univention
be open.

Univention App Center for App Providers

Release 5.2

Mar 24, 2025

TABLE OF CONTENTS:

CHAPTER

ONE

INTRODUCTION

This document is for app providers who want to place their product clearly visible for a broad, professional and growing user group. It covers the steps on how to make the product available as an app for Univention App Center.

1.1 What is Univention App Center?

Univention App Center is an ecosystem similar to the app stores known from mobile platforms like Apple or Google. It provides an infrastructure to build, deploy and run enterprise applications on Univention Corporate Server (UCS). The App Center uses well-known technologies like Docker[1].

1.2 App Center infrastructure

The ecosystem consists of the following components:

The App
: is the software plus the collection of metadata like configuration, text description, logo, screenshots and more for the presentation.

The App Center Repository
: is a central server infrastructure managed by Univention that stores the files and data for the app. It's the installation source for the app.

The App Center Module on UCS
: is part of the web-based management system on UCS. It's the place where UCS administrators install, update and uninstall apps in their UCS environment.

The App Catalog
: presents the app portfolio on the Univention website[2].

The App Provider Portal
: is the self-service portal for app providers where they can create and maintain their app.

The Test App Center
: is the "staging area" for app providers to develop and test their apps.

Univention Configuration Registry
: is the target platform for the app. UCS is technically a derivative of Debian GNU/Linux.

For building an app the app developer works with UCS, the app, the App Provider Portal and the Test App Center.

[1] https://www.docker.com/
[2] https://www.univention.com/products/app-catalog/

CHAPTER

TWO

GET STARTED

This chapter describes the requirements and the steps to create an app for Univention App Center. After reading this chapter an app provider will be able to create their own app and start with a development and test cycle using the Test App Center.

2.1 What does the app provider need?

2.1.1 Software in a Docker image

The software needs to be provided as a Docker image[3]. This is the easiest way to deploy software in Univention App Center. It's also recommended to publish the Docker container to Docker hub[4]. This makes referencing the image later much easier and simplifies the development and test cycle during development.

If public access to the image is not wanted, it can be made private. For use on a UCS test machine during app development, the `docker login` needs to be used on the command-line to grant your machine access to the private image. For the release of the app, the Univention team needs to have access to the image. Please then grant access to the Docker Hub user `univention`. The image then has to be copied manually by the App Center team to the Univention Docker registry, which cannot be browsed. It needs credentials to be accessed. The Docker image should be considered public by the time the app is published in Univention App Center.

The image must have a version tag to distinguish different software versions. It later allows updates for the apps.

2.1.2 Account for App Provider Portal

The App Provider Portal is the app developer's place for self service for all the settings around the app.

1. To start building the app, an account for the App Provider Portal is needed. Please request a personal account[5] on the Univention website and provide some context about the intended app.
2. An email with username and instructions on how to set a password is sent.
3. Afterwards the login can be performed at the App Provider Portal[6].
4. After the login, the *Apps* module needs to be opened to work on the app.

[3] https://docs.docker.com/
[4] https://hub.docker.com/
[5] https://www.univention.com/products/for-solution-providers/
[6] https://provider-portal.software-univention.de/univention/management/

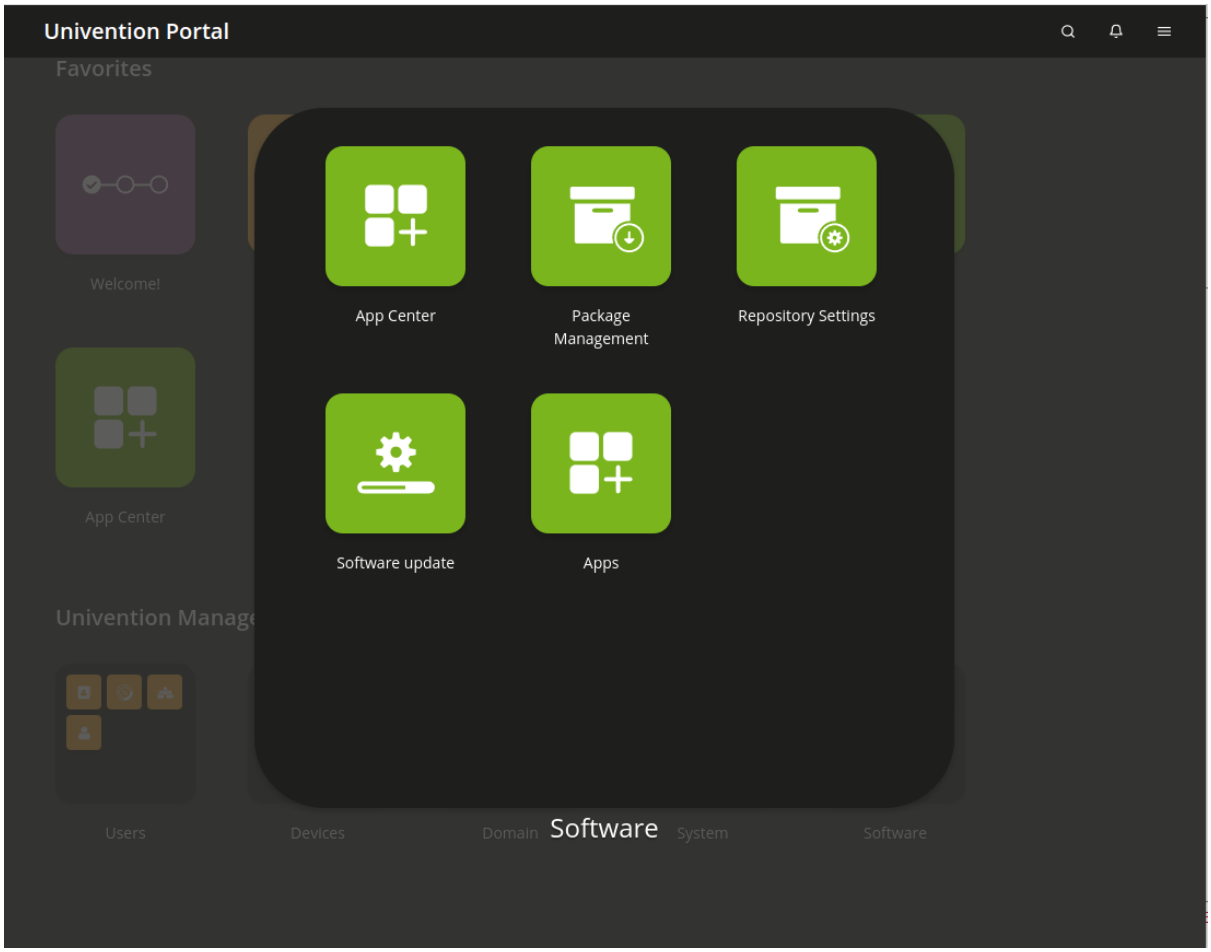

Fig. 2.1: App Provider Portal overview with “Apps” module selected

2.1.3 Where to get help?

App providers that need technical help during their development process are invited to open a topic in Univention Forum[7]. The dedicated section *App Development* is for all questions around app development, debugging and the like.

2.2 Create an app with a Docker image

This section describes how to create the app in the App Provider Portal and use a Docker image. It focuses on a single container setup. For a setup with multiple containers with Docker Compose please see *Create a Multi Container App* (page 9).

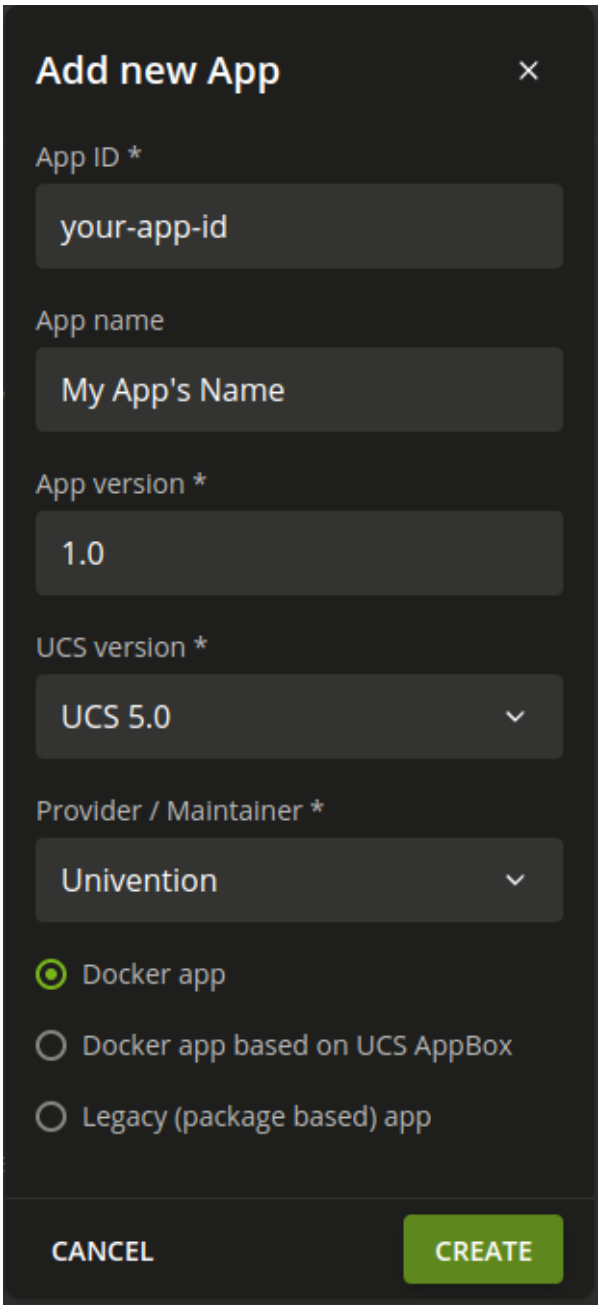

Fig. 2.2: Add new App

[7] https://help.univention.com/c/apps/dev/50

1. In the App Provider Portal select *Favorites* ▸ *Apps* or *Software* ▸ *Apps*.
2. Click on *Add new app* and provide the following settings.

App ID
is like a primary key. Choose it carefully, because it cannot be changed once the app is released to the public. It should only use small capitals, dashes and numbers. Please do not include version strings in here.

App name
is the name of the app. It's used to display the app on the overview pages. This attribute can be changed any time.

App version
is the version of the app. The App Center distinguishes versions and uses them to handle app updates. Once the app is released, this attribute cannot be changed.

UCS version
is the UCS version the app should start to be available on. Simply start with the latest available UCS version. It can also be started with the oldest maintained UCS version to cover the broadest user base of UCS. See the Univention Help 21843 - "UCS maintenance cycle"[8] for an overview of the maintained UCS version. In either case it's recommended to specify the supported UCS versions explicitly, see *Supported UCS versions* (page 6).

Provider / Maintainer
refers to the organization that the app belongs to. Please select your organization here or otherwise the app will not show up in the listing.

Docker app
is for the recommended Docker based app. This documentation only covers single and multi container apps.

2.2.1 Docker image

1. In the app go to the tab *Configuration*.
2. Select the type of Docker app. This chapter discusses the *Single container app*, therefore please select it.
3. Enter the name of the image to *Docker image*. Grab the name of the image from Docker hub for example `python:3.7-bullseye`.

Important: Please add the version tag explicitly. The App Center distinguishes different app versions and handles updates accordingly.

2.2.2 Supported UCS versions

Upon app creation the *UCS Version* has been specified. Please define the supported UCS version explicitly on the *Version* tab in the *Supported UCS versions* section.

Example: The app has been created for *UCS Version* `5.1`. Two entries for Supported UCS versions for App could be made: `5.1-0` and `5.2-0`. This means that for the installation of the app UCS 5.0-4 or UCS 5.2-0 are required.

[8] https://help.univention.com/t/21843

2.2.3 Logo and description

On the app's *Presentation* tab please provide the display name and a description in English and German and logos for the software. Start with a short and a full description. It gives an impression on how it will look like during later testing.

On the same tab two logos can be uploaded: A default icon that is shown on the app tile in the overview. For optimal presentation it should be more of a 1:1 ratio. The second can be more detailed and can for example include the software name. Please provide the logos in SVG format.

Those settings can be changed later. For a more detailed description of the app presentation and notes on the translation, please take a look at *App presentation* (page 23).

2.2.4 Persistent data with volumes

By default files created inside a container are stored in it, but they don't persist when the container is no longer running, removed or is exchanged with a newer version. As solution Docker offers volumes[9], a mechanism for persisting data generated and used by Docker containers. A volume is a directory on the Docker host that is mounted inside the Docker container.

To define volumes for the app, please activate them on the *Overview* tab in the *Modules* section with the option *Docker app defines volumes*. Then go to the *Volumes* tab. Add an entry for each volume and define the directory or file path on the host in the first field and the destination in the container in the second field. Leave the second field empty for the same path.

For example:

Host
`/var/lib/app_etc`

Docker container
`/etc/app`

2.2.5 Web interface

Many Docker apps expose a web interface to the outside world, e.g. via the port 8080. The App Center on UCS takes care to map this web interface from some relative link to this port and adds a reverse proxy configuration to the host's web server configuration.

On the *Web interface* tab, enter the relative path and which ports should be proxied. For example, to map the container's ports 80 and 443 to `/myapp`, the following settings have to be made:

Relative URL to web application
`/myapp`

HTTP port of web application
`80`

HTTPS port of web application
`443`

Supported protocols by the container's web interface
Select *HTTP and HTTPS*, if both protocol schemes should be covered.

[9] https://docs.docker.com/engine/storage/volumes/

2.2.6 Ports

If the app needs to occupy ports on the host that need to be passed along to the container in order to work properly, they can be defined in the *Ports* section on the *Web interface* tab. A list of ports can be defined that the Docker host shall exclusively acquire for the Docker container (*Port to be acquired exclusively*). Ports defined here cannot be used by other services or other Docker containers on the UCS host. A second list can be defined for ports that should be forwarded from the host to the Docker container (*Host port to be forwarded*). Ports defined here will build an implicit conflict list against other apps that want to use these ports.

For example, the solution exposes the API under the dedicated port `5555`. This port would be predestined to be defined here.

With the port definition the App Center also takes care to open them in the UCS firewall. If additional firewall rules for ports are needed, they can be defined in the app join script. Please refer to Network packet filter[10] in the UCS Developer Reference.

2.2.7 Database

Many applications need a relational database management system (RDMS) somewhere in the environment to function properly. If the app needs such a database the App Center takes care of providing one directly to the Docker host. Activate *Docker app needs database* on the *Overview* tab in the *Modules* section and then go to the *Database* tab, where the appropriate settings can be made.

In the *Database* section the settings for the database are defined. MariaDB and PostgreSQL are supported. Database user, database name and the path to the password file can be specified. Upon installation of the app, the App Center installs the defined database on the Docker host, creates a database with the defined settings and saves the password in a file for later use.

In the *Database environment variables* section, the mapping of the database settings to the environment variables in the container are defined. For example, if the container expects the database hostname in `DATABASE_HOST`, it has to be entered into the field *Variable name for the database host*. There are also fields for the database port, user, password, database name and the password file.

2.2.8 Environment

Docker images usually receive environment variables when the container is started. The App Center supports to pass static configuration options to the container. Variables parameterized by Univention Configuration Registry Variables are also supported. An environment file can look like the following example:

```
LDAP_SERVER=@%@ldap/server@%@
FQDN=@%@hostname@%@.@%@domainname@%@
HOME=/var/lib/univention-appcenter/apps/myapp/data/myapp_home
```

The content of the environment file can be entered in the App Provider portal on the *Configuration* tab in the field for *Environment file for Docker container creation*.

[10] https://docs.software-univention.de/developer-reference/5.2/en/misc.html#misc-nacl

2.3 Create a Multi Container App

Univention App Center supports apps that consist of multiple Docker images. It uses Docker Compose[11], a tool for defining and running multi-container Docker applications. The heart of such applications is a YAML file that configures all services for the application. The supported compose file format version is 2.0.

