO URL must be set in `.server.conf`. In the case of a distributed installation, ensure that the FQDN of the component server is entered in `.server.conf`.

### 5.4.6. ToDo: Continue with the installation

The installation can now be continued from the [ToDo: Install components \(as the s3 user\)](#) chapter.



When calling `components_installer.sh`, the message "WARN[0000] The "HOMEDIR" variable is not set. Defaulting to a blank string." will be displayed as no HOMEDIR variable is set on the component server. This message can be ignored. If you want to avoid the message, you can also create the HOMEDIR variable with blank string in `common-custom.env`.

## 5.5. Installation of multiple REST API instances

As of COMPONENTS\_VERSION 2.0.0, you can install multiple REST API instances on one server. A configuration file must be created for each REST API instance.



If you have already installed a REST-API instance in a COMPONENTES\_VERSION prior to 2.1.0, you can leave it in place. However, this will not be taken into account by future updates. It is recommended to delete the instance (and the proxy rule for encrypted instances) and create a new one.

### 5.5.1. Create configuration

To do this, first switch to the `config` directory in the Components main directory. Here you will find the following files:

#### **rest-api-NAME.env.template**

Configuration template for an unencrypted REST-API instance

#### **rest-api-ssl-NAME.env.template**

Configuration template for an SSL-encrypted REST-API instance

Create a copy of the desired template and rename it. You can create as many REST-API configurations as you like. A copied template counts for one instance.

#### Configuration for unencrypted REST-API instance

Copy the file `rest-api-NAME.env.template` and rename the copy to `rest-api-NAME.env`. Replace `NAME` with a suitable name for the REST-API instance:

```
cp rest-api-NAME.env.template rest-api-one.env
```

Now open the file `rest-api-one.env`:

```
#  
# Template file for generating one standalone REST-API instance.  
#  
# To generate one standalone REST-API instance just copy this. Remove the .template path and  
# replace the NAME in your filename with a suitable name for your instance.
```

```
# Then edit this file and fill out all required variables.
#

# Name of your REST-API.
NAME=

# Port which will be exposed.
PORT=

# Comma separated list of all ERP clients. For each client a connection config will be generated.
You could leave this variable empty if you want to setup your connection files by yourself.
CLIENTS=
```

Fill in the variables as follows:

### **NAME**

Name/description of the REST-API instance. This name is used to create the installation directory.

### **PORT**

Port under which the REST-API instance should be accessible.

### **CLIENTS**

Comma-separated list of all ERP clients for which a connection configuration is to be created. These configurations are created automatically and copied to the installation directory. You can leave this field empty if you want to create the configurations manually.

A finished configuration could look like this, for example:

```
...

# Name of your REST-API.
NAME=one

# Port which will be exposed.
PORT=8083

# Comma separated list of all ERP clients. For each client a connection config will be generated.
You could leave this variable empty if you want to setup your connection files by yourself.
CLIENTS=erp,demo
```

## **Configuration for SSL-encrypted REST-API instance**

Copy the file `rest-api-ssl-NAME.env.template` and rename the copy to `rest-api-ssl-NAME.env`. Replace `NAME` with a suitable name for the REST-API instance:

```
cp rest-api-ssl-NAME.env.template rest-api-ssl-two.env
```

Now open the file `rest-api-ssl-two.env`:

```
#
# Template file for generating one ssl protected standalone REST-API instance.
#
# To generate one standalone REST-API instance just copy this. Remove the .template path and
# replace the NAME in your filename with a suitable name for your instance.
# Then edit this file and fill out all requireid variables.
#
# Name of you REST-API. This will also be the context path on the webserver
NAME=
# Comma seperated list of all ERP clients. For each client a connection config will be generated.
# You could leave this variable empty if you want to setup your connection files by yourself.
CLIENTS=
```

Fill in the variables as follows:

### **NAME**

Name/description of the REST-API instance. This name is used to create the installation directory and the context path.

### **CLIENTS**

Comma-separated list of all ERP clients for which a connection configuration is to be created. These configurations are created automatically and copied to the installation directory. You can leave this field empty if you want to create the configurations manually.

A finished configuration could look like this, for example:

```
...
# Name of you REST-API. This will also be the context path on the webserver
NAME=two
# Comma seperated list of all ERP clients. For each client a connection config will be generated.
# You could leave this variable empty if you want to setup your connection files by yourself.
CLIENTS=erp,demo
```

## **5.5.2. Create REST-API instances**

Once you have created all REST-API instance configurations, you can install them by calling `bin/components_installer.sh`. Your standard components can continue to run.

