Sovereign Workplace - Provisioning 0.11

Release 0.11

Univention GmbH

07.12.2023

Inhaltsverzeichnis

1 Versionen

Tab. 1.1: Versionen

Version	Datum	Autor	Inhalt
0.1	28.04.2023	*M. Staps*	Erster Entwurf
0.2	12.06.2023	*M. Staps*	Zwischenstand Einarbeitung Reviews
0.3	14.06.2023	*M. Staps*	Einarbeitung Workshop Entwicklung vom 14.06.2023
0.4	19.06.2023	*M. Staps*	MOM Queues getrennt
0.5	22.06.2023	*M. Staps*	Diskussionsgrundlage Architektur Kernteam
0.6	11.07.2023	*M. Staps*	Einarbeitung 2. Review Dataport, generelles Lösungsdesign
0.7	25.07.2023	*M. Staps*	Umsetzung Lösungsdesign mit UDM
0.8	25.010.2023	*M. Staps*	Einarbeitung Ergebnisse Spikes
0.9	30.10.2023	*M. Staps*	Überarbeitung, erledigte offene Punkte entfernt
0.10	22.11.2023	*M. Staps*	Administratoren getrennt in Systemadministratoren und openDesk Administratoren (Anmerkung Theres Meyer)
0.11	06.12.2023	*M. Staps*	Einarbeitung Anmerkungen Sascha Klosch (Abnahme Dokument)

2 Referenzierte Dokumente

Im nachfolgenden Konzept wird auf folgende Dokumente Bezug genommen:

Tab. 2.1: Referenzen

	ID	Dokument Name	Verantwortlich	Beschreibung
	001	CDR_230710_Architekturkonzept_v34.docx (Architekturkonzept des BMI Version 34)	BMI	https://fs.px.souvap.dphoenixsuite.de/s/TXc9qAD8neCTwNZ?dir=undefined&path=%2FReferenzen&openfile=74499
	002	Authentifizierung und Autorisierung	Univention	**opendesk_architecture_con10.pdf**
	003	UCS Dokumentation Wurzelverzeichnis	Univention	https://docs.software-univention.de/index.html
	004	D-05-UV-104_Provisioning_infrastructure_implementation	Dataport	https://pm.souvap.dphoenixsuite.de/projects/swp-workpackages-2023/work_packages/2049/activity
	005	ADR-001	Univention	https://git.knut.univention.de/univention/decision-records/-/blob/27736a895b5dfc116ac30a67b02337bee95d4024/openDesk/provisioning/0001-mq-backend-evaluation.md (Kopie in Anlage)

3 Einleitung

3.1 Zielsetzung

Im aktuellen IAM von Univention erfolgt die Synchronisation von Benutzer-, Gruppen-, sowie von Asset Objekten mit anderen Komponenten über ein **Listener-Notifiier** Verfahren. Die aktuelle Verfahren entspricht nicht den Anforderungen für einen Betrieb unter K8s.

Das vorliegende Dokument beschreibt die Anbindung und Synchronisation von Benutzer-, Gruppen-, sowie von Asset Objekten, die im zentralen IAM verwaltet werden mit Komponenten, die über einen eigenen Datenhaushalt verfügen.

In der ersten Stufe handelt es sich um alle Objekte, die über die Services der UDM API's im dahinter liegenden Persistenzlayer (aktuell OpenLDAP) verwaltet werden.

Die Anbindung wird als Provisionierung bezeichnet.

Die Provisionierung ist ein zentraler Bestandteil der Integration von Komponenten in den Souveränen Arbeitsplatz.

3.2 Adressaten

Dieses Dokument ist für folgende Adressaten bestimmt

- Architekten der Souveränen Arbeitsplatz zur Ergänzung des Dokuments *„Sovereign Workplace Architecture“*.
- Architekten, Systemadministratoren der Betreiber für die Installation der Komponenten
- Entwickler zur Entwicklung der benötigten Funktionalität

3.3 Abgrenzung des Dokumentes

Folgende Themen sind nicht Bestandteil des Dokumentes:

- Installation und Konfiguration des Souveränen Arbeitsplatz
- Integration anderer Komponenten des Souveränen Arbeitsplatz (z.B., IAM)
- Integration von IAM zur Authentifizierung und Autorisierung
- Erweiterungen der Datenmodelle im OpenLDAP
- Verfahren zur Speicherung von Daten im OpenLDAP (zum Beispiel im Rahmen der IAM Anbindung)

3.4 Relevante Behörden und Firmen

Die folgenden Behörden und Firmen sind für das Dokument relevant:

Tab. 3.1: Behörden und Firmen

Name	Type	Bedeutung	Website
BMI (Bundesministerium des Innern und für Heimat)	Public	Auftraggeber	*www.bmi.bund.de*
Dataport	Public	Betreiber des Souveränen Arbeitsplatz	*www.dataport.de*
Open-Xchange-AG	Unternehmen	OSS Hersteller	*www.open-xchange.com*
Weitere OSS Hersteller	Unternehmen	OSS Hersteller	
BMWK (Bundesministerium für Wirtschaft und für Klimaschutz)	Public	Behörde	*www.bmwk.de*
HZD (Hessische Zentrale für Datenverarbeitung)	Public	Behörde	*www.hzd.hessen.de*

3.5 Umfang

Das Konzept beinhaltet folgende Kapitel:

- Kapitel 1 beschreibt die fachlichen Anforderungen, die aus dem IST Stand sowie den referenzierten Dokumenten abgeleitet wurden.
- Kapitel 2 beschreibt die Lösungsarchitektur

4 Kapitel 1: Anforderungen

Ein zentraler Bestandteil des SouvAP ist ein System für die Verwaltung von Benutzer und Gruppen Objekten, die hauptsächlich für die Autorisierung der User im Portal (Benutzerinterface) sowie in den integrierten Komponenten benötigt werden. **Dieses System wird als IAM System bezeichnet (IAM).**

Das aktuelle System von Univention (Univention Coporate Server - UCS) beinhaltet alle Komponenten des IAM Systems.