2.3.1 Multi container setup

In order to create a Multi Container App, go to the *Configuration* tab in the App Provider Portal, select *Multi container app with Docker compose* and enter the content of your `docker-compose.yml` file. A "flat" YAML file must be used, because the implementation does currently not support references to other files like for example files that should be mounted inside a container or files listing environment variables.

Univention Configuration Registry, UCR for short, is the central tool for managing the local system configuration of UCS (see Administration of local system configuration with Univention Configuration Registry[12]). Settings from UCR can be used in the Docker compose file to parameterize the Docker setup. This comes in very handy when settings like for example the local LDAP server should be passed to a container via its environment variables.

```
[...]
services:
    [...]
    environment:
        ROOT_URL: https://@%@hostname@%@.@%@domainname@%@/$appid
        LDAP_Host: "@%@ldap/server/name@%@"
        LDAP_Port: "@%@ldap/server/port@%@"
        LDAP_BaseDN: "@%@ldap/base@%@"
        LDAP_Authentication_UserDN: "@%@appcenter/apps/$appid/hostdn@%@"
    [...]
```

The example above is an excerpt from a Docker compose file where environment variables are defined for a service. The values of the variables are set to the values of the given UCR variable. *`appid`* needs to be replaced manually by you app id. UCR variables are enclosed by `@%@`. Please mind the double quotes in the example.

You also need to define the *Name of the "main" service within the docker-compose.yml* below the *Contents of the docker-compose.yml file*.

In order to provide access to the application's web interface, please see *Web interface* (page 7).

If the app setup requires exclusive ports and a list of ports needs to get forwarded from the host to the container, please see *Ports* (page 8).

2.3.2 Script execution reference

The App Center allows several scripts to be executed on the host and inside the container during *installation* (page 35), *uninstallation* (page 39) and *upgrade* (page 41). Scripts run inside the container are run inside the container of the "main service".

[11] https://docs.docker.com/compose/
[12] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-administration-of-local-system-configuration-with-univention-configuration

2.3.3 Post processing of Docker Compose file

Before a Multi Container App is started by the App Center, the `docker-compose.yml` is altered by the App Center with the following changes:

1. When a Multi Container App is released, the `docker-compose.yml` is adjusted on the server side and the Docker Image information is changed to point to the Docker Images in the Univention Docker Registry. All Docker Images from published apps are copied to the Univention Docker Registry to be independent of hub.docker.com[13]. This is the only server-side change to the Docker Compose file.
2. The `docker-compose.yml` is itself a UCR template. As such, it will be interpreted by the App Center before being used. See *UCR Template docker-compose file* (page 10) for details.
3. The App Center adds two standard volumes for the main service, as they are also included in Single Container Apps. These are the `/var/lib/univention-appcenter/apps/`*`appid`*`/data/` and `/var/lib/univention-appcenter/apps/`*`appid`*`/conf/` directories on the UCS host. If volumes are defined in the App Provider Portal in the App Configuration, these are also supplemented in `docker-compose.yml` by the App Center for the main service.
4. If ports are defined in the App Provider Portal, they are also added to `docker-compose.yml`. Ports that have already been defined continue to exist. If the same port is defined in the portal and in `docker-compose.yml`, the configuration in the App Provider Portal takes precedence. For example, if `docker-compose.yml` states that port `4500` is provided externally as port `4500`, but the portal defines that this port is to be used as `6500`, `docker-compose.yml` will be modified to map port `4500` to `6500` on the host.
5. If `docker-compose.yml` specifies that port `80` or `443` should be opened to the outside and the App Configuration specifies that these ports should be used by the App Center for the web interface, the App Center will define a port on the fly in `docker-compose.yml`. This is because UCS hosts usually occupy ports 80 and 443 with a web server. The App Center creates an Apache Reverse Proxy configuration. See *Web interface* (page 7) for details.
6. UCS provides a number of environment variables via the App Center, e.g. parameters for an LDAP connection. The necessary variables are also written to `docker-compose.yml` in the *environments* section.
7. Furthermore, in the main service, as in Single Container Apps, all UCR variables defined on UCS are available under `/etc/univention/base.conf`, as well as the password for the so-called machine account under `/etc/machine.secret`.

As a result, Docker Compose starts a configuration on the UCS system that no longer matches 100% of the App Provider's input. The modified `docker-compose.yml` can be found at `/var/lib/univention-appcenter/apps/`*`appid`*`/compose/docker-compose.yml`.

UCR Template docker-compose file

As stated above, the `docker-compose.yml` is a UCR template. This means that you are able to match the file to the environment of the Docker host. The UCS Developer Reference contains more information about UCR templates[14], but the core mechanics are:

1. Although every `docker-compose.yml` is a UCR template, you may not notice it: Where no specific tags are used, the very content is used. So if your file does not need any of the features mentioned below, just use your plain `docker-compose.yml`.
2. You can add specific values of the configuration registry into your file. More importantly, this includes the App settings in *App settings* (page 44) defined by the App itself:

```
environment:
    MY_KEY: "@%@myapp/mysetting@%@"
```

[13] https://hub.docker.com/
[14] https://docs.software-univention.de/developer-reference/5.2/en/ucr/index.html#chap-ucr

Note that App Settings are always added to the main service automatically. But this allows adding them to other containers and using them as part of a composite value.

3. You can do Python scripting within the template, e.g. to read (and write) the content of specific files.

```
environment:
    MY_SECRET: "@!@import uuid; print(uuid.uuid4())@!@"
```

Note that currently, you cannot access App Settings within the Python script.

2.3.4 Finish multi container setup

As soon as all the technical settings are made, please see *App life cycle* (page 13) for the next steps and how to test the app. For app presentation in the App Center please see *App presentation* (page 23).

CHAPTER

THREE

APP LIFECYCLE

This section relates the lifecycle of an app from the first app development, its life in the public to termination. The lifecycle applies to the app overall and to every single version.

3.1 Development

Every app starts with its development. It involves the definition of the app in the App Provider Portal, providing the software solution as a Docker image, see *Get Started* (page 3), and the integration of the solution with UCS, for example in the identity management area, see *Connection with Identity Management* (page 27). This part of the lifecycle can be divided into two steps: setup and integration.

3.1.1 Best practice for app development environment

This section briefly describes best practices on the development and testing environment for apps dedicated to the Univention App Center. The recommendations aim at reducing repeatable time efforts that, for example, are caused by the download of Docker images.

The development of apps mainly involves the App Provider portal and a local instance of Univention Configuration Registry (UCR). To setup your UCS environment the best practice is to download one of the UCS virtual machine images and use VirtualBox or VMware, see *Download UCS* (page 14). After the installation, activate the system[15] and copy your SSH key to the UCS system to save typing the password on each remote console login. Finally, activate the Test App Center, see *Test App Center* (page 14). With that status it's a good time to make a snapshot of the system, which allows to got back to that status. Afterwards, app specific time consuming tasks can be prepared to reduce recurring waiting times. Among them are for example an optional **`docker login`** to gain access to a private Docker image or even the own Docker registry, and the manual download of the app's Docker image(s) with **`docker pull`** (for single container apps) or **`docker-compose -p $appid pull`** (for multi container apps) from the within the `docker-compose.yml` directory of the app, see *Debugging* (page 16). And then also make a new snapshot. Especially, having the Docker image(s) locally saves download time during recurring installation test cycles.

With the last snapshot there is a prepared system that can serve as starting point for app installation and associated testing. The development iteration cycles are a dance of changes in the App Provider portal and testing the installation and app integration. After returning to a snapshot, it's recommended to update the local App Center cache. This is either automatically done during login and opening of the App Center module in the UCS management system or triggered manually with **`univention-app update`** on the console.

[15] https://docs.software-univention.de/manual/5.2/en/central-management-umc/umc.html#central-license

3.2 Testing

Testing the app is the final part of its development. It typically involves a combination of manual and automatic tests. In order to test the app with UCS, please follow these instructions.

3.2.1 Download UCS

Download UCS to get a copy of UCS free of charge from the Univention website[16]. Choose an ISO image or a pre-installed virtual machine image for various virtualization hypervisors, like for example VMware (ESXi and Workstations), VirtualBox and KVM.

3.2.2 Initial UCS setup

Please refer to the Quickstart Guide for Univention Corporate Server[17] for the first steps about installation and initial setup.

3.2.3 Test App Center

Switch to the Test App Center Repository and test the app directly on the UCS system set up before.

```
$ univention-install univention-appcenter-dev
$ univention-app dev-use-test-appcenter
```

3.2.4 Install the app

Up until here, the app is available in the Test App Center and a UCS system is up and running that is configured against the Test App Center. The app can be installed via the Univention Management Console. UMC consists of several modules and one of them is called *App Center*. Open this module and install the app from there.

The following points are mandatory for the App to be published:

- The App has to install without user interaction. Exceptions are configuration parameters obtained through App Settings, see *App settings* (page 44).
- The App may not negatively impact UCS' core services.
- The App has to uninstall without leaving any remainders that interfere with UCS' core services.
- The app is working with the UCS versions that are configured as Supported UCS version, see *Supported UCS versions* (page 6).

When installing the App, also mind `README` information that may have been defined on the *Additional texts* tab in the app in the App Provider Portal after activating it with the option *Additional texts with information for UCS system administrators* on the Overview tab in the *Modules* section. An Administrator expects an app to be fully configured after the installation. If something is needed in order to use the app, such as activating users for the service, it should be described in the appropriate README section, see *README for the administrator* (page 25).

[16] https://www.univention.com/products/download/
[17] https://docs.software-univention.de/quickstart-en-5.2.html

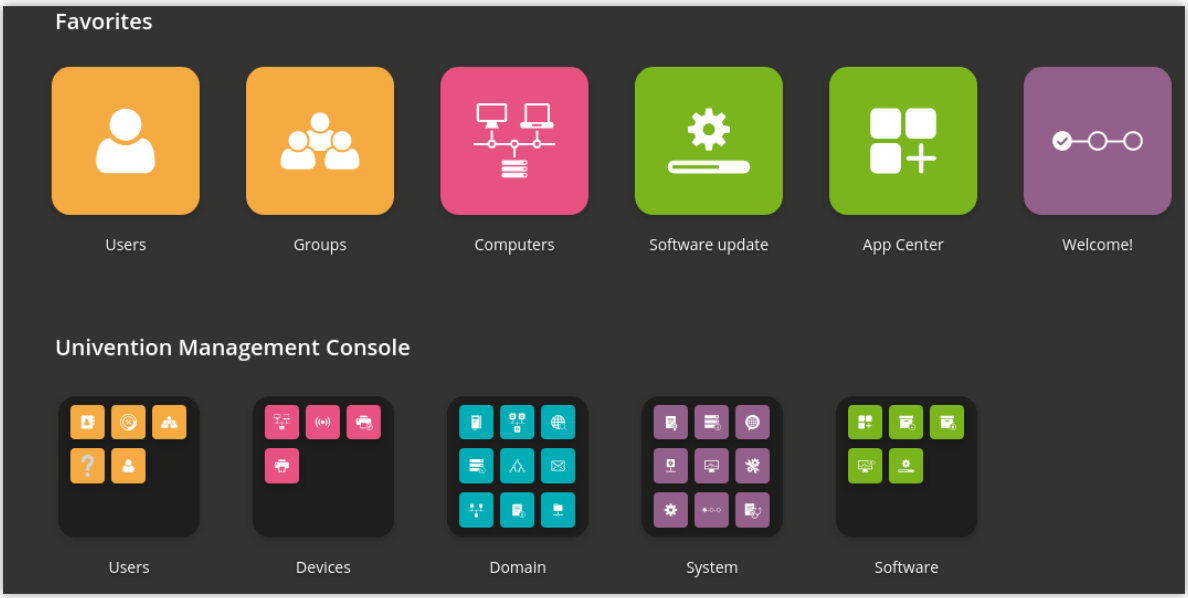

Fig. 3.1: Univention Management Console overview with App Center Module

3.2.5 Upgrade the app

Once several versions of the app exist, the upgrade should be tested, as well. The App Center UMC module only allows the installation of the latest app version. To test an upgrade of the app, please go to the console and install a version explicitly:

```
$ univention-app install myapp=1.0
```

Afterwards, go back to the App Center UMC module. There, the app should show up with the update symbol indicating the availability of an update. Continue and perform the app upgrade.

3.2.6 Test checklist

The following list gives an idea what the app provider should test before the app is approved for release or update. The list cannot cover all possible items, but mentions what is missed repeatedly.

- Integration with UCS identity management: Does the login with activated users work? Will the login be blocked for users that are not activated for the app?
- Environment with proxy: How does the app behave when a proxy is configured in the UCS environment? Does the app take the proxy settings into account? Does the app work as expected? Is the networking still working properly for the app?
- Description and documentation: Are the texts consistent in content? Can they be understood easily? Is the description of technical steps complete and correct, especially path and file names?

3.2.7 Deactivate the Test App Center

To switch back to the productive App Center, please run:

```
$ univention-app dev-use-test-appcenter --revert
```

3.2.8 Automated testing for App Providers

Univention always runs automated tests on an app before it's released. This testing infrastructure can be used by the app provider to increase the test coverage for the app. This can save manual testing efforts.

A test script for the app can be provided in the *Testing* tab. The script needs to be entered in the text area *Test script run after installation*. It's run together with the automatic app tests in the Univention test infrastructure and will be executed after the app has been installed during those tests.

On successful tests, the script has to end with exit code `0`. `--binddn` and `--bindpwdfile` are passed, so that the command line interface of UDM can be used easily. Here is an example:

```
#!/bin/sh
set -e
udm users/user create "$@" \
  --set username=myapp-test-user \
  --set password=s3cr3t-pwd \
  --set lastname=Test \
  --set myappActivated=TRUE
sleep 10
curl http://localhost/myapp/login ...
```

For certain tests user credentials for access to the directory service are needed. They are passed with the parameters `binddn` and `bindpwd`. In many cases the user *Administrator* is used. The Administrator credentials are only important for UDM calls on UCS systems that are not the Primary Directory Node.

The test script can be manually tested.

```
$ univention-install univention-appcenter-dev
$ univention-app dev-use-test-appcenter
$ univention-app install my-app
$ univention-app dev-test-setup  # installs common testing libs like selenium
$ univention-app dev-test \
  --appcenter-server http://appcenter-test.software-univention.de \
  my-app \
  --binddn "$DN" \
  --bindpwd "$BINDPWD"
```

3.2.9 Debugging

During app development or for problem analysis it may be necessary to debug an app. If an app installation fails, the App Center removes the Docker container and thus prevents the developer to have a deeper look into what happened. With the following command the App Center will not throw away a failed app installation. The container is kept on the system.

```
$ univention-app install myapp --do-not-revert
```

For debugging or support cases it can be helpful to enter the app container. The following command opens a shell in the container.

```
$ univention-app shell myapp
```

If the container does not support a shell, the container can be entered with the plain Docker commands.

```
$ CONTAINER="$(ucr get appcenter/apps/myapp/container)"
$ docker exec "$CONTAINER" ...
```

To view the Docker log files for the app, please use the following command:

```
$ univention-app logs $appid  # equivalent to docker logs $CONTAINER
```

For multi container apps using Docker compose, those logging information can be viewed with:

```
$ cd /var/lib/univention-appcenter/apps/$appid/compose
$ docker-compose -p "$appid" logs
```

Important log files on the UCS host for debugging are:

- `/var/log/univention/appcenter.log`
- `/var/log/univention/management-console-module-appcenter.log`
- `/var/log/univention/join.log`
- `/var/log/docker.log`

3.3 Two-way approval

Testing the app also involves giving an approval for the release of the app. Each app in the App Center requires a two-way approval, one from the app provider and one from Univention. The app provider starts with the process and performs the tests to give the approval for its app.

