univention
be open.

Univention Corporate Server 5.2 Architecture

Release 5.2-1

Univention GmbH

Apr 10, 2025

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Welcome to the architecture documentation of UCS (Univention Corporate Server).

This document doesn't cover installation, the usage of UCS or parts of the product. For instructions about how to install and use UCS, see *Univention Corporate Server - Manual for users and administrators* [1].

The document is released step by step after each part is finished. The beginning is at the first, high level.

Your feedback is welcome and highly appreciated. If you have comments, suggestions, or criticism, please send your feedback[1] for document improvement.

For feedback on single sections, use the feedback link next to the section title.

1.1 Audience

This document is for consultants, administrators, solution architects, software developers and system engineers. It describes the technical architecture of UCS on three different detail levels.

The first, high level, *positions UCS in the known IT world* (page 3) and describes the *concepts* (page 7). This view helps readers to understand the principles of UCS. Chapters 2 and 3 assume you are familiar with information technology in general and that you have heard of computer network building blocks and software.

The second, medium level, is for administrators and solution architects. It covers the product components and the numerous services UCS offers to IT infrastructures. You read about the user facing product components and what services UCS runs. You learn what open source software contributes to the capability of UCS and how it interoperates together.

Software developers and system engineers get an overview of the technical parts.

A general understanding of Linux operating systems for servers and IT administration are beneficial for understanding.

For notation, the document uses the *C4 model* notation and the *ArchiMate* notation. For more information, refer to *Architecture notation* (page 70).

The third, low level is about the libraries, internal systems and storage. It describes the pieces a software developer and system engineer needs to know to contribute to UCS. General knowledge of software architecture and software engineering are helpful at this level.

[1] https://www.univention.com/feedback/?architecture=generic

1.2 Learning objectives

After reading this document you have a broad understanding of the UCS architecture. It equips consultants, administrators, and solution architects to better plan their IT environment with UCS. It enables software developers and system engineers to get familiar with software development for UCS.

1.3 How to use the document

This document contains numerous figures. As far as possible, they use SVG format. If you need a larger view of the image, open it in a dedicated tab in your web browser:

To open the figure in the same tab:

1. Click the figure.

Alternatively, to open the figure in a new tab:

1. Right click the figure.
2. Click *Open Image in New Tab* from the context menu.

To open the figure in a new tab:

1. Right click the figure.
2. Click *Open Image in New Tab* from the context menu.

CHAPTER

TWO

POSITIONING IN THE IT WORLD

To comprehend the architecture of UCS, it's important to understand the origin and where it locates in the world of IT (information technology).

2.1 Origin

UCS is a Linux distribution derived from Debian GNU/Linux[2]. Among others, it benefits from the strong software package manager, the high quality maintenance and the long-term stability as operating system for servers. Over the years, Debian has been and is a solid basis for UCS.

UCS is part of the open source family and has strong relations to important projects like for example Samba[3] and OpenLDAP[4].

2.1.1 History

Univention started UCS in 2002 as a collection of scripts that turn a Debian system into a Linux server that offers Windows domain capability. The goal was to offer companies and organizations a standardized Linux server as alternative to Microsoft Windows Server that implements Microsoft's domain concept. Over the time it developed to an enterprise Linux distribution with maintenance cycles that better suited the needs of organizations.

2.1.2 Packages

On UCS software is managed in software packages. The packages on UCS use the deb file format. For more information on the deb file format, see Wikipedia about deb (file format)[5] and Basics of the Debian package management system in the Debian FAQ[6].

UCS—like Debian—uses a package manager, which is a collection of software tools, to automate the process of installation, upgrade, configuration, and removal of computer programs. Packages organize such computer programs on UCS. In UCS the package manager is APT (advanced package tool). For more information about APT, see the Debian package management chapter in the Debian reference[7].

Univention distributes most packages from Debian GNU/Linux for the *amd64* and *all* architecture without changes for UCS. This includes the GNU/Linux kernel and over 98% of unchanged packages from the Debian project. Univention uses the default services from the Debian distribution and delivers custom configurations for UCS.

In the following circumstances, Univention builds and maintains derived packages:

- A later software version of a package is needed for UCS than Debian offers.
- Bug fixes or backports of a specific software are needed for a package.

[2] https://en.wikipedia.org/wiki/Debian
[3] https://en.wikipedia.org/wiki/Samba_(software)
[4] https://en.wikipedia.org/wiki/OpenLDAP
[5] https://en.wikipedia.org/wiki/Deb_(file_format)
[6] https://www.debian.org/doc/manuals/debian-faq/pkg-basics.en.html
[7] https://www.debian.org/doc/manuals/debian-reference/ch02.en.html

Additionally, Univention develops own software responsible for UCS capability that is distributed as Debian package.

Nevertheless, UCS doesn't include packages from the Debian games section, because it would require a content rating for video games. Univention doesn't see added value in the distribution of video games with UCS for the product audience.

2.2 Identity management

The most important functional pillar of UCS is identity management.

Simplified, an IT environment consists of services and users. Services offer capability. Users use capability. Services can also behave as users when they use the capability of another service. Users identify themselves against services to proof that they're eligible to use the capability.

The identification is done with *user accounts* to represent users. User accounts typically have properties like for example username, password, and email address. User accounts that digitally represent a person additionally have for example first name and last name.

Imagine a small IT environment with 20 persons and five systems. Without a central identity management, an administrator would have to maintain 20 user accounts on each of the five systems. The management effort sums up to 100 items. The number of items to manage is a linear function. The function's slope increases with the number of systems that need to know user accounts.

With a central identity management, one service holds the information about the user accounts. All other services have access to that information. An administrator only has to maintain the user accounts on one system. The maintenance effort for the user accounts doesn't anymore multiply with the number of systems that need to know the user accounts. The slope of this linear function is less steep.

Central identity management reduces the maintenance effort of user accounts for administrators.

UCS is a product for central identity management for user accounts, their permissions and the collection of user accounts in groups.

2.3 Infrastructure management

The second important functional pillar of UCS is IT infrastructure management.

IT infrastructure is a set of IT components like computer and networking hardware, various software and network components. It's the foundation of an organization's technology system and drives the organization's success.

UCS provides important infrastructure services to create an IT network infrastructure and connect IT components. For example UCS assigns addresses to computers and other network components through DHCP[8] and resolves hostnames through DNS[9], and much more. Administrators manage various IT components in their IT environment, like different kind of hosts, clients, and printers.

2.4 Connection to the world around

As an operating system that offers many services, UCS interacts with its surrounding peers. Users access the capability of UCS through the following ways:

Web

Persons like administrators and also end users use HTTPS to access the web based UCS management system. In many cases other web-based services provided by other software products delivered through apps are also available through HTTPS.

[8] https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
[9] https://en.wikipedia.org/wiki/Domain_Name_System

Console
: Persons with more technical background and the appropriate permissions can access UCS through a console, either on a local terminal or through a remote secure shell (SSH)[10] session.

Service protocols
: As soon as users use any of the services that UCS offers, they access UCS through one of the service protocols. For example, a user's client requests and IP address through DHCP and later asks for the IP address of the print server through DNS.

As a central system offering identity and infrastructure management UCS has to use and offer numerous ways of connections.

[10] https://en.wikipedia.org/wiki/Secure_Shell

CHAPTER

THREE

CONCEPTS

UCS unites numerous concepts to support administrators with their identity and infrastructure management tasks.

The following concepts explain how UCS uses them:

1. *Domain concept* (page 7)
2. *Replication concept* (page 8)
3. *Role concept* (page 10)
4. *Permission concept* (page 11)
5. *Certificate infrastructure* (page 13)
6. *Univention app ecosystem* (page 13)

3.1 Domain concept

The domain concept is the most important concept in an IT environment operated with UCS. The domain concept offers a way to centrally manage an IT environment where administrators can map their organization's structure to the IT environment.

Simplified, an IT environment consists of computer systems and users. Systems offer services that provide capability. Users use capability. A domain is a single trust context that groups one or more entities like computer systems or users. The domain offers special services called domain services to systems and users. Fig. 3.1 shows the relationship between the actors systems, services, and users.

A trust context uses roles, permissions, and cryptography certificates to ensure secure communication between the domain participants. Domain services and domain participants can rely on the shared trust context when secure and mutually authenticated communication is required.

One key participant in a domain is the identity, a digital representation for persons. An identity represents an account for a user in a domain. It holds information like for example username and password for login. Furthermore, it contains various data associated to the user like for example group memberships, permissions, and different attributes used by services.

User accounts are organized in groups and users can belong to multiple groups. User groups help administrators to apply permissions for domain services to users and are essential to the organization's structure to the domain administration.

All the objects in a domain need to be managed and organized. In a domain a central database called domain database registers all objects, like for example user identities, computer systems, printers, and file shares. See Fig. 3.2 for a graphical interpretation. The database stores the objects in a hierarchical tree-like structure. One or more central systems store the central database and are called domain node.

UCS is a system that operates the central database for the domain. UCS is the central platform that implements the domain concept and helps administrators to manage and organize the IT environment for their organization. For the distinct roles of UCS systems in a domain, see the *role concept* (page 10).

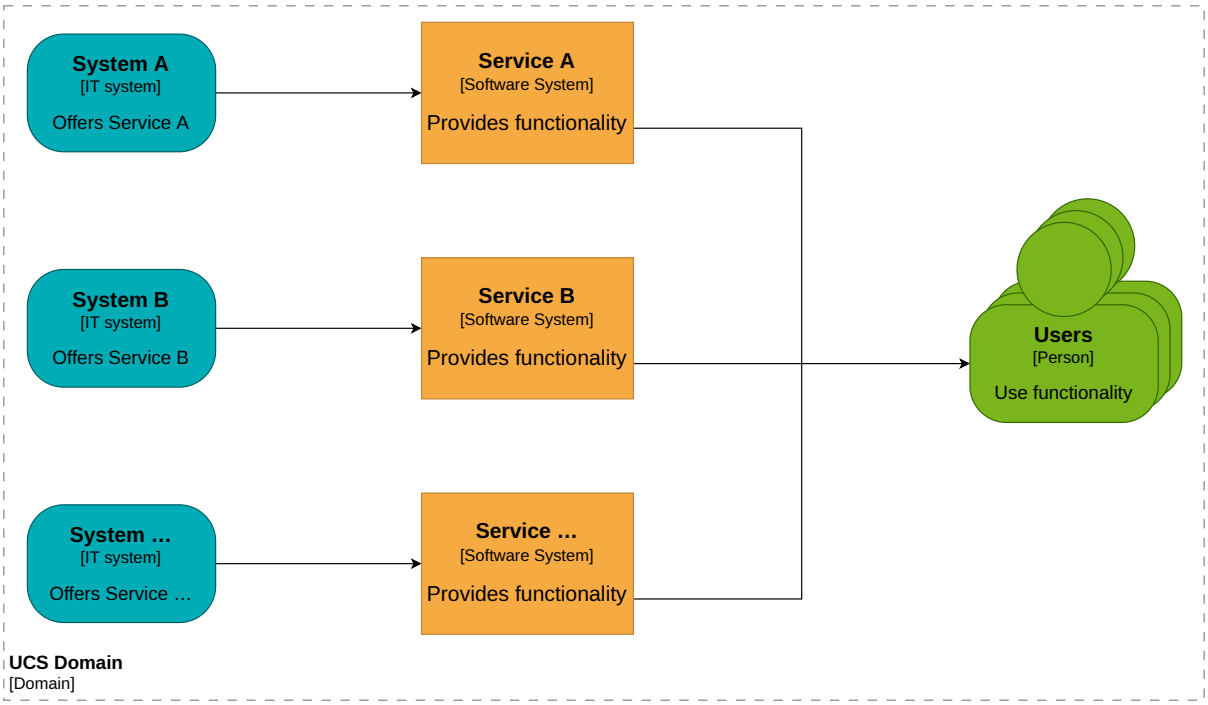

Fig. 3.1: Relationship of the *systems*, *services* and *users* in a domain

3.2 Replication concept

The replication concept ensures the availability and consistency of the central domain database and contributes to its scalability. It's necessary to keep the domain data synchronized across all domain nodes because more than one domain node can have a copy of the central database. For example, domain nodes can get disconnected or need to shutdown for maintenance.

UCS implements the replication concept. The first domain node in the domain has the following tasks:

- It writes domain object data to the database.
- It monitors changes to the database.
- It makes changes available to other domain nodes.

The other domain nodes have a read-only copy of the domain database.

The replication synchronizes a lot of data types. The following list names a few that domain nodes replicate and can't cover all items:

- User identities
- Groups
- Policies
- Permissions
- Information about systems
- Information about printers
- Information about file shares

The domain replication in UCS also ensures that the affected UCS systems run follow-up actions after the changes are replicated. The actions can contain, for example, updates to configurations of services and making the changes available to the users.

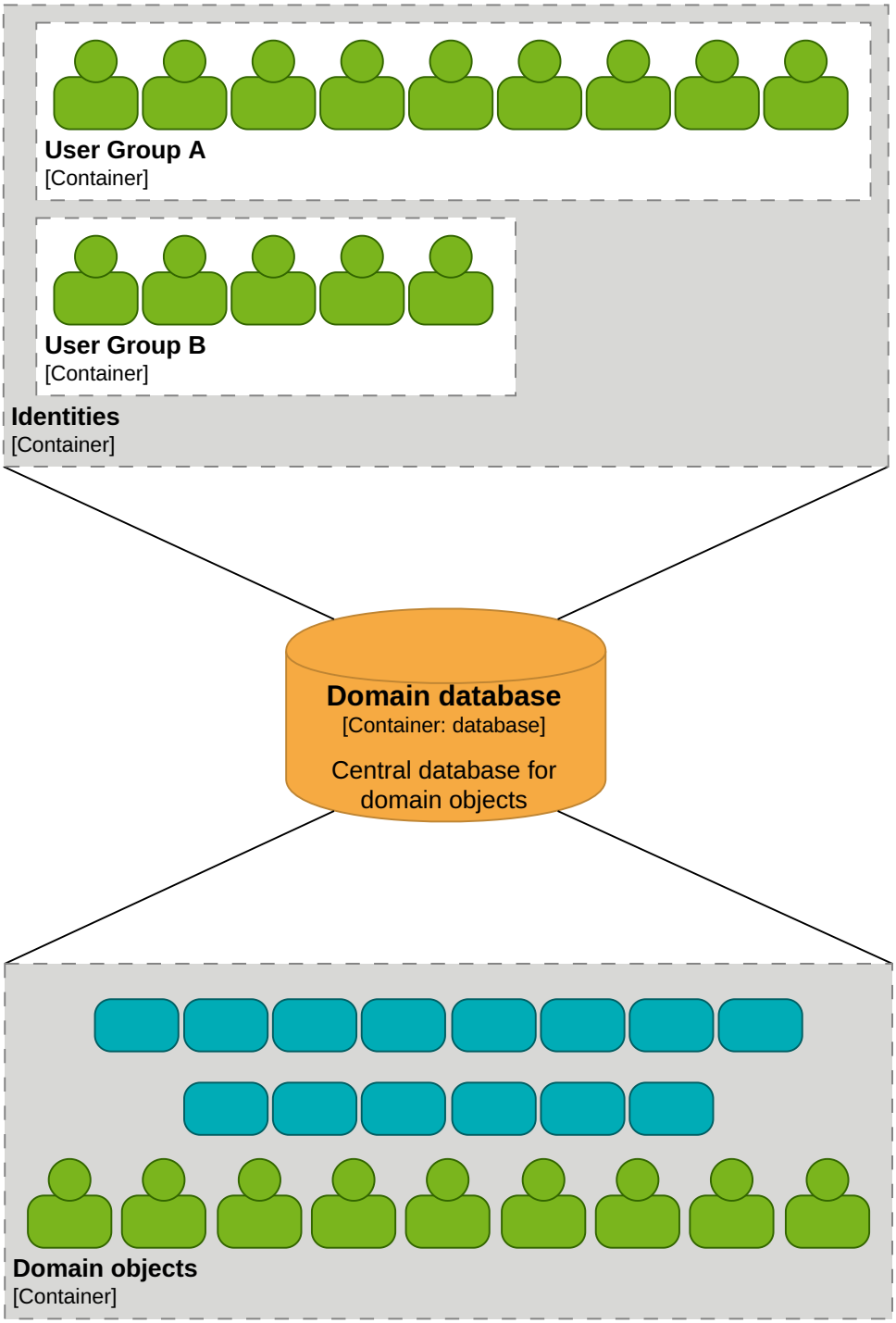

Fig. 3.2: Central domain database with different objects

3.3 Role concept

UCS uses a role concept to assign different roles that include certain tasks to the systems in a domain.

3.3.1 Primary Directory Node

A UCS system with the role *Primary Directory Node* is the first, the primary, domain node in a domain. It's the only system with write permissions to the central domain database and performs all write requests regarding data for the domain database. Only one system in the domain can have the Primary Directory Node role.

3.3.2 Backup Directory Node

A UCS system with the role *Backup Directory Node* has a complete read-only copy of the domain database, including security certificates. More than one UCS system can have the Backup Directory Node role. In case the Primary Directory Node is unavailable, recovery is impossible or needs too much time, an administrator can promote a UCS system in the role Backup Directory Node to a Primary Directory Node. The promotion can't be reversed.

See also:

Converting a Backup Directory Node backup to the new Primary Directory Node[11]
for details on the promotion process of a UCS Backup Directory Node in *Univention Corporate Server - Manual for users and administrators* [1].

3.3.3 Replica Directory Node

UCS systems with the role *Replica Directory Node* have a complete read-only copy of the domain database. Administrators can't promote Replicate Directory Nodes to the Primary Directory Node role or any other role unlike the Backup Directory Node.

A Replica Directory Node optionally allows selective replication, a form of data synchronization that replicates only a subset of the domain database. Selective replication in UCS helps with data minimization, domain protection and permission enforcement.

For example, imagine an organization with office locations in cities like Berlin and Bremen. Each location has a Replica Directory Node as domain node. The Replica Directory Nodes only replicate domain objects like users, groups, and printers that are relevant for their respective location. They don't store objects assigned to other locations.

Replica Directory Nodes are ideally suited as dedicated systems for load intensive services with permanent read operations to the domain database because the read operations run locally instead of across the computer network.

3.3.4 Managed Node

UCS systems with the role *Managed Node* don't have any copy of the domain database. Services on Managed Nodes read domain information over the network from either the Primary Directory Node or from one of the Backup Directory Nodes.

[11] https://docs.software-univention.de/manual/5.2/en/domain-ldap/backup2master.html#domain-backup2master

3.3.5 Clients

Clients are systems in the domain's trust context. They don't have a special role regarding domain services as the other roles described before. In most cases they consume services offered by the domain or other systems.

UCS offers dedicated client roles for desktop systems like Ubuntu, other Linux desktops and macOS. UCS manages IP addresses and DNS entries for systems like network printers and routers with the *IP client* role.

For Microsoft Windows related systems, UCS offers the roles *Domain Trust Account*, *Windows Domaincontroller* and *Windows Workstation / Server*.

See also:

UCS system roles[Page 11, 12]
For more information about the differences of these roles in *Univention Corporate Server - Manual for users and administrators* [1]

3.4 Permission concept

The permission concept in UCS specifies who can read and write domain data. Permissions apply to objects in the domain database like users and systems alike. Policies assign custom permissions to objects. UCS applies default permissions for systems and predefined users and groups.

3.4.1 System roles

UCS system roles imply certain permissions on domain data. Only the Primary Directory Node can write data to the domain database. All other system roles have read-only access. Nevertheless, other systems or users have write permissions for certain operations affecting themselves and they run them on the Primary Directory Node.

For example, when a UCS system joins the domain or an administrator installs an app, the events trigger write operations on the Primary Directory Node.

3.4.2 Administrator and root

Some user accounts also have implicit permissions on domain data and systems. A UCS system knows two administrative user accounts: *Administrator* and *root*.

Administrator
The user account *Administrator* is the first domain user and has all domain permissions. The *Administrator* user account has permission to join systems to the domain and can work with all modules in the UCS management system. The account can only be defined once in the domain and must never be renamed.

The *Administrator* account is only defined once per domain during the installation of the Primary Directory Node. The account password is set during installation.

Think of *Administrator* as the primary administrative account for the UCS **domain**.

root
The user account *root* is the superuser on the local UCS system and has the user ID of `0`. It has all permissions and is equivalent to the *root* account known from other GNU/Linux systems.

The *root* account is defined and the password is set during installation of every UCS system. The account is only for the local UCS system. On other UCS systems administrators should—of course—define different passwords for each *root* account.

Think of *root* as the primary administrative account for the **local** UCS system.

The *root* account has no permissions and is no valid account in the domain context. The account *root* must not be created as domain account.

[12] https://docs.software-univention.de/manual/5.2/en/domain-ldap/system-roles.html#system-roles

3.4.3 Domain users and admins

To simplify the assignment of certain user permissions, UCS has two default user groups in the domain that differ fundamentally: *Domain Users* and *Domain Admins*.

Domain Users
UCS assigns every user to the user group *Domain Users* per default. The group identifies the user account as belonging to a person. The user account only has a minimal set of permissions in the domain.

For example, user accounts in the group can read the domain database, but can't view password hashes. Additional apps in the domain such as **UCS@school** or **Fetchmail** can alter read and write permissions for users and systems. User accounts in the *Domain Users* group also can't sign in to UCS systems for a remote shell by default. The UCS management system yields no modules for them either.

Domain Admins
UCS creates one user account called *Administrator* during the installation of the first UCS system (Primary Directory Node) in a domain. It's the first user account and has all permissions for the domain. The *Administrator* user account is member of the *Domain Admins* group.

Users in *Domain Admins* group have all domain permissions just like the *Administrator* account. To join a UCS system to the domain, administrators need a user account that's member in the groups *Domain Admins* and *DC Backup Hosts*.

See also:

Subsequent domain joins with univention-join Page 12, 13
For more information about subsequent domain joins in [1]

3.4.4 Machine account

All systems part of the domain are actors in a domain like users. Each system has its own account in the domain database. The account is called *machine account*. Depending on the type of system they have different permission sets.

UCS systems can read data from the domain database with their machine account. Every machine account has assigned the following default permissions in the UCS domain:

- The UCS system can read all object information and password hashes for accounts from the domain database. Apps like **UCS@school** and **Fetchmail** limit the read permissions.
- The UCS system can write only information to the domain database that's associated with its account, for example the version of the installed UCS or other apps.

3.4.5 Policies

In addition to the permissions defined for system roles and predefined groups, UCS offers policies for more fine-grained control on administrative settings.

Policies are administrative settings to help administrators with infrastructure management that can be assigned to objects in the domain database. Policies use the inheritance principle as it's known from object oriented software programming. Inheritance allows to set policies to one object in the structured domain database. The policy then applies to all objects that are organized in the structure below.

[13] https://docs.software-univention.de/manual/5.2/en/domain-ldap/domain-join.html#domain-ldap-subsequent-domain-joins-with-univention-join

3.5 Certificate infrastructure

The certificate infrastructure in a domain operated with UCS ensures the trust context between all participants. The first domain node creates its own CA (certificate authority) for the domain. For more information, see the Wikipedia article Certificate authority[14].

UCS uses TLS (Transport Layer Security). The UCS Primary Directory Node creates the CA on behalf of the domain during its installation and signs certificates for other systems that join the domain. All certificates have an expiration date. Backup Directory Nodes in the domain repeatedly pull all certificates from the Primary Domain Controller to allow administrators to promote one of them to a Primary Directory Node any time, if needed.

Services in the UCS domain also use the certificates created by UCS. Administrators can configure alternative certificates for end-user or internet facing services with certificates issued by third parties, for example Let's Encrypt[15].

The domain systems use the certificates for secure communication between each other over the computer network, for example for domain database replication and the web interface of the UCS management system. Communication clients need to know the public key of the domain's CA to validate the public key of the certificate.

3.6 Univention app ecosystem

Univention App Center is one of the most important parts of UCS. This section describes the Univention app ecosystem, where UCS is just one part. The Univention app ecosystem consists of actors, an infrastructure, and artifacts.

This section provides information about the following aspects around the App Center:

- *App Center purpose* (page 14) about the why of the App Center for administrators and app providers.
- *App ecosystem actors* (page 15) in the context of the App Center ecosystem.
- *App artifacts* (page 16) for an overview of the content in the App repository.
- *Univention app infrastructure* (page 19) for the App Center.

Hint: For architecture notation, this part of the document uses ArchiMate® 3.2, a visual language with a set of default iconography for describing, analyzing, and communicating many concerns of enterprise architectures. For more information about how the document uses the notation, refer to *ArchiMate* (page 70).

Continue reading

App Center **(page 26)**
for the description of the App Center component in UCS.

See also:

App Center service **(page 57)**
for the architecture of the App Center on UCS

[14] https://en.wikipedia.org/wiki/Certificate_authority
[15] https://letsencrypt.org

3.6.1 App Center purpose

Depending on the direction you look at the App Center, it has different purposes and provides different benefits.

First, the App Center provides value to administrators in terms of software management:

- Maintained enterprise software integrated with UCS identity management. The existing integration reduces the effort for customers to maintain such an integration.
- Software lifecycle management with simplified installation and updating of software applications in a server infrastructure.
- Reliable delivery infrastructure to serve the software lifecycle management.

Fig. 3.3 shows the purposes of the App Center for customers and app providers.