Eine Beschreibung der Module und Funktionen ist hier zu finden: UCSManual for users and administrators[1].

Für das Verständnis des Konzeptes sind folgende UCS Komponenten wichtig:

- *OpenLDAP* - Persistenzlayer für die Verwaltung der Objekte
- *Unvention Dirictory Manager (UDM)* - Fassade für den Zugriff auf das OpenLDAP. Beinhaltet u.a. das Mapping von fachlichen Werten auf technische Felder.

[1] https://docs.software-univention.de/manual/5.0/en/index.html

4.1 Übergreifende Anforderungen

Die Provisionierung ist eine Bestandteil der Integrationsfähigkeit des SouvAP. Sie benutzt dabei die durch einen MOM bereitgestellte Capability Message Brokerage.

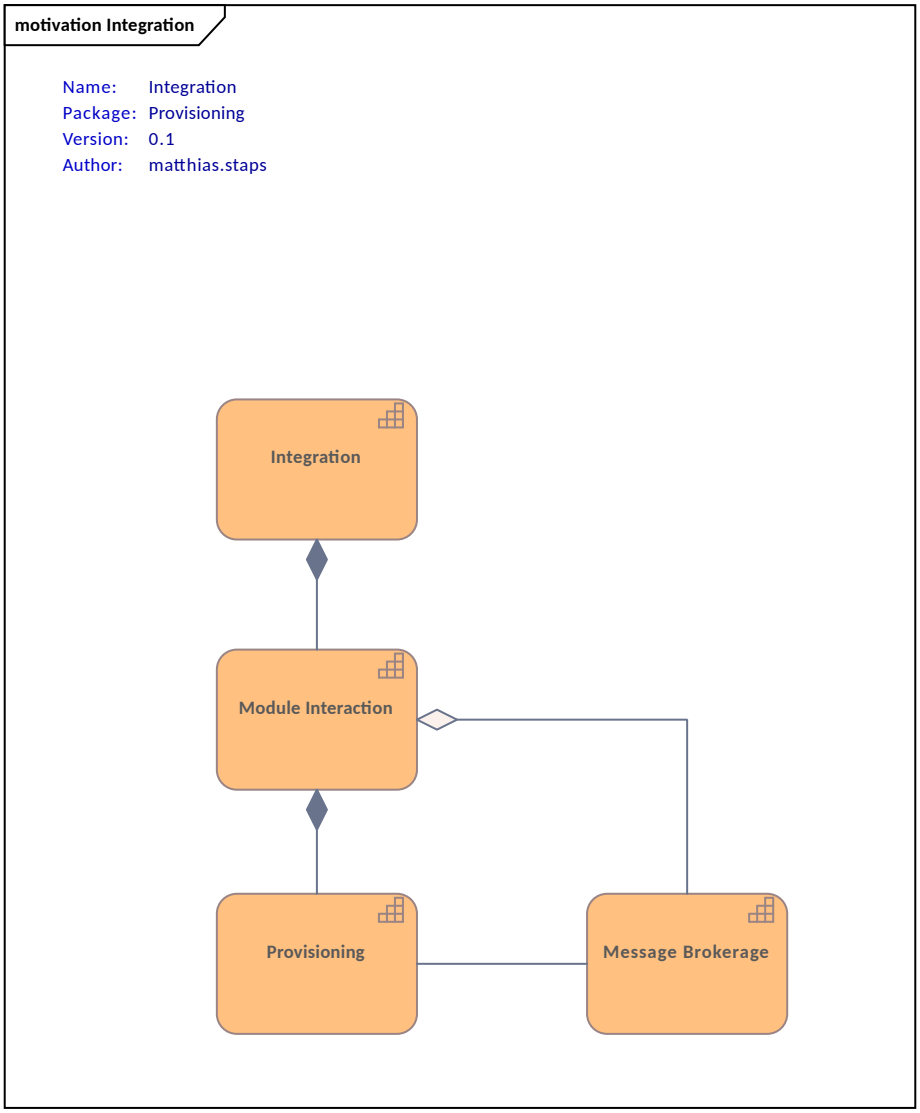

Abb. 4.1: **Figure** Capability Interoperabilität

Für die Integration von Module in den SouvAP sind folgende Standards zu verwenden (see also *Ref. ID 001* (Seite 2)):

Tab. 4.1: Anforderungen Integration

ID	Beschreibung	Quelle	Status
INT_001	Einsatz von standardisierten Protokollen für den Nachrichtenaustausch	see also *Ref. ID 001* (Seite 2) - Kapitel 6.1.2.2.1	bestätigt
INT_002	Einsatz von standardisierte Protokollen für den Datenaustausch	see also *Ref. ID 001* (Seite 2) - Kapitel 6.1.2.2.3.	bestätigt
INT_003	Einsatz von standardisierten Datenformaten	see also *Ref. ID 001* (Seite 2) - Kapitel 6.1.2.2.4.	bestätigt
INT_004	Austauschbarkeit von OSS Modulen	see also *Ref. ID 001* (Seite 2) - Kapitel 6.1.2.9.5.	bestätigt
INT_005	Anpassung der Komponentenschnittstelle mit einem Adapter	see also *Ref. ID 001* (Seite 2) - Kapitel 6.1.5.3.	bestätigt
INT_006	Einsatz von Konfigurations- und Anwendungsprotokoll Standards	see also *Ref. ID 001* (Seite 2) - Kapitel 6.2.2.8.	bestätigt

An die Komponenten/ Module des Interop Layers bestehen folgende allgemeine Anforderungen. (see also *Ref. ID 001* (Seite 2)):

Tab. 4.2: Anforderungen Interop Layer

ID	Beschreibung	Quelle	Status
INTOP_001	sorgt für eine nahtlose, effiziente und sichere Kommunikation zwischen verschiedenen Softwaresystemen	see also *Ref. ID 001* (Seite 2) - Kapitel 5.9.2	bestätigt
INTOP_002	sorgt für den Datenaustausch zwischen Anwendungen bzw. technischen Komponenten	see also *Ref. ID 001* (Seite 2) - Kapitel 5.9.2	bestätigt
INTOP_003	unterhält Sicherheitstechniken zur Vermeidung von Unterbrechungen, Verzögerungen und Informationsverluste vermeiden	see also *Ref. ID 001* (Seite 2) - Kapitel 5.9.2	bestätigt

4.2 Technische Anforderungen

Anforderungen aus der aktuellen Integration des IAM.