The app provider's approval is given by clicking on *Approve for release* in the App Provider Portal. The Univention App Center team is notified and they will start the automated tests. As soon as the automated tests are successful, the second approval is available and the app is ready to be released.

The automated tests conducted by Univention test for installation, un-installation and re-installation of the app for different UCS system roles. They check for proper operation of the UCS services. No app functionality will be tested. The tests are for ensuring that the app does not break UCS functionality.

Besides the technical testing, the first release also undergoes a manual review process on the app presentation. The app description is checked for plausibility, understandability and for the App Center context.

3.4 Release

As soon as the app is ready for release, it's copied to the productive App Center by the Univention App Center team. This involves the app's presentation material as well as the Docker image. Univention maintains an own Docker registry. The Docker image is copied to that registry and the reference to the Docker image is updated accordingly in the app definition. The app will be visible in the App Center UMC module immediately for all UCS users after release. The app catalog is updated automatically and shows the new app.

3.5 Updates

After the release of the first version the app enters "maintenance mode" and receives updates from the app provider. The evolution of an app is represented by its version. In order to provide an update for the app, follow these steps:

1. Go to the *Apps* UMC module in the App Provider Portal and search for the app.
2. To create a new version of the app, mouse over the app's tile, right click and select *New App Version*.

Fig. 3.2: Menu to add a new App

3. Enter the values for source and target version. The App Provider Portal will then copy the source app configuration to the target.
4. Open the new app version and make the appropriate changes, e.g. the version tag for the Docker image. Usual places for changes involve the scripts and maybe the integration. Texts and logos should be checked if they are still up to date.

3.6 Termination (end of life)

If the app provider decides to discontinue an app and stop maintenance, the app can be terminated by setting it *end of life*. As soon as an app is published in status end of life, UCS system administrators won't be able to install it anymore. UCS systems that already have the app installed, will show a notification in the App Center that informs the UCS system administrator that they should look for alternative solutions.

App versions are not explicitly terminated. They are superseded by the next version. The termination applies to the whole app.

The app can be terminated by going to the app's configuration in the App Provider Portal. Please go to the *Advanced* tab to the section *Custom configuration*. Create a new custom configuration with the *Custom key* `EndOfLife` and the *Custom value* `True`. Afterwards press *Save* and *Approve for release*. Please provide a custom message in the publish dialog and let the App Center team know about the reasons for the termination.

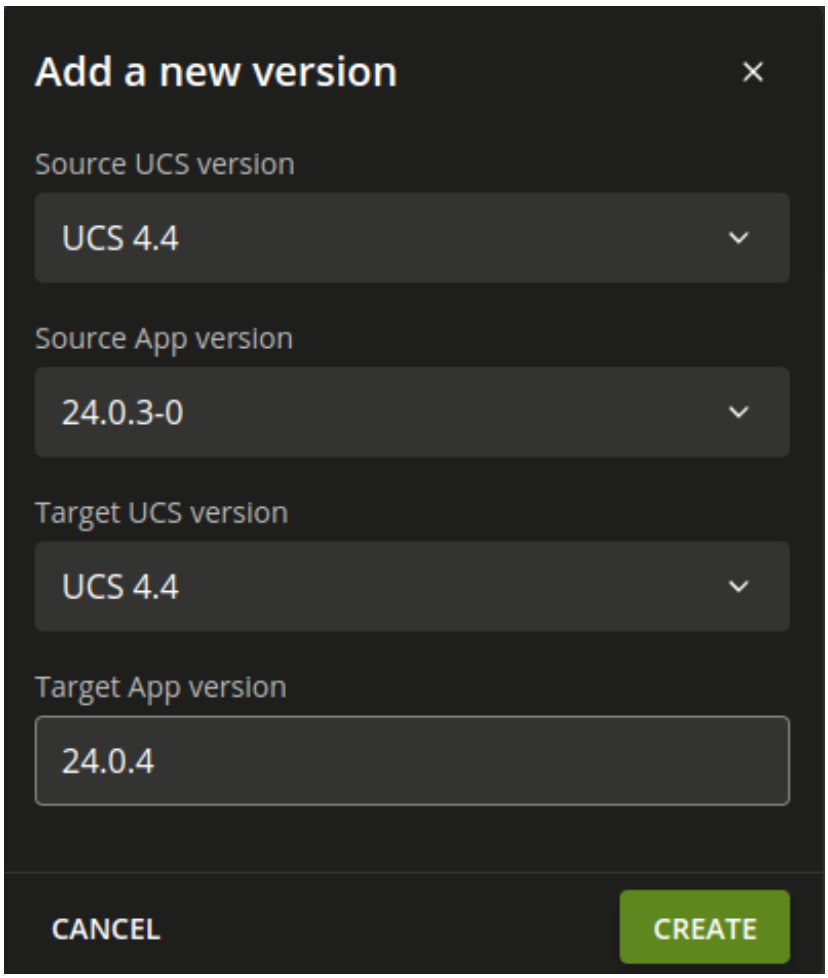

Fig. 3.3: Add a new App version

3.7 App Provider Portal upload interface

An app is configured via the App Provider Portal, which offers a web interface. There is also an upload interface available. If the developer prefers to keep the app configuration in a version control system, the upload interface offers a good way to push changes for the app to the Test App Center. The script can also be used in non-interactive mode within other scripts. In such cases username and password should be stored in separate files (no new line at the end) that are passed as parameters.

The scripts needs to be downloaded[18]. It requires Python 3.7 and cURL to run. For a list of available actions and their parameters, use the script's help:

```
$ ./univention-appcenter-control --help
$ ./univention-appcenter-control upload --help
```

The following examples show how the script can be used.

```
# creates a new version based on the latest version of myapp
$ ./univention-appcenter-control new-version 5.2/myapp

# finds out on which UCS versions this Docker App may be installed
# Note: jq is an external tool: apt-get install jq
# you may parse JSON without it, of course
$ ./univention-appcenter-control get --json 5.2/myapp |
  jq '._ini_vars.SupportedUCSVersions'

# creates version 2.0 of myapp based on the (formerly) latest version
$ ./univention-appcenter-control new-version 5.2/myapp 5.2/myapp=2.0

# sets the DockerImage of the new app
$ ./univention-appcenter-control set 5.2/myapp=2.0 \
  --json '{"DockerImage": "mycompany/myimage:2.0"}'

# copies myapp Version 1.0 from UCS 4.4 to UCS 5.2.
$ ./univention-appcenter-control new-version 4.4/myapp=1.0 5.2/myapp=1.0

# uploads all README files in the current folder to the latest version
# myapp in UCS 4.2
$ ./univention-appcenter-control upload 5.2/myapp README*

# uploads an image. Will be rejected if this image is not specified
# somewhere in the ini file. Note: This may overwrite the logo for other
# version using the same logo name, too.
$ ./univention-appcenter-control upload 5.2/myapp=1.0 myapp.svg
```

The script can also be used to automate the creation of a new app version for an app update. The generics steps are the following:

1. Create a new version in the provider portal via the upload script based on the latest published version.
2. For single container apps update the reference to the app's Docker image and its version. For multi container apps, download the compose file, update the references to the Docker images of the services there and upload the compose file.
3. Perform the usual tests before approving an app update for release.
4. Send an email to the App Center team and approve the release. Please provide information about the UCS section, the app's version and the internal component reference. It's best to provide this information in the email's subject and it can look like this: *"Regarding 4.4/myapp=4.74 (myapp_20201106183244)"*

The following examples show how to run the steps with the upload interface. The examples assume that the password to the portal is stored in a password file which is given as parameter to the script:

[18] https://provider-portal.software-univention.de/appcenter-selfservice/univention-appcenter-control

```
$ APP_UPGRADE_FROM="12.1"
$ APP_VERSION="12.2"
$ UCS_MINOR="4.4"
$ MY_APP="myapp"
$ MY_USERNAME="my_username"
$ PWD_FILE="portal_password"

# 1. Create a new version in the app provider portal
$ ./univention-appcenter-control new-version \
  --username "$MY_USERNAME" \
  --pwdfile "$PWD_FILE" \
  "$UCS_MINOR/$MY_APP" \
  "$UCS_MINOR/$MY_APP=$APP_VERSION"

## First example for single container apps
# 2. Update the reference to the app Docker image
$ ./univention-appcenter-control set \
  --username "$MY_USERNAME" \
  --pwdfile "$PWD_FILE" \
  "$UCS_MINOR/$MY_APP=$APP_VERSION" \
  --json '{"DockerImage": "my_company/'"$MY_APP:$APP_VERSION"'"}'

# 3. Obtain the component id of the new app version.
# The command asumes the latest component is the new app.
$ COMPONENT=$(./univention-appcenter-control status \
  --username "$MY_USERNAME" \
  --pwdfile "$PWD_FILE" \
  "$UCS_MINOR/$MY_APP" | sed -rne 's/^ *COMPONENT: +//p;T;q')

# 4. Send the email
$ SUBJECT="Regarding $UCS_MINOR/$MY_APP=$APP_VERSION ($COMPONENT)"

####
# Second example for multi container apps
# 2. Get app configuration data

$ ./univention-appcenter-control get \
  "$UCS_MINOR/$MY_APP=$APP_VERSION" \
  --json \
  --username "$MY_USERNAME" \
  --pwdfile "$PWD_FILE" > "$MY_APP.json"

# 3. Extract the compose content
$ jq -r .compose < "$MY_APP.json" > compose

# Edit the compose file accordingly. A custom script can help to automate this␣
↪step.
# This script depends on the app and the compose file content
# Replace the "image: " lines and refer to the upstream Docker images and their␣
↪respective tags

# 4. Upload altered compose file
$ ./univention-appcenter-control upload \
  --username "$MY_USERNAME" \
  --pwdfile "$PWD_FILE" \
  --non-interactive \
  "$UCS_MINOR/$MY_APP=$APP_VERSION" compose

# 5. Send the mail and with subject as described above
```

CHAPTER

FOUR

APP PRESENTATION

This chapter is about how the app is presented to the user with texts, logos, screenshots and videos. The contents are part of the app configuration. They are shown to the UCS system administrator in the App Center UMC module in UCS and to users on the Univention website in the App Catalog[19].

All changes are made in the App Provider portal. They need to be saved by pressing the button *Save*. Then a release can be requested via the button *Approve for release*. The items mentioned in this chapter can be changed and published any time and do not require a new version of the app.

4.1 Logos

All logos uploaded to the App Provider Portal have to be SVG format, which is most flexible for the presentation purposes.

Important: When SVG files are created or exported, please make sure that fonts are converted to paths before export. Otherwise the text in the logos is not rendered properly and the logo may look odd.

Please also do not simply import a bitmap graphic into SVG and export it. Results after scaling may not look good, because the logo is basically a bitmap in SVG apparel.

The logos can be uploaded on the *Presentation* tab in the *Logos* section. Two icons are needed: One for the app tile on the overview page and a more detailed logo for the app page. The tile has only limited space in square format. Please make sure, the logo can still be recognized. The detailed logo is not limited. Most logos for this slot have a landscape orientation. The App Center and the App Catalog take care of the appropriate scaling. SVG allows a very good result due to its nature as a vector graphics format.

4.2 Screenshots and videos

Screenshots and videos are a good way to introduce the solution to the user. To add screenshots please go to the *Screenshots & YouTube videos* section on the *Presentation* tab. Screenshots can be in PNG or JPG format. Videos have to be published on YouTube and the full YouTube link has to be provided in the App Provider Portal. Please keep in mind to provide the material for English and German speaking audience.

If the same screenshots exist in German, it's recommended that they are added, as well.

Comprehensive visualizations can be added optionally which make it easier to understand the app's description. Give the image files sensible names including the keywords. An example of a bad file name for an image would be `app_76bz3.jpg`, whereas `app_name.jpg` would be much better.

[19] https://www.univention.com/products/app-catalog/

4.3 Description

The full description is the text introducing the solution to the user and thus is very important for getting their attention. Here are some tips intended to help to present the app in a user-friendly, customer-oriented, and search-engine-optimized manner.

- Unique content with at least 300 words. Not a copy from the solutions web page.
- Content: What does the app do? The added value and benefits should be described and examples be provided.
- The app is running on UCS. What is the added value that the combination of UCS and the app offers to the customer?
- It's important for the user to understand which "edition" of the solution is installed and what features or limitations are included. Please also provide information on how to "upgrade" to the next "edition".
- The text should be structured in paragraphs. Subheadings and lists should be used.
- Search engines should be kept in mind and keywords be used.
- Links should be furnished with all HTML attributes.
- Include supporting images/screenshots and define them in the app configuration (not within description).

The description is provided in HTML format. If more control on the HTML is needed, switch to HTML source mode and edit the HTML directly. Here sections and headings can be added.

Note: Custom styles in CSS should not be used, because they may distract from the overall impression. The App Center UMC module and the App Catalog already have respective CSS style definitions.

The length of the description depends on how much there is to say and how much explanation the app requires. Ideally the description should be at least 300 words long. The text should be structured and paragraphs should be used to make it easier to read. The target group are potential customers. Subheadings (HTML: *h2*, *h3*) should be used to divide the text into logical sections. It's very helpful for the reader to be able to see the advantages of the app and its combination with UCS at a glance. For this reason, presentation of the advantages in lists (HTML: *ol*, *ul*, *li*) is particularly practical.

If links are used in the app description (e.g., to pages on the solution's own website), please always use the `target="_blank"` (open in new tab) and assign the link a title attribute. Please keep the use of links to a minimum and ideally use the fields provided especially for this purpose in the app metadata.

4.4 Categories

On the *Presentation* tab the app can be given one or more categories from a given set in the *App categories* section. Users can filter the app overview in the App Center and in the App Catalog accordingly. App categories help to group apps together by topic and give a better overview for the various apps available.

4.5 Contact

For the users it's important to know who is the producer of the app. For this purpose there is the *Responsibility: Contact information* section on the *Presentation* tab in the App Provider Portal. Please provide product contact information like an email address and a website to the solution. Please also link to a website, where the app provider's support options and pricing is explained to potential customers and place the link in the field Link to website with product support options.

4.6 License

In the *License* section on the *Presentation* tab license information can be defined; for example, a license agreement. This has to be read and accepted by a UCS system administrator before the app is installed. If the text is not accepted, the installation process is aborted and no app is installed. The license agreement is mostly used by app providers for legal information that needs confirmation by the administrator before anything is installed. If such a text is not needed, leave it empty.

UCS system administrators have to register with a valid email address in order to use the App Center. If the app provider configures an email address in the field *E-mail address for App install notifications* it receives information on a daily basis about who installed the app. The App Center UMC module informs the user that the app provider may contact them. App providers can use the address, for example, for lead management.

The last setting is intended to provide the users a rough imagination about the license type of the software. One option best fitting to the solution should be chosen:

- `Empty`: If no value is given, the App Center UMC module and the App Catalog will show the text "Please contact the App provider for further license details".
- `Free commercial use`
- `Free commercial use. Some functions or services are liable to costs.`
- `Liable to Costs with Free Trial`
- `Liable to Costs`

4.7 README for the administrator

In the tab *Additional texts* further information for an app can be provided that show up at certain stages of the app lifecycle. Those README files are also in HTML like the description and content can be provided the same way, see *Description* (page 24). The App Provider Portal describes when each README file shows up.

It's highly recommended to use the README files to show information that should not go in the app description, like for example configuration details, hints before and after an update, etc. Please also keep in mind to provide a proper German translation.

4.8 Translations

All texts, screenshots and videos should be entered in English. Translations to German should be only made in the appropriate field next to the English text. It should be made sure that translations for the texts that have an English version are provided. Otherwise, English text will show up for a user with German language settings.