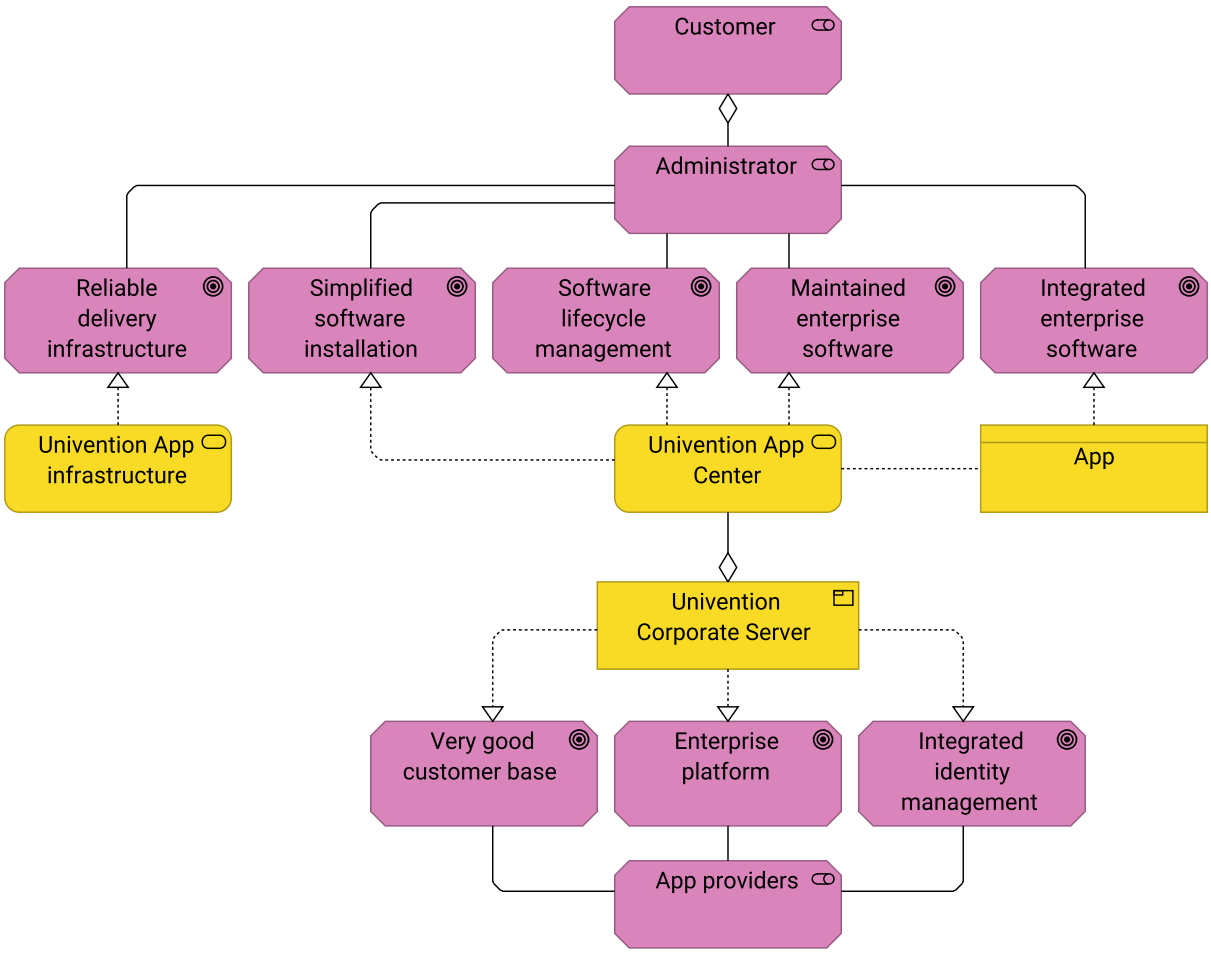

Fig. 3.3: Purpose of the App Center

Second, because the App Center is part of UCS, app providers benefit from a good customer base and an enterprise platform with integrated identity management. With integrated identity management at their fingertips, app providers don't have to worry about identity management on their own. They can rely on the offered interfaces such as LDAP, SAML, and OpenID Connect.

3.6.2 App ecosystem

On the one hand, the App Center is a user-facing product component in UCS. The *App Center service* (page 57) covers the architecture and technology in more detail. On the other hand, the App Center is also an ecosystem with services, actors, artifacts, and infrastructure.

This section provides an overview of the ecosystem.

App ecosystem actors

Fig. 3.4 shows the actors involved in the Univention App Center ecosystem. For the sake of brevity, the figure shows a subset of the responsibilities.

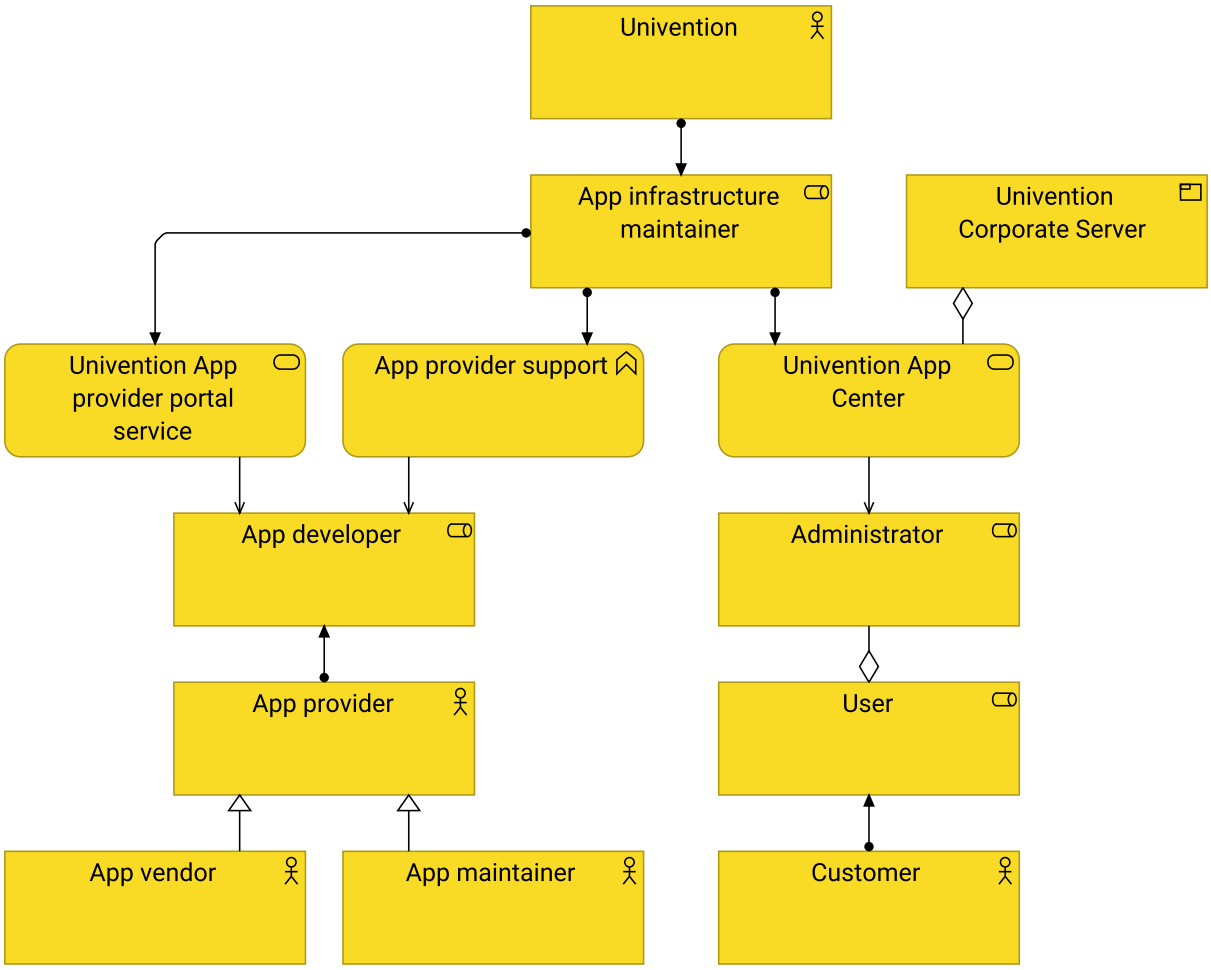

Fig. 3.4: Actors in the App Center ecosystem

App infrastructure maintainer

Univention fulfills the responsibility of the *App infrastructure maintainer* and as such is responsible for *Univention App Center*. For example, Univention operates the infrastructure so that administrators can install software through the App Center.

The *App infrastructure maintainer* also fulfills the responsibilities to operate the *Univention App provider portal service* and to provide *App provider support*. Both serve the *App developer* during on-boarding and app maintenance.

App provider

The next actor in the App Center ecosystem is the *App provider* in the following specializations:

App maintainer
: The *App maintainer* doesn't own the software, but maintains the app with the software in the App Center.

  The App Center also contains open source apps. Organizations that act as *App maintainer* don't own the open source software. They invest their knowledge of UCS and the software in an app, its integration with UCS, and the maintenance of the app for the benefit of customers and to promote open source software.

App vendor
: The *App vendor* owns the software. Organizations that own software and maintain their own app in Univention App Center act in both ways, as *App vendor* and *App maintainer* at the same time.

App developer
: The role *App developer* is the primary role that interacts with the *Univention App provider portal service* and uses the *App provider support*.

Customer

The third actor is the customer in the role of the user and especially the *Administrator*. They use *Univention App Center* with the associated services and apps to cover their software needs for their business.

App artifacts

The artifacts in the App Center are apps. At the technology level an *App Center app* consists of the parts shown in Fig. 3.5.

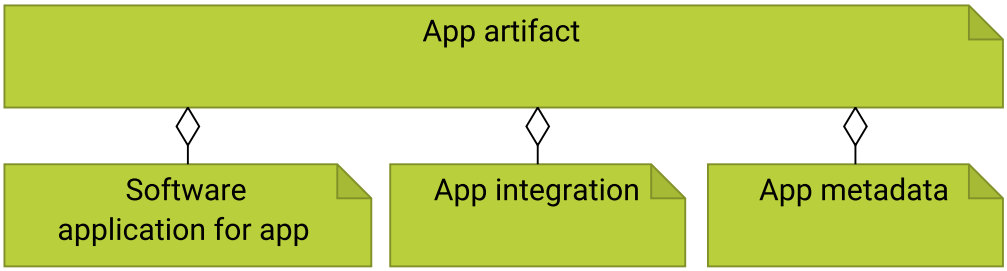

Fig. 3.5: Parts of an app

Software application for app
: *Software application for app* is the software itself, the binary artifact as provided by the vendor.

App integration
: *App integration* includes scripts and software tailored to the integration needs of the software application and UCS. They take care of the proper setup so that the app is ideally ready to use after installation. For example, the integration may consist of:

  - Setup for single sign-on configuration between the software application and UCS.

- Configuration to set up the web server.
- Script to populate a database with the database schema and required data.
- Environment setup for configuring the software application.

App metadata
App metadata is the content responsible for properly presenting the app to the user in the App Center. It includes name, description, logo, and contact information for the app provider.

The App Center recognizes the *Software application for app* in the form in which the vendor distributes the binary artifact, as shown in Fig. 3.6.

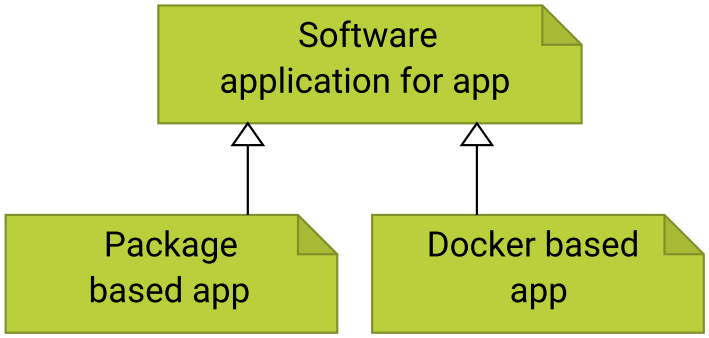

Fig. 3.6: Kinds of software distribution for the App Center

Package based app
Package based app refers to software distributed using *Debian packages* (page 3). Apps that extend the core capabilities of UCS use Debian packages for software distribution. The App Center installs the packages from dedicated repositories per app and handles the repository configuration.

Docker based app
Docker based app refers to software distributed through Docker images, a data format for containerized software. Docker based apps decouple the software runtime from the underlying UCS operating system and reduce the complexity of app maintenance for app providers.

Important: The App Center prefers Docker based apps over package based apps.

Finally, a *Docker based app* can be either a *Single container app* or a *Multi container app*, as shown in Fig. 3.7.

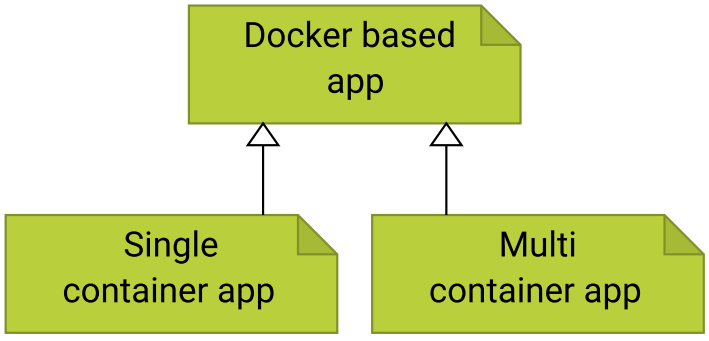

Fig. 3.7: Kinds of Docker apps

Single container app
Individual single container apps consist of a Docker image. UCS uses the Docker engine to run them.

Multi container app
Multi container apps, on the other hand, consist of more than one Docker image. UCS uses Docker compose[16]

[16] https://docs.docker.com/compose/

and the Docker engine to run them. App providers that offer their app as multi container app often provide the required parts as micro services for better decoupling and dependency control. They also typically offer this type of deployment anyway, independent of the App Center.

Fig. 3.8 shows the overall model, its parts and what an app consists of. On the application level the App Center differentiates an *App* into *Package based app* and *Docker based app* and handles both.

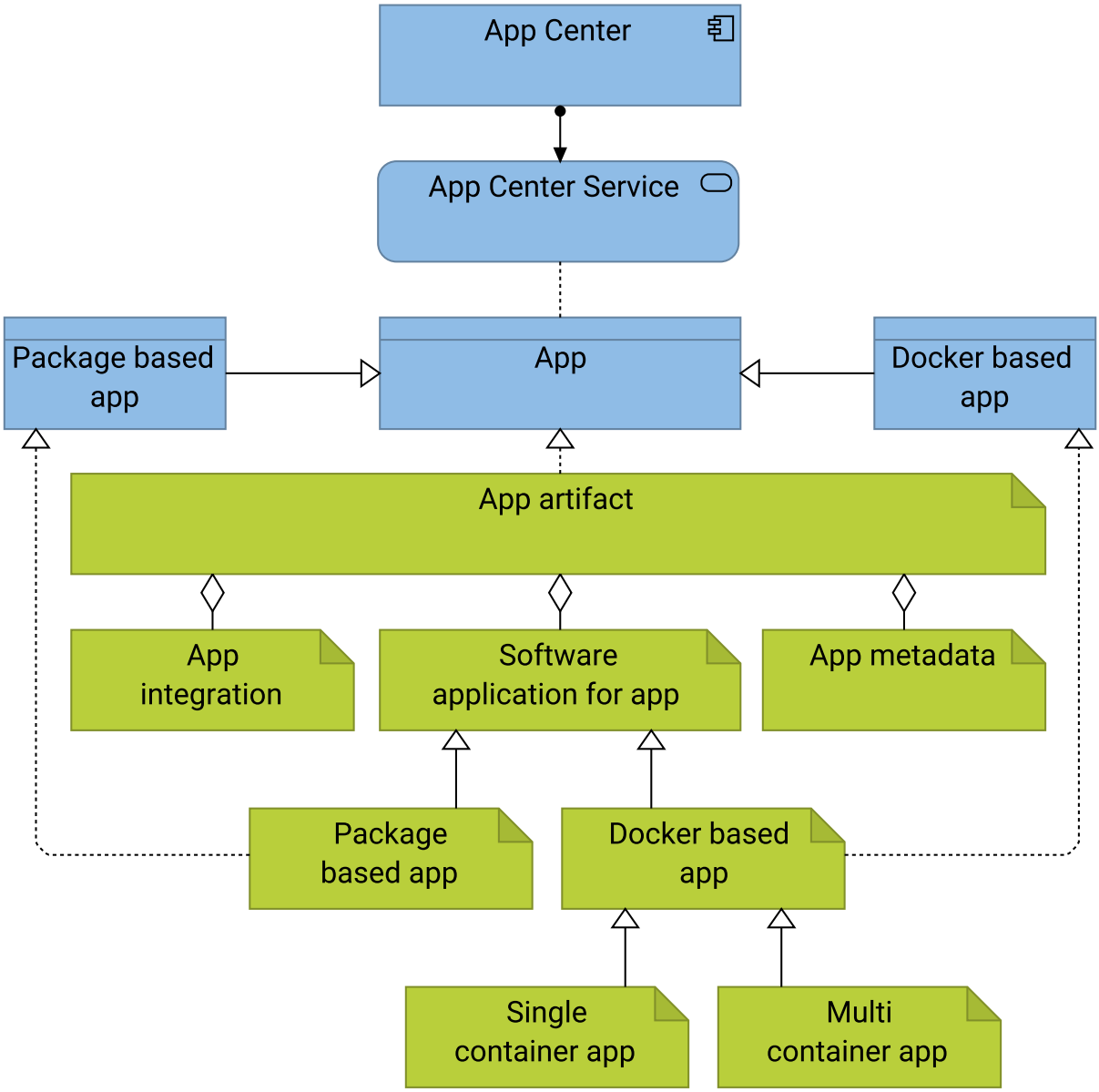

Fig. 3.8: Apps as content in the App Center ecosystem

Univention app infrastructure

The App Center requires a dedicated infrastructure consisting of several elements to function properly. Fig. 3.9 shows the infrastructure, and the description of each element follows.

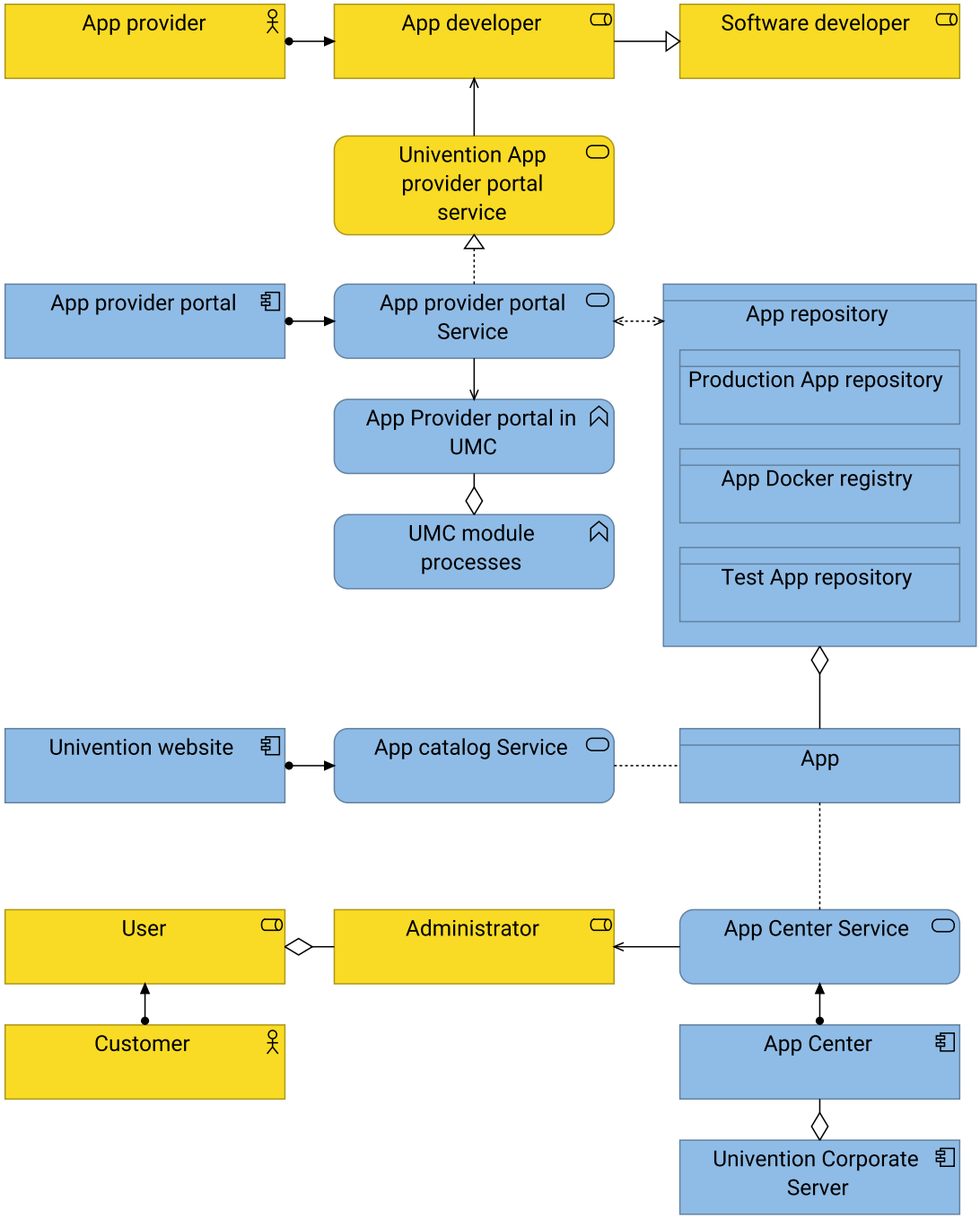

Fig. 3.9: App Center infrastructure model

App developer

An *App developer* is a software developer who is responsible for creating and maintaining an app. The *App developer* belongs to the *App provider* (page 16).

Univention App provider portal service

The *Univention App provider portal Service* is the entry point for app developers who create and maintain an app in the App Center. App developers use the *App provider portal* that handles authentication and access rights to the app definitions for app developers. And, it uploads the app software to the *App repository*.

Technically, the *App Provider portal* is a *UMC module* (page 52) running on a dedicated UCS system to manage the *App repository*.

App repository
The *App repository* is the storage location for the app artifacts. UCS systems connect to the *App repository* to load the app metadata for presentation and to download the app for installation on a UCS system.

The *App repository* consists of the following parts:

- *Production App repository* is the location where all UCS systems download the apps. It contains the publicly available apps.
- *App Docker registry* is the location for the Docker images of Docker based apps.
- *Test App repository* is the location for apps under development. Only app developers use it during app development. After an app release completes, the app appears in the *Production App repository*.

App Catalog
The *App Catalog* is part of the Univention website and provides an overview of the available apps and their descriptions. It's a representation of the app metadata for user information purposes. The *App Catalog* loads the data from the *App repository*.

App Center
In the context of Fig. 3.9, the term *App Center* of the ArchiMate application component refers to everything on a local UCS system that makes up the App Center. The *App Center Service* loads the app information from the *App repository*.

For the architecture of the *App Center*, refer to *App Center service* (page 57).

Administrator
The *Administrator* is the primary *User* role that interacts with the *App Center Service* on a UCS system. The *Administrator* has the user permissions to install, update, and remove apps on a UCS system.

See also:

Univention App Center[Page 20, 17]
for more information for administrators about how to use the App Center in *Univention Corporate Server - Manual for users and administrators* [1].

Univention App Center for App Providers[18]
for more information for app developers about how to develop apps for Univention App Center in *Univention App Center for App Providers* [2]

App Catalog[19] **on the Univention website**
for an overview about available apps in the App Center.

[17] https://docs.software-univention.de/manual/5.2/en/software/app-center.html#software-appcenter
[18] https://docs.software-univention.de/app-center/5.2/en/index.html
[19] https://www.univention.com/products/app-catalog/

CHAPTER

FOUR

PRODUCT COMPONENTS

In this part of the document, you learn about the second, medium, and detail level of the architecture of UCS. You learn about UCS product components that you face directly when you use UCS. The product components typically act as entry points for your tasks.

The product components descriptions are intended for administrators and solution architects. For software developers and system engineers, it provides the context for the architectural details of UCS. Make sure you are familiar with the *Concepts* (page 7) behind UCS.

Hint: For architecture notation, this part of the document uses ArchiMate® 3.2, a visual language with a set of default iconography for describing, analyzing, and communicating many concerns of enterprise architectures. For more information about how the document uses the notation, refer to *ArchiMate* (page 70).

The following product components from Fig. 4.1 introduce themselves in the order you most likely encounter them when you work with UCS:

1. *UCS portal* (page 22)
2. *UCS management system* (page 22)
3. *App Center* (page 26)

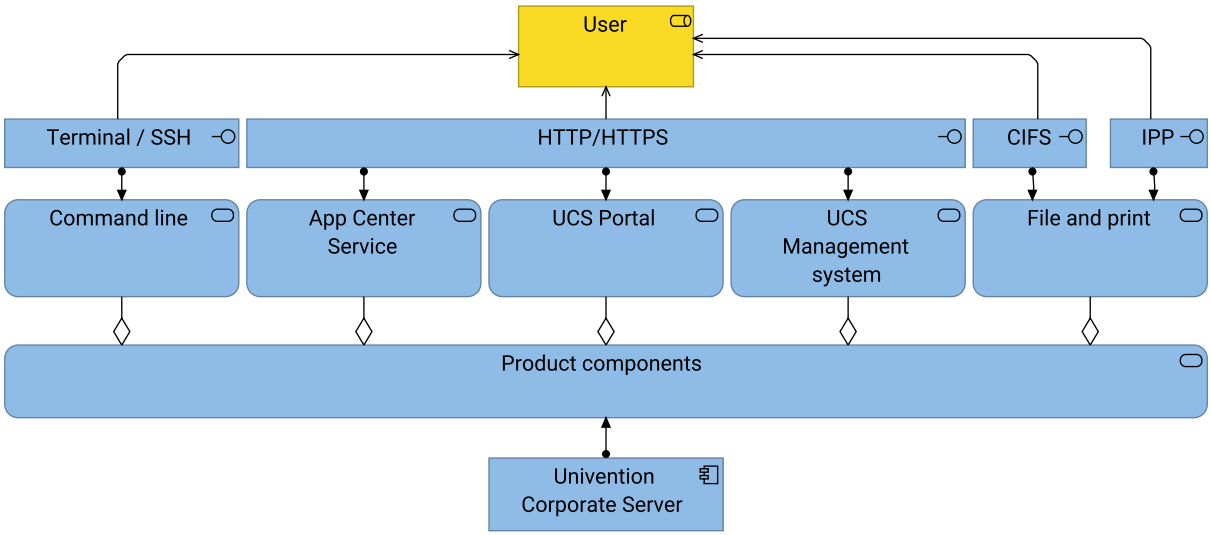

Fig. 4.1: User facing product components of UCS

Hint: The section is work in progress. Later updates of the document explain the concepts *Command line* and *File and print*. For the sake of completeness Fig. 4.1 already shows them.

4.1 UCS portal

The UCS portal is the central entry point to UCS and home page for the work place for domain users and administrators. The web page shows tiles with icons, text, and web links to various services and applications of the domain and external resources.

Every UCS system can have a portal page, regardless of its system role. A domain can have multiple portal configurations with different content. The portal configuration controls the following aspects:

- Which portal shows up on which UCS system in the domain?
- Which user groups see which tile on which UCS system?