Tab. 4.3: Anforderungen aus aktuellem IAM

ID	Beschreibung	Quelle	Status
UNI_001	Für die Verarbeitung von Events benötigen Komponenten den alten sowie den neuen Zustand der Objekte	Analyse aktueller Funktionsumfang	bestätigt
UNI_002	Änderungen an OpenLDAP Objekten erfolgen nur über die UDM API'so	Analyse aktueller Funktionsumfang	offen
UNI_003	UDM REST API benötigt die Berechtigungen für den Zugriff auf das OpenLDAP. (Einsatz von OPA) Sie benötigt also in jedem Fall auch eine extra LDAP `cn=admin` Connection.	Analyse aktueller Funktionsumfang	bestätigt
UNI_004	Die Sicherstellung der Abarbeitung der Reihenfolge der `Events` durch den `Consumer` erfolgt grundsätzlich nach dem `first-in - first-out` Prinzip, das bei Bedarf durch Filter und Sortierung übersteuert werden kann.	Analyse aktueller Funktionsumfang	bestätigt
UNI_005	Durch die UDM API ist eine Klammer (wenn möglich als Transaktion) über fachlich zusammenhängende Einzeloperationen im OpenLDAP zu bilden.	Workshop DEV 14.06.2023	bestätigt

4.3 Fachliche Anforderungen

Der Souveräne Arbeitsplatz beinhaltet Module (Produkt im Sinne des SouvAP), die zur Sicherstellung ihrer Funktionalität einen eigenen Datenhaushalt mit Benutzer- und Gruppeninformationen verwalten. Ein Beispiel dafür ist OX, das unter anderem für jeden User des SouvAP eine Mailbox bereitstellt und verwaltet.

Es handelt sich damit um einen konkreten Fall, in dem Daten zwischen Komponenten ausgetauscht werden.

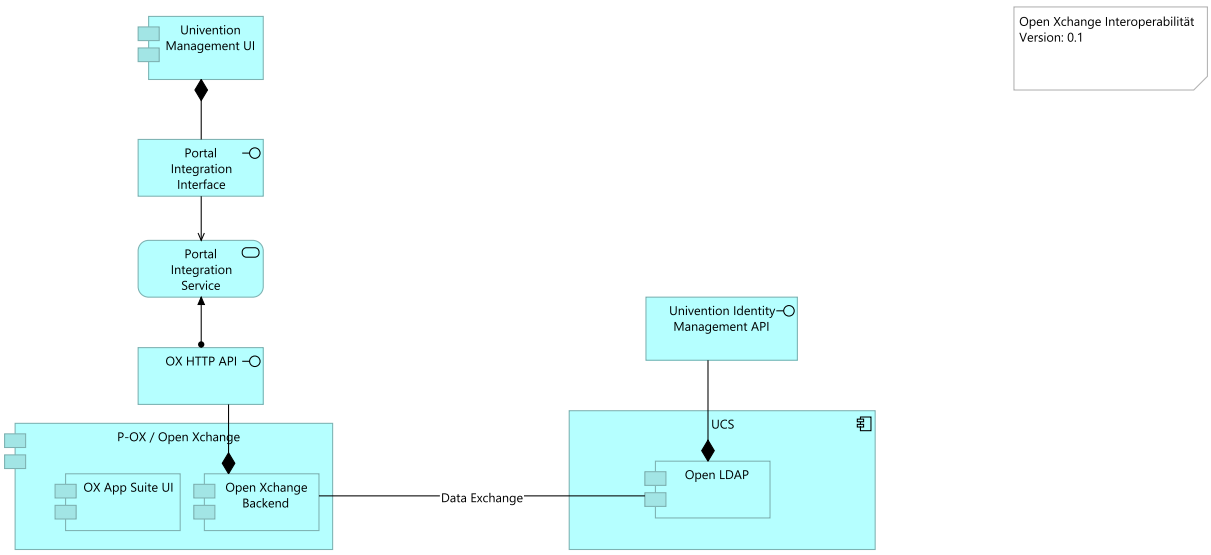

Abb. 4.2: **Figure** Beispiel Interoperabilität OX

Fachlich gibt es zwei zentrale Anforderungen an die Lösung:

1. Im Rahmen des **User-Lifecycle** werden in das zentrale IAM des SouvAP neue Benutzer aufgenommen, Berechtigungen geändert oder Benutzer deaktiviert und gelöscht. Es ist die Erwartung eines Benutzers, dass er bei seiner ersten Anmeldung an das Portal in allen Komponenten des SouvAP die erforderliche Autorisierung für die Nutzung der Funktionalitäten der Komponenten besitzt.
2. Über UDM erfolgt die Verwaltung von Asset Objekten im OpenLDAP, die für die Funktion der Komponenten

relevant sind. Der **Lifecycle** dieser Objekte ist ebenfalls über das Provisioning an die Komponenten weiter zu geben.

Im konkreten Beispiel mit OX ist die Erwartungshaltung, dass der Benutzer z.B. E-Mails senden und empfangen kann. Um das zu ermöglichen, müssen in OX die benötigten Mailpostfächer angelegt werden. Für Termine können Besprechungsräume gebucht werden, die als Asset über UDM angelegt und gepflegt werden und die im OX verfügbar sein müssen.

Die Provisionierung stellt die benötigten Daten bereit.