4.9 Recommended Apps Badge

Apps can be awarded with different badges and are therefore especially highlighted in the App Center. One of those badges is the "Recommended Apps"[20] award for the use in professional environments. Apps with the "Recommended Apps" award meet the below listed quality criteria. The functional scope of the software solution is not evaluated. The award is assigned by the Univention App Center Team and the criteria serve as decision guidelines.

- The app can be installed and uninstalled cleanly and does not alter the UCS system against the rules.
- Univention is not aware of any open security vulnerabilities for the app or the app provider has promised to remedy the vulnerabilities soon. In principle, Univention does not carry out any active security monitoring for apps in the App Center. If Univention becomes aware of security vulnerabilities, the App provider will be informed and a deadline for an update will be agreed upon.

[20] https://www.univention.com/products/app-catalog/?recommended_app=1

- The version of the software solution offered in the App Center is maintained by the app provider.
- If the software solution requires user accounts to identify users, the app uses UCS Identity Management as a source of user accounts.
- The app provider makes updates of its software for the app available regularly and promptly to UCS via the App Center.
- If the app provider offers update paths for its software solution, the app also supports these update paths.
- The app vendor ensures that the app deploys its software solution to new UCS versions within a short period of time, ensuring that administrators can update UCS.
- Commercial support is available for the app.
- The app has been available in the App Center for at least 3 months.
- For the app, there are virtual app appliances that are linked on the app vendor's website for download. This makes commissioning the app on UCS extremely easy.

CHAPTER

FIVE

CONNECTION WITH IDENTITY MANAGEMENT

One of the key features of UCS is the integrated identity management (IDM). With this central identity management, users benefit, among other things, from a single login independent of which services or systems they use. It's highly recommended to integrate the app into the identity management system.

If the app should benefit from the identity management, the flag *The administrator needs to enable users for the app* should be activated in the App Provider Portal on the *Identity management* tab under the *User rights management* section. This extends the IDM by a checkbox and an administrator of the UCS system can activate or deactivate each user individually for the app. The setting can then be found in the Users UMC module and is called *Apps*. It's also possible to make significantly more complex settings. See *User rights management* (page 33) for more details.

5.1 Provisioning

There are different ways in which applications can access provisioning information. The following describes a pull and push-based procedure.

5.1.1 Automatically via LDAP connection (Pull)

UCS stores the user and group information in an OpenLDAP based directory. Thus, the default information can be accessed via the LDAP protocol. Objects are identified by an LDAP filter. The following filter can be used to search for users `(univentionObjectType=users/user)` and for groups the filter `(univentionObjectType=groups/group)` can be used.

If the user activation is used (*The administrator needs to enable users for the app*), the following LDAP filter can be used: `(&(univentionObjectType=users/user)(myappActivated=TRUE))`. The string `myapp` has to be replaced with the *appid*.

The parameters for the LDAP server can be read from the environment variables:

LDAP_SERVER_NAME
: The fully-qualified host name of the OpenLDAP server the app may connect to.

LDAP_SERVER_PORT
: The port of the OpenLDAP server the app may connect to.

LDAP_SERVER_ADDITION
: A list of alternative OpenLDAP servers. These values should be used for failover.

LDAP_BASE
: The base for the whole LDAP database, e.g., `dc=mydomain,dc=intranet` or `o=mydomain <o=mydomain>`.

Important: As a rule, the LDAP base should not be further restricted. Many environments store users below `cn=users <cn=users>` but this is not the case in all environments.

By default, the OpenLDAP server in UCS doesn't allow anonymous connections. For every app a user account is created. The account has read access to the LDAP directory. The username is passed as the environment variable `LDAP_HOSTDN`. The password is written in the file `/etc/machine.secret`. The credentials are not changed when an app is upgraded. But they change if an app is reinstalled.

5.1.2 Automatically via IDM notifications (Push)

An app can be notified by the IDM system when users or groups are created, modified or deleted. For each change, a file is created in a specific directory. The app can either poll the directory or register a command that is executed when a file is created.

Setup in App configuration

The configuration for these IDM notifications can be done on the *Identity management* tab in the *Provisioning* section in the App Provider Portal. It can be configured which object types are watched. Currently, users and groups are supported.

A script should be specified in the App Provider Portal. The script is copied from the App Center into the Docker container and executed in the context of the container there. If a script is already part of the container, this script can be called accordingly, e.g.

```
#!/bin/sh
exec /usr/sbin/app-connector
```

How the mechanism works

The JSON files are created in the directory `/var/lib/univention-appcenter/apps/appid/data/listener/`. As soon as any attribute of the watched object types is changed a JSON file is created in the directory. The script is called in a defined and configurable interval by the App Center, if at least one JSON file has been written. Once the script has finished a JSON file, the script must delete the JSON file.

New in version 5.0-3: `ListenerUDMVersion`

`ListenerUDMVersion` is a custom configuration for an app. It defines the format version, that the listener uses to pass data from UDM to the app. Possible values are `1` and `2`. If `ListenerUDMVersion` isn't defined as custom configuration in the app metadata, the listener uses version `2`.

Univention recommends to use `ListenerUDMVersion` 2, because it uses the UDM HTTP REST API representation.

Configuration in App Provider Portal

To set `ListenerUDMVersion` in the App Provider Portal, open the app of interest and navigate to *Advanced* ▸ *Custom configuration*. Add a new custom configuration with the key `ListenerUDMVersion`.

Migration to `ListenerUDMVersion` 2

For using version `2`, app developers need to compare what data they process and how they handle the representation. In best case, they don't need to adjust the listener integration.

All files are JSON with one dictionary and the following content. You find logging information about the listener in `/var/log/univention/listener_modules/appid.log`.

`id`
: A unique identifier for the object holding the value of `entry_uuid` attribute of the LDAP object. It does not change even if the object is moved. The script certainly wants to identify objects by this attribute.

dn
: The distinguished name of the LDAP object.

udm_object_type
: The type of the object, for example `users/user`, or `groups/group`.

object
: A dictionary of the attributes of this object. If `object` is `null`, the object was deleted.

  The listener passes the data in the UDM HTTP REST API representation to the `object` dictionary.

  For example, refer to *JSON example for ListenerUDMVersion 2* (page 29).

  The listener passes the data in the UDM representation to the `object` dictionary. The representation uses strings for boolean values such as `"OK"`, `"1"`, `"0"`, `"TRUE"`, or `"FALSE"`.

  For example, refer to *JSON example for ListenerUDMVersion 1* (page 31).

What should the script cover?

- The mechanism does not filter the data. Every change will be saved in JSON files. If only a subset of users, e.g. a certain user type like students, shall be processed, the script should filter on it and only continue with the relevant data.
- UCS can re-synchronize a listener. In this case, each and every object appears once again as a JSON file. The script needs to cover the case where no real modification to the object has been made.
- The script has to exit with exit code = 0 on success and != 0 on failure.
- The script has to delete the JSON file that has already been processed. If the files are not deleted, the script should detect duplicates and make sure to handle the same change accordingly.
- If a mapping between the `id` of the JSON file and the primary user key in the solutions database is not possible, consider maintaining a mapping table by the script, if necessary. The `id` is the only attribute that remains the same for an object.
- It may happen that the same `id` appears twice in the set of JSON files. This means that multiple modifications on the object have been made since the last time your script processed the object.

JSON example

This is an example of a JSON file for a user change. It's not complete, but should clarify the idea.

JSON example for `ListenerUDMVersion` 2

```
{
    "dn": "uid=Administrator,cn=users,dc=demo,dc=univention,dc=de",
    "id": "b2f13544-e3cb-1037-810e-23ad4765aade",
    "properties": {
        "accountActivationDate": {
            "activation-date": null,
            "activation-time": null,
            "activation-timezone": "Europe/Berlin"
        },
        "birthday": null,
        "city": null,
        "country": null,
        "departmentNumber": [],
        "description": null,
        "disabled": false,
        "displayName": "Administrator",
```

(continues on next page)

(continued from previous page)


```
        "e-mail": [],
        "employeeNumber": null,
        "employeeType": null,
        "firstname": null,
        "gecos": "Administrator",
        "gidNumber": 5000,
        "groups": [
            "cn=Domain Admins,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Domain Users,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=DC Backup Hosts,cn=groups,dc=demo,dc=univention,dc=de"
        ],
        "homePostalAddress": [],
        "homeShare": null,
        "homeSharePath": "Administrator",
        "homeTelephoneNumber": [],
        "homedrive": null,
        "initials": null,
        "jpegPhoto": null,
        "lastbind": null,
        "lastname": "Administrator",
        "locked": false,
        "lockedTime": "0",
        "mailAlternativeAddress": [],
        "mailForwardAddress": [],
        "mailForwardCopyToSelf": "0",
        "mailHomeServer": null,
        "mailPrimaryAddress": null,
        "mobileTelephoneNumber": [],
        "objectFlag": [
            "hidden"
        ],
        "organisation": null,
        "overridePWHistory": null,
        "overridePWLength": null,
        "pagerTelephoneNumber": [],
        "password": null,
        "passwordexpiry": null,
        "phone": [],
        "physicalDeliveryOfficeName": null,
        "postOfficeBox": [],
        "postcode": null,
        "preferredDeliveryMethod": null,
        "preferredLanguage": null,
        "primaryGroup": "cn=Domain Admins,cn=groups,dc=demo,dc=univention,dc=de",
        "profilepath": null,
        "pwdChangeNextLogin": null,
        "roomNumber": [],
        "sambaLogonHours": null,
        "sambaPrivileges": [],
        "sambaRID": 500,
        "sambaUserWorkstations": [],
        "sambahome": null,
        "scriptpath": null,
        "secretary": [],
        "serviceprovider": [],
        "shell": "/bin/bash",
        "street": null,
        "title": null,
        "uidNumber": 2002,
        "umcProperty": {
            "appcenterSeen": "false",
```


(continues on next page)

(continued from previous page)

```
            "udmUserGridView": "tile"
        },
        "unixhome": "/home/Administrator",
        "unlock": false,
        "unlockTime": "",
        "userexpiry": null,
        "username": "Administrator"
    },
    "udm_object_type": "users/user"
}
```

JSON example for `ListenerUDMVersion` 1

```
{
    "dn": "uid=Administrator,cn=users,dc=demo,dc=univention,dc=de",
    "id": "b2f13544-e3cb-1037-810e-23ad4765aade",
    "object": {
        "description": "Built-in account for administering the computer/domain",
        "disabled": "0",
        "displayName": "Administrator",
        "gecos": "Administrator",
        "gidNumber": "5000",
        "groups": [
            "cn=Domain Admins,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Domain Users,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=DC Backup Hosts,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Schema Admins,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Enterprise Admins,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Group Policy Creator Owners,cn=groups,dc=demo,dc=univention,dc=de",
            "cn=Administrators,cn=Builtin,dc=demo,dc=univention,dc=de"
        ],
        "lastname": "Administrator",
        "locked": "0",
        "lockedTime": "0",
        "mailForwardCopyToSelf": "0",
        "mailPrimaryAddress": "admin@sparka-43.intranet",
        "mailUserQuota": "0",
        "passwordexpiry": null,
        "primaryGroup": "cn=Domain Admins,cn=groups,dc=demo,dc=univention,dc=de",
        "sambaRID": "500",
        "shell": "/bin/bash",
        "uidNumber": "2002",
        "umcProperty": [
            [
                "appcenterDockerSeen",
                "true"
            ],
            [
                "appcenterSeen",
                "2"
            ],
            [
                "udmUserGridView",
                "default"
            ]
        ],
        "unixhome": "/home/Administrator",
        "unlockTime": "",
        "userexpiry": null,
```

(continues on next page)

(continued from previous page)

```
        "username": "Administrator",
        "webweaverActivated": "TRUE"
    },
    "udm_object_type": "users/user"
}
```

5.2 Authentication

There are different ways in which applications can authenticate against the UCS identity management system.

5.2.1 LDAP

UCS stores the user and group information in an OpenLDAP based directory. Thus, the default information can be accessed via the LDAP protocol. Objects are identified by an LDAP filter. The following filter can be used to search for users `(univentionObjectType=users/user)` and for groups the filter `(univentionObjectType=groups/group)` can be used.

If the user activation is used (The administrator needs to enable users for the app), the following LDAP filter can be used: `(&(univentionObjectType=users/user)(myappActivated=TRUE))`. The string `myapp` has to be replaced with the *appid*.

The parameters for the LDAP server can be read from the environment variables:

LDAP_SERVER_NAME
: The fully-qualified host name of the OpenLDAP server the app may connect to.

LDAP_SERVER_PORT
: The port of the OpenLDAP server the app may connect to.

LDAP_SERVER_ADDITION
: A list of alternative OpenLDAP servers. These values should be used for failover.

LDAP_BASE
: The base for the whole LDAP database, e.g., `dc=mydomain,dc=intranet` or `o=mydomain <o=mydomain>`.

Important: As a rule, the LDAP basis should not be further restricted. Many environments store users below `cn=users <cn=users>` but this is not the case in all environments.

By default, the OpenLDAP server in UCS does not allow anonymous authentications. For every app a user account is created. The account has read access to the LDAP directory. The username is passed as the environment variable `LDAP_HOSTDN`. The password is written in the file `/etc/machine.secret`. The credentials are not changed when an app is upgraded. But they change if an app is reinstalled.

5.2.2 Kerberos

UCS integrates a Kerberos server by default. As usual with Kerberos, the data for the Kerberos configuration can be obtained from DNS. By default, the DNS domain name is passed through the `DOMAINNAME` environment variable. The following settings can then be queried via DNS:

Kerberos Realm
: It can be queried by the TXT record `\_kerberos.DOMAINNAME`.

Kerberos KDC
: It can be queried by the SRV records `\_kerberos._tcp.DOMAINNAME` and `\_kerberos._udp.DOMAINNAME`.

5.3 User rights management

The flag *The administrator needs to enable users for the app* can be activated in the App Provider Portal on the *Identity management* tab in the *User rights management* section. This adds a checkbox to the user administration and a schema extension for the IDM is created, so that the status of the checkbox is stored in an attribute in the IDM. This allows each user to be activated or deactivated separately.

If the app requires more settings in the IDM, an own LDAP schema can be uploaded into the App Provider Portal on the *Identity management* tab in the *User rights management* section in the field *Schema extension for LDAP*.

In this case, it's also possible to create individual extended attributes during the setup process. This should be done in the join script. Further information on extended attributes can be found in the Univention Developer Reference[21].

[21] https://docs.software-univention.de/developer-reference/5.2/en/index.html

CHAPTER

SIX

APP CONFIGURATIONS

The App Center offers the possibility to add scripts at various points during the installation and configuration of the app. The scripts are described below. They can be edited in the app in the App Provider portal in the following sections:

- Scripts → Join & unjoin script
- Scripts → Scripts called before App Center action on apps
- Scripts → Scripts for data store and restore (`store_data`, `restore_data_before_setup`, `restore_data_after_setup`)
- Scripts → Setup script
- Scripts → Configure scripts. (`configure`)
- Scripts → Configure scripts. (`configure_host`)

6.1 Installation scripts

During the installation of an app, various scripts are called. Please see the overview below about of the involved scripts and the parameters they are called with. More information on the scripts themselves can be found in the following sections.

During installation process you may need to run commands inside a container of your app, for example, to properly prepare it. For more information, refer to *Command univention-app* (page 61).

6.1.1 Script called before installation to verify that App may be installed

The `preinst` script is executed on the UCS host system before the app is initialized, even before the app image is downloaded. Its purpose is to check whether installation will be successful or not. Any exit code other than 0 will result in cancellation of the installation process. This script is also executed if the app is upgraded.

The script receives the LDAP bind DN of the Administrator account and its password, the version of the app that should be installed, the locale and an error log file for log output as parameters. Error messages in the passed error log file will be passed to the UCS management system and thus to the administrator performing the installation. Proper error messages can thus be passed to the administrator.