Organizations can configure multiple portals. They can brand and customize them individually to specific user groups.

Continue reading

UCS portal service **(page 53)**
for architecture details about the UCS portal service

See also:

UCS portal page[Page 22, 20]
for instructions about how to configure and customize the UCS portal page in *Univention Corporate Server - Manual for users and administrators* [1]

4.1.1 Benefits

For the portal as primary entry point to a UCS domain, users like administrators or end users only need to remember or bookmark one web address. After login with their web browser, users see their personal portal. Some tiles only show up after login.

With single sign-on, users provide their credentials onetime per session and can use services and apps without additional authentication.

4.1.2 Single sign-on for the UCS portal

To use single sign-on with a service, the service needs to support and integrate single sign-on in the UCS domain. UCS supports the standards SAML and OpenID Connect.

4.2 UCS management system

The UCS management system is the central administration interface for users with administrative tasks to operate a UCS domain and maintain the UCS systems. It provides different interfaces, for example for web browsers, command line and programming, although the term *UCS management system* generally refers to the web interface. Administrators usually open the management system in their web browser through the *UCS portal* (page 22) after they sign in.

The UCS management system provides for the following purposes:

1. Intuitive, easy to use, and central web application for all administrative tasks around identity and IT infrastructure management
2. Low entrance barrier for system administrators
3. Aggregation and representation of data stored in the domain database

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

4. Management of UCS systems through a web based interface

It consists of the following parts as shown in Fig. 4.2:

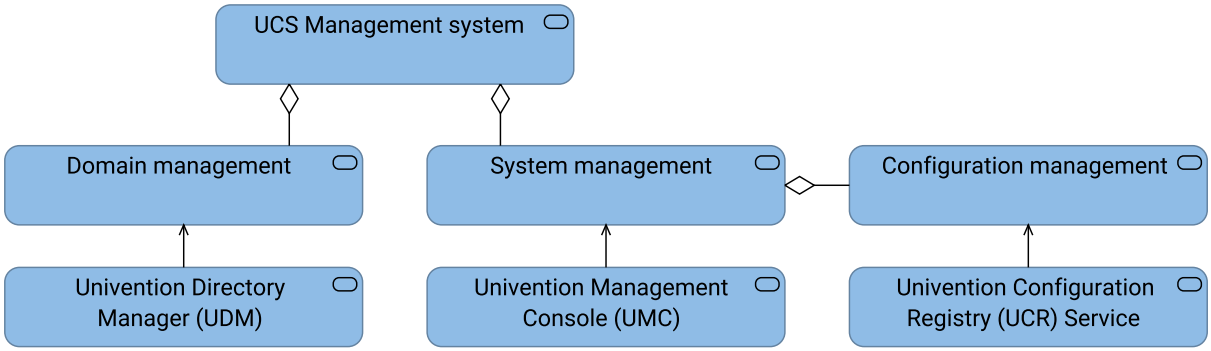

Fig. 4.2: Parts of the UCS management system

- *Domain management* (page 23) to centrally manage the domain settings and objects in the domain database.
- *System management* (page 24) to manage UCS systems, services and apps.
- *Configuration management* (page 25) to store configuration settings for each UCS system.

Talking about the UCS management system comprises all previously mentioned parts. Administrators rely on the UCS management system, because it simplifies their daily work and improves the usability of UCS.

4.2.1 Domain management

Domain management covers all administrative areas related to identities, devices, and services across the domain.

Identities
Identities include users and their collection in user groups. A user can be member of different groups.

Devices
Devices include computers such as other UCS systems, user clients such as notebooks or printers. Furthermore, it also includes device monitoring.

Services
Services include basic infrastructure services such as networking with DHCP and DNS, infrastructure for applications and users such as email and file shares, and other domain services such as domain join, directory manager, and policies.

In UCS, UDM (Univention Directory Manager) is responsible for domain management. You can imagine UDM as abstraction layer to the domain database. UDM provides modules for each area. Services and apps can extend the domain management.

As abstraction layer the purposes of UDM are the following:

Data aggregation
Return multiple objects from the domain database as one object in UDM.

Data consistency
Ensure data consistency between different objects in the domain database, such as references between different objects and ensure that the references are always valid.

Data presentation
Enhance the data from the domain database for appropriate presentation to the user on the command line and in the web interface.

Atomic operations
Provide a locking mechanism so that operations with multiple actions run as atomic operation. The domain

database doesn't support transactions. For example, creating a user with a unique primary email address requires the reservation of username and email address before UDM can create the user object.

Input value validation
Validate user input to ensure correct and consistent data in the domain database. UDM is the interaction layer between the user and the domain database.

Process logic
Process logic ensures that UDM automatically applies default values to properties when users don't set values for properties. In addition, the process logic prevents inconsistent state of data.

User interface enhancements
UDM provides an interface for enhancement with additional properties in UDM. *Extended attributes* and *extended options* provide the interfaces.

Usability
UDM enhances the usability when working with data from the domain database. For example, the domain database maintains group memberships at the group only. In contrast, in UDM administrators can maintain group memberships at the group and at the user alike.

Continue reading

Univention Directory Manager (UDM) (page 35) for description of the architecture of UDM.

See also:

Administrators refer to *Univention Corporate Server - Manual for users and administrators* [1]:

- User management[21] for identity management of users
- Group management[22] for identity management of user groups

See also:

Software developers refer to *Univention Developer Reference* [3]:

- Package extended attributes[23]
- Extended options[24]

4.2.2 System management

System management includes all administrative tasks related to the underlying UCS system. These tasks include, for example, UCS system updates, management of apps such as lifecycle, configuration, and certificate handling. The purpose of system management is to simplify the daily tasks of administrators when managing multiple UCS systems.

The component *Univention Management Console (UMC)* provides the capabilities for system management on UCS and is part of the UCS management system. It offers the technology stack for the web interface of the UCS management system. UMC (Univention Management Console) consists of modules for various management tasks. Apps and software packages can provide custom UMC modules and extend the UCS management system.

UMC is a central component in UCS for the following reasons:

- UMC provides the technology stack for the web interface of the UCS management system.
- UMC provides user authentication interface to the UCS management system and *UCS portal service* (page 53).
- UMC allows extension of the UCS management system with custom UMC modules.

As component serving the web interface for the UCS management system, UMC involves a web frontend and a backend as shown in Fig. 4.3.

[21] https://docs.software-univention.de/manual/5.2/en/user-management/index.html#users-general
[22] https://docs.software-univention.de/manual/5.2/en/groups.html#groups
[23] https://docs.software-univention.de/developer-reference/5.2/en/udm/package-extended-attributes.html#udm-ea
[24] https://docs.software-univention.de/developer-reference/5.2/en/udm/package-extended-attributes.html#udm-ea-option

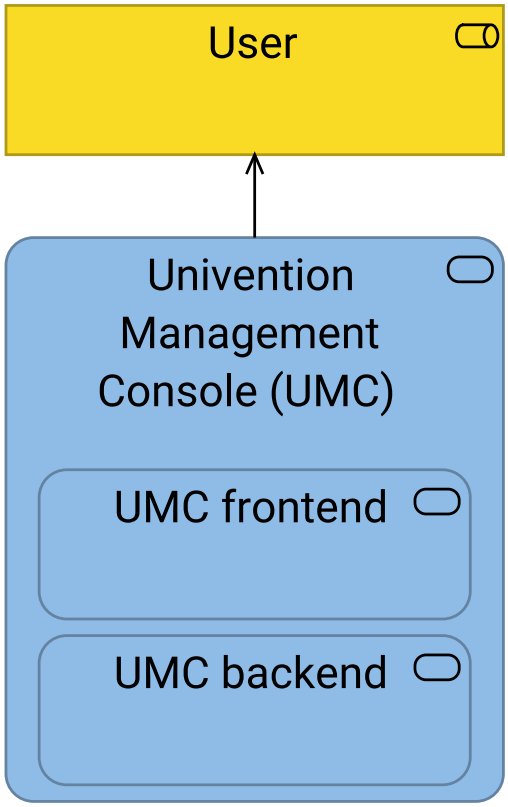

Fig. 4.3: *UMC web frontend* and *UMC backend* realize Univention Management Console

Continue reading

UMC - Univention Management Console (page 45) for description of the architecture of UMC

See also:

System administrators refer to *Univention Corporate Server - Manual for users and administrators* [1]:

- Univention Management Console modules[25] for details about UMC modules
- Expansion of UMC modules with extended attributes[26] for details about how to enhance with *extended attributes*

Software developers and system engineers refer to *Univention Developer Reference* [3]:

- Univention Management Console (UMC)[27] for technical details about UMC for software developers

4.2.3 Configuration management

Configuration management is a collection of tasks used to configure software systems. For example, changing the system's mail relay server requires updates to several configuration text files. With configuration management, an administrator changes the configuration setting in one place. The change then triggers updates to the associated configuration files.

The component *Univention Configuration Registry (UCR)* covers the local configuration management on all Univention Corporate Server systems. Services, scripts, and apps use UCR as a central configuration store. And administrators use UCR to adapt their UCS system to their needs.

[25] https://docs.software-univention.de/manual/5.2/en/central-management-umc/umc.html#central-user-interface
[26] https://docs.software-univention.de/manual/5.2/en/central-management-umc/extended-attributes.html#central-extended-attrs
[27] https://docs.software-univention.de/developer-reference/5.2/en/umc/index.html#chap-umc

UCR consists of a non-hierarchical key-value store called *UCR variables*. It provides a common interface to system settings. UCR decouples configuration settings from specific file formats such as plain text, XML, or JSON. UCR also consists of a template system and mechanisms to generate configuration files from templates and UCR variables.

UCS uses UCR variables for all configuration settings on a system. And UCS provides many templates for service configuration files.

Continue reading

Univention Configuration Registry (UCR) (page 29) for description of the architecture of UCR

See also:

Administration of local system configuration with Univention Configuration Registry[Page 26, 28]
For information about how to use UCR in *Univention Corporate Server - Manual for users and administrators* [1]

Univention Management Console (UMC)[29]
For detailed information about UCR in *Univention Developer Reference* [3]

4.3 App Center

Univention App Center is one of the most important parts of UCS. It's responsible for the lifecycle management of UCS components and third party applications that add enterprise software to the UCS domain. The App Center simplifies the installation and integration of software with UCS. In this respect, the App Center is similar in principle to the app stores on mobile platforms, with the difference that it applies to the server infrastructure.

Apps is short for software applications. Univention offers components for UCS as apps. And third party vendors, so-called app providers, offer software solutions as apps. Apps consist of software, integration with UCS and metadata, such as texts and logos for presentation. A central idea of the apps is the tight integration with UCS, especially the integration with identity management. For more information about app artifacts, refer to *App artifacts* (page 16).

Like many other product components, administrators interact with the App Center either through the web based management system or a terminal as shown in Fig. 4.4.

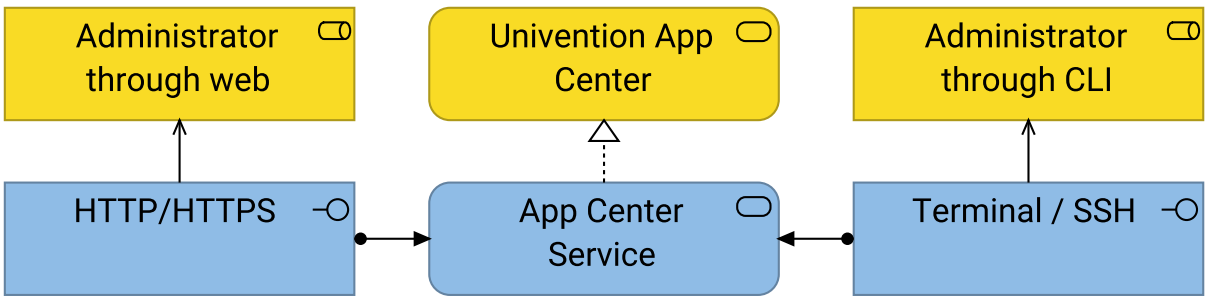

Fig. 4.4: Architecture overview of the App Center

Abstractly speaking, the application service *App Center Service* offers a web interface through *HTTP/HTTPS* and a command line interface through *Terminal / SSH*.

The following list demarcates the App Center from its capabilities. The App Center isn't:

- a tool to distribute software specific to customers or projects.
- a solution for every use case. It has limitations for example in large environments that require setups for a cluster or load balancing.

[28] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-administration-of-local-system-configuration-with-univention-configuration
[29] https://docs.software-univention.de/developer-reference/5.2/en/umc/index.html#chap-umc

Think of the App Center as a global entity in the UCS domain. The App Center addresses all UCS systems. Administrators can view and install any available app.

Continue reading

App Center service **(page 57)**
for the next detail level with description of the architecture of the App Center.

See also:

Univention app ecosystem **(page 13)**
for information about the Univention app ecosystem.

Univention App Center[Page 27, 30]
for information for administrators about the App Center in *Univention Corporate Server - Manual for users and administrators* [1]

Univention App Center for App Providers[31]
for information about how to develop apps for Univention App Center in *Univention App Center for App Providers* [2]

[30] https://docs.software-univention.de/manual/5.2/en/software/app-center.html#software-appcenter
[31] https://docs.software-univention.de/app-center/5.2/en/index.html

CHAPTER

FIVE

SERVICES

This section covers numerous services that UCS offers to IT infrastructures. It's for administrators and solution architects.

Hint: For architecture notation, this part of the document uses ArchiMate® 3.2, a visual language with a set of default iconography for describing, analyzing, and communicating many concerns of enterprise architectures. For more information about how the document uses the notation, refer to *ArchiMate* (page 70).

5.1 Univention Configuration Registry (UCR)

This section describes the architecture of UCR (Univention Configuration Registry). For a general overview about system management and the role of UCR, refer to *System management* (page 24).

You find the source code for UCR at UCS source: base/univention-config-registry/[32].

Every UCS system installs UCR per default, regardless of the system role. UCR stores all configuration settings of a UCS system, and also some other information for quick lookup, for example the validity of certificates. Administrators set values for configuration settings. Also, packages set configuration values with default values during their installation. As shown in Fig. 5.1 UCR is an important component used everywhere, for example by *UCS Packages*, *Services*, *Scripts* and also *Apps*.

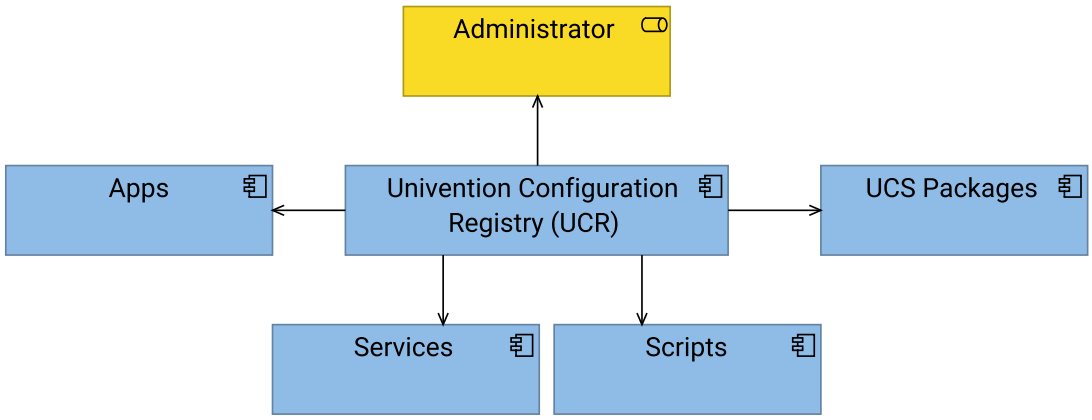

Fig. 5.1: Consumers of UCR

[32] https://github.com/univention/univention-corporate-server/tree/5.2-1/base/univention-config-registry/

5.1.1 UCR architecture

Fig. 5.2 shows the architecture overview of UCR.

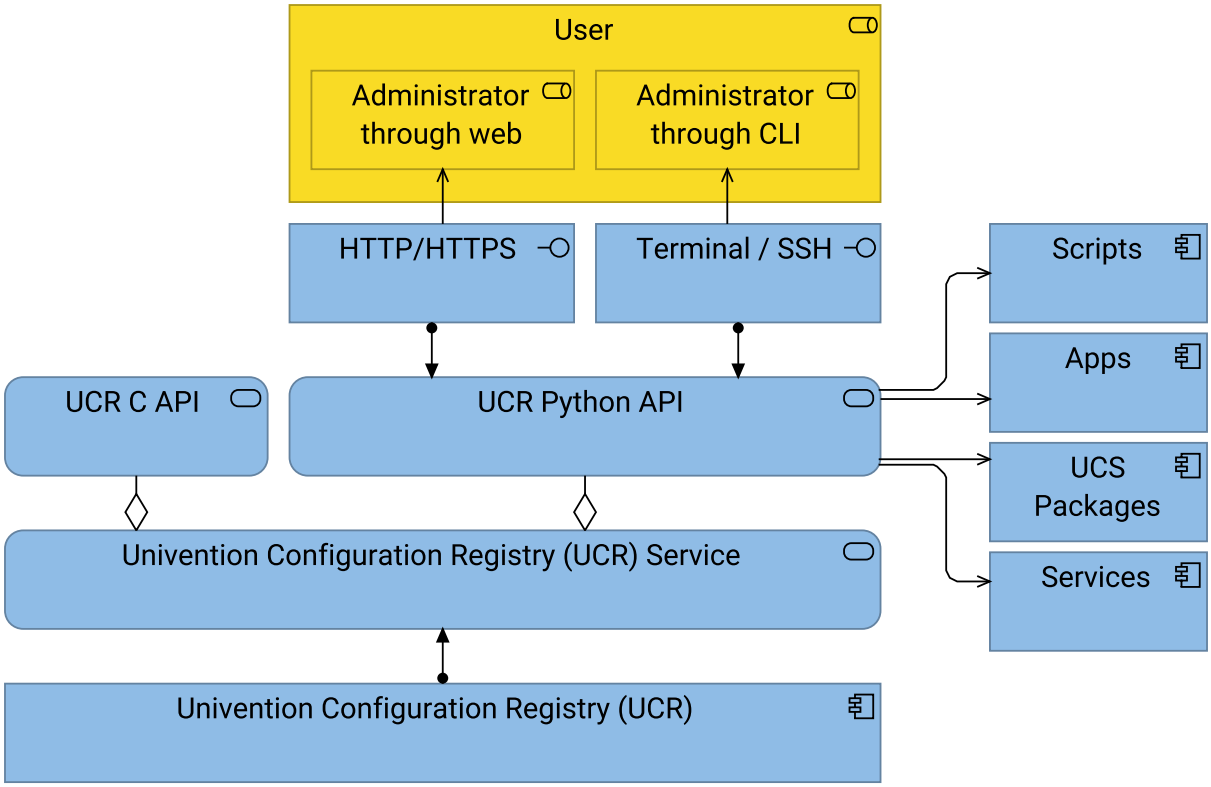

Fig. 5.2: Architecture overview of UCR

UCR provides configuration values to *Scripts*, *Apps*, *UCS Packages* and *Services* with its various configuration values from the UCR variables. *Users* such as administrators use UCR through the web interface of *UMC - Univention Management Console* (page 45) with *HTTP/HTTPS* and through the command line with *Terminal / SSH*. And *UCR Python API* offers a programming interface for UCS components and other Python programs. *UCR C API* is a small API in C for only getting and setting UCR variables.

See also:

Administrators, refer to *Univention Corporate Server - Manual for users and administrators* [1]:

- Using the Univention Management Console module[33]
- Using the command line frontend[34]

See also:

Software developers and system engineers, refer to *Univention Developer Reference* [3]:

- Using UCR from shell[35]
- Using UCR from Python[36]

From *Univention Corporate Server Python API 5.2 documentation* [4]:

- `univention.config_registry`[37] for *UCR Python API*

[33] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-using-the-univention-management-console-web-interface
[34] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-using-the-command-line-front-end
[35] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage-shell
[36] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage-python
[37] https://docs.software-univention.de/ucs-python-api/univention.config_registry.html#module-univention.config_registry

5.1.2 UCR persistence layer

Fig. 5.3 shows the relation between the active *Univention Configuration Registry (UCR) [application component]* and the passive *UCR variables*, *UCR templates* and *System configuration files*.

UCR variables
UCR is independent from any LDAP directory service. Instead, UCR uses plain text files as its storage backend for UCR variables and saves them in `/etc/univention/base*.conf`. Most UCR commands read UCR variables. The *UCR set / unset* command changes UCR variables.

The variables don't follow a hierarchy. The slash (`/`) separator exists for readability.

UCR templates
UCR templates are text file templates for configuration files of various services in UCS. They include placeholders for the UCR variables. Additionally, they can include Python code for algorithms and more complex use cases.

The template files locate at `/etc/univention/templates/files/`.

The mapping between which UCR template uses which UCR variables locates at `/etc/univention/templates/info/`.

System configuration files
When UCR variables change or administrators run the `UCR commit`[38] command, the *UCR configuration manager* determines the affected system configuration files. The manager reads the respective *UCR templates*, parses them, replaces the variable placeholders with the values from the *UCR variables*, and writes *System configuration files*. UCR commands like **`ucr set`** and **`ucr unset`** automatically trigger *UCR commit* on all affected *System configuration files* referencing the changed *UCR variables*.

UCR usually doesn't reload services affected by configuration file changes, because only the administrator knows when configuration tasks are complete and safe for restart.

Exceptions to this behavior exist. For example, changes to UCR variables starting with `interfaces/` trigger a restart of the networking service, unless you set UCR variable `interfaces/restart/auto` to `no`. Also, the Docker service restarts when UCR variables starting with `proxy/*` change.

> **Caution:** Beware that UCR overwrites any manual changes to configuration files that are under control of UCR. Such configuration files include a header with a warning. Overwriting can happen during system updates or other events that trigger a rewriting of configuration files.

Fig. 5.4 shows this general workflow after an administrator sets a UCR variable. Other actors can be *UCS Packages*, *Scripts*, or *Services*.

The *Administrator* triggers the event *UCR set variable* by using the UCR command. *UCR set / unset* writes one of the *UCR variables* and triggers a *UCR commit*. The *UCR commit* uses the *UCR variable priority*, the *UCR variables*, and the *UCR templates* to write and update the *System configuration*. After *UCR commit* finished, it triggers the *Configuration written* event.

See also:

Administration of local system configuration with Univention Configuration Registry[Page 31, 39]
for more information about using UCR in *Univention Corporate Server - Manual for users and administrators* [1].

See also:

Software developers and system engineers, refer to *Univention Developer Reference* [3]:

- Using UCR[40] for more information about how to extend or develop with UCR

[38] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#cmdoption-ucr-arg-commit
[39] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-administration-of-local-system-configuration-with-univention-configuration
[40] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage

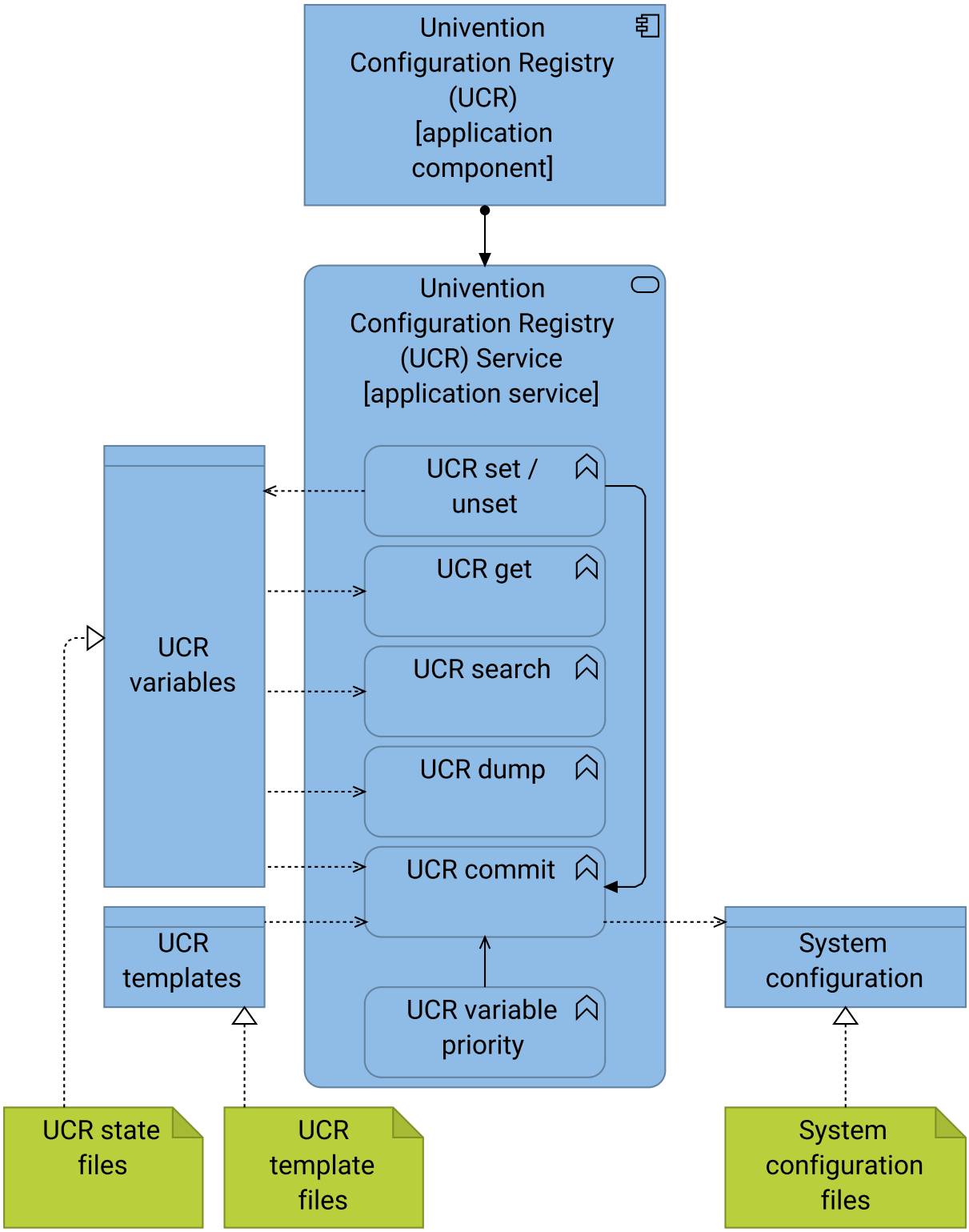

Fig. 5.3: Architecture of Univention configuration registry persistence layer

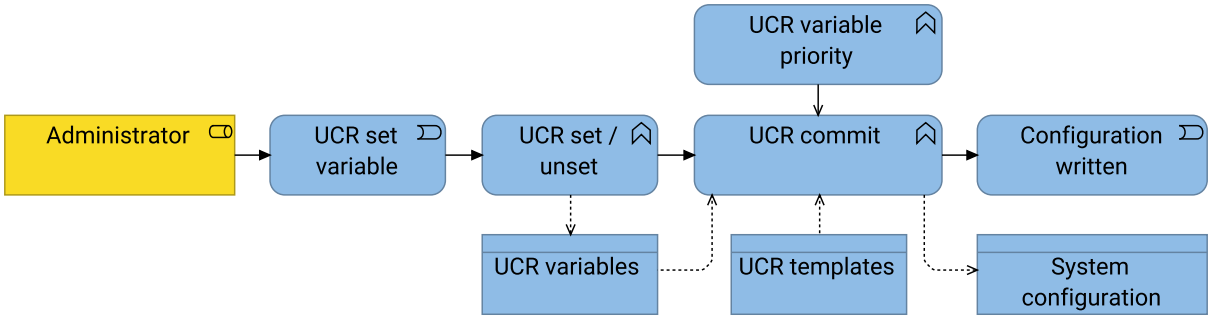

Fig. 5.4: Workflow after setting a UCR variable

- UCR Template files conffiles/path/to/file[41] for more information about writing UCR template files

For more information about how to run code or programs when specific UCR variables change, refer to the following documentation:

- Script[42] for more information about how to call external programs
- Module[43] for more information about how to run Python modules
- File[44], refer to `Preinst`, `Postinst`, and `/etc/univention/templates/scripts/`.