Tab. 4.4: Fachliche Anforderungen

ID	Beschreibung	Quelle	Status
PROV_001	Alle Komponenten des SouvAP verfügen bei der ersten Anmeldung eines Benutzers über die erforderlichen Informationen zur Autorisierung	Analyse aktueller Funktionsumfang	Bestätigt
PROV_002	Änderung sich Benutzerdaten, so ist diese Informationen allen Komponenten zur Verfügung zu stellen, um eine Auswertung/ Nutzung in den Komponenten zu ermöglichen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_003	Werden Benutzer gelöscht, so ist diese Information allen Komponenten zu Verfügung zu stellen, um alle relevanten Daten zu diesem Benutzer ebenfalls zu löschen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_004	Werden im IAM Gruppen neu angelegt, so ist diese Information allen Komponenten zur Verfügung zu stellen, um eine Auswertung/ Nutzung in den Komponenten zu ermöglichen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_005	Werden Gruppen gelöscht, so ist diese Information allen Komponenten zu Verfügung zu stellen, um alle relevanten Daten zu dieser Gruppe ebenfalls zu löschen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_006	Werden im IAM neue Objekte, z.B. Assets angelegt, so ist diese Information allen Komponenten zur Verfügung zu stellen, um eine Auswertung/ Nutzung in den Komponenten zu ermöglichen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_007	Werden Assets gelöscht, so ist diese Information allen Komponenten zu Verfügung zu stellen, um alle relevanten Daten zu diesen Assets ebenfalls zu löschen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_008	Änderung sich Assetdaten, so ist diese Informationen allen Komponenten zur Verfügung zu stellen, um eine Auswertung/ Nutzung in den Komponenten zu ermöglichen	Analyse aktueller Funktionsumfang	Bestätigt
PROV_009	Änderungen von Objekten können bei Notwendigkeit mehrere Events generieren, so dass mehrere Aktionen möglich sind	Ist-Analyse	Bestätigt
PROV_010	Den Komponenten darf nur die Information verfügbar gemacht werden, die sie wirklich benötigen. Die Definition welche Komponente Berechtigungen für den Zugriff auf welche Daten bekommt muss mit dem Berechtigungsmodell des IAM so integriert sein, das Berechtigungen nur an einer Stelle definiert werden.	Analyse aktueller Funktionsumfang	Bestätigt

4.4 Rollen

Die Einrichtung der Provisionierung als festen Bestandteil des openDesk erfolgt durch Systemadministratoren.

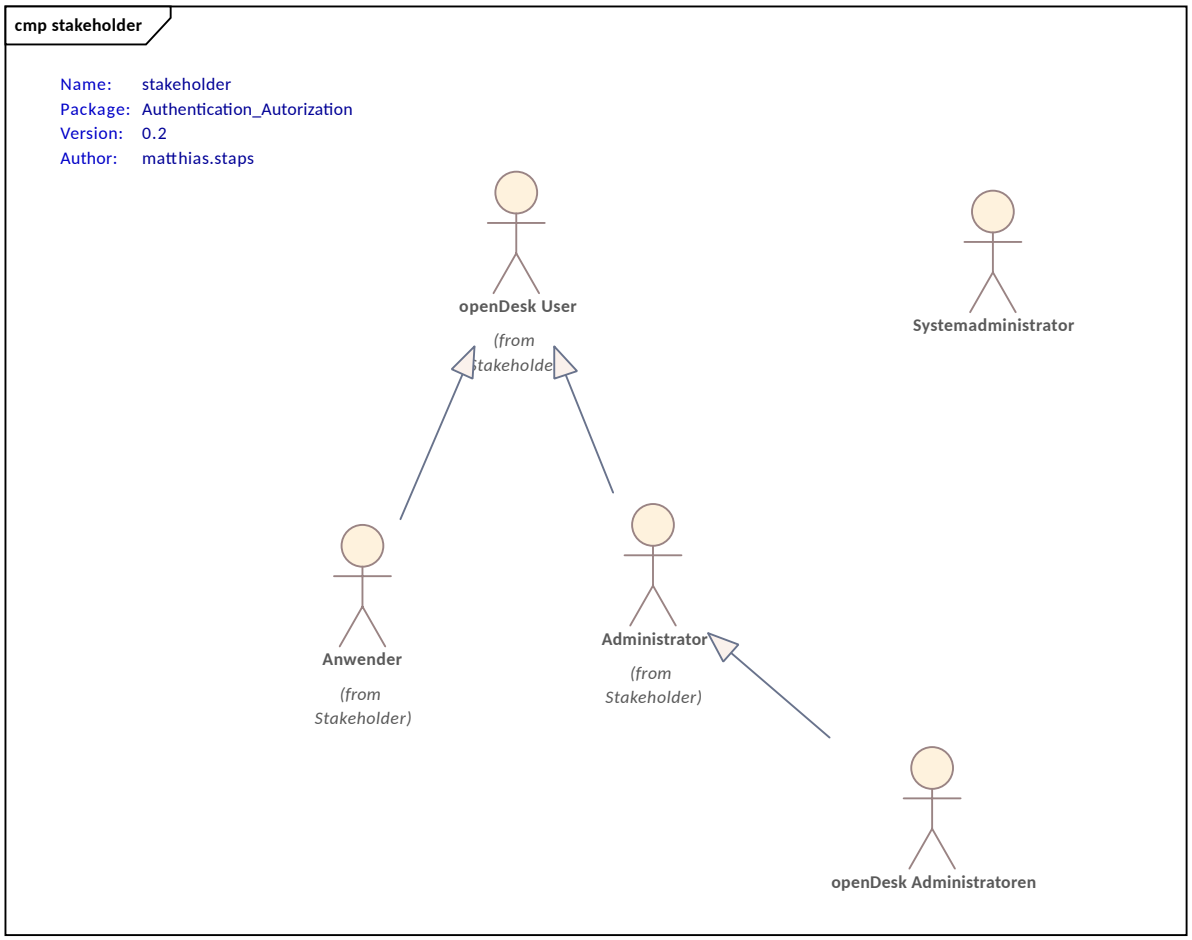

Abb. 4.3: **Figure** Stakeholder

Die Verwaltung einer Instanz, wie zum Beispiel das Einrichten von Benutzern und Gruppen erfolgt durch openDesk Administratoren.

4.5 Use Cases

Zum besseren Verständnis wird die Umsetzung der Anforderungen in Form von generischen Use Cases beschrieben.