The call performs the following actions for each REST-API configuration file:

- Copy the templates from `templates/dynamic/rest-api` or `templates/dynamic/rest-api-ssl` into your installation directory `installation` with the NAME as postfix (e.g. `rest-api-one`).
- Add the REST-API instance `docker-compose.yml` to the global `docker-compose.yml` (include from `./rest-api-one/docker-compose.yml` in `installation/docker-compose.yml`).



- If it is an SSL-encrypted instance, a proxy rule is created for the web server (e.g., [installation/webserver/templates/modules/rest-api-ssl-two.conf.template](#)). The NAME ( `two` in the example) is used as the context path.

### 5.5.3. Start/stop REST-API instances

To start the REST-API instances, change to the installation directory and execute `docker-compose up -d SERVICE_NAME`. For the two configurations from the previous chapters, these would be the following commands:

```
cd installation
docker-compose up -d rest-api-one
docker-compose up -d rest-api-ssl-two
```

To activate the proxy rule from the web server, the container must be recreated:

```
docker-compose down webserver && docker-compose up -d webserver
```

If you had used the configurations from the previous chapters, the REST-API instances would be accessible under the following URLs:

- Unencrypted instance: [http://<HOST\\_NAME>:8083/mw/r](http://<HOST_NAME>:8083/mw/r)
- SSL-encrypted instance: [https://<WEBSERVER\\_HOST>:<WEBSERVER\\_PORT>/two/mw/r](https://<WEBSERVER_HOST>:<WEBSERVER_PORT>/two/mw/r)

If you want to stop an instance, you can do this with the following call:

```
docker-compose down rest-api-one
```

## 5.6. SSL encryption of the full text search

To encrypt your full text search using SSL, you can create additional proxy rules for the web server.

### 5.6.1. Requirements

- The full text search is already installed and running.
- If you have several clients and want to use the full text search, you must install a separate full text search for each client. These must be accessible via different ports.

### 5.6.2. Create configuration

To do this, first switch to the `config` directory in the Components main directory. Here you will find the following file:

#### **vts-ssl-proxy-NAME.env.template**

Configuration template for a VTS SSL proxy rule

Create a copy of the template and rename it. You can create any number of VTS SSL proxy rule configurations. A copied template counts for an ERP client.

For example, if you want to create a VTS SSL proxy rule for the ERP client **erp**, copy the template as follows:

```
cd s3components/config
cp vts-ssl-proxy-NAME.env.template vts-ssl-proxy-erp.env
```

If the full-text search of the ERP client runs on the same server under port 16011, you can adapt the file **vts-ssl-proxy-erp.env** as follows:

```
#
# This file contains all connection data for one VTS instance.
# To use this file just copy it and remove the '.tempalte' in the filename.
#
# Abas ERP client for VTS
VTS_CLIENT=erp
VTS_PROTOCOL=http
VTS_HOST=${HOST_NAME}
VTS_DOMAIN=${HOST_DOMAIN}
VTS_PORT=16011
```

### 5.6.3. Install proxy rule

Once you have created all the VTS SSL proxy rule configurations, you can install them by calling **bin/components\_installer.sh**. Your standard components can continue to run.

The call performs the following actions for each configuration file:

- Creation of a proxy rule under installation/webserver/templates/modules/fulltext\_search\_<VTS\_CLIENT>.conf.template

### 5.6.4. Create new web server container

To activate the proxy rule, you must recreate the web server container. To do this, go to the **s3components/installation** directory and run the following command:

```
docker-compose down webserver && docker-compose up -d webserver
```

To check whether the full text search is now accessible via SSL, call the URL `<a href="https://&lt;host_fqdn&gt;;&lt;webserver_port&gt;/&lt;client&gt;/fulltext-search/ui/en/Search60011.html#otp=&lt;abas_password&gt;" class="bare">https://&lt;host_fqdn&gt;;&lt;webserver_port&gt;/&lt;client&gt;/fulltext-search/ui/en/Search60011.html#otp=&lt;abas_password&gt;</a>`;

## 5.6.5. Use encrypted full text search in the Abas GUI

To ensure that the encrypted full text search is also used in the Abas GUI, open the `$HOMEDIR/mandantdir.env` file and add the entry `VTS_BASEURL = https://<host_fqdn>:<webserver_port>/<client>/fulltext-search`.

Now rebuild your environment using `envmake`. If you now open the Abas GUI, the encrypted full text search is used.