5.1.3 UCR variable priorities

UCR uses priority layers to determine what value actually becomes effective. The following layers from low priority to high priority exist:

Default
: The lowest priority represents the default value for UCR variables. The package that introduces the UCR variable sets the default value. This priority layer avoids default values scattered across the program code in UCS.

  New in version 5.0: *Default* layer added to UCR

  Packages must explicitly register a default value in its UCR info file, so that the UCR variables uses the *Default* layer.

  The package's `postinst` may still set the default value of UCR variables using **`ucr set name?value`**. This command stores the UCR variable in the *Normal* layer.

  Changing a UCR variable default value the "old way" without the *Default* layer requires updates in multiple code locations resulting in a major drawback with increased effort.

Normal
: The priority layer *normal* becomes effective after an administrator, a package installation or something else explicitly sets a value for a UCR variable. UCR uses this layer by default, when a role like administrator or script uses none of the options `--force`, `--schedule`, or `--ldap-policy` to explicitly use another layer.

LDAP
: By default each UCS system has its own independent UCR. For managing multiple UCS systems, administrators can define the same *UCR policies* in LDAP and apply them to several UCS systems consistently. UCS stores the values of these settings in the priority layer *LDAP*, which takes precedence over both previous layers.

[41] https://docs.software-univention.de/developer-reference/5.2/en/ucr/templates.html#ucr-conffiles
[42] https://docs.software-univention.de/developer-reference/5.2/en/ucr/configuration.html#ucr-script
[43] https://docs.software-univention.de/developer-reference/5.2/en/ucr/configuration.html#ucr-module
[44] https://docs.software-univention.de/developer-reference/5.2/en/ucr/configuration.html#ucr-file

By default, UCS systems apply *UCR policies* once per hour, but not at a fixed minute to avoid load peaks on the LDAP server. You can change the default value of *once per hour* with the UCR variable `ldap/policy/cron`[45].

Scheduled
The priority layer *scheduled* is specific to UCS@school. It temporarily overwrites UCR variables.

Forced
The priority layer *forced* has the highest priority for a regular UCS system by default. It applies to UCR variables set with the option `--force`.

Custom
The priority layer *custom* is an internal detail and not used by default. This layer applies **only** when the environment variable `UNIVENTION_BASECONF` has a value and points to a file. Then the *custom* layer has the highest priority for those processes only.

See also:

System administrators refer to *Univention Corporate Server - Manual for users and administrators* [1]:

- Policy-based configuration of UCR variables[46] for more information about how to set UCR variables with a policy
- Policies[47] for more information about *Policies* in UCS

See also:

Software developers and system engineers, refer to *Univention Developer Reference* [3]:

- debian/package.univention-config-registry[48] for more information about the UCR info file.

5.1.4 UCR limitations

UCR has the following limitations:

1. UCR variables store and return string values.
2. Values must not contain newlines (`\n`, `\r`) or zero bytes (`\0`).
3. UCR variable names should be restricted to alpha-numeric characters from the ASCII alphabet.

   UCR commands validate the variable name using the function `validate_key()`, that prohibits using many shell control characters. For more information, refer to UCS source: base/univention-config-registry/python/univention/config_registry/misc.py#L131[49].
4. It's recommended, that UCR variables shouldn't exceed the length of `1024` characters counting the length of the key and the length of the value plus 3: $key.length + value.length + 3 <= 1024$

   The underlying C implementation of UCR is the reason for the limitation. The limit isn't enforced in the implementation.
5. Access management:

   Write
   On the command line, only the user `root` can change UCR variables. UMC policies can grant proper rights to users, so that a *normal* user can also set UCR variables through *UMC - Univention Management Console* (page 45).

[45] https://docs.software-univention.de/manual/5.2/en/appendix/variables.html#envvar-ldap-policy-cron
[46] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#ucr-templates-policy
[47] https://docs.software-univention.de/manual/5.2/en/central-management-umc/policies.html#central-policies
[48] https://docs.software-univention.de/developer-reference/5.2/en/ucr/configuration.html#ucr-info
[49] https://github.com/univention/univention-corporate-server/blob/5.2-1/base/univention-config-registry/python/univention/config_registry/misc.py#L131

See also:

See also the note about the path and access rights in Using UCR from shell[50] in *Univention Developer Reference* [3].

Read

Any user or process on a UCS system can read UCR variables, because `/etc/univention/base*.conf` are world-readable.

Warning: Don't access UCR variables directly from the files. Always use the interfaces such as:

- For administrators, see *Univention Corporate Server - Manual for users and administrators* [1]:
  - web interface[51]
  - command line interface[52]
- For developers, see *Univention Developer Reference* [3]:
  - shell scripts[53]
  - Python interface[54]

5.2 Univention Directory Manager (UDM)

This section describes the technical details for UDM. For a general overview about the UCS management system and the role of UDM, see *UCS management system* (page 22).

You find the source code for UDM at UCS source: management/univention-directory-manager-modules/[55].

Other packages in UCS can also define UDM modules. The respective packages include the sources for their UDM modules. For example, the following packages also provide UDM modules:

- *App Center* (page 26) at UCS source: management/univention-appcenter/[56]
- *UCS portal service* (page 53) at UCS source: management/univention-portal/[57]
- S4 Connector at UCS source: services/univention-s4-connector/[58]

5.2.1 UDM architecture

Fig. 5.5 shows the architecture for UDM. A description of the elements follows.

LDAP directory

The data persistence layer consists of the LDAP directory, that provides the domain database, the persistence layer and data source for UDM. For communication with the LDAP directory, UDM uses the *Lightweight Directory Access Protocol (LDAP)*.

UDM uses a two layer architecture for abstraction as shown in Fig. 5.5. Except for the *LDAP directory*, all shown elements belong to UDM. The first abstraction layer at the bottom is the *UDM Python library* with the following elements:

[50] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage-shell
[51] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-using-the-univention-management-console-web-interface
[52] https://docs.software-univention.de/manual/5.2/en/computers/ucr.html#computers-using-the-command-line-front-end
[53] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage-shell
[54] https://docs.software-univention.de/developer-reference/5.2/en/ucr/usage.html#ucr-usage-python
[55] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-directory-manager-modules/
[56] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-appcenter/
[57] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-portal/
[58] https://github.com/univention/univention-corporate-server/tree/5.2-1/services/univention-s4-connector/

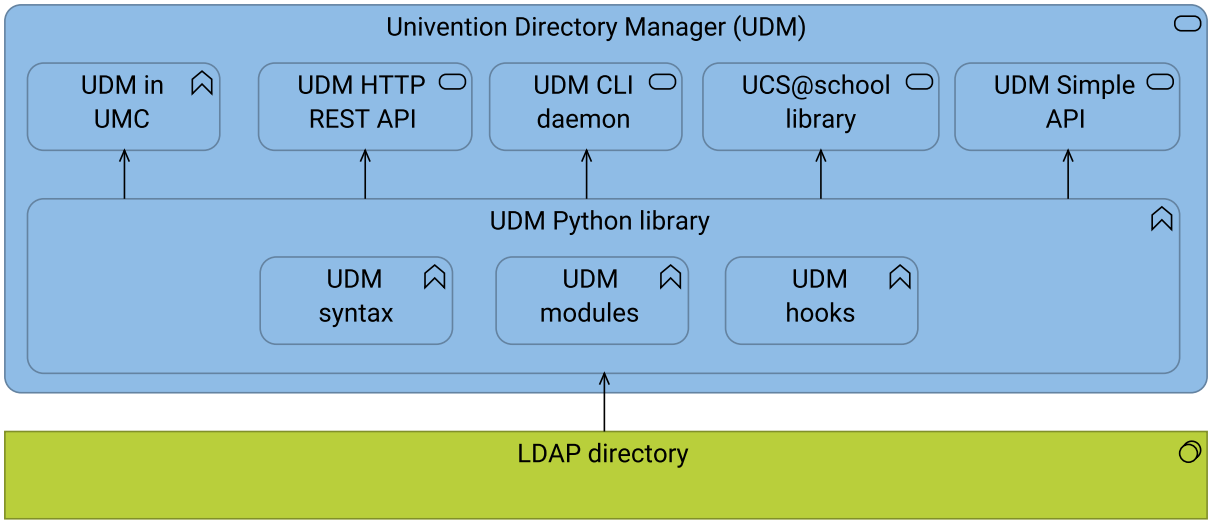

Fig. 5.5: Architecture of UDM

UDM Python library
Provides the library for abstraction and the environment for *UDM syntax*, *UDM modules*, and *UDM hooks*. *UDM Python library* uses the *LDAP directory*. You can imagine something similar to an object relational mapper for SQL. *UDM Python library* provides Python modules and classes below `univention.admin.*`[59]:

UDM Syntax
UDM syntax provides the following capability:

- Perform syntax validation on user input data.
- Present static values from a predefined list of possible values.
- Calculate possible values dynamically upon use.
- Specify the layout and widget type for presentation in UMC.

UDM modules
UDM modules translate LDAP objects to UDM objects and back. They ensure data consistency, validate user input, implement process logic and improve the usability of UCS.

- For more information about UDM modules, refer to *UDM modules* (page 38).
- For more information about UDM data, refer to *UDM data* (page 38).

UDM hooks
UDM hooks are Python classes with methods that can integrate into existing UDM modules together with *extended attributes*. They offer an alternative to customize UDM.

The second abstraction layer in Fig. 5.5 uses the *UDM Python library* and offers *UDM in UMC*, UDM HTTP REST API, the *UDM CLI daemon*, the *UCS@school library*, and the *UDM Simple API*.

UDM in UMC
Runs the UDM modules inside UMC and presents them to the user over HTTP through the web browser. It creates one process per user session for all UDM modules. *UDM in UMC* uses the *UDM Python library*.

UDM HTTP REST API
Provides the UDM HTTP REST API interface to UDM as a separate service. UDM offers HTTP access through the UDM HTTP REST API to use UDM through a remote interface.

UDM CLI Daemon
Provides the command-line interface to UDM through one system wide process for each user. The process

[59] https://docs.software-univention.de/ucs-python-api/univention.admin.html#module-univention.admin

terminates itself after a default idle time of 10 minutes. The command-line interface uses the *UDM Python library*.

UCS@school library
Provides an abstraction in Python for UCS@school. The UCS@school library uses the *UDM Python library*.

UDM Simple API
Allows to use UDM capability and objects directly in Python programs. For example, *UCS portal service* (page 53) uses the API. *UDM Simple API* provides Python modules and classes below `univention.udm.*`[60].

As mentioned before, UDM is highly customizable to the needs of environments, custom services and apps. Custom UDM modules, extended attributes and UDM hooks offer different possibilities for the customization of UDM.

See also:

Administrators, refer to *Univention Corporate Server - Manual for users and administrators* [1]:

- Expansion of UMC modules with extended attributes[61]
- Command line interface of domain management (Univention Directory Manager)[62]

See also:

Software developers and system engineers, refer to *Univention Developer Reference* [3]:

- UDM syntax[63]

From *Univention Corporate Server Python API 5.2 documentation* [4]:

- `univention.admin`[64]
- `univention.udm`[65]

5.2.2 Dependencies for UDM

UDM depends on LDAP. You can resolve the other detailed dependencies with the package manager.

The following services in UCS need UDM:

- UCS@school library
- Active Directory Connector
- S4 Connector
- *UCS portal service* (page 53)

Following the chain, *UDM in UMC* and UDM HTTP REST API wouldn't work without UDM either. From the items mentioned in *UDM architecture* (page 35) and Fig. 5.5, UDM needs the following to work properly:

- *UDM Python library*
- *UDM syntax*
- *UDM modules*
- *UDM hooks*

And UDM offers its capability to the following items:

- *Python UDM API*
- *UDM CLI daemon*

[60] https://docs.software-univention.de/ucs-python-api/univention.udm.html#module-univention.udm
[61] https://docs.software-univention.de/manual/5.2/en/central-management-umc/extended-attributes.html#central-extended-attrs
[62] https://docs.software-univention.de/manual/5.2/en/central-management-umc/udm-command.html#central-udm
[63] https://docs.software-univention.de/developer-reference/5.2/en/udm/syntax.html#udm-syntax
[64] https://docs.software-univention.de/ucs-python-api/univention.admin.html#module-univention.admin
[65] https://docs.software-univention.de/ucs-python-api/univention.udm.html#module-univention.udm

- *UCS@school library*

5.2.3 UDM modules

UDM modules represent a set of LDAP object classes and their corresponding attributes in UDM objects. They ensure data consistency, validate user input, implement process logic and improve the usability of UCS.

UDM modules exist for almost every LDAP object class. For example, UDM objects `users/user` represent different LDAP object classes like `person`, `organizationalPerson`, `inetOrgPerson`, `posixAccount`, or `shadowAccount`. Another example is the password field at a UDM object `users/user`, that creates several password hash types in the different LDAP object classes for users. UDM presents one password to the user. In the background it ensures password consistency for different services, that need different password hash types.

Python is the programming language for UDM modules. During installation UDM modules register themselves in the LDAP directory. The UCS domain replicates the UDM modules to UCS systems across the domain. On the UCS systems, the Univention Directory Listener writes the UDM modules to the systems' file system. The replication ensures the availability of all UDM modules in the UCS domain alike.

Domain administrators can grant permission to use particular UDM modules in UMC to other users. UDM modules access the LDAP directory with the permissions of the user so that LDAP *access control lists* for read and write actions apply to the user.

See also:

UDM modules[Page 38, 66]
For information about UDM modules for software developers in *Univention Developer Reference* [3].

5.2.4 UDM data

Talking about UDM modules requires a distinction between data describing a UDM object and an LDAP object:

- The term *properties* refers to data fields in UDM objects.
- The term *attributes* refers to data fields in LDAP objects.

UDM modules map between LDAP objects and UDM objects. They format data upon read and write operations to and from the LDAP directory for representation to the user as shown in Fig. 5.6. UDM modules are in the center of the data mapping and emphasize their translation role. For example, widgets in UMC show a human readable representation of the data. Fields that represent a date value offer a calendar widget to the user.

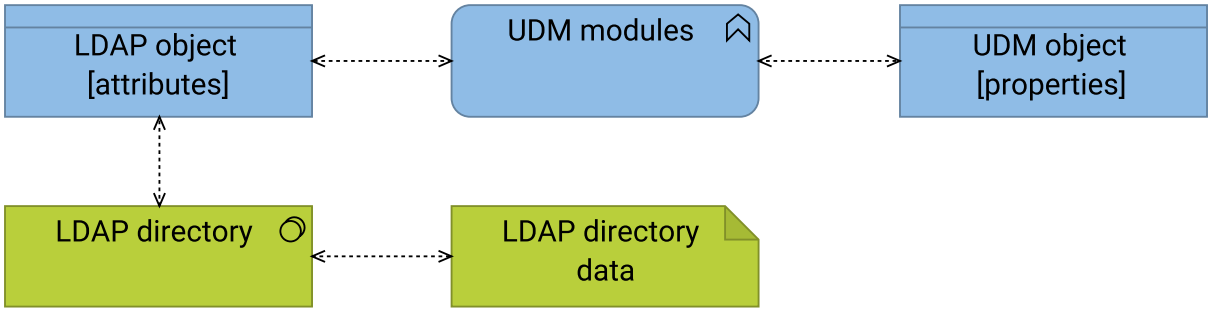

Fig. 5.6: UDM modules map data between LDAP objects and UDM objects

Extended attributes provide the capability to add and customize properties in UDM. They define a mapping between UDM properties and LDAP attributes.

See also:

[66] https://docs.software-univention.de/developer-reference/5.2/en/udm/udm-modules.html#udm-modules

Expansion of UMC modules with extended attributes[Page 39, 67]
How to use extended attributes, *Univention Corporate Server - Manual for users and administrators* [1]

5.3 UDM HTTP REST API

This section describes the technical architecture for the UDM HTTP REST API, a JSON (JavaScript Object Notation) HTTP (Hyper-Text Transfer Protocol) interface to interact with *Univention Directory Manager*. It follows a *RESTful* architecture, adheres to REST (Representational State Transfer) principles and provides an *OpenAPI* schema.

For a general overview about UDM, see *UCS management system* (page 22) and *Univention Directory Manager (UDM)* (page 35).

For information about how to use the API (application programming interface) as a developer, see UDM HTTP REST API[68] in *Univention Developer Reference* [3].

You find the source code for UDM HTTP REST API at UCS source: management/univention-directory-manager-rest/[69].

Tip: This section uses various concepts of the *ArchiMate* (page 70) notation. To avoid confusion, have a close look at the figures and make yourself familiar with the different concepts in the *Application layer* (page 73) and the *Relationships* (page 76) in the appendix.

Fig. 5.7 shows the relation of UDM HTTP REST API down the line from the UCS *Product components* with the *UCS management system* (page 22), and its *Domain management* (page 23) implemented by *Univention Directory Manager (UDM)*.

The *Univention Corporate Server* application component has *Product components* services assigned to it. *Product components* are an abstract container for the main product building blocks, see *Product components* (page 21). *Product components* consist of the *UCS Management system* and other application services. *UCS Management System* application service consists of the *Domain Management* application service and others. Univention Directory Manager (UDM) application service serves the application services *Domain management* and UDM HTTP REST API.

Hint: Fig. 5.7 isn't a layer diagram, because it uses composition and aggregation relations between the different concepts.

5.3.1 Architecture

Fig. 5.9 provides an overview of the architecture of UDM HTTP REST API.

The main building blocks are the following concepts:

UDM HTTP REST API application component
The central part of the UDM HTTP REST API is the application component that contains the respective application services for communication with the outside world, the server, and the gateway.

The package `univention-directory-manager-rest` provides this application component and all the pieces outlined later.

UDM HTTP REST API application service
The application service that the UDM HTTP REST API explicitly exposes. It's an abstraction of the other application processes that realize it.

[67] https://docs.software-univention.de/manual/5.2/en/central-management-umc/extended-attributes.html#central-extended-attrs
[68] https://docs.software-univention.de/developer-reference/5.2/en/udm/rest-api.html#udm-rest-api
[69] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-directory-manager-rest/

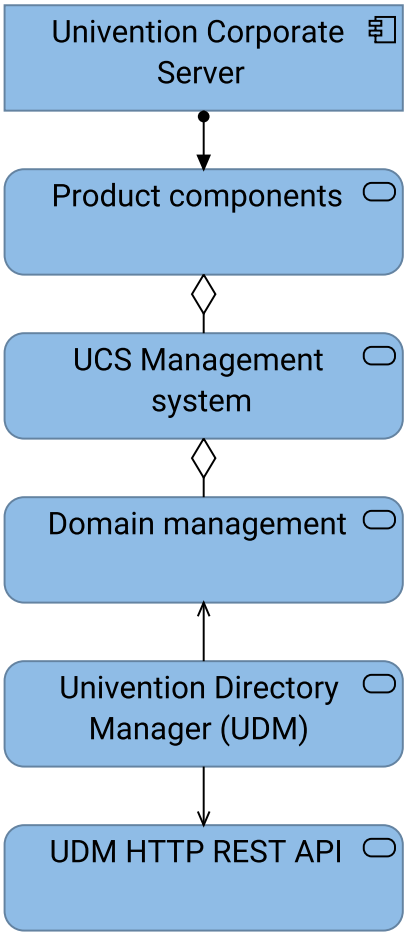

Fig. 5.7: UDM HTTP REST API as part of UDM in the domain management

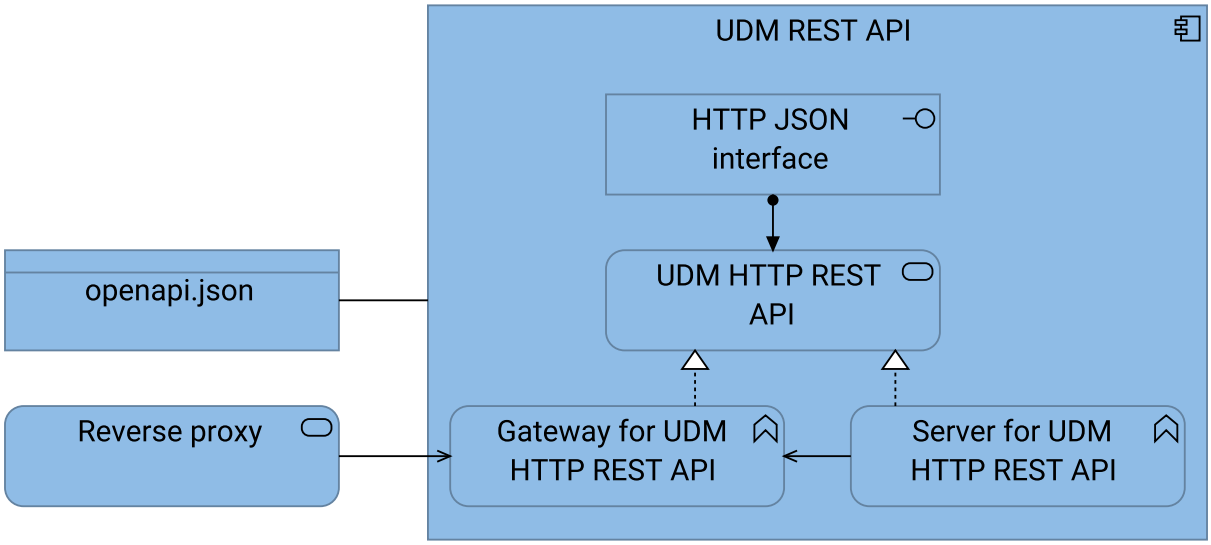

Fig. 5.8: UDM HTTP REST API overview in a nested view

***Gateway* for UDM HTTP REST API application process**
: UDM HTTP REST API launches one *Gateway* process. It forwards each request from the *Reverse Proxy* to the appropriate *Server* for UDM HTTP REST API process with the required locale.

***Server* for UDM HTTP REST API application process**
: *Server* for UDM HTTP REST API is a dedicated server process for each configured natural language. It serves the content accordingly.

***Reverse proxy* application service**
: The *Reverse proxy* functions as gateway. It adds HTTP security headers and forwards HTTP requests to the *Gateway* for UDM HTTP REST API service. It also handles errors in case the *Server* for UDM HTTP REST API is unreachable. It's part of the web server on UCS.

***HTTP JSON interface* application interface**
: UDM HTTP REST API can answer requests in the `HAL+JSON` format.

  Hypertext Application Language (HAL) provides hypermedia controls to navigate the API efficiently and independently.

openapi.json
: The *openapi.json* describes the *HTTP JSON interface* in the OpenAPI schema following the OpenAPI specification. The JSON file allows to auto-generate RPC clients.

5.3.2 Technology

Fig. 5.9 shows the architecture in a non-nested view with some more concepts around the reverse proxy. It also adds the technology layer with *Tornado*, *Apache HTTP server* and *Apache module mod_proxy*.

Tornado implements the server and the gateway application process for the UDM HTTP REST API. As other services also use *Apache HTTP server*, so does the UDM HTTP REST API.

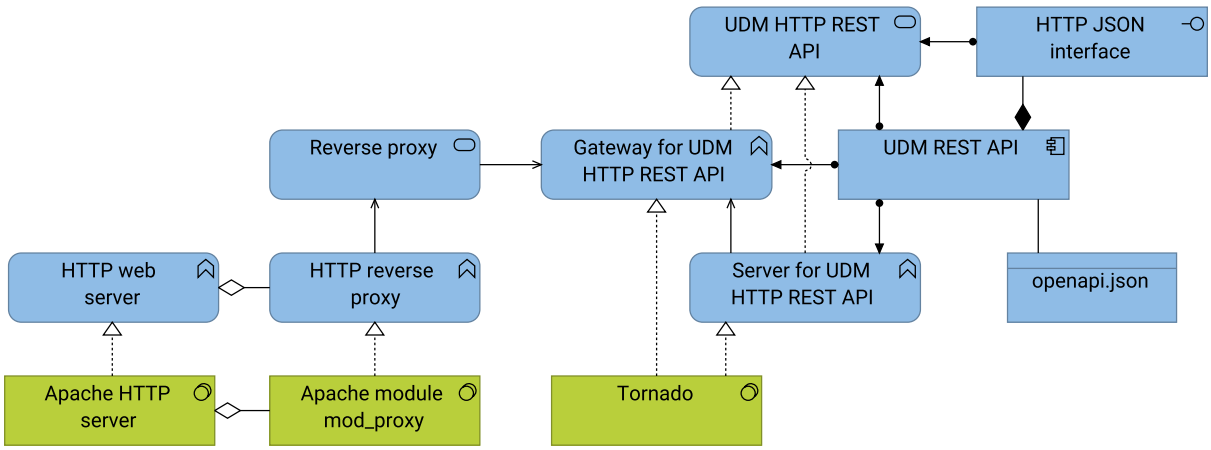

Fig. 5.9: UDM HTTP REST API and its relation to the web server

You can see in Fig. 5.9, that the UDM HTTP REST API application services is an abstraction for the application processes *Gateway* for UDM HTTP REST API and *Server* for UDM HTTP REST API. All three concepts are assigned to the UDM HTTP REST API application component.