Bemerkung: Die Use Cases für den User-Lifecycle im IAM sind Ausgangspunkt für die folgenden Use-Cases. Sie werden auf drei grundlegende Anwendungsfälle zusammengefasst:

- **create**
- **update**
- **delete**

Tab. 4.5: Use Cases Provisionierung

ID	Beschreibung	umgesetzte Anforderung
UC_PROV_001	Neuaufnahme eines Benutzers	PROV_001
UC_PROV_002	Änderung Gruppenzugehörigkeit eines Benutzers	PROV_001 PROV_002
UC_PROV_003	Änderung Attribute zu einem Benutzer	PROV_002
UC_PROV_004	Deaktivieren eines Benutzer	
UC_PROV_005	Löschen eines Benutzer	PROV_003
UC_PROV_006	Anlegen einer Gruppe	PROV_001 PROV_004
UC_PROV_007	Löschen einer Gruppe	PROV_005
UC_PROV_008	Anlegen eines Asset	PROV_00
UC_PROV_009	Bearbeiten eines Asset	PROV_007 PROV_008
UC_PROV_010	Löschen eines Asset	PROV_007 PROV_008
UC_PROV_011	Hinzufügen einer Komponente zu einem bestehenden IAM	PROV_001 PROV_002 PROV_004
UC_PROV_012	Anlegen/Ändern einer Projektgruppe	UC_PROV_009

Im folgenden werden die Use Cases im einzelnen beschrieben:

UC_PROV_001 - Neuaufnahme eines Benutzers

Tab. 4.6: UC_PROV_001

ID	UC_PROV_001
Name	Neuaufnahme eines Benutzers
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert
Nachbedingung (Erfolg)	Neuer Benutzer wurde in den Komponenten angelegt
Trigger	Neuer Benutzer im IAM angelegt
Haupterfolgsszenario	1. Neuer Benutzer wird im IAM angelegt 2. Event über den neuen Benutzer wird allen konfigurierten Komponenten bereitgestellt 3. Die benötigten Daten für das Anlegen des Benutzer in der jeweiligen Komponente werden bereitgestellt 4. Benutzer wird in den Komponenten angelegt und autorisiert
Alternativen/ Ergänzungen	keine

UC_PROV_002 - Änderung Gruppenzugehörigkeit eines Benutzers

Tab. 4.7: UC_PROV_002

ID	_UC_PROV_002
Name	Änderung Gruppenzugehörigkeit eines Benutzers
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert • Benutzer ist im IAM und in den Komponenten vorhanden
Nachbedingung (Erfolg)	Geänderte Gruppenzugehörigkeit wurde verarbeitet
Trigger	Benutzer wird einer Gruppe hinzugefügt oder aus einer Gruppe entfernt.
Haupterfolgsszenario	1. Benutzer wird einer Gruppe hinzugefügt oder aus einer Gruppe entfernt. 2. Event über die Änderung wird allen konfigurierten Komponenten bereitgestellt. 3. Die benötigten Daten für die Verarbeitung in den jeweiligen Komponente werden bereitgestellt. 4. Verarbeitung der Daten in den Komponenten (z.B. Änderung Berechtigung).
Alternativen/ Ergänzungen	keine

UC_PROV_003 - Änderung Attribute zu einem Benutzer

Tab. 4.8: UC_PROV_003

ID	_UC_PROV_003
Name	Änderung Attribute zu einem Benutzer
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert • Benutzer ist im IAM und in den Komponenten vorhanden
Nachbedingung (Erfolg)	Geänderte Benutzerattribute wurde verarbeitet
Trigger	Benutzerattribut wurde geändert
Haupterfolgsszenario	1. Benutzerattribut wurde geändert 2. Event über die Änderung wird allen konfigurierten Komponenten bereitgestellt 3. Die benötigten Daten für die Verarbeitung in den jeweiligen Komponente werden bereitgestellt 4. Verarbeitung der Daten in den Komponenten (z.B. Änderung Name)
Alternativen/ Ergänzungen	keine

UC_PROV_004 - Deaktivieren eines Benutzer

Tab. 4.9: UC_PROV_004

ID	_UC_PROV_004
Name	Deaktivieren eines Benutzer
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert • Benutzer ist im IAM und in den Komponenten vorhanden • SSO ist an einer zentralen Stelle konfiguriert
Nachbedingung (Erfolg)	keine Anmeldung möglich
Trigger	Benutzer wird deaktiviert
Haupterfolgsszenario	1. Benutzer wird deaktiviert 2. keine Bereitstellung des Events 3. Authentifizierung des Benutzers nicht möglich (keine Anmeldung am Portal)
Alternativen/ Ergänzungen	keine

UC_PROV_005 - Löschen eines Benutzer

Tab. 4.10: UC_PROV_005

ID	_UC_PROV_005
Name	Löschen eines Benutzer
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert • Benutzer ist im IAM und in den Komponenten vorhanden
Nachbedingung (Erfolg)	Benutzer mit allen Daten wurde gelöscht
Trigger	Benutzer im IAM gelöscht
Haupterfolgsszenario	1. Benutzer wurde gelöscht 2. Event über die Löschung wird allen konfigurierten Komponenten bereitgestellt 3. Löschen des Benutzers sowie aller Daten in den Komponenten (z.B. Mailbox in OX)
Alternativen/ Ergänzungen	keine

UC_PROV_006 - Anlegen einer Gruppe

Tab. 4.11: UC_PROV_006

ID	_UC_PROV_006
Name	Anlegen einer Gruppe
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert
Nachbedingung (Erfolg)	Gruppe in den Komponenten angelegt
Trigger	Neue Gruppe im IAM angelegt
Haupterfolgsszenario	1. Neue Gruppe wird im IAM angelegt 2. Event über die neue Gruppe wird allen konfigurierten Komponenten bereitgestellt 3. Die benötigten Daten für die Verarbeitung in den jeweiligen Komponente werden bereitgestellt 4. Gruppe in den Komponenten angelegt
Alternativen/ Ergänzungen	keine