Note that *App settings* (page 44) with scope `outside` are already set by the time the script is run and can therefore be checked against.

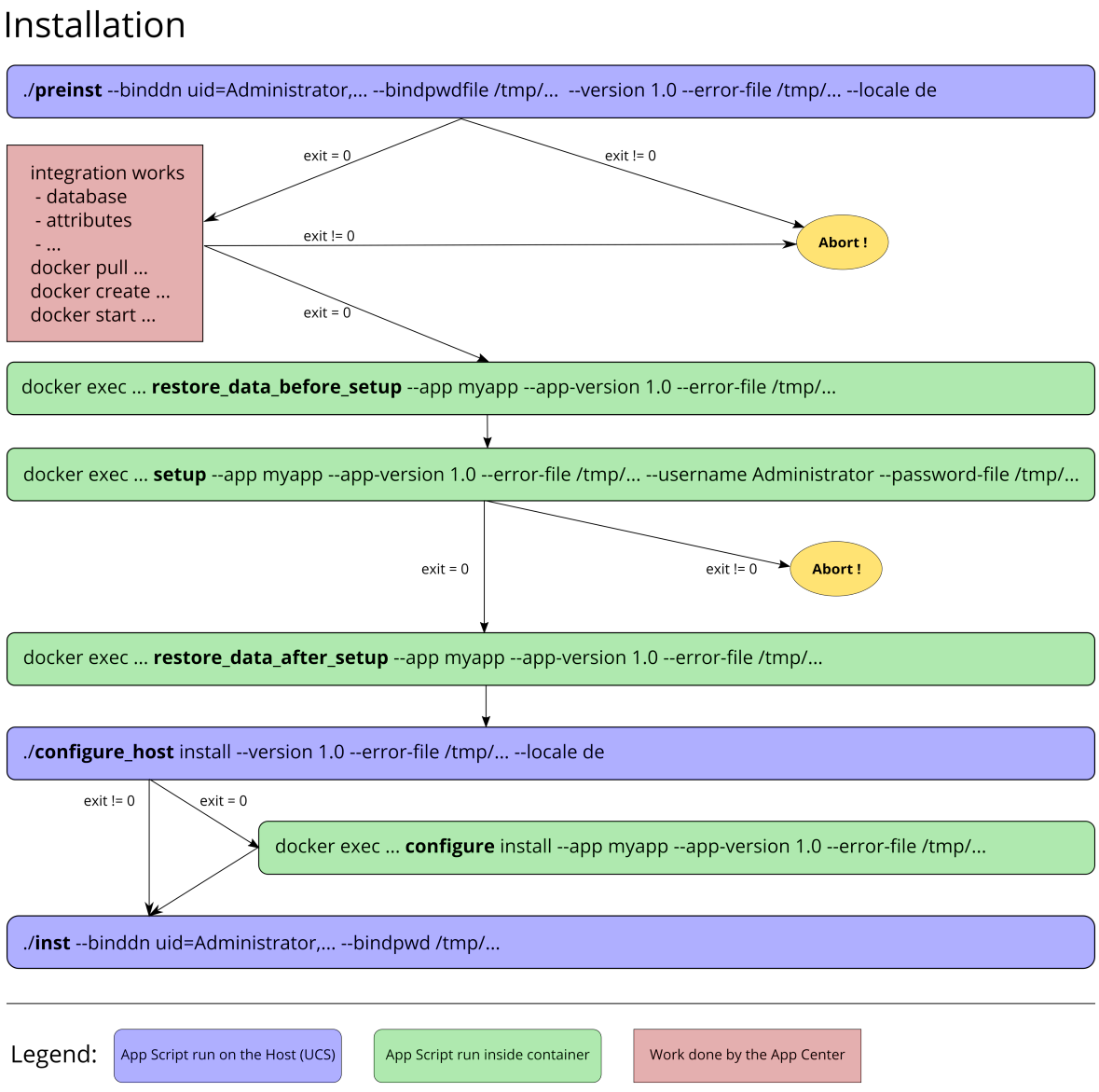

Fig. 6.1: App workflow for installation

6.1.2 Docker script restore_data_before_setup

The lifecycle script `restore_data_before_setup` is executed inside the Docker container before the `setup` script is run. Its purpose is to restore the data which has been stored by the `store_data` script. The parameters are the *appid*, the app version and a filename for error logging.

6.1.3 Docker script setup

The lifecycle script `setup` is executed inside the Docker container. It's the heart of the initial app configuration and typically used to make environment specific settings to the container or apply certain changes that require the container environment. If the script fails (`exit code != 0`) the installation is aborted.

The parameters given to the script are the *appid*, the app version, a filename for error logging and the username and credentials for the Administrator user.

6.1.4 Docker script restore_data_after_setup

The lifecycle script `restore_data_after_setup` is executed inside the Docker container after the `setup` script is run. Its purpose is to restore the data which has been stored by the `store_data` script. The parameters are the *appid*, the app version and a filename for error logging.

6.1.5 Settings script run on Docker host

The settings script `configure_host` is executed on the Docker host after the `restore_data_after_setup` script is run. Its purpose is to make environment specific settings on the UCS host regarding the app. The parameters are the app action `install`, the app version, a filename for error logging and the locale.

6.1.6 Settings script run in Docker container

The settings script `configure` is executed inside the Docker container after the `configure_host` script. Its purpose is to make environment specific settings in the app container. The parameters are the app action `install`, the *appid*, the app version and a filename for error logging.

6.1.7 Join script

The joinscript `inst` is executed on the UCS host system after the Docker container is configured. Please refer to Domain join[22] in the UCS Developer Reference about how to write a join script. In principle a join script is a script that runs after the installation of an app and it has write access to the LDAP directory service. If it runs successfully, the join script may save this information in a status file. If this does not happen, the user is constantly reminded to re-run the join script. So the join script does not need to run successfully. The installation will not be aborted at this point. But of course at some point it should run through successfully.

[22] https://docs.software-univention.de/developer-reference/5.2/en/join/index.html#chap-join

Join script helper

Apart from the functions documented in the Developer Reference, the below listed functions are available in join scripts for dealing with apps. They require the following line in the script:

```
. /usr/share/univention-appcenter/joinscripthelper.sh
```

Furthermore, this call provides access to the following variables:

$APP
app id

$APP_VERSION
app version

$SERVICE
app name

$CONTAINER
Docker container id

Join script functions

joinscript_add_simple_app_system_user
Adds a domain wide user to the LDAP directory that is not a real Domain User and offers an authentication account. It can be used as bind user for the app to connect to the LDAP directory. The password will be stored on the Docker Host at `/etc/APP.secret`. The DN will be `uid=APP-systemuser,cn=users,ldap_base`.

```
joinscript_add_simple_app_system_user "$@" --set mailPrimaryAddress=...
```

joinscript_container_is_running
Returns whether or not the Docker container is currently running.

- 0: Yes
- 1: No

Can be used in an if statement.

```
joinscript_container_is_running || die "Container is not running"
```

joinscript_run_in_container
Runs one command inside the container. Returns the return code of the command.

```
joinscript_run_in_container service myapp restart ||
die "Could not restart the service"
```

joinscript_container_file
Prints the absolute path for the Docker host for the filename given inside the container.

```
FILENAME="$(joinscript_container_file "/opt/$APP/my.cnf")"
```

joinscript_container_file_touch
Creates a file inside the container. Directories are created along the way. Prints the resulting filename just like "joinscript_container_file".

joinscript_register_schema
Registers a LDAP schema file semi automatically. The schema file allows to extend LDAP objects with new attributes. The file will be copied to the Docker host's `/usr/share/univention-appcenter/apps/APPID/APPID.schema` during installation. See the LDAP documentation[23] for the syntax of a schema file.

[23] https://www.openldap.org/doc/admin24/schema.html

If an official object identifier (OID) namespace is needed, Univention can provide one. It's important to note that shipping the schema file alone is not enough. It has to be registered with the mentioned function in the join script.

The schema file content can be provided in the App Provider portal on the *Identity management* tab in the *User rights management* section, in the field for *Schema extension for LDAP*.

```
joinscript_register_schema "$@"
```

Join script boilerplate

The following boilerplate can be used as a starting point for the app's own join script.

```
#!/bin/bash
VERSION=1

. /usr/share/univention-appcenter/joinscripthelper.sh
joinscript_init
eval "$(univention-config-registry shell)"
ucs_addServiceToLocalhost "$SERVICE" "$@" || die

... # Place for the app's join script code

joinscript_save_current_version
exit 0
```

6.2 Uninstall scripts

During the process to uninstall an app, various scripts are called. Please see the overview below about the involved scripts and the parameters they are called with. More information on the scripts themselves can be found in the following sections.

6.2.1 Script called before uninstalling to verify that App may be removed

The `prerm` script is executed on the UCS host system. Its purpose is to check the prerequisites to uninstall an app or abort if they are not met. For example, the `prerm` may fail if other software still depends on it. Any exit code other than `0` will result in cancellation of the uninstall process. The given parameters are the LDAP bind DN of the Administrator account and its password, the version of the app that should be uninstalled/removed, the locale and an error log file for log output. Error messages in the passed error log file will be passed to the UCS management system and thus to the administrator performing the installation. Proper error messages can thus be passed to the administrator.

6.2.2 Settings script run on Docker host

The settings script `configure_host` is executed on the Docker host after the `prerm` script is run. Its purpose is to make environment specific settings on the UCS host during the removal of the app. The parameters are the app action `remove`, the app version, a filename for error logging and the locale.

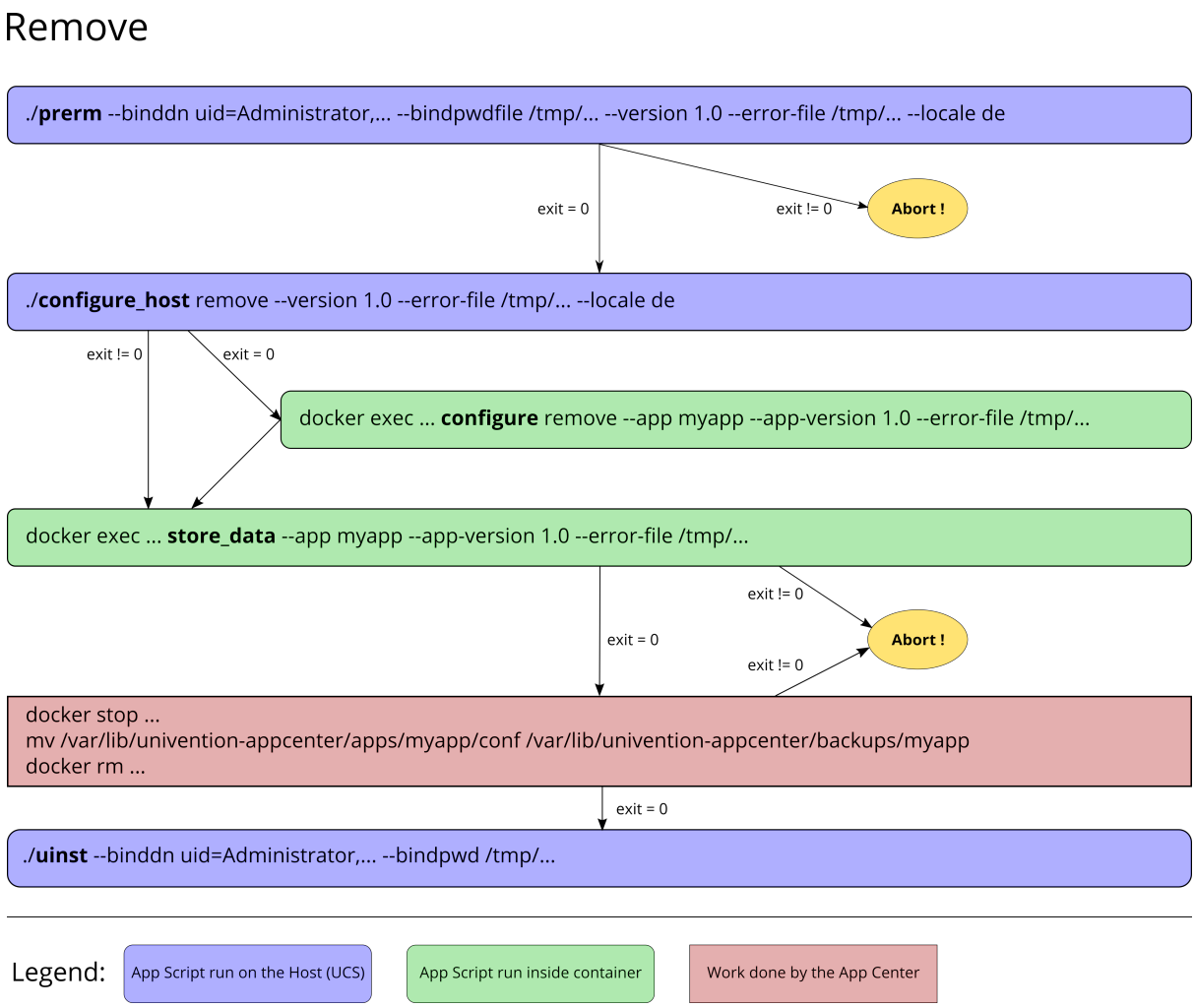

Fig. 6.2: App workflow for Removal

6.2.3 Settings script run in Docker container

The settings script `configure` is executed inside the Docker container after the `configure_host` script. Its purpose is to make environment specific settings in the app container before it's removed. The parameters are the app action `remove`, the *appid*, the app version and a filename for error logging.

6.2.4 Docker script store_data

The lifecycle script `store_data` is required if data exists in the container which should not be removed when the container is replaced with a new container or if the app is uninstalled. The script is not required if all the data is stored outside of the container for example in a database or a mapped volume. It's executed inside the Docker container and it should copy the relevant data to `/var/lib/univention-appcenter/apps/APPID/data/`. Afterwards, the data can be restored by one of the `restore_data*` scripts. The parameters are the *appid*, the app version and a filename for error logging.

6.2.5 Unjoin script

The unjoin script `uinst` is executed on the UCS host system after the Docker container is removed. See Writing unjoin scripts[24] in UCS Developer Reference for how to write an unjoin script. It should revert most (if not all) changes done in the join script. With the notable exception of schema registration. An LDAP schema extension should never be removed once it was registered.

6.3 Upgrade scripts

It may be necessary to move data from the old container to the new container when the app container is replaced during an upgrade or when the app is uninstalled. The upgrade scripts can be used for this purpose. Please see an overview of the involved scripts and the parameters they are called with in the figure below. More information on the scripts themselves can be found in the following sections.

During upgrade process you may need to run commands inside a container of your app, for example, to run a data migration. For more information, refer to *Command univention-app* (page 61).

6.3.1 Script called before upgrade to verify that App may be upgraded

The `preinst` script is executed on the UCS host system before the app upgrade is initialized, even before the Docker image is downloaded. Its purpose is to check whether the requirements for the upgrade are fulfilled. Any exit code other than 0 will result in cancellation of the upgrade process.

The script receives the LDAP bind DN of the Administrator account and its password, the old version of the app and the new version, the locale and an error log file for log output as parameters. Error messages in the passed error log file will be passed to the UCS management system and thus to the administrator performing the installation. Proper error messages can thus be passed to the administrator.