## 5.7. Creating additional proxy rules

If you want to encrypt other applications with SSL, it is possible to create additional proxy rules for the nginx web server. You can either use the script `bin/create_webserver_proxy_rule.sh` or create the proxy rule manually. Both ways are described in the following subchapters.



A proxy rule that has been created is not modified by an upgrade. If your proxy rule contains entries that are no longer supported by the nginx version you are using, you will need to adjust and test the rule manually.

### 5.7.1. Using the script `bin/create_webserver_proxy_rule.sh`

You can create a new proxy rule for the nginx web server using the `bin/create_webserver_proxy_rule.sh` script. The following proxy rules are created:

```
location /${WEBSERVER_PATH} {
    proxy_set_header X-Real-IP $remote_addr;

    resolver 127.0.0.11; # Local resolver
    set $upstream
    ${TARGET_PROTOCOL}://${TARGET_HOST}${TARGET_DOMAIN}:${TARGET_PORT}/${TARGET_PATH};
    proxy_pass $upstream;
}
```

You can customize the proxy rule using the script parameters `--webserver-path`, `--target-protocol`, `--target-host`, `--target-domain`, `--target-port` and `--target-path`. The script parameter `--file-name` must be used to specify a name for the proxy rule.

Call up the script as follows, for example:

```
bin/create_webserver_proxy_rule.sh --file-name example --target-protocol http --target-host example
--target-domain ".com" --target-port 8080 --target-path test webserver-path test2
```

The file `example.conf.template` with the following content is created under ``installation/webserver/templates/modules``:

```
location /test2 {
```

```
proxy_set_header X-Real-IP $remote_addr;

resolver 127.0.0.11; # Local resolver
set $upstream http://example.com:8080/test;
proxy_pass $upstream;
}
```

If the web server is now restarted and the proxy rule is active, a proxy pass from [https://<webserver\\_host>:<webserver\\_port>/test2](https://<webserver_host>:<webserver_port>/test2) to <http://example.com:8080/test> takes place.

### 5.7.2. Manual creation of the proxy rule

Alternatively, you can also create the proxy rule manually. To do this, you must create the file [installation/webserver/templates/modules/NAME.conf.template](#). In this file, you can customize the proxy rule according to your wishes. You must then restart the web server container for the proxy rule to become active.

## 6. Temporary configuration docker containers

Since COMPONENTS\_VERSION 2.0.0, a Docker container called **keycloak-init** starts temporarily and takes over the configuration of Keycloak.

Since COMPONENTS\_VERSION 2.2.3, Docker containers named **java-issued-store**, **java-root-store** and **\*java-ldap-store\*** are started temporarily, which generate certificates in p.12 format in a shared Docker volume named 'certificates'. These are used by Keycloak, the DMS-Rest-Api and the RabbitMQ for DMS.

All temporary containers are automatically closed after configuration. This means that the containers are no longer displayed in the list of running containers (`docker ps`), but they remain in the list of terminated containers (`docker ps -a`).

To **remove** all stopped containers, you can use the following command (Attention: Only execute this command if you no longer need any of the stopped containers, as this can also affect Docker containers that are not part of the components!):

```
docker container prune
```

Alternatively, you can delete containers that have already been stopped by specifying the name or ID of the container:

```
docker container rm <container_name>  
docker container rm <container_id>
```

## 7. Additional components

There are components that are not started by default. These can be activated via profiles.

The `COMPOSE_PROFILES` variable can be used in `components-installer-custom.env` for this purpose. The additional profiles can be listed in this variable, separated by commas.

The additional components are briefly presented in the following sub-chapters.

### 7.1. BPM (with-bpm)

The `with-bpm` profile activates additional containers for BPM.



To use BPM, add `COMPOSE_PROFILES=with-bpm` in the `components-installer-custom.env` **before generation by `components_installer.sh`**.

Add the variables `EDP_CONFIG` and an `ADMIN_USER_EMAIL` to **After generation** in `s3components/installation/bpm/service-custom.env` to make your customization persistent (the `components_install.sh` does not overwrite custom files).

The BPM backend creates a user with this mail address in the BPM Postgres database. The `ADMIN_USER_EMAIL` should therefore match a user in Keycloak who has the rights to handle workflows in Abas. If BPM was not started with a correctly configured `ADMIN_USER_EMAIL`, either delete the fresh BPM Postgres database and configure the values appropriately in the `s3components/installation/bpm/service-custom.env` (then restart the BPM container) or customize the user in BPM directly at the following URL: [http://<host\\_fqdn>:8088/camunda/app/welcome/default/#/login](http://<host_fqdn>:8088/camunda/app/welcome/default/#/login)

### 7.2. DMS-Rest-API and RabbitMQ for DMS (with-dms)

The `with-dms` profile activates the DMS Rest API and an additional RabbitMQ container that is used for DMS communication.