UC_PROV_007 - Löschen einer Gruppe

Tab. 4.12: UC_PROV_007

ID	_UC_PROV_007
Name	Löschen einer Gruppe
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Benutzer und Gruppen sind konfiguriert • Gruppe ist in den Komponenten vorhanden
Nachbedingung (Erfolg)	Gruppe sowie relevante Daten in den Komponenten gelöscht
Trigger	Gruppe im IAM gelöscht
Haupterfolgsszenario	1. Gruppe wird im IAM gelöscht 2. Event über die Löschung der Gruppe wird allen konfigurierten Komponenten bereitgestellt 3. Gruppe sowie relevante Daten werden gelöscht
Alternativen/ Ergänzungen	keine

UC_PROV_008 - Anlegen eines Asset

Tab. 4.13: UC_PROV_008

ID	UC_PROV_008
Name	Anlegen eines Asset
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Nutzung Asset ist konfiguriert
Nachbedingung (Erfolg)	Neues Asset wurde in den Komponenten angelegt
Trigger	Neues Asset im IAM angelegt
Haupterfolgsszenario	1. Neues Asset wird im IAM angelegt 1. Event über das neue Asset wird allen konfigurierten Komponenten bereitgestellt 2. Die benötigten Daten für das Anlegen des Assets in der jeweiligen Komponente werden bereitgestellt 3. Asset wird in den Komponenten angelegt und autorisiert
Alternativen/ Ergänzungen	keine

UC_PROV_009 - Bearbeiten eines Asset

Tab. 4.14: UC_PROV_009

ID	UC_PROV_009
Name	Bearbeiten eines Asset
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Nutzung Asset ist konfiguriert • Asset im IAM vorhanden
Nachbedingung (Erfolg)	Asset wurde in der Komponenten aktualisiert
Trigger	Asset im IAM geändert
Haupterfolgsszenario	1. Asset wird im IAM geändert 1. Event über das geänderte Asset wird allen konfigurierten Komponenten bereitgestellt 2. Die benötigten Daten für das Assets in der jeweiligen Komponente werden bereitgestellt 3. Asset wird in den Komponenten aktualisiert
Alternativen/ Ergänzungen	keine

UC_PROV_010 - Löschen eines Asset

Tab. 4.15: UC_PROV_010

ID	_UC_PROV_010
Name	Löschen eines Asset
Akteur	System
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Komponenten mit eigenem Datenhaushalt für Assets sind konfiguriert • Asset ist im IAM und in den Komponenten vorhanden
Nachbedingung (Erfolg)	Asset mit allen Daten wurde gelöscht
Trigger	Asset wird im IAM gelöscht
Haupterfolgsszenario	1. Asset wurde gelöscht 1. Event über die Löschung wird allen konfigurierten Komponenten bereitgestellt 2. Löschen des Assets sowie aller Daten in den Komponenten (z.B. Besprechungsraum in OX)
Alternativen/ Ergänzungen	keine

UC_PROV_011 - Hinzufügen einer Komponente zu einem bestehenden IAM

Tab. 4.16: UC_PROV_011

ID	_UC_PROV_011
Name	Hinzufügen einer Komponente zu einem bestehenden IAM
Akteur	System, Systemadministrator
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Im IAM sind Benutzer und Gruppen vorhanden • Neue Komponenten ist konfiguriert
Nachbedingung (Erfolg)	Datenstand in der Komponente ist synchron zum Datenstand im IAM
Trigger	Neue Komponente wird hinzugefügt
Haupterfolgsszenario	1. Neue Komponente installiert 2. Gruppen werden als Event (neu) bereitgestellt 3. Die benötigten Daten für das Anlegen der Gruppen in der Komponente werden bereitgestellt 4. Benutzer werden als Event (neu) bereitgestellt 5. Die benötigten Daten für das Anlegen der Benutzer in der Komponente werden bereitgestellt 6. Benutzer werden in der Komponente autorisiert 7. Komponente wird für Provisionierung aktiviert
Alternativen/ Ergänzungen	keine

UC_PROV_012 - Anlegen/ Ändern einer Projektgruppe

Tab. 4.17: UC_PROV_011

ID	_UC_PROV_012
Name	Anlegen/ Ändern einer Projektgruppe
Akteur	System, OpenDesk Administrator
Vorbedingung	• SouvAP ist konfiguriert und funktionsfähig • Im IAM sind Benutzer und Gruppen vorhanden • Projektgruppe ist im IAM als Objekt bekannt • Mehrere Komponenten sind für die Nutzung von Projektgruppen konfiguriert
Nachbedingung (Erfolg)	Datenstand in allen Komponente ist synchron zum Datenstand im IAM
Trigger	Projektgruppe wird aufgenommen
Haupterfolgsszenario	1. Neue Projektgruppe wird aufgenommen 2. Events für eine neue Gruppe werden generiert 3. Events für die Mitglieder der Gruppen werden generiert 4. Komponente 1 wertet die Events aus, liest die benötigten Daten nach und legt die entsprechenden Objekte an. 5. Komponente 2 wertet die Events aus, liest die benötigten Daten nach und legt die entsprechenden Objekte an. 6. Komponente N.. wertet die Events aus, liest die benötigten Daten nach und legt die entsprechenden Objekte an.Benutzer werden als Event (neu) bereitgestellt 7. Daten sind in allen Komponenten vorhanden und autorisiert
Alternativen/ Ergänzungen	keine

5 Kapitel 2: Lösungsarchitektur

Das Kapitel beschreibt das Lösungsdesign der Provisionierung.

5.1 Annahmen

1. Das IAM (UCS) ist das führende System und zu jedem Zeitpunkt repräsentiert der Datenstand im IAM den aktuellen Zustand.
2. Das Verfahren dient der Bereitstellung von Änderungen von User und User- und Gruppenobjekten des IAM in andere Komponenten.
3. Der Persistenzlayer des IAM ist OpenLDAP
4. Das Verfahren dient ebenfalls der Bereitstellung von Änderungen von komponentenspezifischen Objekten, die im OpenLDAP gespeichert sind und über die Services von UDM bereitgestellt werden.
5. Alle Änderungen von Objekten erfolgen über die REST Services der UDM API's. Es gibt keine schreibenden Zugriffe außerhalb UDM auf das OpenLDAP.