[24] https://docs.software-univention.de/developer-reference/5.2/en/join/write-unjoin.html#join-unjoin

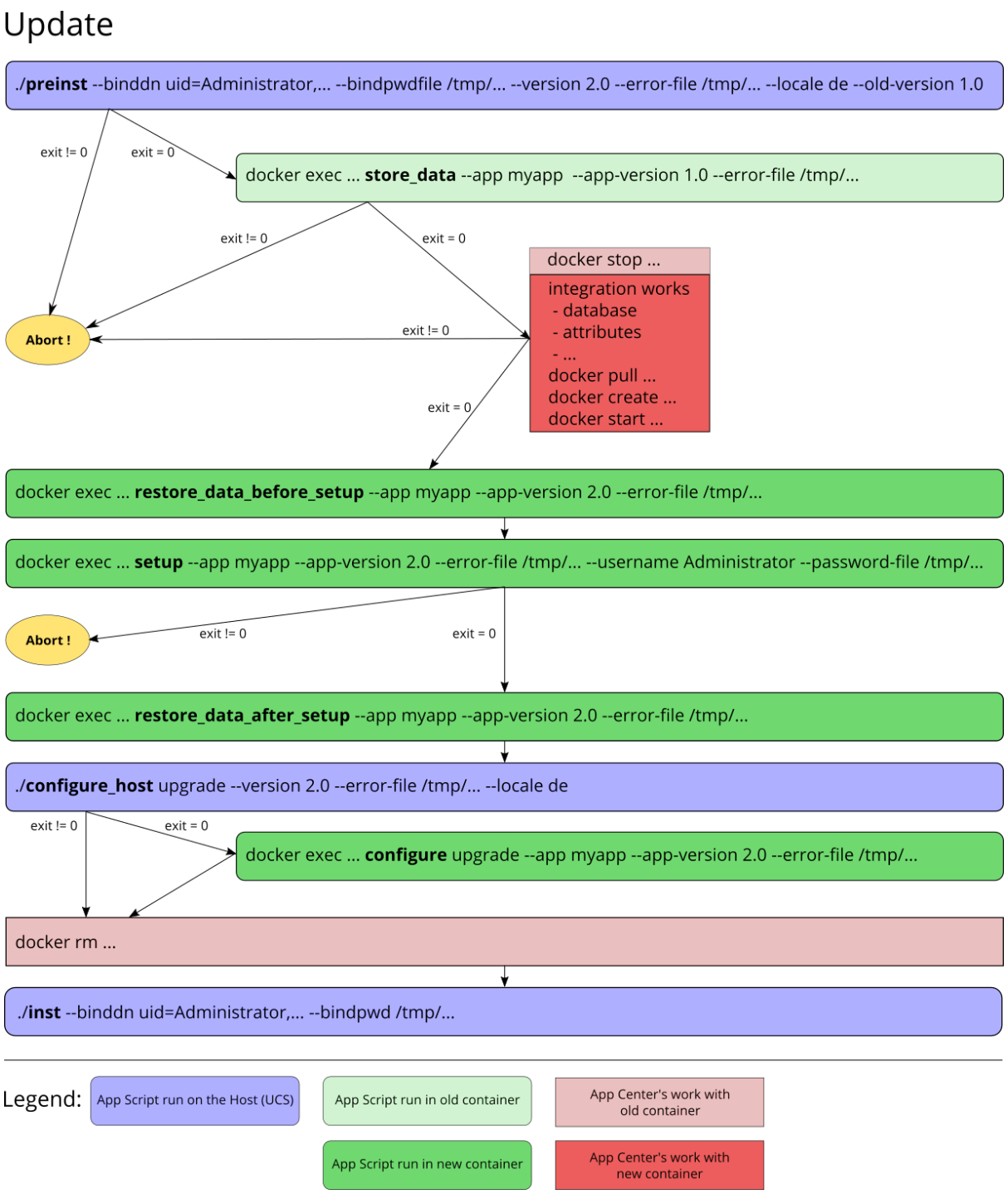

Fig. 6.3: App workflow for upgrade

6.3.2 Docker script store_data

The lifecycle script `store_data` is required if data exists in the container which should not be removed when it's replaced with a new container or if the app is uninstalled. It isn't required if all the data is stored outside the container for example in a database or a mapped volume. The script is executed inside the Docker container and it should copy the relevant data to `/var/lib/univention-appcenter/apps/APPID/data/`. Afterwards, the data can be restored by one of the `restore_data*` scripts when they are executed in the new container.

6.3.3 Docker script restore_data_before_setup

The lifecycle script `restore_data_before_setup` is executed inside the Docker container before the `setup` script is run. Its purpose is to restore the data which has been stored by the `store_data` script.

6.3.4 Docker script setup

The lifecycle script `setup` is executed inside the Docker container. 't is used to make environment specific settings to the new container or apply certain changes that require the container environment. If the script fails (`exit code != 0`) the upgrade is aborted.

The parameters given to the script are the *appid*, the app version, a filename for error logging and the username and credentials for the Administrator user.

6.3.5 Docker script restore_data_after_setup

The lifecycle script `restore_data_after_setup` is executed inside the Docker container after the `setup` script is run. Its purpose is to restore the data which has been stored by the `store_data` script in the old container.

6.3.6 Settings script run on Docker host

The settings script `configure_host` is executed on the Docker host after the `restore_data_after_setup` script is run. Its purpose is to make environment specific settings on the UCS host regarding the app during the upgrade. The parameters are the app action `upgrade`, the app version, a filename for error logging and the locale.

6.3.7 Settings script run in Docker container

The settings script `configure` is executed inside the Docker container after the `configure_host` script is run. Its purpose is to make environment specific settings in the app container during the upgrade. The parameters are the app action `upgrade`, the *appid*, the app version and a filename for error logging.

6.3.8 Join Script

Finally, the join script `inst` is called to end the upgrade. With an updated join script changes can be made to the environment that require the necessary execution permissions or access to the UCS directory service. When a join script should run during the upgrade, please keep in mind to increment the `VERSION` counter. For more information on the join script in general see *Join script* (page 37).

6.4 App settings

The App settings allow the user to configure the app during its runtime. The App Provider Portal can be used to define which settings are displayed to the user. The app can react accordingly to the changes.

If App settings are defined for an app, the user can reach these settings in the app configuration, see *App settings button* (page 44).

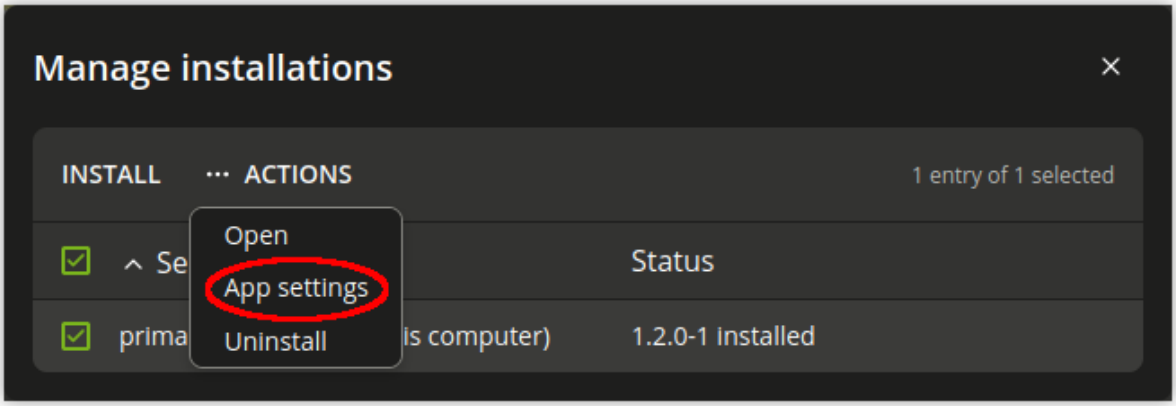

Fig. 6.4: App settings button

An example for an App settings dialog is in *App settings example* (page 45).

The App settings can be defined on the tab *Advanced* in the section *App settings* in the App Provider Portal.

6.4.1 React on App settings

The settings are saved inside the Docker container in the file `/etc/univention/base.conf` in the format *key: value*. After the settings are changed, two scripts are executed. First, the script `configure_host`. This script is run on the Docker host. Second, the script `configure` is executed. It's executed inside the Docker container. In the App Provider Portal, the path of the script can be given (*Configure scripts*) or the script code can be uploaded (*Path to script inside the container (absolute)*).

6.4.2 App settings configuration

The App settings are defined in the ini format. The definition can be done in the field *Settings* that can be used to configure the app ini file format. One ini file can contain several settings.

The name of a setting is the name of the section in the ini file, for example

```
[myapp/mysetting]
```

It's recommended to use the app ID as a prefix to prevent collisions.

The type of the attribute is defined with the keyword *Type*. The following types are supported:

`String`
: A standard input field with no restrictions. This is used by default.

`Int`
: A number field which is validated accordingly.

`Bool`
: A checkbox. The value `true` or `false` is set.

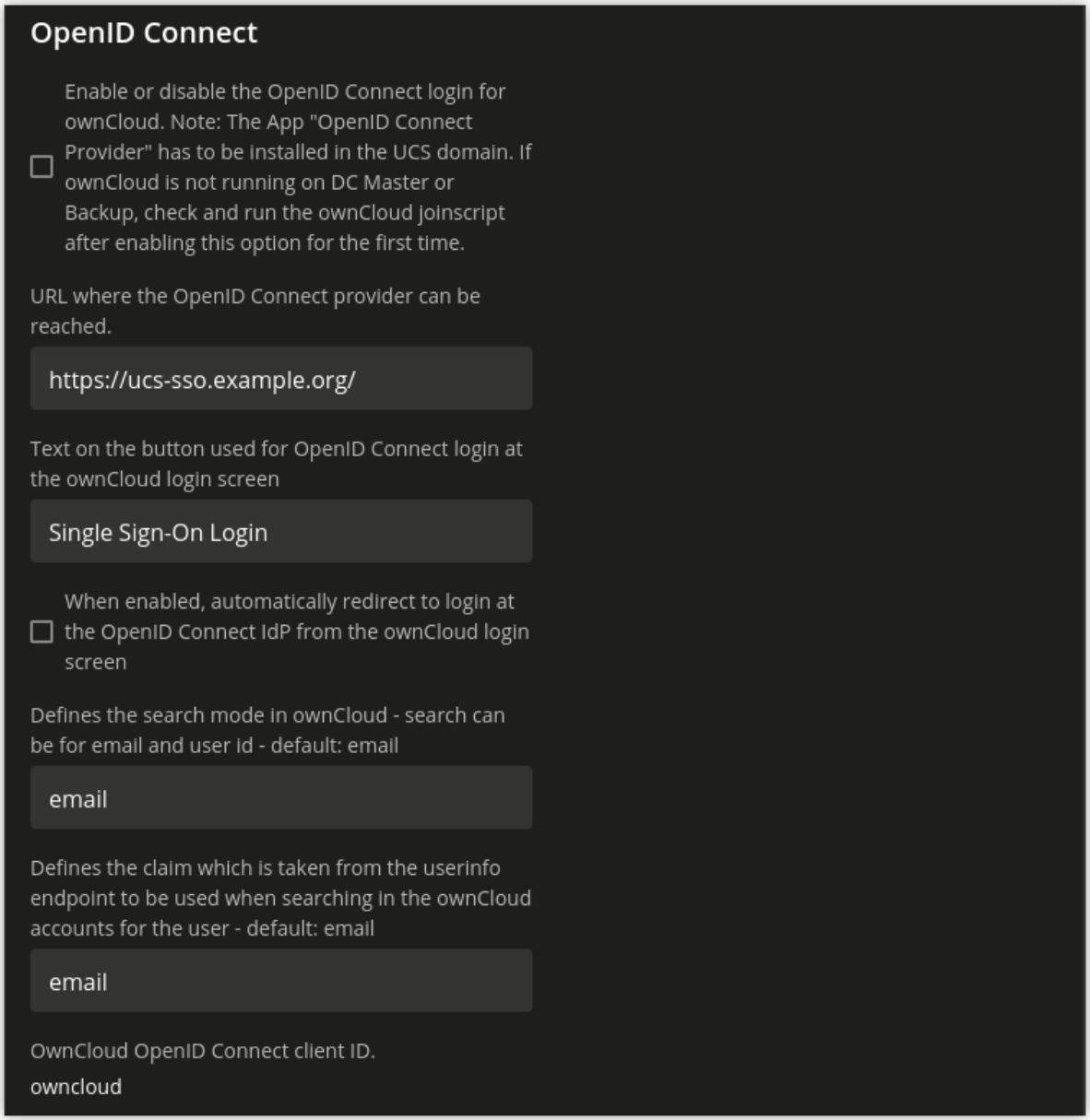

Fig. 6.5: App settings example

List
A widget that lets the user choose from a predefined set of values.

Password
A password input.

> **Note:** The content will be stored as clear text value inside the Docker container.

File
An upload widget. The content is stored directly in a file according to the definition of the setting.

PasswordFile
As a File, but shown as a password input.

Status
A read-only settings that is actually meant as a feedback channel for the user. This does not render a widget, but instead just writes a text with whatever was written into this variable. Writing to it's up to the App Provider (e.g., by using the configure script).

The attribute `Description` is used to define the description of the setting. It's shown next to the widget so that the user knows what to do with this form. It can be localized by also defining `Description[de]` and so on.

The attribute `Group` can be used to group settings. All settings sharing one group will be put under that label. The default group is `Settings`. It's also possible to localize it for example `Group[de]`.

The attribute `Show` can be used to define when the setting should be shown. By default the setting attribute is shown when the app is up and running. It's also possible to show the setting attribute during the installation. The following values are possible `Install`, `Upgrade`, `Remove` and `Settings`. It's possible to specify more than one value which must be separated by comma.

The attribute `ShowReadOnly` can be used in the same way as `Show`. The difference is that the value is not changeable.

The attribute `InitialValue` can be used during the installation. If no value for this attribute was given during the installation, the defined value is set.

The attribute `Required` can be used to define if this setting has to be set or not.

The attribute `Scope` is used to specify if the value is set inside the Docker container (`inside`), on the Docker host (`outside`) or on both (`inside, outside`). The default is `inside`. Values in the scope `inside` can be referenced in the `docker-compose.yml` for multi container apps just like Univention Configuration Registry Variables. For an example see *Post processing of Docker Compose file* (page 10).

The attributes `Labels` and `Values` are used if a type List is defined. The attribute `Labels` defines the values shown to the user and the attribute `Values` defines the values which are stored. The lists are comma separated and should have the same size. If a comma is necessary inside a label or value, it can be escaped with a `\\`.

The attribute `Filename` can be used to define the absolute path where the file should be stored. This attribute is needed in case the types `File` or `PasswordFile` are used.

6.4.3 App settings examples

This is a minimal settings definition:

```
[myapp/mysetting]
Type = String
Description = This is the description of the setting
Description[de] = Das ist die Beschreibung der Einstellung
```

These are two more advanced settings

```
[myapp/myfile]
Type = File
Filename = /opt/myapp/license
Description = License for the App
Description[de] = Lizenz der App
Show = Install, Settings
Group = License and List
Group[de] = Lizenz und Liste
```

```
[myapp/list]
Type = List
Description = List of values
Show = Install
ShowReadOnly = Settings
Values = value1, value2, value3
Labels = Label 1, Label 2, Label 3
InitialValue = value2
Scope = inside, outside
Group = License and List
Group[de] = Lizenz und Liste
```

The first of these two settings will upload a file to `/opt/myapp/license` inside the container. The second will save *myapp/list: value2* (or another value) inside the container and on the Docker host. Both settings will be shown before the installation. On the App settings page, the list setting will be read-only.

6.5 Certificates

UCS provides a certificate infrastructure for secure communication protocols. See SSL certificate management[25] in the UCS manual.

Apps may need access to the UCS certificate infrastructure or need to be aware of changes to the certificates. The Univention App Center provides a simple way to manage certificates inside an app. The script **`update-certificates`** is executed on the UCS host automatically during the installation and upgrade of apps (but can also be executed manually) and provides apps a simple way to gain access to certificates and to react to changes to certificates.

```
# update all apps
univention-app update-certificates

# update app "my-app"
univention-app update-certificates my-app
```

What happens with `update-certificates`?