5.3.3 Request flow

Fig. 5.10 shows the abstract flow of a request through the different concepts to the data store in the domain database *LDAP directory*. The flow emphasizes the dependency of the UDM HTTP REST API to UDM. For more information about the UDM architecture and how *UDM Python library* relates to it, see *UDM architecture* (page 35).

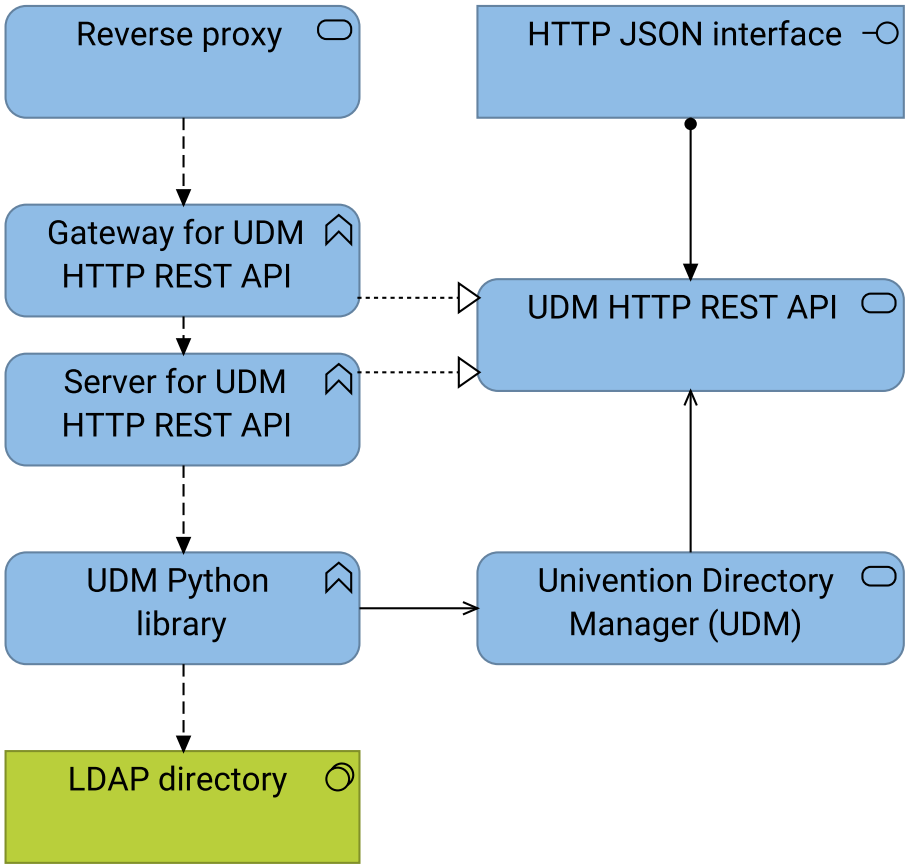

Fig. 5.10: Request flow for UDM HTTP REST API

5.3.4 Capabilities

UDM HTTP REST API provides capabilities as shown in Fig. 5.11. Different concepts of the UDM HTTP REST API realize different capabilities, so that all of them serve a dedicated purpose.

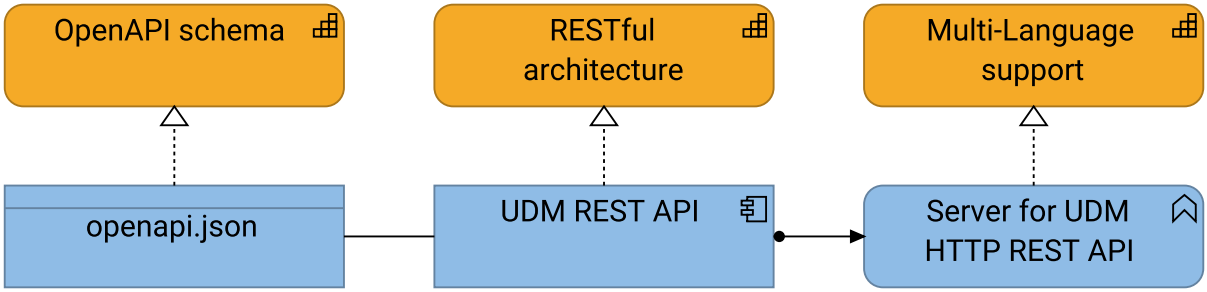

Fig. 5.11: UDM HTTP REST API capabilities

Hint: A capability in *ArchiMate* (page 75) represents an ability that an active structure element possesses.

In the Fig. 5.11, you see different relations such as realization, aggregation, and assignment. Be aware of their different meaning.

OpenAPI schema
The *OpenAPI schema* provides the definition of the UDM HTTP REST API in a programming language agnostic manner. It uses the OpenAPI specification and helps to transfer the knowledge about the API from the API provider to the API consumer.

RESTful architecture
For more information, see *RESTful architecture* (page 43).

Multi-Language support
The *UCS management system* (page 22) supports multiple languages, such as English and German. UDM HTTP REST API belongs to the UCS management system and therefore supports the same set of languages. Language support is important for UDM HTTP REST API to provide localized messages to the client and the user.

See also:

What is OpenAPI?[Page 43, 70]
for more information about OpenAPI and the specification.

5.3.5 RESTful architecture

The UDM HTTP REST API adheres to the *RESTful* architecture as defined in Fielding [5]. The term *REST* stands for **Re**presentation **S**tate **T**ransfer and includes six architectural and four interface constraints that make a service *RESTful*.

See also:

UCS source: /management/univention-directory-manager-rest/README.md - "UDM HTTP API"[71]
for a detailed description about the RESTful architecture, the rationale of the constraints, compliance and compliance violations, and the OpenAPI interface.

Architectural constraints

The six architectural constraints are the following

1. Client-server

   The *client-server constraint* enforces a clear separation between a passive server component and an active client component. The server component has the authority over the entire service realm and its meaning. The client component must not make any assumptions about the server logic.

   The client-server constraint allows clients and servers to evolve independently, because it supports separation of concerns and reduces interdependencies. Clients focus on the user interface and hypermedia. Servers focus on business logic and the representation of resources.

2. Stateless

   The *stateless constraint* enforces a stateless communication between clients and servers. This means that each request must contain all the information necessary for the server to fully understand and process the request. The client is responsible for handling all session state. This separation allows scalability by adding server instances or processes, since each server can handle requests independently.

   Stateless communication simplifies the server implementation and enables service scalability.

3. Cache

   The *cache constraint* forces data in a response to be either explicitly or implicitly enabled for caching. Caching improves performance by reducing the need for repeated requests to the server.

[70] https://www.openapis.org/what-is-openapi
[71] https://github.com/univention/univention-corporate-server/blob/5.2-1//management/univention-directory-manager-rest/README.md

4. Uniform interface

   The *uniform interface constraint* requires that components communicate using generic and standardized data formats that all participating components understand. The interface must satisfy the interface constraints described later.

   The server must provide the same unified interface that satisfies the data manipulation constraint of all server data. Clients, servers, or other intermediaries can work seamlessly with the API using the same standardized interface. The API doesn't require application-specific data formats or schemas.

   The standardized data format JSON focuses on structure and representation of data. The lacks of mechanisms for semantic and hypermedia interaction make JSON unsuitable as uniform interface.

5. Layered system

   The *layered system constraint* extends the client-server constraint by introducing intermediate components that have the ability to fully understand and manipulate messages. The intermediate components use the principles of *stateless* and *self-describing messages* to extend the architecture. Crucially, each layer operates behind a unified interface that hides layer specifics from clients and components. This layer opacity gives the system a remarkable degree of flexibility and adaptability.

6. Code-on-demand (optional)

   The *code-on-demand constraint* gives servers the optional ability to extend client functionality by embedding code in representations. This optional constraint comes with the trade-off of potentially limiting availability to clients capable of running the embedded code.

Interface constraints

The four interface constraints are the following:

1. Identification of resources

   The *identification of resources constraint* means that the server abstracts all information as a resource. Each resource must have one or more names or identifiers, typically represented by a unique HTTP URI. The server manages the URIs and has the authority to assign them. URIs serve as straightforward identifiers and don't carry any additional semantic information.

   Clients access resources using resource identifiers only. Clients should refrain from manually constructing URIs unless the server provides URI templates. Clients navigate through state transitions using links found within retrieved representations, allowing them to follow hypermedia links and traverse the API without hardcoded URIs. The server can change URIs without disrupting clients.

2. Manipulation of resources through representations

   A resource represents a set of entities that the API reflects through representations or identifies through URIs when a concrete realization of the concept doesn't yet exist. This fundamental principle implies that the state and representation of a resource can change dynamically over time while remaining the same resource.

   It's important to understand that a representation of a resource isn't the resource itself. The API represents a resource in various formats, such as HTML, XML, JSON, LDIF representing it's current state, key-value pairs representing the wanted state, images, or even error conditions such as `404 Not Found`. In REST, the client achieves state changes by examining the response and the ways the response provides to modify the representation. This involves selecting a transformation, creating, or modifying a representation, and sending it back to the server.

3. Self-descriptive messages

   The *self-descriptive message constraint* ensures that the API transmits messages as representations consisting of resource or request data metadata and control data.

   The MIME media type of the request data plays a critical role in specifying both the syntax and semantics of message payloads.

   Metadata, presented in the form of key-value pairs, describes how to interpret the message, defines caching rules, provides authentication information, specifies encodings, languages of representation, and more.

Control data, a form of metadata, describes metadata, and enables various capability.

4. Hypermedia as the Engine of Application State

   The *hypermedia as the engine of application state (HATEOAS) constraint* means that representations must not only convey data, but also contain information to control the state of the application. Each response should include all available state transfer capabilities, such as HTML forms, state changes links, URI templates, or other relevant resources.

   Hypermedia refers to data formats that can include hyperlinks and other hypermedia elements. Specifications such as *JSON-LD*, *UBER*, *SIREN*, *HAL*, *Collection+JSON*, and *Hydra* extend JSON to include hypermedia elements.

   HATEOAS has the following requirements:

   - The client must know the media type and it must be rich enough to describe all possible client-server interactions.
   - The client should only follow links contained in the representation, and shouldn't construct identifiers without user interaction.

5.3.6 Dependencies

You can resolve the other detailed dependencies with the package manager. UDM HTTP REST API depends on the following elements:

- *Univention Directory Manager (UDM)* (page 35)
- *UMC - Univention Management Console* (page 45) for providing the components for the caching of LDAP connections
- UDM-UMC module, a dedicated *UMC module* (page 52) that provides the common abstraction of UDM modules.
- *Tornado*

The following *server roles* (page 10) need UDM HTTP REST API:

- UCS Primary Directory Node
- UCS Backup Directory Node

5.4 UMC - Univention Management Console

This section describes the technical architecture of the Univention Management Console (UMC). For a general overview and its relation to system management, refer to *System management* (page 24).

You find the source code at the following locations:

- UCS source: management/univention-management-console/[72]
- Web interface presentation layer at UCS source: management/univention-web/[73]
- Packages with UMC modules usually include a `umc` directory, for example:
  - UCS source: base/univention-quota/umc/[74]
  - UCS source: base/univention-system-setup/umc/[75]
  - UCS source: base/univention-updater/umc/[76]

[72] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console/
[73] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-web/
[74] https://github.com/univention/univention-corporate-server/tree/5.2-1/base/univention-quota/umc/
[75] https://github.com/univention/univention-corporate-server/tree/5.2-1/base/univention-system-setup/umc/
[76] https://github.com/univention/univention-corporate-server/tree/5.2-1/base/univention-updater/umc/

- UCS source: management/univention-appcenter/umc/[77]
- UCS source: management/univention-management-console-module-join/umc/[78]
- UCS source: management/univention-management-console-module-adtakeover/umc/[79]
- UCS source: management/univention-management-console-module-diagnostic/umc/[80]
- UCS source: management/univention-management-console-module-ipchange/umc/[81]
- UCS source: management/univention-management-console-module-reboot/umc/[82]
- UCS source: management/univention-management-console-module-services/umc/[83]
- UCS source: management/univention-management-console-module-top/umc/[84]
- UCS source: management/univention-management-console-module-ucr/umc/[85]
- UCS source: management/univention-management-console-module-udm/umc/[86]
- UCS source: management/univention-management-console-module-welcome/umc/[87]
- UCS source: management/univention-self-service/umc/[88]
- UCS source: management/univention-server-overview/umc/[89]
- UCS source: management/univention-system-info/umc/[90]
- UCS source: services/univention-ad-connector/umc/[91]
- UCS source: services/univention-admin-diary/umc/[92]
- UCS source: services/univention-pkgdb/umc/[93]
- UCS source: services/univention-printserver/umc/[94]

Every UCS system installs UMC and its dependencies per default. UMC consists of the *UMC frontend* and the *UMC backend*. Fig. 5.12 shows the simplified architecture of Univention Management Console and the description thereafter.

The *UMC frontend* has the following items:

- *UMC web frontend*
- *UMC client*

The *UMC backend* has the following items:

- *Static HTTP server*

[77] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-appcenter/umc/
[78] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-join/umc/
[79] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-adtakeover/umc/
[80] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-diagnostic/umc/
[81] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-ipchange/umc/
[82] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-reboot/umc/
[83] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-services/umc/
[84] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-top/umc/
[85] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-ucr/umc/
[86] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-udm/umc/
[87] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-management-console-module-welcome/umc/
[88] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-self-service/umc/
[89] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-server-overview/umc/
[90] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-system-info/umc/
[91] https://github.com/univention/univention-corporate-server/tree/5.2-1/services/univention-ad-connector/umc/
[92] https://github.com/univention/univention-corporate-server/tree/5.2-1/services/univention-admin-diary/umc/
[93] https://github.com/univention/univention-corporate-server/tree/5.2-1/services/univention-pkgdb/umc/
[94] https://github.com/univention/univention-corporate-server/tree/5.2-1/services/univention-printserver/umc/

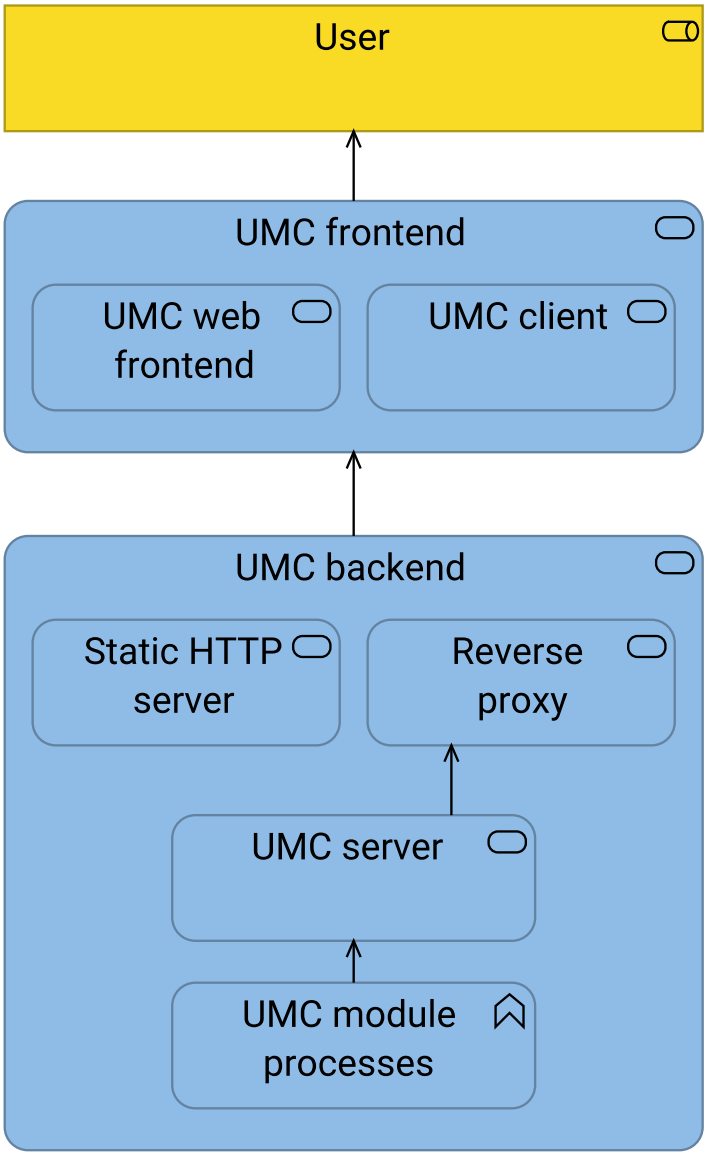

Fig. 5.12: Architecture overview of Univention Management Console

- *Reverse proxy*
- *UMC server*
- *UMC modules*

The user facing parts of the *UMC frontend* are the *UMC web frontend* and the *UMC client*. *Reverse proxy* handle and transform the requests and pass them to the *UMC server* at the backend.

5.4.1 UMC communication

This section focuses on the communication within UMC. Fig. 5.13 shows the architecture with the communication interfaces *HTTP/HTTPS*, *HTTP*, *Terminal/SSH*. The following sections describe the interfaces.

HTTP/HTTPS in UMC

The user interacts with the *UMC web frontend* in their web browser. The *UMC web frontend* communicates through *HTTP/HTTPS* with the *UMC backend*. The *Reverse proxy* receives requests, handles SSL/TLS, and forwards the requests through *HTTP* to the *UMC server*.

Terminal and SSH in UMC

The *UMC client* communicates with *UMC backend* through `HTTPS`. Administrators use UMC through the *UMC web frontend* or through specific command-line tools.

Caution: Although UMC offers a *Command line* through *Terminal/SSH*, only software developers use the interface for example for software testing. Interaction with the interface requires knowledge about the internals of *UMC modules*.

5.4.2 Authentication

UMC provides the web and authentication interface of the UCS management system. Users authenticate through a regular form-based login, basic HTTP authentication or SAML (Secure Authentication Markup Language).

In UMC, the *UMC server* implements SAML in the *SAML service provider* role. The *UMC server* considers SAML authenticated users as authenticated.

Fig. 5.14 shows how the *UMC server* handles user authentication.

Successful authentication
: *UMC server* creates a session and returns a session cookie.

Unsuccessful authentication
: *UMC server* denies the connection and answers with a denied request towards the user. The reasons can be manifold, for example:

  - Wrong username and password combination
  - Deactivated user account
  - Expired password
  - Locked account because of too many failed login attempts

The *UMC server* uses the PAM (pluggable authentication module) stack on UCS to validate and authenticate users for usual login and for SAML authentication. *UMC server* evaluates ACL (access control list)s to grant or deny the usage of UMC modules. To find the user object for the authenticating user, *UMC server* runs an LDAP search for the username. It also allows to authenticate users with their email address. Furthermore, PAM recognizes deactivated user accounts, expired passwords, and allows to change an expired password during sign-in.

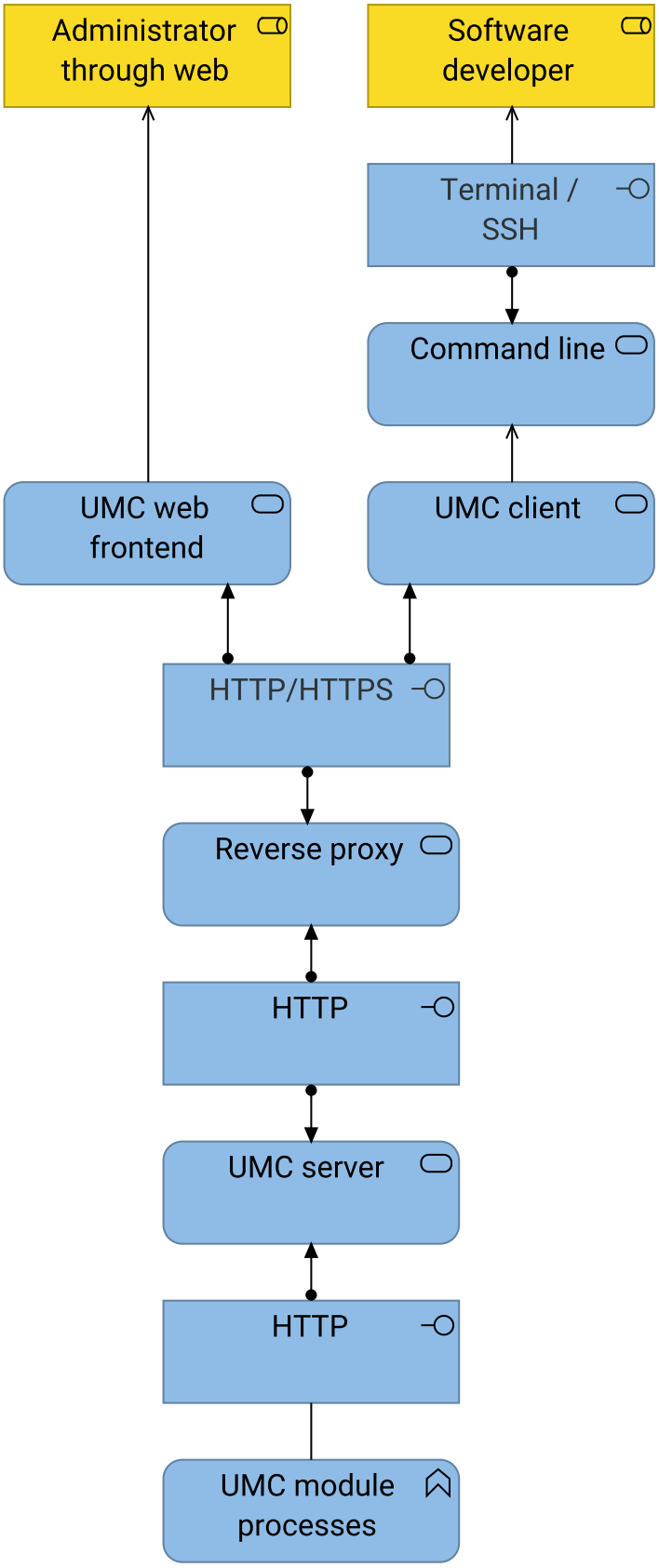

Fig. 5.13: Architecture of Univention Management Console with communication interfaces

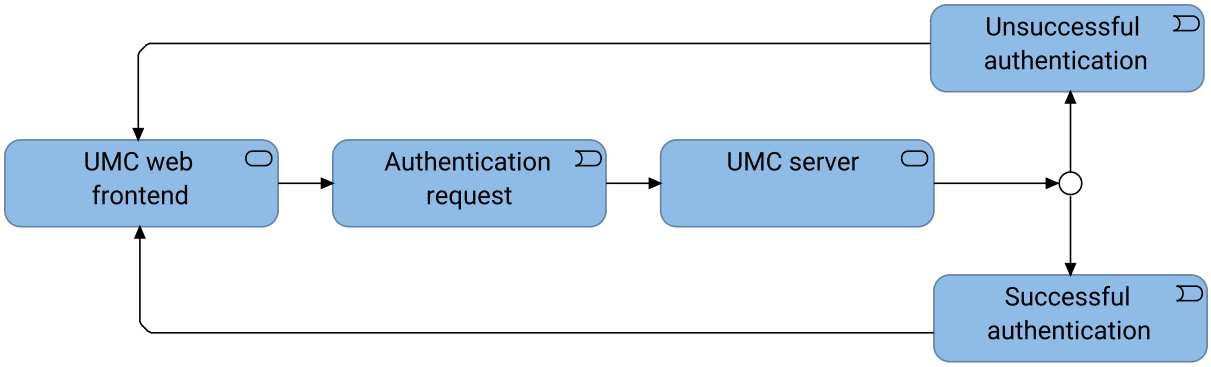

Fig. 5.14: Authentication chain in UMC

See also:

Administrators, refer to *Univention Corporate Server - Manual for users and administrators* [1]:

User management module - Account tab[Page 50, 95]
for information about deactivated and expired user accounts

Automatic lockout of users after failed login attempts[96]
for information about failed login attempts and how UCS handles them in Samba, PAM and OpenLDAP

5.4.3 UMC backend

The *UMC backend* consists of the following items as shown in Fig. 5.12:

- *Reverse proxy*
- *UMC server*
- several *UMC modules*

In Fig. 5.15 you also see the *Reverse proxy*. In fact, the web server offering the *Reverse proxy* consists of more parts.

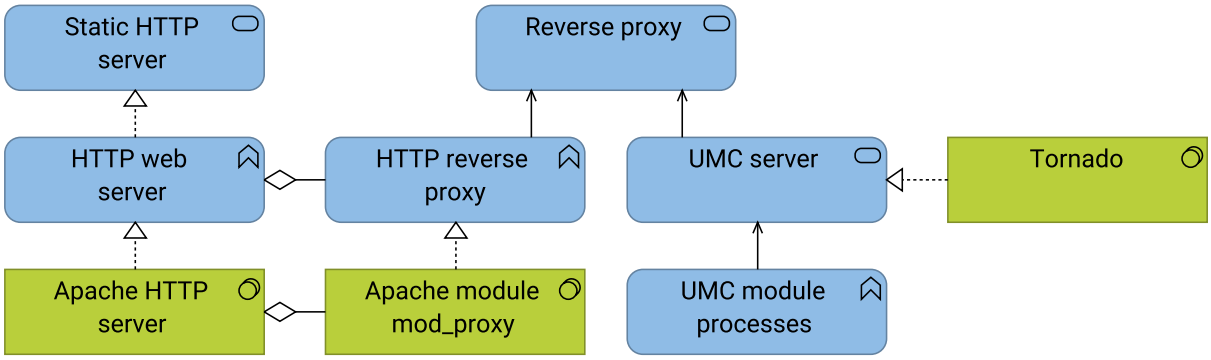

Fig. 5.15: Parts of the *UMC backend*

Static HTTP server
First is the web server realized by *Apache HTTP server*. The web server provides the *Static HTTP server* that delivers the static files for the *UMC web frontend*. And the *Static HTTP server* responds with important HTTP headers for caching rules of the static files and security related headers like for example content security policy[97].