6. Die Autorisierung und Authentizierung der technischen Zugriffe auf die Services wird durch den Betreiber, z.B. mittels OPA, sichergestellt.

5.2 Abgrenzung

1. Der User Lifecycle im IAM, z.B. über die Anbindung an ein zentrales IAM System ist nicht Bestandteil der Lösung
2. Das Verfahren dient nicht der Replikation von im OpenLDAP gespeicherten Objekten/ Daten auf andere OpenLDAP Instanzen (Lastverteilung, Skalierung)
3. Das Verfahren dient nicht der Protokollierung (Tracing) von Zugriffen und Änderungen auf das zentrale IAM
4. Die Bereitstellung von funktionalen programmatischen Erweiterungen (Deployment) ist nicht Bestandteil des Verfahrens

5.3 Lösungsdesign

Das Design beschreibt die Anbindung einer Komponente (Consumer) an den Provisioning Service.

Folgendes generelles Lösungsdesign lässt sich aus den Anforderungen ableiten:

Provisioning Module

Tab. 5.1: Provisioning Module

Name	Beschreibung
`Provisioning Service`	Service für die Verarbeitung von generierten Events.
`Consumer Regsiter Service`	Service für Registrierung von Komponenten (Consumer)
`Message Dispatcher`	Verteilung der Messages an die Consumer
`Consumer Adapter`	Komponentenspezifische Implementierung der Provisioning Schnittstelle.
`Consumer Worker`	Verarbeitet die Messages aus der Consumer Queue
`Provisioning Worker`	Verarbeitet die Messages der Eingangs Queue
`Incomming Provisioning Queue`	Eingangs Queue des Provisioning Services
`Provisioning Configuration`	Persistenzlayer für die Speicherung der Konfiguration der Consumer

Provisioning Service

Nimmt den Event der Quelle entgegen, generiert die Message im SKIM Format und stellt diese in die Eingangsqueue des Provisioning Systems.

Consumer Register Service

Der Service wird bei der Registrierung einer Komponente beim Provisioning System aufgerufen. Es wird der Consumer mit den zu benachrichtigenden Objekt Typen registriert, die die `Consumer Queues` werden angelegt und die Konfiguration wird persistiert.

Message Dispatcher

Verarbeitet Messages aus der Provisioning Eingangsqueue und legt für jeden Consumer entsprechend des abonnierten Objekt Typ die Message in die `Consumer Queues`. Dazu nutzt er die Konfiguration aus der persistierten `Provisioning Configuration`.

Consumer Adapter

Komponentenspezifische Implementierung der Provisionierung für Consumer, die nicht SKIM als Standard unterstützen.

Consumer Worker

Liest die `Consumer Queue` aus und übergibt die Message an den `Component Adapter` zur Verarbeitung. Nach erfolgreichem Abschluss der Verarbeitung wird die Message aus der Queue gelöscht. Im Fehlerfall wird die Message in der Queue gelassen (alternativ Error Queue) und es wird ein Event an das Monitoring System generiert.

Provisioning Worker

Liest die Eingangsqueue aus und übergibt die Message zur Verarbeitung an den `Message Dispatcher`. Nach erfolgreicher Verarbeitung wird die Message aus der Queue gelöscht. Im Fehlerfall wird die Message in der Queue gelassen (alternativ Error Queue) und es wird ein Event an das Monitoring System generiert.

Incomming Provisioning Queue

Eingangsqueue für alle zu relevanten Messages. Die Queue wird im angebundenen MOM bei der Bereitstellung des Provisioning Services angelegt und ist Bestandteil der `Provisioning Configuration`.

Provisioning Configuration

Persistenzschicht für die Konfiguration des Provisioning Services.