This is **not** the standard RabbitMQ container that is provided with the Abas installation.

#### 7.2.1. Requirements for the DMS Rest API

For the DMS Rest API, it is necessary that Keycloak has already been started. You need access to the Keycloak administration interface to extract information from the abas Realm in Keycloak:

##### 1. Get and set RabbitMQ Client Secret

Retrieve the RabbitMQ Client Secret from Keycloak and set it in the `components-installer-custom.env`:

```
RABBITMQ_CLIENT_SECRET=<rabbitmq_client_secret>
```

*Steps to retrieve Keycloak:*

Log in to the Keycloak Admin Panel.

- Navigate to Clients in the abas Realm.
- Search for the client rabbitmq.
- Open the Credentials tab.
- Copy the value Secret and set it as RABBITMQ\_CLIENT\_SECRET.

## 2. Retrieve and set Rest API Client Secret

Retrieve the Rest API Client Secret from Keycloak and set it in the `components-installer-custom.env`:

```
RESTAPI_CLIENT_SECRET=<restapi_client_secret>
```

*Steps to retrieve Keycloak:*

Log in to the Keycloak Admin Panel.

- Navigate to Clients in the abas Realm.
- Search for the client rest-api.
- Open the Credentials tab.
- Copy the Secret value and set it as RESTAPI\_CLIENT\_SECRET.

## 3. Set database encryption key

Set the database encryption key in the `components-installer-custom.env`:

```
DATABASE_ENCRYPTION_PASSWORD=<any_non_empty_string>
```

## 4. Activate the 'with-dms' profile

Activate the profile `with-dms` in the `components-installer-custom.env` and run the installer again so that your configuration in the `components/installation` directory is updated. The DMS-Rest-Api and the RabbitMQ container are then taken into account when using `docker-compose` commands in the installation directory.

## 7.3. Prodaso Abas Connector (with-pac)

The `with-pac` profile activates three containers that are used for communication between Abas and Prodaso.

In addition to the profile, the following variables can be set for the PAC containers in `components-installer-custom.env`:

```
PAC_REST_API_CLIENT_SECRET=
```

```

PAC_REST_API_CLIENT_ID=
PAC_ABAS_TENANT=
PAC_ABAS_CLIENT=
PAC_PRODASO_GRAPHQL_URL=
PAC_PRODASO_SUBSCRIPTION_URL=
PAC_PRODASO_GRAPHQL_APIKEY=

```

Variable	Description	Comment
PAC_REST_API_CLIENT_SECRET	REQUIRED - Tenant management	Tenant management → Click the eye next to the correct tenant → Users → Applications top left in the hamburger menu, add application
PAC_REST_API_CLIENT_ID	REQUIRED	Tenant management
PAC_ABAS_TENANT	REQUIRED	Should have a /configuration.json format <a href="https://subdomain.domian.abas.cloud/">https://subdomain.domian.abas.cloud/</a>
PAC_ABAS_CLIENT	REQUIRED	Abas system to connect to
PAC_PRODASO_GRAPHQL_URL	REQUIRED	Example: <a href="https://dev.prodaso.ai/graphql">https://dev.prodaso.ai/graphql</a> , COMES FROM PRODASO
PAC_PRODASO_SUBSCRIPTION_URL	REQUIRED	Example: <a href="wss://dev.prodaso.ai/subscriptions">wss://dev.prodaso.ai/subscriptions</a> , COMES FROM PRODASO
PAC_PRODASO_GRAPHQL_APIKEY	REQUIRED, COMES FROM PRODASO	

This is another necessary step to ensure the PAC interface can communicate with the Rest API.

Client credentials configuration for the REST API: [Click here](#)



## 8. FAQ

### 8.1. How is the configuration structured?

See [Structure of the components configuration](#).

### 8.2. When should which configuration file be adjusted?

**Variables defined in the components standard** should be customized in `s3components/components-installer-custom.env`.

The `components_installer.sh` script should then be called to update the configuration known to it.

Variables that **only affect certain components and are not part of the standard configuration** should be adjusted in the respective `service-custom.env` files of the components **after the `components_installer.sh` call** in the 'installation' directory: For example, an individual setting for Keycloak in `s3components/installation/keycloak/service-custom.env`.

Variables that **apply to all components** and are not part of the standard configuration should be adjusted in `s3components/installation/common-custom.env`.