[95] https://docs.software-univention.de/manual/5.2/en/user-management/umc.html#users-management-table-account
[96] https://docs.software-univention.de/manual/5.2/en/user-management/user-lockout.html#users-faillog
[97] https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP

Reverse proxy
Second is the reverse proxy capability from the *Apache HTTP server* with the *Apache module mod_proxy*. The *Reverse proxy* also responds with important HTTP headers similar to the *Static HTTP server*.

The *Reverse proxy* redirects the following URI paths to the *UMC web server*:

- `/univention/set/.*` as regular expression
- `/univention/auth`
- `/univention/logout`
- `/univention/saml/.*` as regular expression
- `/univention/oidc/.*` as regular expression
- `/univention/command/.*` as regular expression
- `/univention/upload/.*` as regular expression
- `/univention/get/.*` as regular expression

UMC server
Further down the chain is the *UMC server* realized by *Tornado*, that only allows connections from the *Reverse proxy*. For example, it provides session management for signed in users.

The *UMC server* accepts requests with `HTTP`. For example, the *UMC client* uses it as connection endpoint. When a `HTTP` request reaches the *UMC server*, the *UMC server* maps the request to a dedicated UMC module depending on the URL and answers the request accordingly. The *UMC server* opens an IPC (interprocess-communication) socket to the UMC module and they talk `HTTP`. It handles some requests directly, for example `get/` and `set/`, and takes care of authentication and the language setting for the web content.

UMC module processes
UMC modules extend UCS with capability. For the description, refer to *UMC modules* (page 52).

5.4.4 UMC web frontend

The *UMC web frontend* is responsible for the presentation layer of UMC and runs in the user's web browser. It uses the modular JavaScript framework *Dojo Toolkit* to create dynamic widgets. And it uses the *Bootstrap* CSS framework for responsive designed web pages.

Fig. 5.16 provides a detailed view on the model of the *UMC web frontend*.

The *UMC web frontend* consists of static files for JavaScript, HTML and CSS. The *UMC backend* sends the static files to the user's web browser, where the web browser presents UMC as a web application. The following packages from UCS source: management/univention-web/[98] contain the artifacts for the web front user interface:

`univention-web-js`
Contains the ready-to-use JavaScript files built with *Dojo Toolkit*.

`univention-web-styles`
Contains the ready-to-use CSS files for the web design including the graphical theme built with *Bootstrap*.

`univention-management-console-frontend`
Contains the HTML files for the *UMC web frontend*. More packages like **`univention-server-overview`**, **`univention-management-console-login`**, **`univention-system-setup`**, **`univention-portal`** and others also contain HTML files for the *UCS management system*.

[98] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-web/

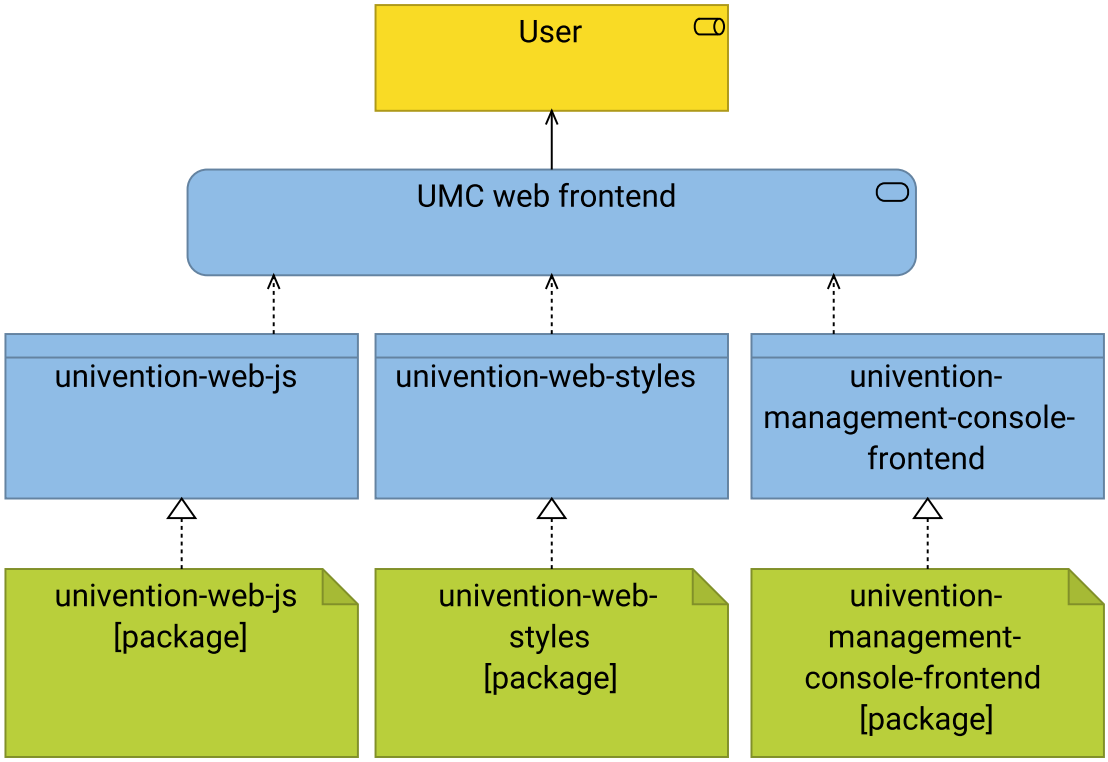

Fig. 5.16: Model for UMC web frontend

5.4.5 UMC modules

This section covers *UMC modules*. For the context of *UMC modules*, refer to *UMC backend* (page 50).

UMC modules extend UCS with capability. Each UMC module defines its command behavior with a Python implementation and its web frontend presentation with JavaScript as shown in Fig. 5.17.

Depending on the system role, UCS already installs UMC modules per default during installation. Such modules are for example the *App Center*, or *Package Management*. Furthermore, apps from the App Center can also extend UMC with additional modules, for example the *OX License Manager* or *OpenVPN4UCS*.

Every UMC module runs its own module process per user session on UCS with the user permission according to the requesting user. The encapsulation with separate processes ensures that UMC modules don't interfere with each other. One disadvantage is the additional memory consumption of every UMC module process.

UMC module processes don't run continually. After an idle time of ten minutes and if no open requests exist and no additional requests came in, module processes stop. The *UMC server* checks for running UMC module processes for every request. If the requested process doesn't run, the *UMC server* starts the UMC module process.

Tip: Use `umc/module/timeout` to configure the idle time for the UMC module processes. The default value is 10 minutes.

See also:

Development and packaging of UMC modules[Page 52, 99]
for information about development and packaging for UMC modules in *Univention Developer Reference* [3]

[99] https://docs.software-univention.de/developer-reference/5.2/en/umc/local-system-module.html#umc-module

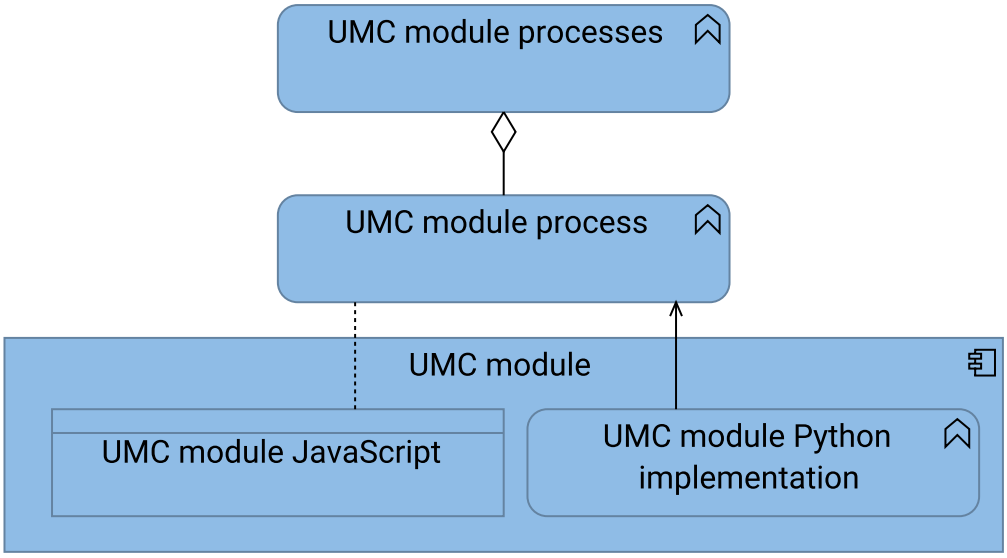

Fig. 5.17: Architecture of a UMC module

5.5 UCS portal service

This section describes the technical architecture of the UCS portal service. For a general overview, see *UCS portal* (page 22).

Every UCS system role installs the UCS portal and its dependencies per default. The UCS portal generates structured data in the JSON format. The data persistence layer consists of cache files with structured data in the JSON format. The UCS portal needs information about the tiles on the portal and about user memberships in user groups. Portal frontend and backend use HTTP for communication.

You find the source code at UCS source: management/univention-portal/[100].

Fig. 5.18 shows the architecture of the UCS Portal and the description below.

The *User* uses the *UCS Portal* through a web browser with HTTP/HTTPS. The *Portal frontend* (page 55) and the *backend* together realize the *UCS portal*. The *Portal backend* (page 55) validates the user login with the *UMC server* and uses structured data from the *UCS Portal tile cache* and the *UCS group cache*.

The UCS Portal uses the following technology:

HTTP request handler
: The UCS Portal backend uses *Tornado* to handle the HTTP requests from the frontend and to serve the data to the frontend. Tornado[101] is a Python web framework and asynchronous networking library.

Single-page application
: *Vue.js* with *TypeScript* is the technology behind the web frontend of the portal. It serves the single-page application of the portal to the user. The decision came to *Vue.js*, because it's flexible, painless, and not owned by a company. The implementation began with *Vue.js* 3, because it has full *TypeScript* support and many improvements compared to *Vue.js* 2.

[100] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-portal/
[101] https://www.tornadoweb.org/en/stable/

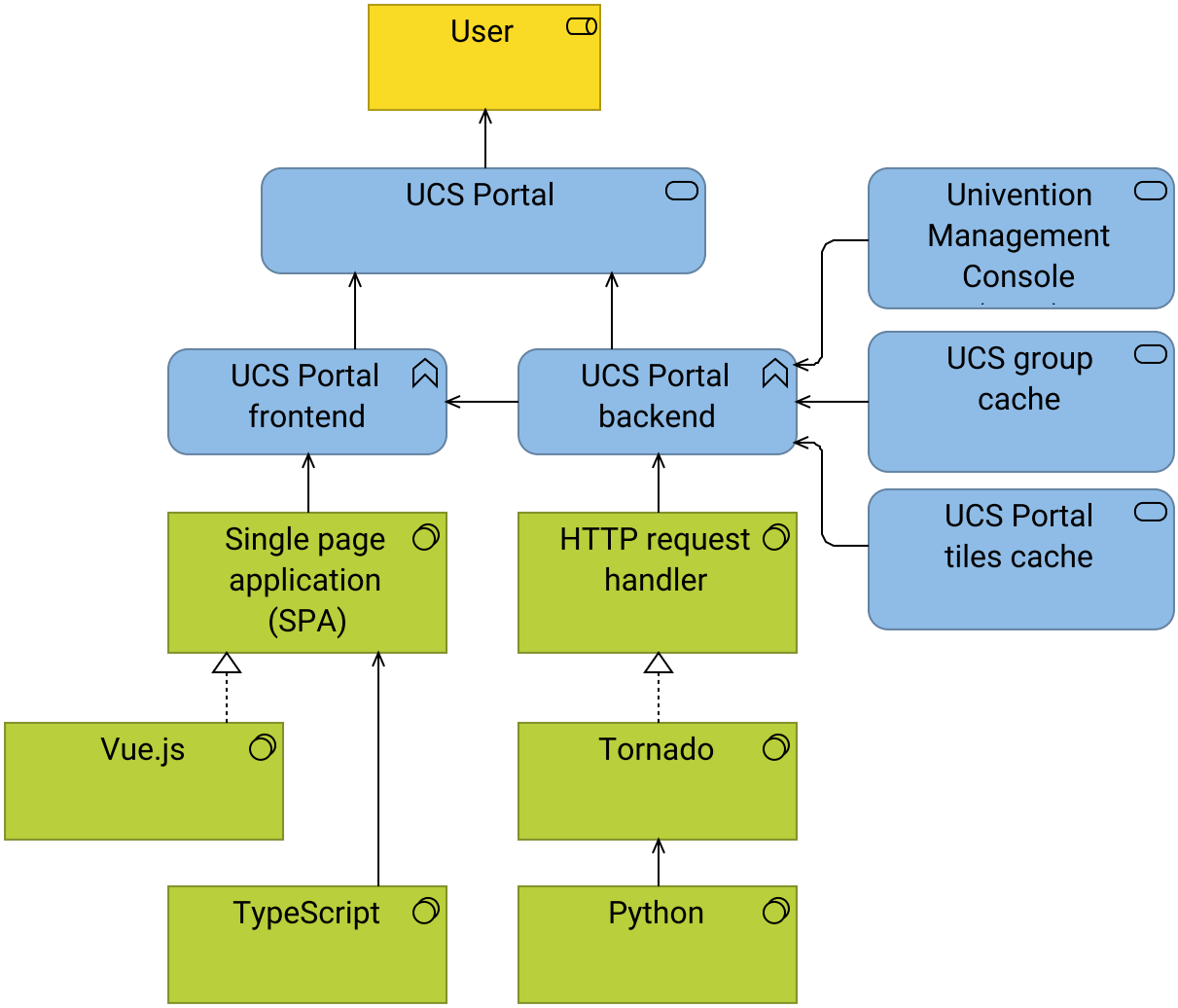

Fig. 5.18: Architecture of the UCS Portal

5.5.1 Portal frontend

The portal frontend is a single-page application[102] and renders the UCS portal in the users' web browser. Users see for example the portal header, background image, a menu and various tiles consisting of logo, title, and description.

The portal requests the structured data in `portal.json` about what to render from the *Portal backend* (page 55).

5.5.2 Portal backend

The portal backend generates the data about what portal the frontend renders for the user.

The portal backend delegates the user authentication to the UMC server. It maintains internal caches for the portal content and the user group memberships. It doesn't request LDAP or *Univention Directory Manager (UDM)* (page 35) directly.

Fig. 5.19 shows the architecture of the portal backend. A description about the elements and their responsibility follows.

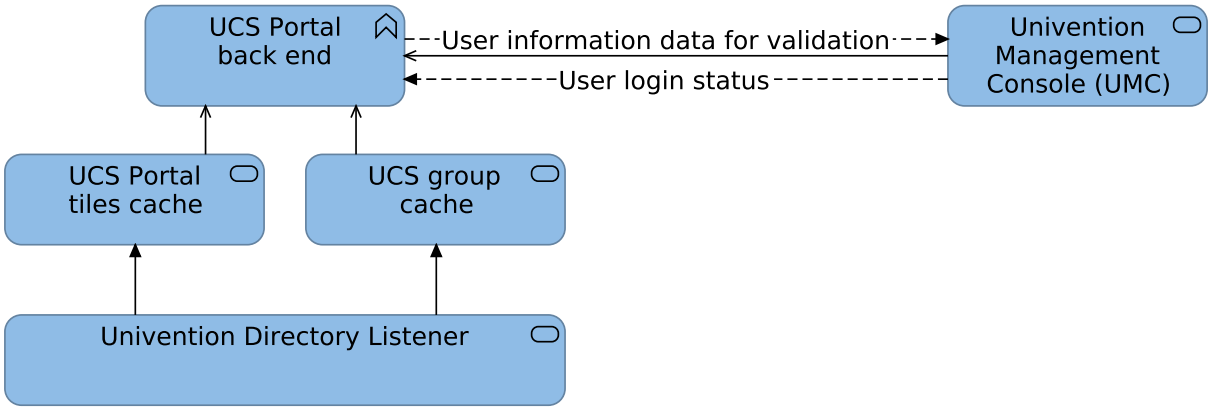

Fig. 5.19: Architecture of the UCS Portal backend

UCS Portal tiles cache
: Provides structured data about the tiles configured for every portal in the domain. Every tile has assignments to user groups.

UCS group cache
: Provides structured data to resolve a user and its group memberships including nested groups.

UMC server
: Validates user authentication for a given user.

Univention Directory Listener
: In the context of the UCS Portal, the Univention Directory Listener triggers the update of the *UCS portal tile cache* (page 57) and the *UCS group cache* (page 57).

[102] https://en.wikipedia.org/wiki/Single-page_application

User identification

Fig. 5.20 shows the basic model of the user identification. The description follows below.

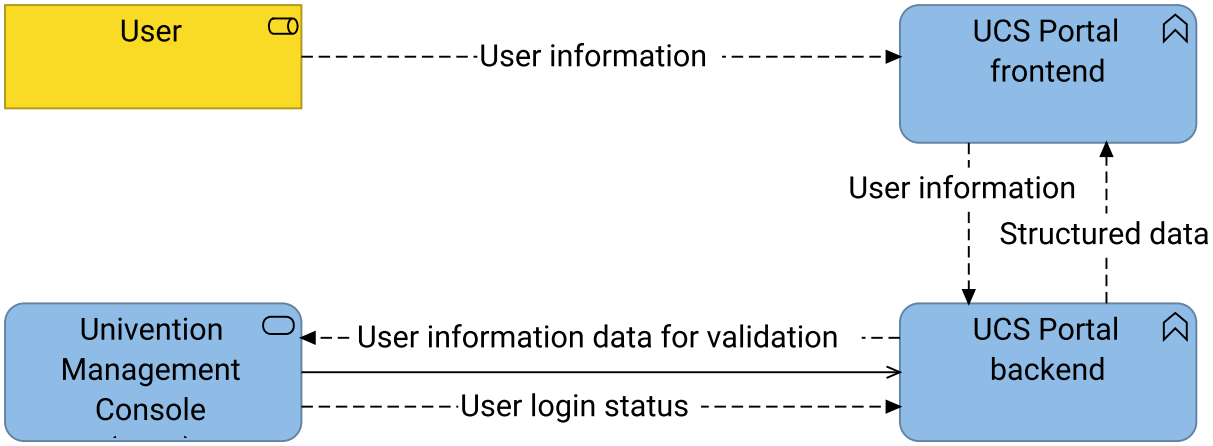

Fig. 5.20: User identification in the UCS Portal

1. The user is either an anonymous user or has user information from a login.
2. The portal frontend sends an HTTP request with user information to the portal backend.
3. The portal backend delegates the user validation to the UMC server.
4. The UMC server returns the login status.
5. Based on the login status the portal backend generates the structured data for the portal frontend.

Structured data for portal content

The structured data in `portal.json` for the portal frontend has information for example about folders in the menu, categories in the portal main area, portal design, the entries for the menu and the portal tiles. For example, the anonymous portal data[103] from the UCS demo system[104].

The content depends on the user login status:

Anonymous users
: Anonymous users see portal content that's publicly available.

Signed in users
: Signed in users see public content and content depending on their group memberships. One user may also see different tiles than another user.

The portal backend uses the caches in the following sections to generate the structured data.

[103] https://demo.univention.de/univention/portal/portal.json
[104] https://demo.univention.de/univention/portal/

UCS portal tile cache

The portal tile cache has information about the content of every tile like name, description, logo, and category. Furthermore, it knows the group assignment for every tile.

When administrators create or modify a portal in the UMC module *LDAP directory*, the Univention Directory Listener reacts on this change and triggers the listener module responsible for the portal tile cache. The module then uses UDM and recreates the portal tile cache.

The portal tile cache uses structured data, as well. The listener module saves it in a JSON file in the file system of the UCS system.

UCS group cache

The *User identification* (page 56) returns information about the user without data about the users' group memberships and nested groups. The group cache steps in and provides a mapping for users to their groups.

Running the user's group resolution on the fly is an expensive operation especially for large environments.

To mitigate the expensive operation, the Univention Directory Listener triggers the respective listener module in the *post-run* when no more changes happen to user groups for 15 seconds. The group cache retrieves the necessary information from the key-value store of the UCS group membership cache.

5.5.3 Dependencies for UCS portal

The UCS portal depends on the Univention Directory Listener, *Univention Directory Manager (UDM)* (page 35), the *UCS group membership cache*, and the *UCS Portal tile cache*. Table 5.1 lists the depending services and their packages:

Table 5.1: Dependencies for UCS portal

Service	Package name
UCS configuration manager	`univention-config`
Univention Directory Listener	`univention-directory-listener`
UCS command-line based administration tools	`univention-directory-manager-tools`
UCS group membership cache	`univention-group-membership-cache`
UCS management console server (page 45)	`univention-management-console-server`

5.6 App Center service

This section describes the architecture of the App Center service focused solely on UCS.

For a general overview of the App Center, its ecosystem, the participating actors, and the infrastructure, see *Univention app ecosystem* (page 13). For the overview of the App Center as product component, see *App Center* (page 26).

You can find the source code at UCS source: management/univention-appcenter/[105].

Fig. 5.21 shows the architecture overview of the App Center on a UCS system.

For the abstract context description about the first two rows in Fig. 5.21, refer to *App Center* (page 26).

In the notation in Fig. 5.21 the application service *App Center Service* summarizes all behavior that relates to the App Center and apps on a UCS system. The center piece for all behavior in the *App Center* are the *App Center actions* that use the *Python App Center library* to do "stuff" with *App*s.

The App Center is a complex component in UCS. As you continue reading, the concepts unfold and reveal their internal parts.

[105] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-appcenter/

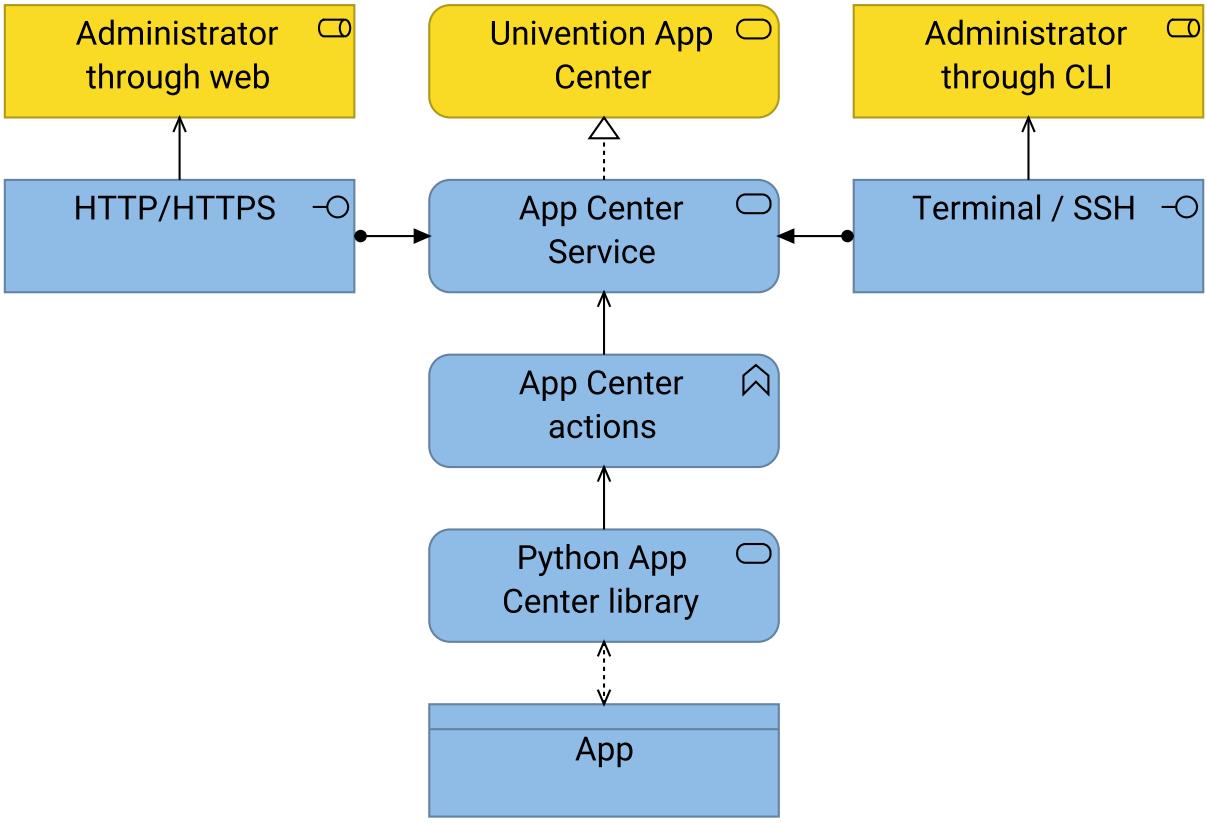

Fig. 5.21: Architecture overview of the App Center on a UCS system

See also:

`Python App Center library`[Page 58, 106]
for detailed information about the Python library for the App Center in *Univention Corporate Server Python API 5.2 documentation* [4].

5.6.1 App Center interfaces

First, this section continues with the App Center connections to the external world. Fig. 5.22 shows the interfaces to the user for the App Center and how the App Center relates to other parts of UCS.

The left side shows the path for the web interface of the App Center. Like many other components, the App Center uses *UMC - Univention Management Console* (page 45) for the web interface. The App Center provides the following *UMC modules* (page 52):

App Center in UMC
The UMC module *App Center in UMC* provides the web interface to the user. Administrators can list, show, install, update, and remove apps. It presents all available apps to the administrator in a nice overview. It's also responsible for the app presentation with information like description, screenshots and videos, contact and app provider information.

Apps in UMC
The UMC module *Apps in UMC* provides a proper view in the UCS management system for every installed app. It shows the app description, detailed information and offers actions like update or remove on the app to the administrator.

The right side shows the path to the command line interface of the App Center.