- The UCS root CA certificate is copied to `/usr/local/share/ca-certificates/ucs.crt` inside the container.
- update-ca-certificates is executed in the Docker container, if it exists, to update the CA certificate list.
- The UCS root CA certificate is copied to `/etc/univention/ssl/ucsCA/CAcert.pem` inside the container.
- The Docker host UCS certificate is copied to `/etc/univention/ssl/docker-host-certificate/{\cert.perm,private.key}` and `/etc/univention/ssl/`*`FQDN_DOCKER_HOST`*`/{cert.perm,private.key}`.

Every app can define a `update_certificates` script. In the app provider portal it can be added on the tab *Advanced* in the section *Certificates*.

Example:

[25] https://docs.software-univention.de/manual/5.2/en/domain-ldap/ssl.html#domain-ssl

```
#!/bin/sh
# cat the UCS root CA to the app's root CA chain
cat /etc/univention/ssl/ucsCA/CAcert.pem >> /opt/my-app/ca-bundle.crt
service my-app-daemon restart
```

The script has to be uploaded via the upload API (section *App Provider Portal upload interface* (page 20)). The script should be written locally and then uploaded with the following command:

```
./univention-appcenter-control upload --username "$your_username" 5.2/myapp=1.0␣
↪update_certificates
```

6.6 Mail integration

Univention Corporate Server (UCS) provides a complete mail stack with the *Mailstack* app in the App Center. It includes Postfix as *MTA* for SMTP and Dovecot for IMAP. If the app relies on an existing mail infrastructure, it's one option to use the mail stack app and require its installation in the UCS domain. This can be configured for the app in the App Provider portal on the *Version* tab in the section *Required apps* by adding the *Mailserver* app and setting `Installed in domain`. With this configuration the App Center on the system administrator's UCS system will check, if the *Mailserver* app is installed somewhere in the domain and asks the administrator to install it accordingly.

Next the app needs to be configured to use the UCS SMTP and IMAP servers. This is done in the Join Script, see *Join script* (page 37). The following snippet gives an example what should be included in the Join Script:

```
...
eval "$(univention-config-registry shell)"
...
# use the first IMAP server as smtp and imap server
mailserver="$(univention-ldapsearch -LLL '(univentionService=IMA)' cn |
sed -ne 's/^cn: "//p;T;q')"
if [ -n "$mailserver" ]; then
  mailserver="$mailserver.$domainname"

  # for Docker Apps the helper script joinscript_run_in_container
  # can be used to run commands in the container
  . /usr/share/univention-appcenter/joinscripthelper.sh
  joinscript_run_in_container my-app-setup --config imap="$mailserver"
  joinscript_run_in_container my-app-setup --config smtp="$mailserver"
  joinscript_run_in_container my-app-setup --config sieve="$mailserver"
fi
...
```

The snipped searches the UCS LDAP directory for the host with the service IMAP and sets the FQDN of this host as IMAP, SMTP and SIEVE server for the app. This is a good default and may not be correct for some setups.

The best practice mail settings when the UCS mail stack is used, are the following.

IMAP:

- TLS
- Port 143
- Authentication is possible for domain users with a primary mail address.
- The user's uid or the primary mail address are both valid for authentication.

SMTP:

- TLS
- Port 587 (submission) for authentication
- Mechanism Login or Mechanism Plain

6.6.1 Provide mail with Docker Apps

For the intended app it may be necessary to provide SMTP and IMAP with a custom setup for the app. To provide SMTP and/or IMAP services in a Docker app, these services have to be stopped on the Docker host. This can be done in the app's preinst Docker script, see *Script called before installation to verify that App may be installed* (page 35). Example:

```
#!/bin/sh

# stop imap/smtp on docker host
systemctl stop postfix dovecot
ucr set postfix/autostart=no dovecot/autostart=no
```

To map SMTP and/or IMAP ports from the container to the host to be able to use the Docker host as IMAP/SMTP server exclusive ports for the container have to be set to the relevant ports (e.g. `110`, `143`, `993`, `995`, `587`, `25`, `465`, `4190` for *POP3(S)*, *IMAP(S)*, *SMTP(S)*, submission and sieve). See *Ports* (page 8) on how to set an exclusive port.

Firewall exceptions for these ports are create automatically.

Best practice is to at least map the IMAP data store to the Docker host to provide a separation of data and container (important for migration to Docker and Docker image updates). See *Persistent data with volumes* (page 7).

6.6.2 Use local mail on Docker host

With a stopped Postfix on the Docker host, mail can no longer be delivered locally. If that is a problem, the following setup can help.

Install the *extremely simple MTA* **`ssmtp`** and configure this MTA to use the `localhost` (our Docker container is listening on `localhost:25`).

```
univention-install --yes ssmtp
# add mailhub=localhost:25 in to /etc/ssmtp/ssmtp.conf
```

Now configure Postfix in the Docker container to deliver mails from the Docker host locally by adding the FQDN of the Docker host to `mydestination`:

```
ucr set mail/postfix/mydestination="\$myhostname, localhost.\$mydomain, localhost,
↪$DOCKER_HOST_NAME"
```

6.7 Subdomains / dedicated FQDN for an App

There may be reasons why an App needs to have its own FQDN within the UCS domain. Some Apps may not be able to configure a web interface that integrates well into the default Apache sites of UCS, see *Web interface* (page 7).

To avoid naming collisions, the App's FQDN should reference the Docker Host's FQDN, e.g, `myapp.ucs-primary.domain.tld`. UCS can do the following to allow this scenario to work as smooth as possible:

- Add a dedicated FQDN for the App and make it known to the internal DNS. That means that the new FQDN is an alias for the actual FQDN of the Docker host.
- Generate a certificate for this FQDN. Technically, a wildcard certificate is created.
- Generate a virtual host for Apache with that new FQDN. Thus, requests to that FQDN will be handled by the *VHost*. The skeleton configuration can be easily extended by writing a configuration file that is then included in the *VHost* entry.

For this to work, this snippet can be used in the join script (*Join script* (page 37)):

```
univention-add-vhost \
    "myapp.$(ucr get hostname).$(ucr get domainname)" 443 \
    --conffile /var/lib/univention-appcenter/apps/myapp/data/apache.conf \
    "$@"  # "$@" is used to pass credentials
# write the apache.conf, maybe by using the App Settings
systemctl reload apache2
nscd -i hosts  # only needed if the new fqdn should be used immediately by the␣
↪system
systemctl reload named  # same here
```

This will create the following entry in `/etc/apache2/sites-available/univention-vhosts.conf`

```
# Virtual Host for myapp.ucs-primary.domain.tld/443
<IfModule mod_ssl.c>
<VirtualHost *:443>
    ServerName myapp.ucs-primary.domain.tld
    IncludeOptional /var/lib/univention-appcenter/apps/myapp/data/apache.con[f]
    SSLEngine on
    SSLProxyEngine on
    SSLProxyCheckPeerCN off
    SSLProxyCheckPeerName off
    SSLProxyCheckPeerExpire off

    SSLCertificateFile /etc/univention/ssl/*.ucs-primary.domain.tld/cert.pem
    SSLCertificateKeyFile /etc/univention/ssl/*.ucs-primary.domain.tld/private.key
    SSLCACertificateFile /etc/univention/ssl/ucsCA/CAcert.pem
</VirtualHost>
</IfModule>
```

Note: Although this seems convenient for some Apps, this feature creates an *internal* name. It may still be inconvenient for testers that run UCS in a virtual environment where their browser is not part of UCS' DNS.

Warning: This method may not work in the "AD member mode". There, a Windows Domaincontroller is the leading system and provides the DNS. The DNS alias has to be added by the Admin manually there as our script cannot add it for them.

6.8 Firewall

This section describes how the local Univention Firewall based on **`iptables`** is changed by apps and how it can be customized. Docker containers have access to the Docker host. And the Docker containers can be made available for external clients with *Ports redirection* settings, see *Ports* (page 8).

If MariaDB or PostgreSQL are used as database, those ports will be opened automatically for the Docker container (section *Database* (page 8)).

Every app can provide additional custom rules to open required ports. This can be done in the join script (section *Join script* (page 37)). In the example the port 6644 is opened for TCP and UDP:

```
$ univention-config-registry set \
  "security/packetfilter/package/$APP/tcp/6644/all=ACCEPT" \
  "security/packetfilter/package/$APP/tcp/6644/all/en=$APP" \
  "security/packetfilter/package/$APP/udp/6644/all=ACCEPT" \
  "security/packetfilter/package/$APP/udp/6644/all/en=$APP"

$ systemctl try-restart univention-firewall
```

Please also add corresponding **ucr unset** commands in the unjoin script so that the firewall rules will be removed when the app is removed from the system (section *Unjoin script* (page 41)).

CHAPTER

SEVEN

APP APPLIANCES

App Appliances are pre-defined images which consist of the App, the UCS management system and the UCS runtime environment. They are run as a virtual machine within a hypervisor and are currently provided as VMware, VMware ESXi, VirtualBox and KVM images. By default the UCS branding is used, but it's possible and recommended to use a custom branding.

7.1 Create an app appliance

To create an appliance, select an app version that has already been published and activate the *Create App appliance* checkbox on the *Appliance* tab. If the solution needs a minimum size of memory, please specify the needed mega bytes.

7.1.1 Additional software

If the appliance should include additional apps, please specify them in the *Additional software* section.

7.1.2 Customize setup wizard in appliance

The appliance allows customization of the UCS setup wizard and controls which setup pages and setup fields should be hidden. For simplicity towards the user, it's recommended to hide the `software` page.

7.1.3 Customize app listing in App Center

The listing of apps in the App Center UMC module in the appliance can be customized to either whitelist or blacklist certain apps. For example, if the solution is a groupware, other groupware solutions can be hidden from the overview listing. System administrators can only install the whitelisted apps or are not allowed to install the black listed apps.

7.1.4 First steps information

The appliances are usually configured in such a way that the user can start using them right away. In some cases it may be necessary to provide some information for the first steps. For example, the user needs to know that a user object has to be created and activated for the app first. This could be briefly described in this section. The German translation should be kept in mind and provided.

7.1.5 Customize UMC favorite category

The appliance also allows to customize the UMC modules which should be pre-configured for the favorites section in the UCS management system. The favorites section comes up first, after a UCS system administrator logs onto the UCS management system. It's recommended to have the modules `Users`, `Groups` and `App Center` listed here.

7.1.6 Appliance build

As soon as the settings are made, *Save*, click the *Approve for release* button and provide a custom message to let the Univention team know that an appliance is ready to be build. This will create a ticket which helps to keep the communication in one place.

7.1.7 Test and release

The appliance is automatically built in the Univention build infrastructure. After the build is finished, automatic tests will be started. Build and testing will approximately need four hours.

After the automatic tests have finished successfully, the app provider is informed. A link to the appliance download is sent and a few days are given for testing. If no veto is sent, the Appliance is usually published after the veto deadline. After the release, the appliance will show up for download on the app page in the App Catalog. App Providers are recommended to also place a link to the app detail page from their download page. The link is one criteria for the recommended apps badge in the App Center overview and the App Catalog.

7.2 Custom branding

With a customized branding of an appliance the boot loader, the boot splash, the system setup wizard and the portal page can be modified. Please look at the screenshots below and the explanations of the options that control the look.

7.2.1 Boot loader

The background color can be configured for the boot loader. Please define in the *Primary color* setting.

7.2.2 Boot splash

The boot splash can have a custom background and a logo. The logo is defined in *Logo for the bootsplash during system boot*. Please provide a SVG file and mind the recommendations in *Logos* (page 23).

The background color is defined in *CSS definition of the appliance background in bootsplash & welcome screen*. For a black background, simply define `#000000`. A gradient can for example be defined with `linear-gradient(to bottom, #345279 0%, #1d2c41 100%)`. For more information on how to use a gradient, see CSS linear-gradient() function on w3schools[26].

[26] https://www.w3schools.com/cssref/func_linear-gradient.php

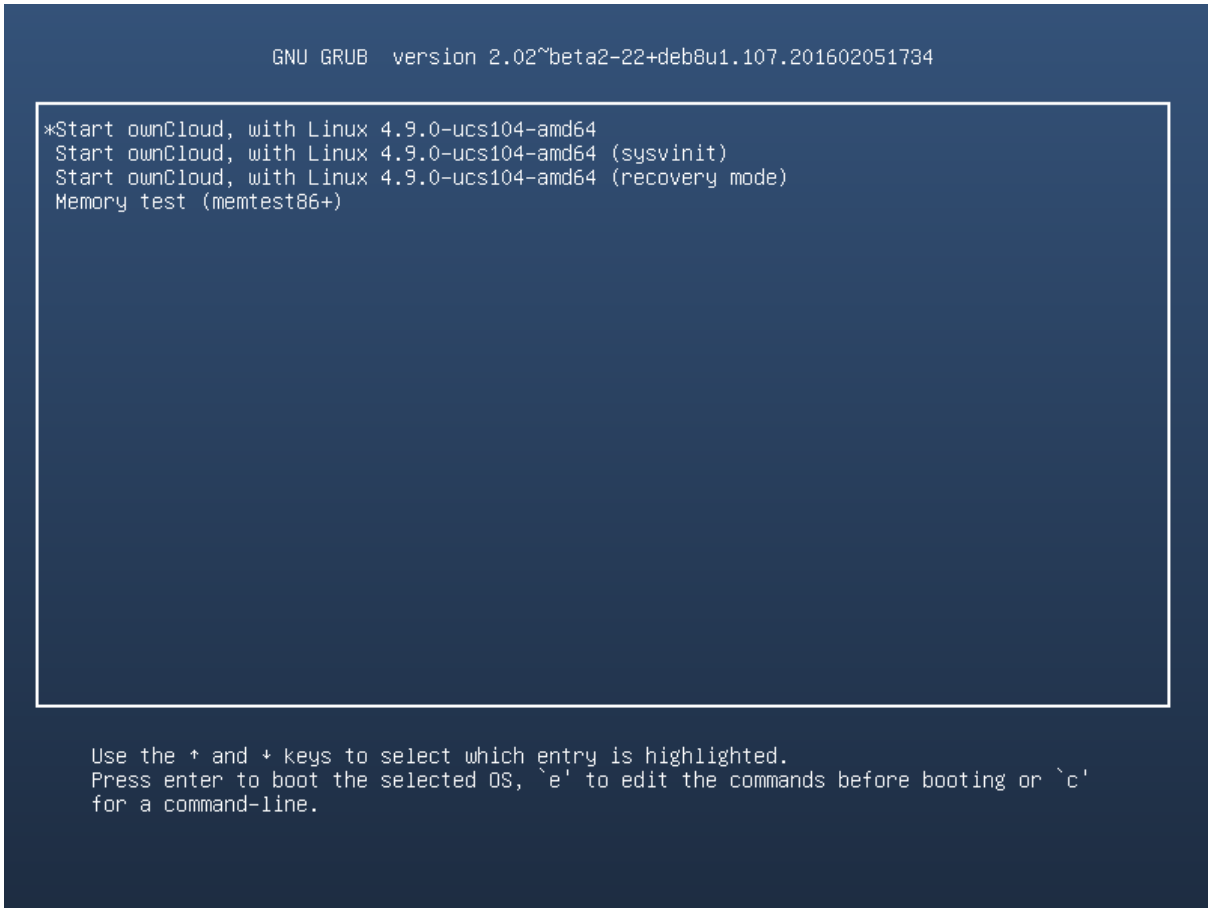

Fig. 7.1: Boot loader

Fig. 7.2: Boot splash

7.2.3 System setup wizard

The system setup wizard allows several slots to be customized. In the *Configuration* section in the App Provider Portal, the `Appliance name` (the word "appliance" is automatically appended) can be set. It controls the heading in the system setup wizard. The *Logo for the first page of the setup wizard* shows up on the first page of the system setup wizard.

In the *Branding* section, the logo on the top left can be changed with the *Logo for header in setup wizard* setting. The *Primary color* controls the background color of the UMC header.

The *Secondary color* is used as color for smaller graphical elements throughout the setup wizard, see *System setup summary screen* (page 58).

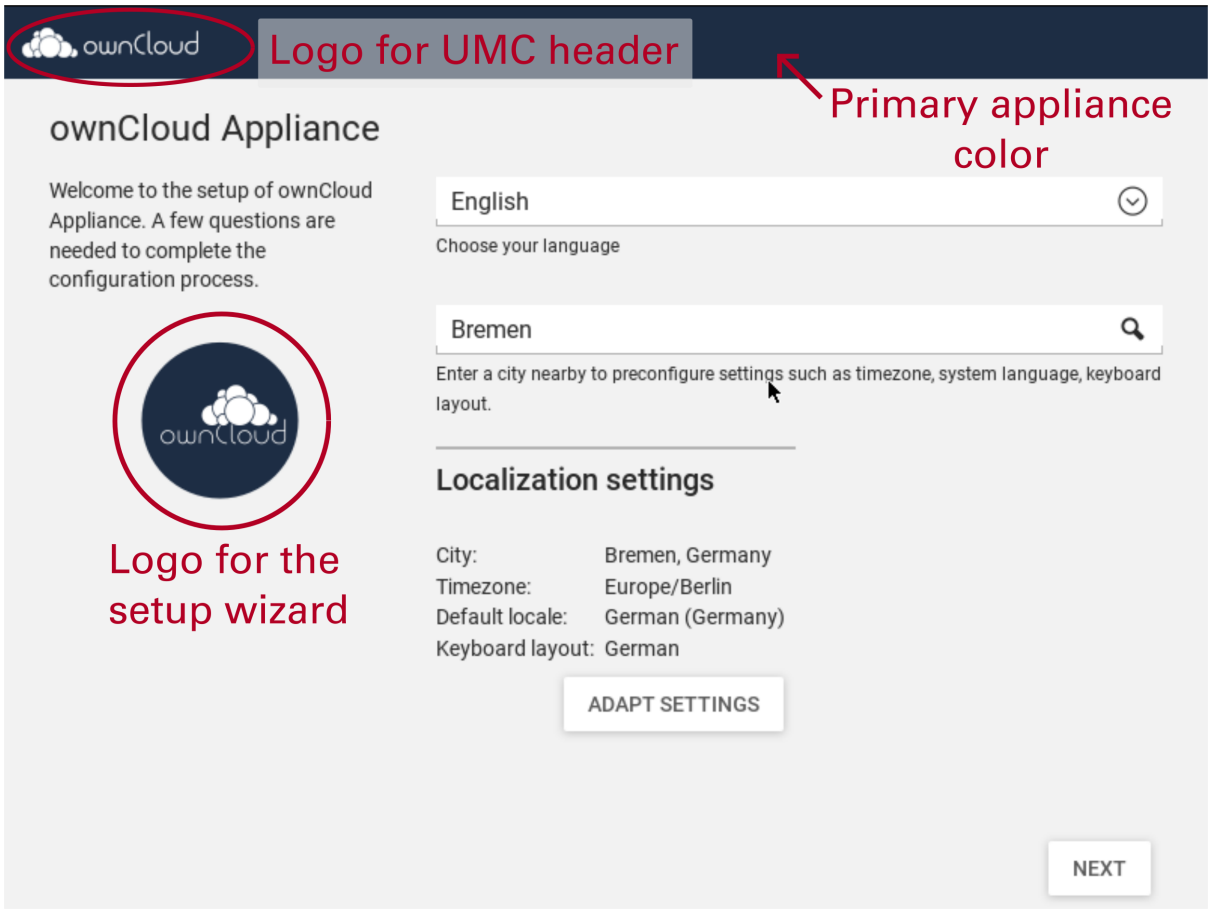

Fig. 7.3: System setup first screen

7.2.4 Welcome screen

The welcome screen is shown after the appliance setup has been finished and also every time the appliance is started. It offers information on how the user can access the appliance. It uses settings like the Appliance name and the CSS definition of the appliance background. The *Logo for the welcome screen* needs to be a SVG file that is slightly wider than high and which has the fonts converted to paths. Please mind the recommendations in *Logos* (page 23).

Depending on the character of the welcome screen background (bright or dark), the *Font color for welcome screen* should be either set to `White` or `Black`.

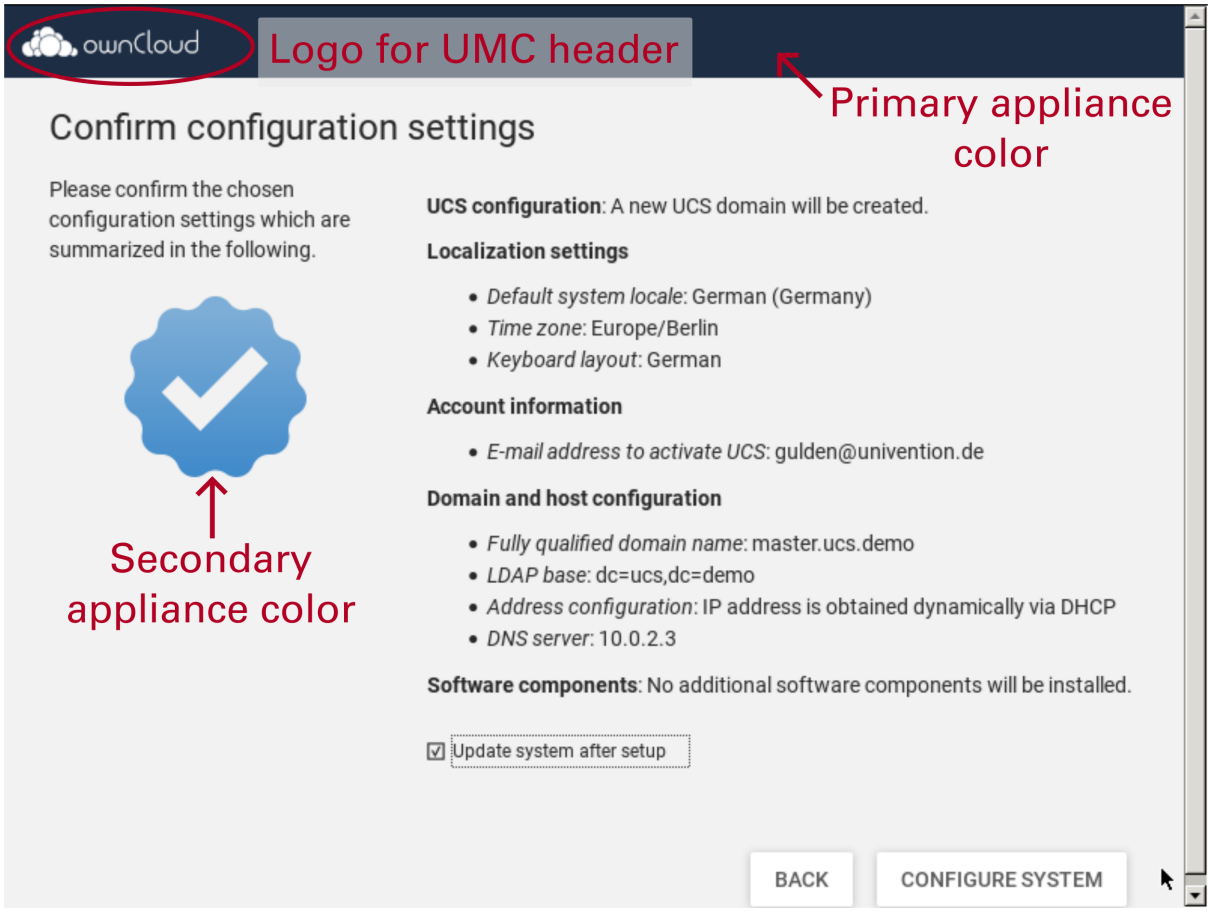

Fig. 7.4: System setup summary screen

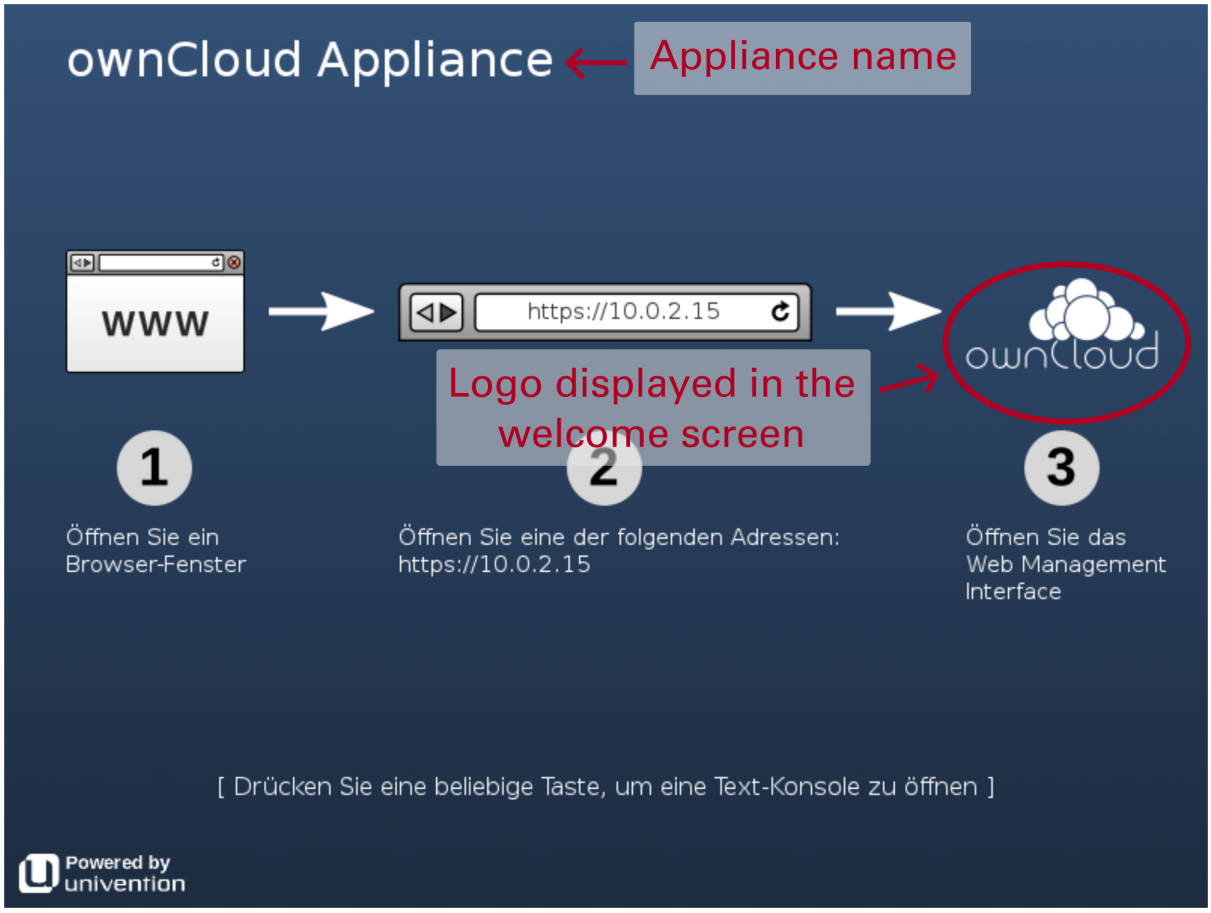

Fig. 7.5: Appliance welcome screen

7.2.5 Portal page

The branding of the portal page is independent from the other sections. The *Title for the UCS portal page in the appliance* can be defined and it can be configured if the font color shall be black or white. The *Logo for the portal page* controls which logo shall be set in the first tile on the portal page. The background can either consist of a background image or a background color or a background color gradient as described in *Boot splash* (page 54).

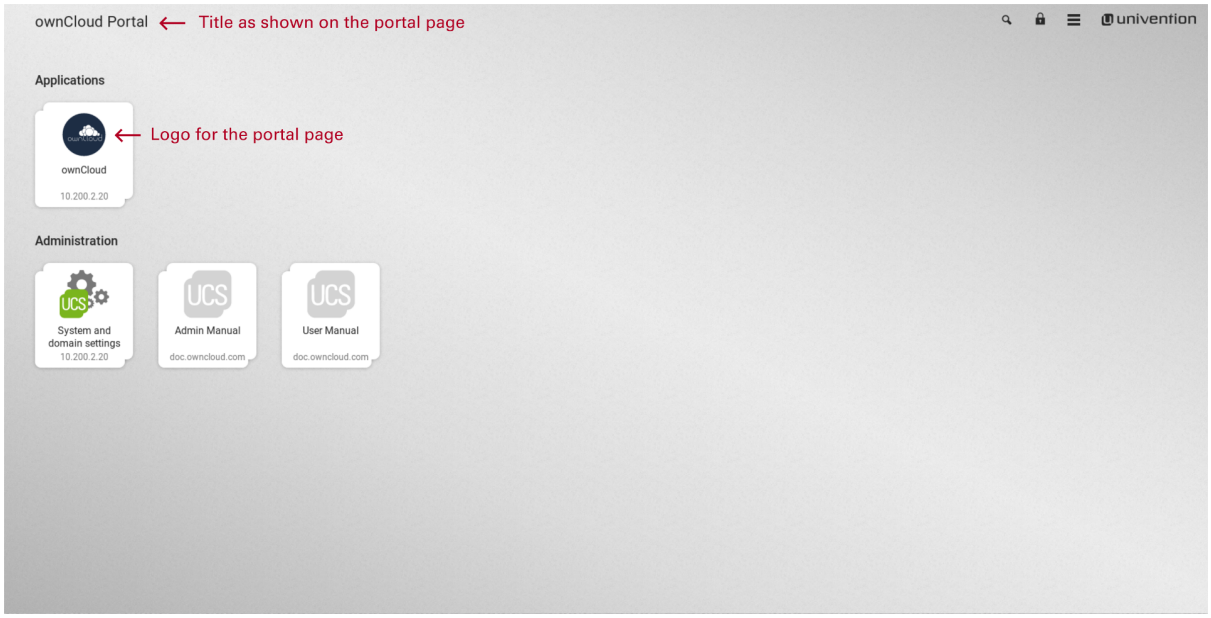

Fig. 7.6: Appliance portal page

CHAPTER

EIGHT

COMMAND `UNIVENTION-APP`

`univention-app` is a program to manage apps from Univention App Center on the command line on a UCS system.

This section describes a selection of the available commands. For a complete list of available commands, run **`univention-app`** without any arguments and `univention-app <command> --help` for details on a specific command.

`shell` `[-h] [-u USER] [-i] [-t] [-s SERVICE_NAME] app command`

The *`univention-app shell`* (page 61) command allows to run commands within a Docker app.

For multi container apps, **`univention-app shell`** runs the command in the main container which is the default behavior. The app metadata defines the app's "main service". For more information about referencing containers, see *Create a Multi Container App* (page 9).

To run a command in the main service of the app:

Listing 8.1: Run a shell command inside the app's main service

```
$ univention-app shell <app> <command>
```

To run a command in a specific service of the app, such as creating a backup of a **`MongoDB`** database:

Listing 8.2: Run a shell command inside the app's main service

```
$ univention-app shell -s <service-name> <app> mongodump --out /data/backup
```

For descriptions of the outline options and arguments of *`univention-app shell`* (page 61), refer to the console output of:

Listing 8.3: Show options and arguments of *`univention-app shell`* (page 61)

```
$ univention-app shell --help
```

INDEX

S

U