[106] https://docs.software-univention.de/ucs-python-api/univention.appcenter.html#module-univention.appcenter

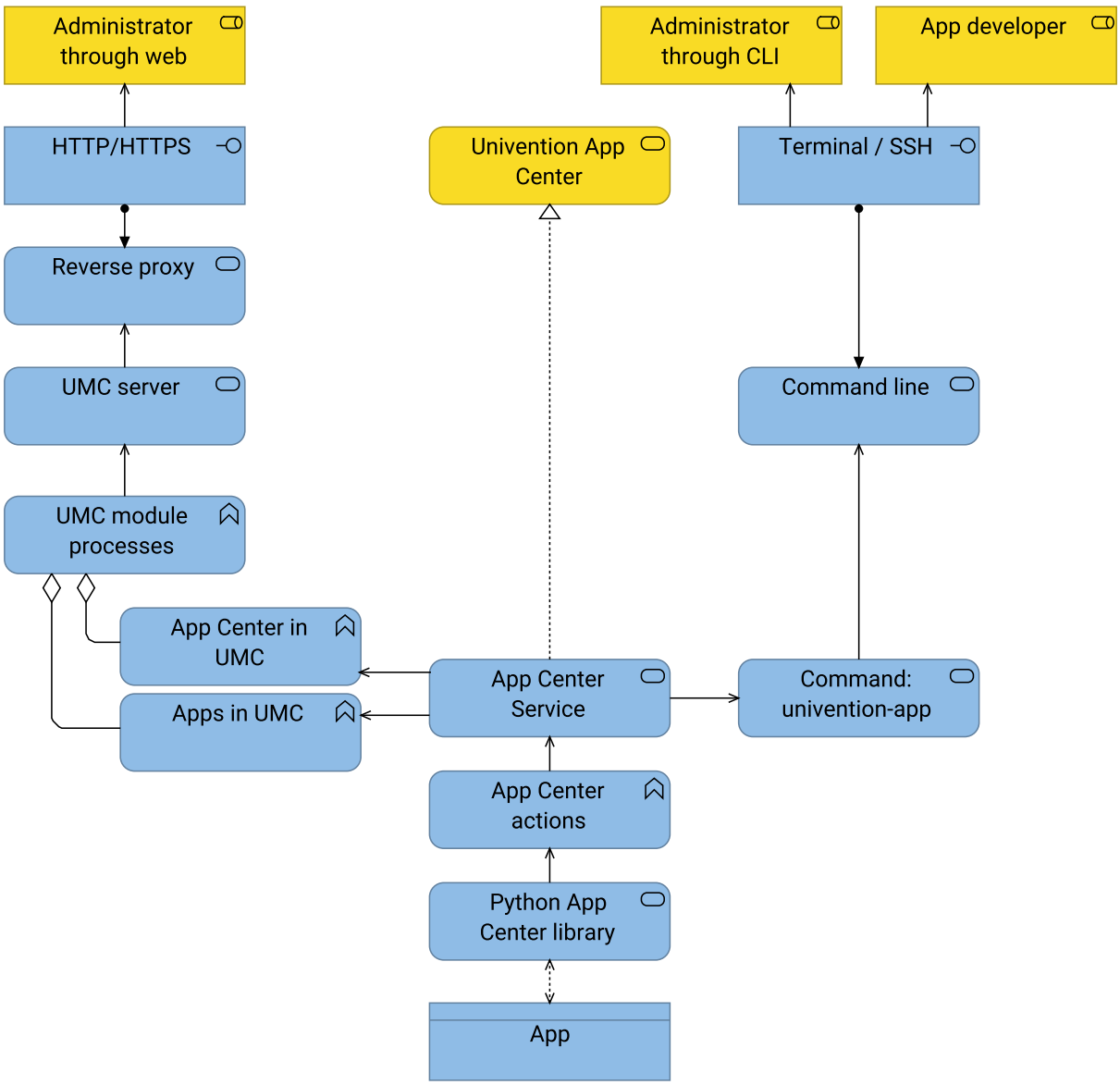

Fig. 5.22: App Center interfaces to the user
The figure extends Fig. 4.4.

The items *App Center*, *Python App Center library*, and *App* in the middle are the core of the App Center. The following sections describe them in more detail.

See also:

App presentation[Page 60, 107]
for information about how app providers can define the data for app presentation in *Univention App Center for App Providers* [2].

UCS source: management/univention-appcenter/umc/[108]
for the source code of the UMC module *App Center in UMC*.

5.6.2 App Center actions

App Center actions are the center piece for all behavior in the *App Center*. Figure Fig. 5.23 shows the most important actions.

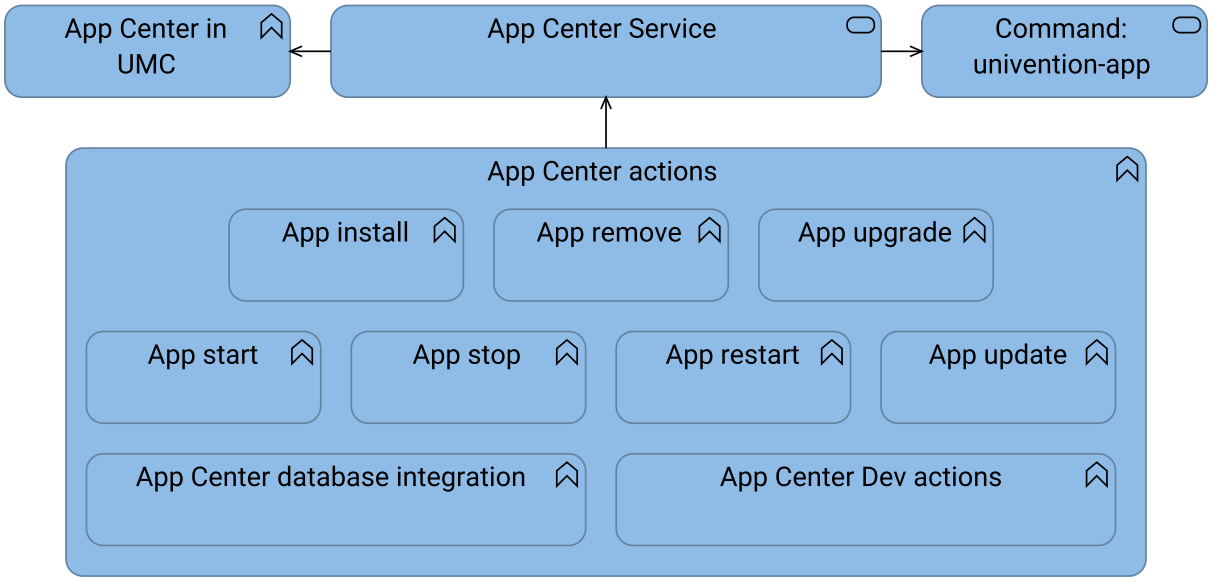

Fig. 5.23: App Center actions

To get a list of all actions, take a look into the checked out source code in the directory UCS source: management/univention-appcenter/python/[109] of the UCS repository and run the following command:

Listing 5.1: Get a list of available *App Center actions* from the sources

```
$ find | grep actions
```

The core actions that administrators encounter when working with UCS are actions to manage the app lifecycle and control their operational status. These are actions such as:

- *App install*
- *App remove*
- *App upgrade*
- *App start*
- *App stop*

[107] https://docs.software-univention.de/app-center/5.2/en/presentation.html#app-presentation
[108] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-appcenter/umc/
[109] https://github.com/univention/univention-corporate-server/tree/5.2-1/management/univention-appcenter/python/

- *App restart*
- *App update*

And the App Center has other actions, for example, they run during installation like the *App Center database integration* or handle a listener module dedicated to the app. Furthermore, app developers use the *App Center Dev actions* during app development.

The *App Center actions*' purpose is manifold:

- They abstract lifecycle actions for apps for the various distribution flavors like *Package based app* and *Docker based app*.
- They hide the complexity of lifecycle management and standardize the needed procedures.

See also:

App artifacts **(page 16)**
for information about the various distribution flavors *Package based app* and *Docker based app*.

5.6.3 App Center apps cache

This section covers the *Apps Cache*, a part of the *App Center* that exists on every UCS system. Fig. 5.24 shows the *Apps Cache* relationship to the *App Center actions*.

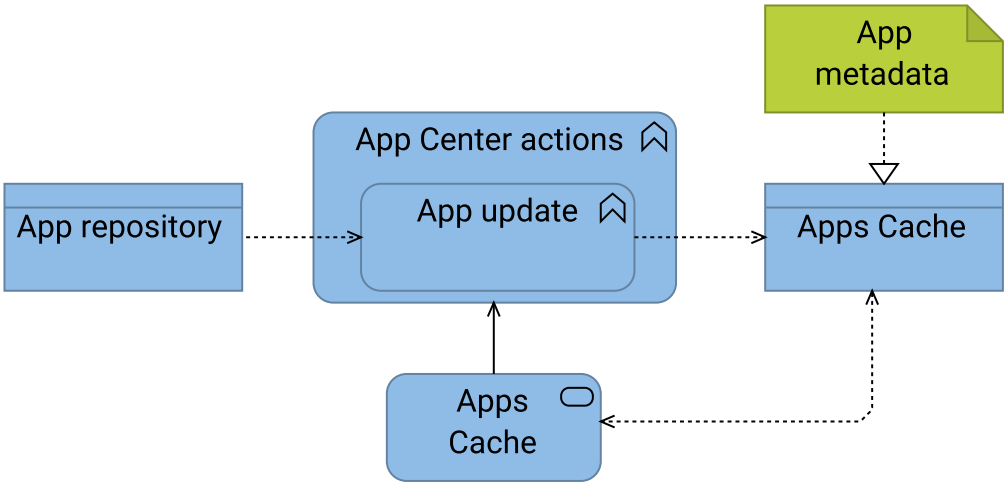

Fig. 5.24: App Center *Apps cache*

The App Center has the action *App update* that downloads information from the *App repository* and writes the *Apps Cache* on a UCS system. It has the following purposes:

- Download all the *App metadata* from the *App repository*. For information about the infrastructure, refer to *Univention app infrastructure* (page 19).
- Consolidate the app metadata in a JSON file.

The app metadata locates in the directory `/var/cache/univention-appcenter/` on a UCS system. The data from the *Apps Cache* is then available to all other *App Center actions* that need any kind of information related to apps. For example, the UMC module *App Center in UMC* reads the data from the *Apps Cache* to display it in the web interface.

5.6.4 App integration

The App Center offers various integration points for apps to simplify the app setup and the integration into the UCS environment.

Web server integration

For apps that offer their own web interface, the App Center provides a web server integration as shown in Fig. 5.25.

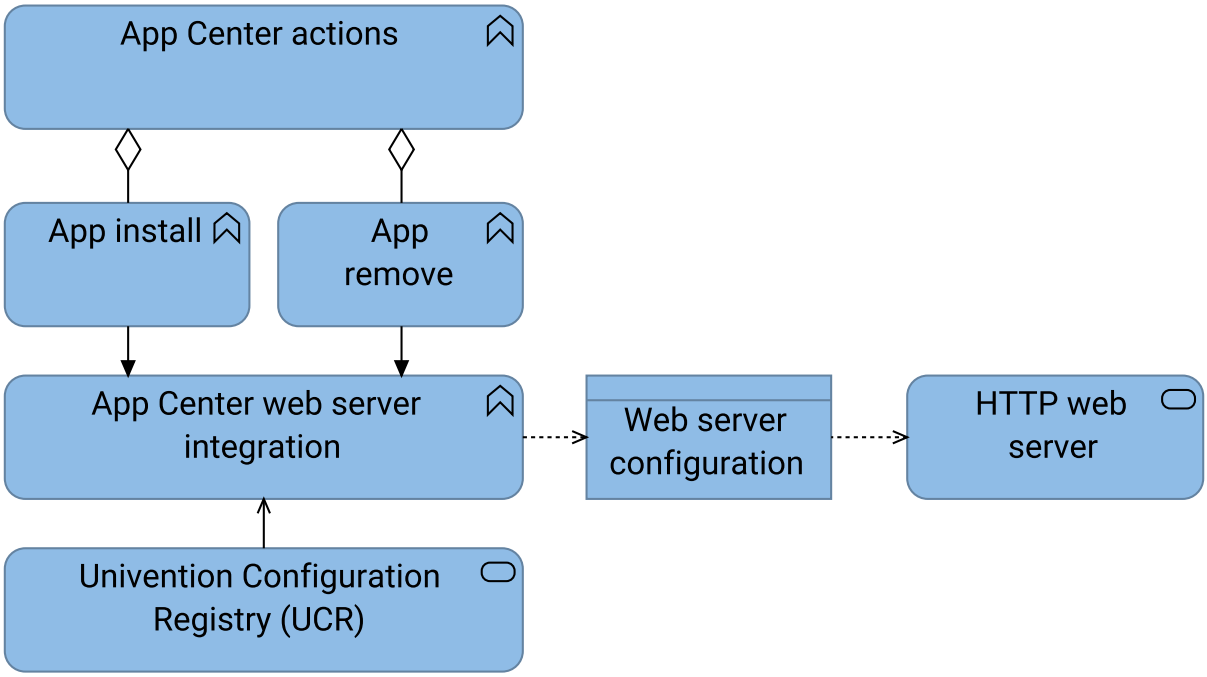

Fig. 5.25: App Center web server integration

The *App Center web server integration* appends the *Web server configuration* and adds the path to the app's web interface. The procedure uses *Univention Configuration Registry (UCR)* (page 29). The *App Center web server integration* removes the appended configuration upon app removal.

Apps can also provide a complex web server integration by adding their own configuration to the *HTTP web server*. App developers handle the configuration lifecycle on their own in the app.

See also:

Web interface[Page 62, 110]
for more information about how to expose the app's web interface in *Univention App Center for App Providers* [2].

[110] https://docs.software-univention.de/app-center/5.2/en/get_started.html#create-app-with-docker-web-interface

Portal integration

Apps that offer a web interface and use the *web server integration* (page 62) automatically use the portal integration to add a tile to the *UCS portal* (page 53), as shown in Fig. 5.26.

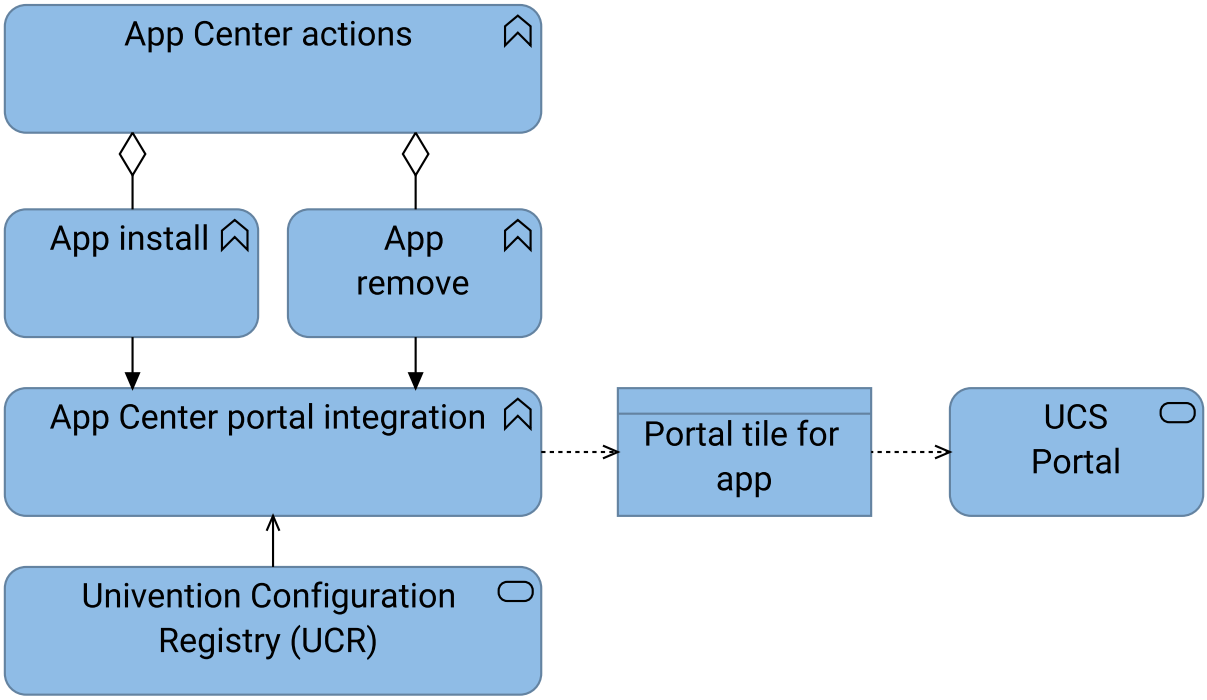

Fig. 5.26: App Center portal integration

Upon installation, the App Center adds a portal tile with the icon, name, and link to the app's web interface. Upon removal, the App Center removes the portal tile.

Database integration

For apps that need a RDBMS (relational database management system) like *MariaDB* or *PostgreSQL* the App Center installs the respective packages from the UCS package repository during app installation, as shown in Fig. 5.27.

Apps using the databases provided with UCS benefit from the following advantages:

- Univention maintains the packages for the databases and provides security updates.
- The databases integrate with the UCS system. For example, the App Center creates a database for the app together with a database user and password.
- The App Center provides the connection settings to the app. The app can start with creating the database schema.

Nevertheless, the *App Center database integration* has the following limitations:

- UCS installs the RDBMS on the same host as the app and creates one database.
- The App Center doesn't use the RDBMS on a remote host or in a Docker environment.
- Apps have limited possibilities to configure the RDBMS.
- If the UCS system with the app has multiple apps installed that use a database, they share the RDBMS and its configuration.

Docker based apps, that need more flexibility, can provide their app as *Multi container app* and add the RDBMS as Docker container with the required configuration. The app provider is responsible for maintenance and security updates for the RDBMS as Docker container.

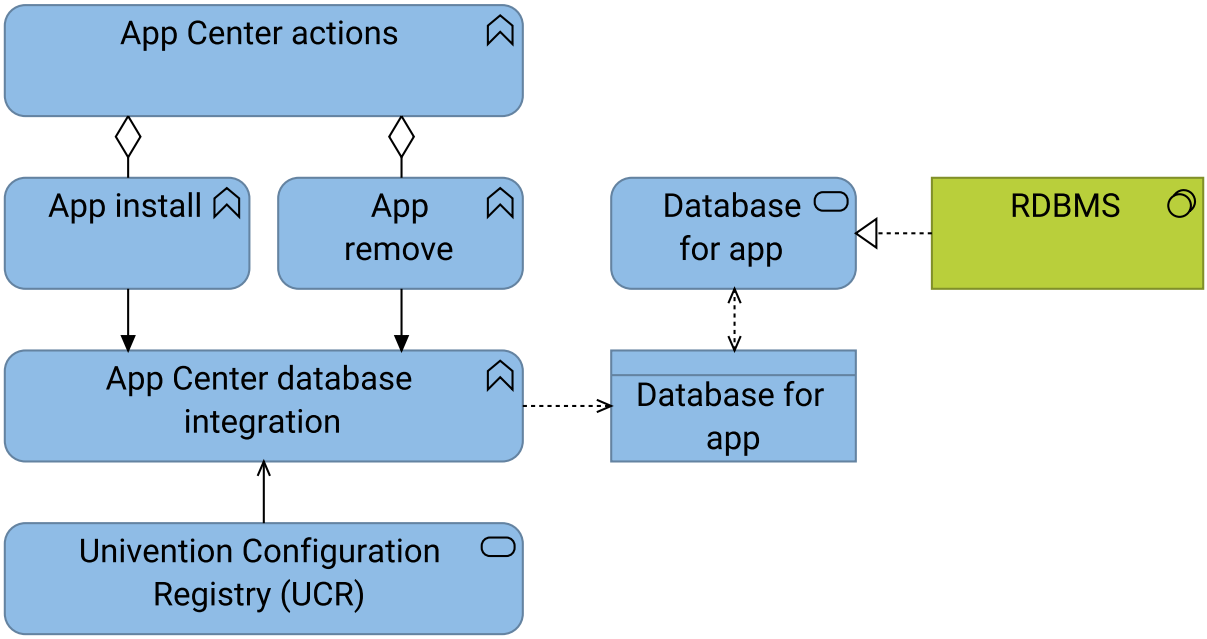

Fig. 5.27: App Center database integration

Fig. 5.28 shows the maintenance relations for the RDBMS. Although the model might imply that either role maintains the database software, it's not the case. Instead, they cover the distribution of the RDBMS.

See also:

Database[Page 64, 111]
for more information about how to configure the app integration in an app in *Univention App Center for App Providers* [2].

Identity management integration

Many app providers integrate their app with the identity management in UCS. The identity management integration consists of the following aspects:

User provisioning
Provisioning means that the app gains knowledge about user account information and can, for example, create a user account in its own data structure and map it with the user account in the UCS identity management. Each app handles the mapping individually.

The preferred provisioning method is *push*. Upon changes in the LDAP directory, the Univention Directory Listener creates information for the app to handle.

In contrast, the pull method through direct LDAP connection requires periodic pulls. The app must then identify and handle changes on its own.

User authentication
Authentication means that the app uses one of the different authentication protocols in UCS like for example Kerberos, LDAP, SAML, or OpenID Connect.

To use the identity management integration in the app, the app developer can activate it in the app metadata.

Fig. 5.29 shows the App Center generating the listener module upon app installation for user provisioning using the *push* method. The key items have a less strong filled background color.

[111] https://docs.software-univention.de/app-center/5.2/en/get_started.html#create-app-with-docker-database

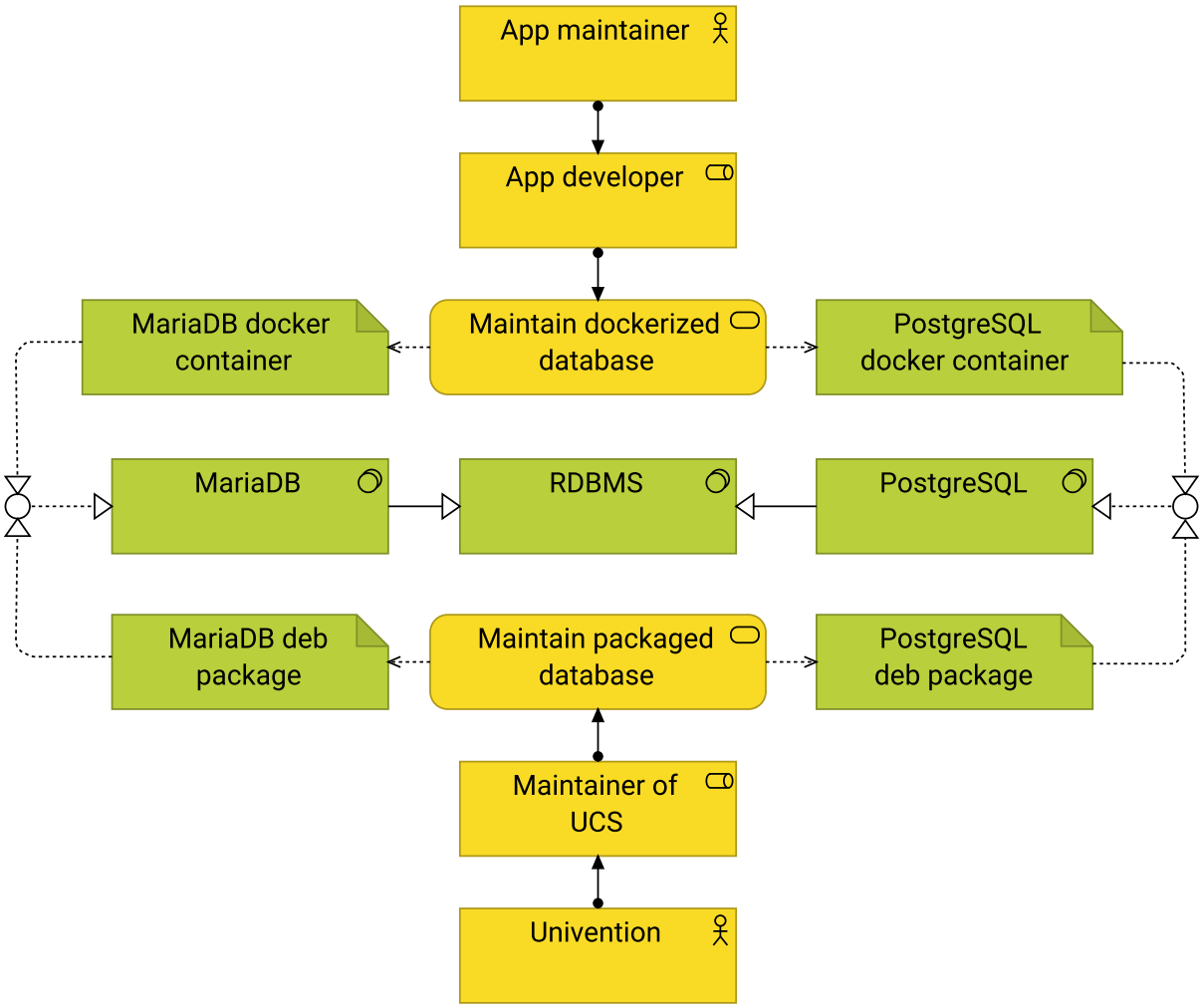

Fig. 5.28: Maintenance of databases for Apps
Consider the *OR* junction as *XOR* for the realization relation.

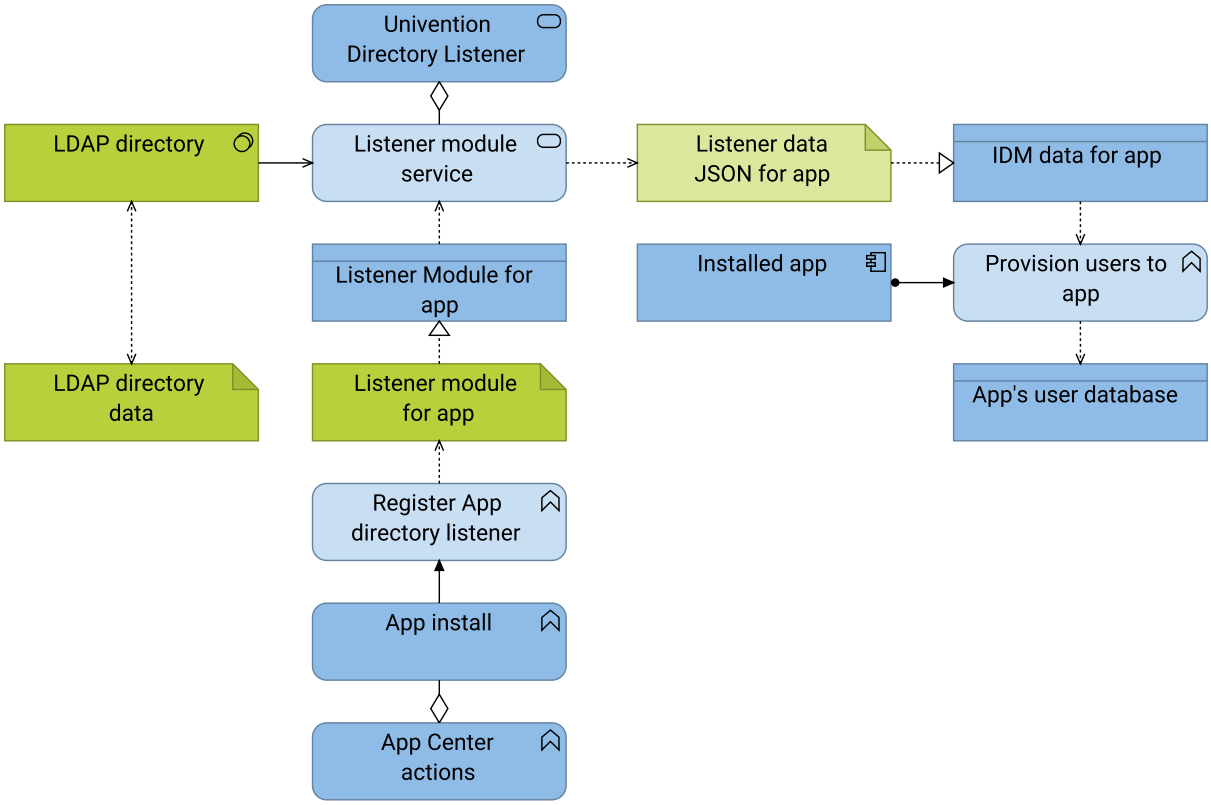

Fig. 5.29: *Register App directory listener* for user provisioning with *push* method

Register App directory listener

Upon app installation, the App Center generates a listener module for the app and starts a service for the Univention Directory Listener.

- *Register App directory listener* creates the artifact *Listener module for app*
- *Listener modules for app* realizes the service *Listener Module for app*.
- The service *Listener module service* runs the listener module for the app and belongs to the service *Univention Directory Listener*.

For example, on a UCS system with five installed apps that use the identity management integration, the App Center generates five listener modules and services.

Listener module service

The listener listens for changes in the LDAP directory service. The listener module consists of two parts:

1. Part one creates change information relevant to the app based on changes in the LDAP directory. Such changes are, for example, *user account created*, *user account modified*, or *user account removed*.
2. Part two takes the information about the changes and creates a JSON file, the artifact *Listener data JSON for app*, with information about the user account and about the kind of change. It periodically looks for the file from part one to generate the JSON file.

Listener data JSON for app

Is the artifact created by the *Listener module service*. From an architecture perspective the artifact realizes the data object *IDM data for app*.

Provision users to app

Provision users to app reads the *IDM data for app*, handles them accordingly, and writes the relevant information to the *App's user database*. For example, the app creates a user account in its database to internally refer to the user. The *Installed app*, that has *Provision users to app* assigned, is responsible to handle the JSON files written by the *Listener module service*.

See also:

For app software developers, refer to the following content in *Univention App Center for App Providers* [2]:

- Connection with Identity Management[112] for information about how to connect an app with the identity management.
- Provisioning[113]
  - Automatically via LDAP connection (Pull)[114]
  - Automatically via IDM notifications (Push)[115]
- Authentication[116]
  - LDAP[117]
  - Kerberos[118]

Extended attributes

The App Center uses *extended attributes* for every app upon installation when the app requires the administrator to enable user accounts for the app.

Extended attributes require an LDAP schema extension. The App Center creates that schema extension automatically and registers it in the LDAP directory service. And it also generates the extended attribute accordingly to use the extra fields added with the schema extension and map them to respective fields in UDM.

For more information about extended attributes from the architecture perspective, refer to *UDM data* (page 38).

Beyond the default schema extension, the App Center also registers schema extensions provisioned with the app. Apps that use the LDAP directory as their user database make use of schema extensions and extended attributes to enable a respective user administration for the system administrator. An LDAP schema extension ensures that the third party software can use the required LDAP attributes.

See also:

Administrators refer to the following content in *Univention Corporate Server - Manual for users and administrators* [1]:

Expansion of UMC modules with extended attributes[Page 67, 119]
: How to use extended attributes

See also:

App software developers, refer to the following content in *Univention App Center for App Providers* [2]:

User rights management[120]
: for more information about user rights management for apps.

[112] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#connection-idm
[113] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#provisioning
[114] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#provisioning-pull
[115] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#provisioning-push
[116] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#authentication
[117] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#authentication-ldap
[118] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#authentication-kerberos
[119] https://docs.software-univention.de/manual/5.2/en/central-management-umc/extended-attributes.html#central-extended-attrs
[120] https://docs.software-univention.de/app-center/5.2/en/identity_management.html#user-rights-management

5.6.5 Dependencies for the App Center

As complex component in UCS, the service *App Center* has the following dependencies:

- *Univention Configuration Registry (UCR)* (page 29)
- *Univention Directory Manager (UDM)* (page 35)
- *UMC - Univention Management Console* (page 45)
- Univention Directory Listener
- *UCS portal service* (page 53)
- Univention updater
- *Docker.io* with the *Docker Engine* and *Docker compose*

Fig. 5.30 shows the direct dependencies in the model.

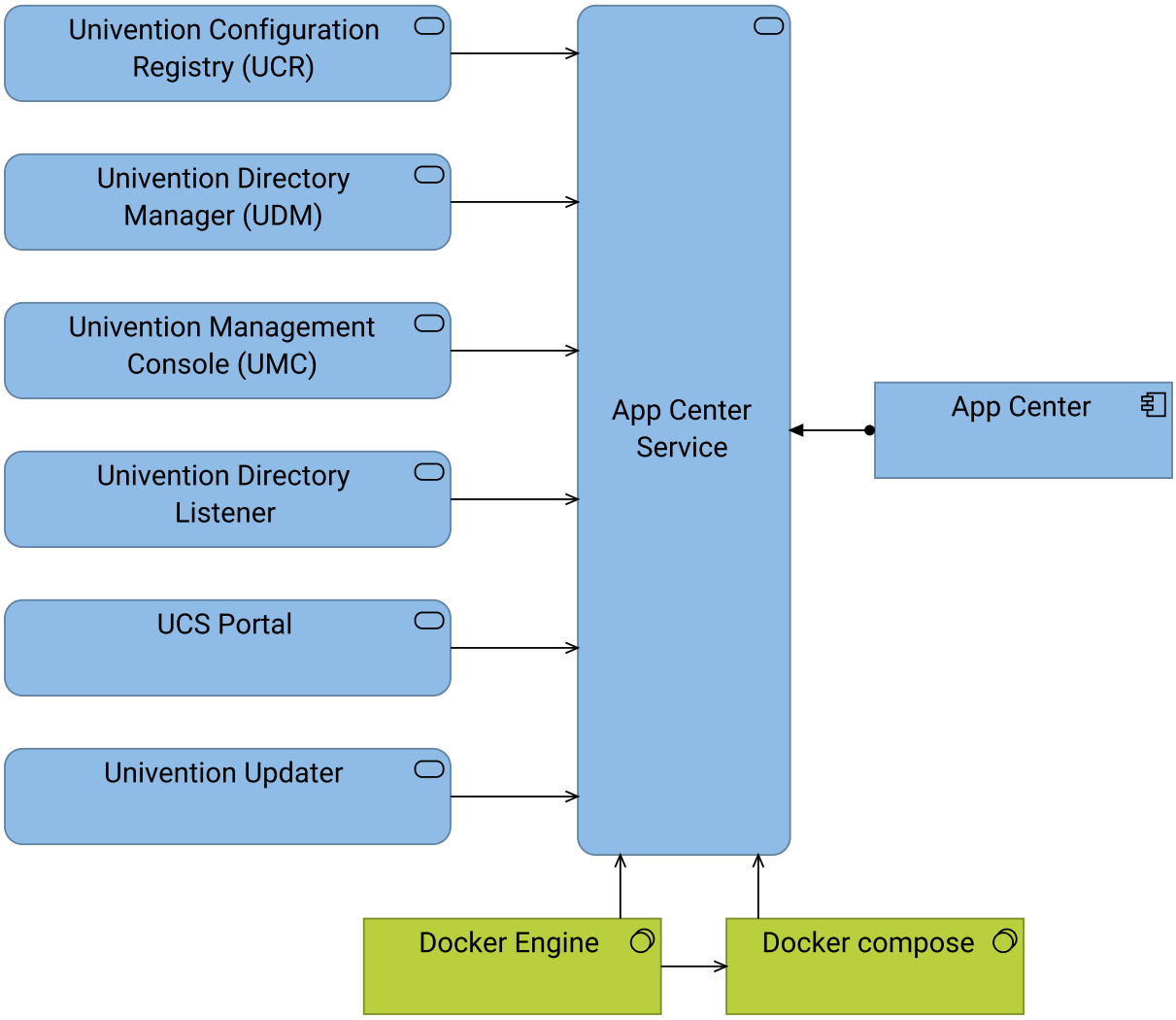

Fig. 5.30: Dependencies of the App Center

The dependency to the *Univention updater* comes from the App Center's handling of the *Package based Apps* and for example the *App Center database integration*.

CHAPTER

SIX

APPENDIX

The following sections provide additional information to this document. They deliver additional information for the content.

6.1 Third party software

This section lists the third party software referenced in this document.

Apache HTTP server
: The Apache HTTP Server Project is an effort to develop and maintain an open source HTTP server for modern operating systems. The goal of the project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

  For the website, see Apache HTTP server project[121].

Apache module mod_proxy
: **mod_proxy** and related modules implement a proxy and gateway for *Apache HTTP server*, supporting a number of popular protocols as well as several different load balancing algorithms.

  For the website, see Apache module mod_proxy[122].

Bootstrap
: Powerful, extensible, and feature-packed front end toolkit.

  For the website, see Bootstrap - The most popular HTML; CSS, and JS library[123].

Dojo Toolkit
: Modular JavaScript framework

  For the website, see Dojo Toolkit[124].

Tornado
: **Tornado Web Server** is a Python web framework and asynchronous networking library.

  For the website, see Tornado Web Server[125].

TypeScript
: **TypeScript** is the programming language of choice for the frontend, because it helps to achieve a unified codebase through typing and linting features. Furthermore, *TypeScript* avoids common JavaScript mistakes and helps software developers to write cleaner code.

  For the website, see TypeScript: JavaScript with Syntax For Types[126].

Vue.js
: **Vue.js** is a versatile JavaScript framework for building web user interfaces.

[121] https://httpd.apache.org/
[122] https://httpd.apache.org/docs/current/en/mod/mod_proxy.html
[123] https://getbootstrap.com/
[124] https://dojotoolkit.org/
[125] https://www.tornadoweb.org/en/stable/
[126] https://www.typescriptlang.org/

For the website, see Vue.js - The Progressive JavaScript Framework[127].

6.2 Architecture notation

A notation helps a lot to understand complex architectures and to explain the software architecture of UCS. A standardized notation is key to communicate an architecture.

This section describes the notation and elements used in this document. It's intended to help you as reader to understand the notations. This section tries to duplicate as little content as possible and instead provide deep links to the corresponding resources on the internet.

6.2.1 C4 model

The document uses the C4 model in the *Concepts* (page 7) section.

> The C4 model is a lean graphical notation technique for modeling the architecture of software systems. It's based on a structural decomposition of a system into containers and components and relies on existing modeling techniques such as the Unified Modeling Language (UML) or Entity Relation Diagrams (ERD) for the more detailed decomposition of the architectural building blocks.
>
> ---Wikipedia contributors, "C4 model"[128], Wikipedia, The Free Encyclopedia, (accessed January 24, 2023).

The C4 model is notation independent and provides:

1. A set of hierarchical abstractions for software systems, containers, components, and code.
2. A set of hierarchical diagrams for system context, containers, components, and code.

The C4 model is a good to learn, developer friendly approach to software architecture diagramming. But, it comes to limits when it comes to model the architecture. Diagramming tools draw just boxes and lines and can't answer questions like "What dependencies does component X have?"

See also:

The C4 model for visualizing software architecture[Page 70, 129]
for a description of the C4 model from Brown [6]

6.2.2 ArchiMate

ArchiMate is a notation and modeling standard for enterprise architecture maintained by *The OpenGroup*. This document uses the ArchiMate® specification 3.2[130].

Why ArchiMate?

> ArchiMate isn't about standard boxes and lines, it's all about a common language that provides the foundations for a good architecture description.
>
> Using the language without its notation is already of great value as it allows people to understand each other.
>
> ---Jean-Baptiste Sarrodie, "Why ArchiMate?"[131], 28. September 2018

Regarding the simplified language analogy:

[127] https://vuejs.org
[128] https://en.wikipedia.org/w/index.php?title=C4_model&oldid=1115745383
[129] https://c4model.com/
[130] https://pubs.opengroup.org/architecture/archimate32-doc/
[131] https://www.archimatetool.com/blog/2018/09/25/why-archimate/

- ArchiMate contains a vocabulary that covers most domains of Enterprise Architecture. This document focuses on technology, application and business.
- ArchiMate uses a grammar similar to natural language with *subject*, *verb*, and *object* to describe what *people* or *things* do, and adds an external, service oriented, view of those activities.
- The ArchiMate default notation is similar to spelling as it provides a way to "save ideas on paper".

See also:

"ArchiMate", Wikipedia, The Free Encyclopedia[Page 71, 132]
for an overview of the ArchiMate frameworks, language and viewpoints

ArchiMate reader's guide

This document uses the ArchiMate concepts *element*, *relationship*, and *relationship connector* mentioned in the later sections. The following sections provide specific links to the corresponding resources in the ArchiMate specification with summarized definitions. They help to pick out the parts needed to understand the notation.

To properly read ArchiMate, it's recommended to read parts of the ArchiMate specification about the following:

1. The ArchiMate Core Framework[133] section, that refers to the layers[134] *Business*, *Application*, and *Technology*. Imagine the layers as rows in a table.
2. The ArchiMate Core Framework section explains the three *Aspects*. Think of an aspect as columns in a table:
   - The *Active Structure Aspect* represents structural elements, the actors. Think of it as the subject in a natural language sentence.
   - The *Behavior Aspect* represents behavior performed by actors. Think of it as the verb in a natural language sentence.
   - The *Passive Structure Aspect* represents objects, the targets of the actors' behavior. Think of it as the object in a natural language sentence.
3. You find the ArchiMate concepts used in the document in the sections below, organized by layer. To read a short definition for each element, follow the links to the corresponding summaries in the specification.

See also:

ArchiMate® specification 3.2[135]
for the complete *ArchiMate® 3.2 Specification* [7]

Free ArchiMate 3.2 Overview PDFs in multiple languages[136]
for overview PDF files about ArchiMate 3.2 in different languages, such as English and German.

Mastering ArchiMate Edition 3.1[137]
for a free PDF excerpt of the book from Wierda [8]

[132] https://en.wikipedia.org/wiki/ArchiMate
[133] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Language-Structure.html#sec-The-ArchiMate-Core-Framework
[134] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Language-Structure.html#sec-Layering-of-the-ArchiMate-Language
[135] https://pubs.opengroup.org/architecture/archimate32-doc/
[136] https://ea.rna.nl/archimate/free-archimate-overview-pdf/
[137] https://ea.rna.nl/mastering-archimate-edition-3-1/

Business layer

Business Layer elements model the operational organization of an enterprise in a technology-independent manner.

For the business layer the document uses the ArchiMate concepts as shown in Fig. 6.1.

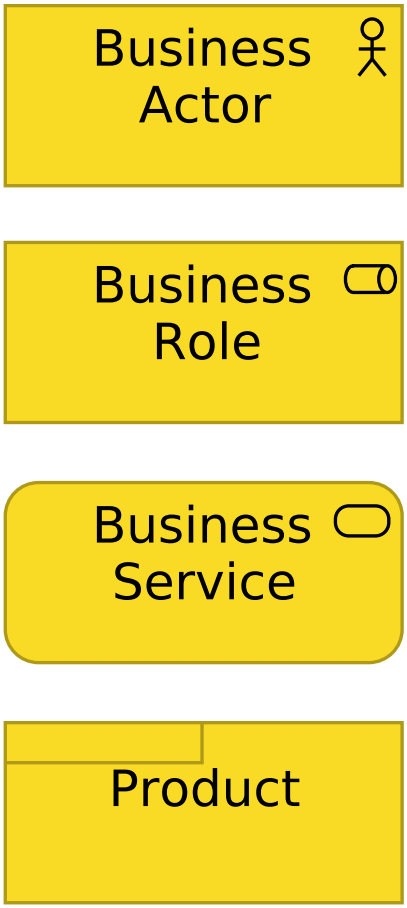

Fig. 6.1: ArchiMate business layer concepts used in this document

Meanings in one sentence

Summary of Business Layer Elements[Page 72, 138]
for a table with a summary of business layer elements

See also:

ArchiMate business layer[139]
for the specification of the business layer

[138] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Business-Layer.html#sec-Summary-of-Business-Layer-Elements
[139] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Business-Layer.html

Application layer

Application Layer elements typically model the application architecture that describes the structure, behavior, and interaction of the applications of the enterprise.

For the application layer the document uses the ArchiMate concepts as shown in Fig. 6.2.

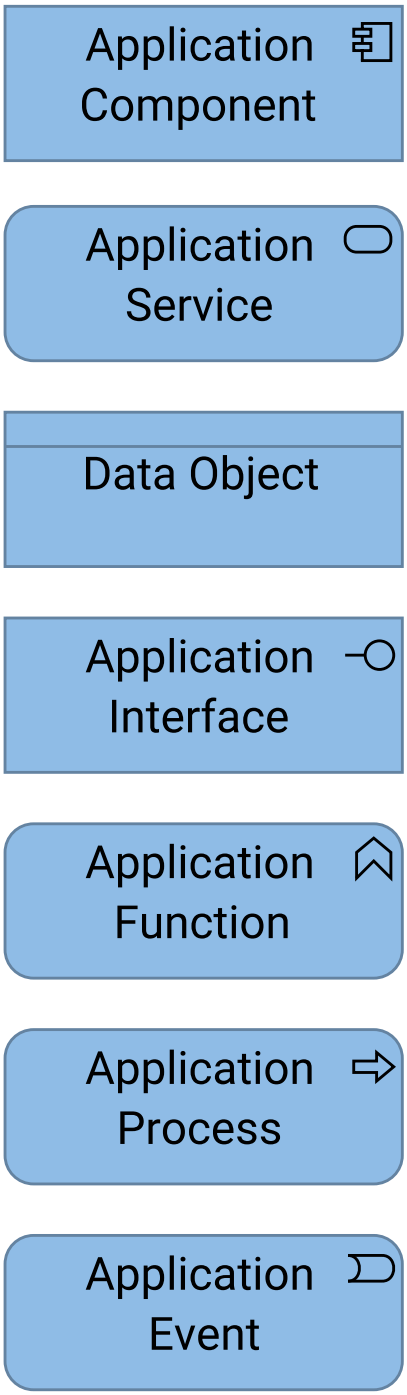

Fig. 6.2: ArchiMate application layer concepts used in this document

Meanings in one sentence

Summary of Application Layer Elements[Page 74, 140]

for a table with a summary of application layer elements

See also:

ArchiMate application layer Page 74, 141
for the specification of the application layer

Technology layer

The *Technology Layer* elements typically model the technology architecture of the enterprise, describing the structure and behavior of the technology infrastructure of the enterprise.

For the technology layer the document uses the ArchiMate concepts as shown in Fig. 6.3.

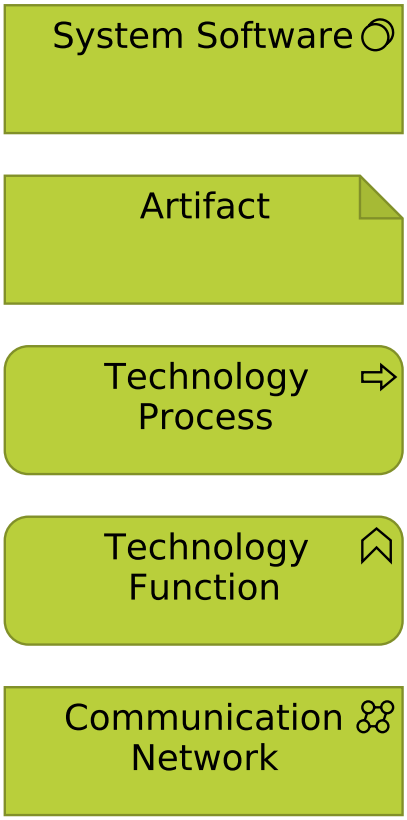

Fig. 6.3: ArchiMate technology layer concepts used in this document

Meanings in one sentence

Summary of Technology Layer Elements[142]
for a table with a summary of technology layer elements

See also:

ArchiMate technology layer[143]
for the specification of the technology layer

[140] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Application-Layer.html#sec-Summary-of-Application-Layer-Elements
[141] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Application-Layer.html
[142] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Technology-Layer.html#sec-Summary-of-Technology-Layer-Elements
[143] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Technology-Layer.html

Motivation elements

Motivation elements model the motivations, or reasons, that guide the design or change of an enterprise architecture.

The motivation elements belong to the ArchiMate full framework[144]. From the motivation elements the document uses the ArchiMate concepts as shown in Fig. 6.4.

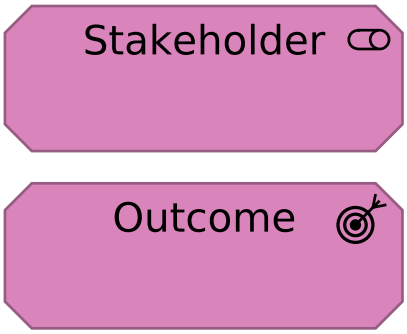

Fig. 6.4: ArchiMate motivation elements used in this document

Meanings in one sentence

Summary of Motivation Elements[Page 75, 145]
for a table with a summary of motivation elements

See also:

ArchiMate motivation elements[146]
for the specification of the motivation elements

Strategy elements

The strategy elements are typically used to model the strategic direction and choices of an enterprise, as far as it concerns the impact on its architecture. They express how the enterprise wants to create value for its stakeholders, the capabilities it needs, the resources needed to support these capabilities, as well as how it plans to configure and use these capabilities and resources to achieve its aims.

The strategy elements belong to the ArchiMate full framework[147]. From the strategy elements the document uses the ArchiMate concepts as shown in Fig. 6.5.

Fig. 6.5: ArchiMate strategy elements used in this document

Meanings in one sentence

Summary of Strategy Elements[148]
for a table with a summary of strategy elements

[144] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Language-Structure.html#sec-The-ArchiMate-Full-Framework
[145] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Motivation-Elements.html#sec-Summary-of-Motivation-Elements
[146] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Motivation-Elements.html
[147] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Language-Structure.html#sec-The-ArchiMate-Full-Framework
[148] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Strategy-Layer.html#sec-Summary-of-Strategy-Elements

See also:

ArchiMate strategy elements[Page 76, 149]
for the specification of the strategy elements

Relationships

The document uses almost all relations from the ArchiMate Core framework.

As reader you may find views that don't repeat concepts and relationships in between two concepts in focus. Such views are abstractions and they use the derivation of relationships. ArchiMate provides derivation rules to create abstract views.

Meanings in one sentence

Summary of Relationships[150]
for a table with a summary of relationships

See also:

Derivation of Relationships[151]
for an introduction to derivation of relationships

ArchiMate Relationships[152]
for the specification of relationships

ArchiMate Specification of Derivation Rules[153]
for the specification of derivation rules for valid and potential relationships

[149] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Strategy-Layer.html
[150] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Relationships-and-Relationship-Connectors.html#sec-Summary-of-Relationships
[151] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Relationships-and-Relationship-Connectors.html#sec-Derivation-of-Relationships
[152] https://pubs.opengroup.org/architecture/archimate32-doc/ch-Relationships-and-Relationship-Connectors.html
[153] https://pubs.opengroup.org/architecture/archimate32-doc/ch-relationships-Normative.html

Summary of Relationships

Table 3 gives an overview of the ArchiMate relationships with their definitions.

Table 3: Relationships

Structural Relationships		**Notation**	**Role Names**
Composition	Represents that an element consists of one or more other concepts.		← composed of → composed in
Aggregation	Represents that an element combines one or more other concepts.		← aggregates → aggregated in
Assignment	Represents the allocation of responsibility, performance of behavior, storage, or execution.		← assigned to → has assigned
Realization	Represents that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.		← realizes → realized by
Dependency Relationships		**Notation**	**Role Names**
Serving	Represents that an element provides its functionality to another element.		← serves → served by
Access	Represents the ability of behavior and active structure elements to observe or act upon passive structure elements.		← accesses → accessed by
Influence	Represents that an element affects the implementation or achievement of some motivation element.	+/-	← influences → influenced by
Association	Represents an unspecified relationship, or one that is not represented by another ArchiMate relationship.		associated with ← associated to → associated from
Dynamic Relationships		**Notation**	**Role Names**
Triggering	Represents a temporal or causal relationship between elements.		← triggers → triggered by
Flow	Represents transfer from one element to another.		← flows to → flows from
Other Relationships		**Notation**	**Role Names**
Specialization	Represents that an element is a particular kind of another element.		← specializes → specialized by
Relationship Connectors		**Notation**	**Role Names**
Junction	Used to connect relationships of the same type.	(And) Junction Or Junction	

Fig. 6.6: Screenshot from the table with a summary of relationships in the ArchiMate specification
For a link, refer to *Summary of Relationship* in the *See also* box.

BIBLIOGRAPHY

[1] *Univention Corporate Server - Manual for users and administrators*. Univention GmbH, 2021. URL: https://docs.software-univention.de/manual/5.2/en/.

[2] *Univention App Center for App Providers*. Univention GmbH, 2021. URL: https://docs.software-univention.de/app-center/5.2/en/.

[3] *Univention Developer Reference*. Univention GmbH, 2021. URL: https://docs.software-univention.de/developer-reference/5.2/en/.

[4] *Univention Corporate Server Python API 5.2 documentation*. Univention GmbH, 2021. URL: https://docs.software-univention.de/ucs-python-api/.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[6] Simon Brown. *The C4 model for visualising software architecture*. 2018. URL: https://c4model.com/.

[7] *ArchiMate® 3.2 Specification*. The Open Group, Oct 2022. URL: https://pubs.opengroup.org/architecture/archimate32-doc/_archimate_3_2_specification.html.

[8] Gerben Wierda. *Mastering ArchiMate Edition 3.1: A serious introduction to the ArchiMate® enterprise architecture modeling language*. R&A, second edition, 2021. ISBN 978-90-831434-0-8. URL: https://ea.rna.nl/mastering-archimate-edition-3-1/.

INDEX

A

M

P

R

S

T

U

V

W