Generieren der Message

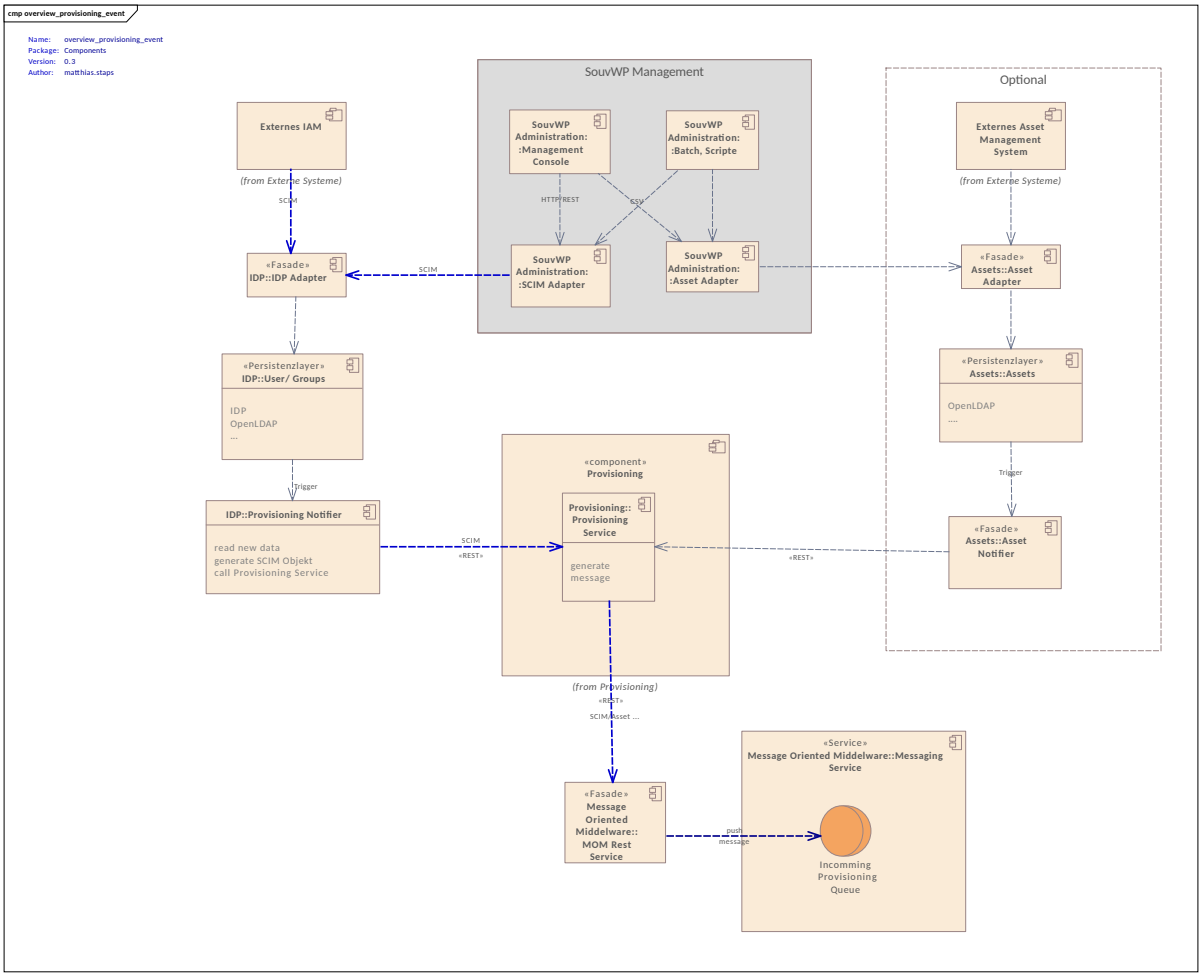

Abb. 5.1: **Figure** Lösungsdesign Generieren Message

Der Identity Provider (IDP) verwaltet alle User und Gruppen Objekte. Bestandteil des IDP ist ein `Provisioning Notifier`, der Events bei Änderungen der verwalteten Objekte (Create, Update, Delete) eine Message an den `Provisoning Service` schickt. Die Message ist im SCIM Format.

Optional können für weitere Objekt Typen nach dem gleichen Verfahren Provisioning Messages erzeugt werden. Das Format der Messages ist dabei Objekt Typ spezifisch.

Dispatching

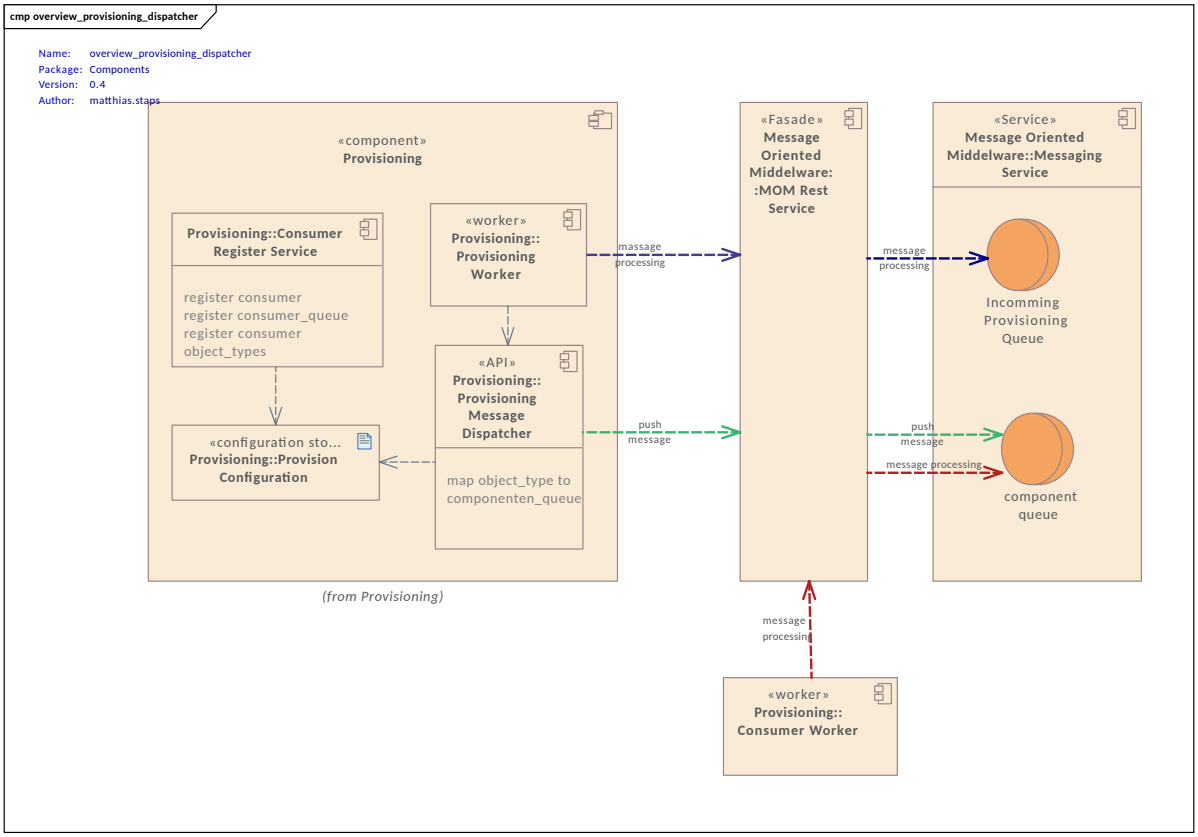

Abb. 5.2: **Figure** Lösungsdesign Dispatching Message

Der `Provisioning Message Dispatcher` verteilt die Messages in die registrierten `Consumer Queues`. Grundlage dafür die ist die `Provisioning Configuration`.

Consumer Register Service Der `Consumer Rergister Service` wird bei der initialen Bereitstellung eines Consumer (Deployment der Komponente) aufgerufen. Er definiert den Endpunkt für den Zugriff des `Consumer worker` und gibt diesen zurück.

Der Endpunkt wird als MOM REST Service bereitgestellt und ermöglicht ein `pull` aller Messages für den Consumer.

Der Endpunkt sowie die benötigte Konfiguration für den `Provisioning Dispatcher` wird in der `Provisioning Configuration` außerhalb des Containers persistiert.

Message Processing

Der `Consumer Worker` liest die Consumer Queue aus und übergibt die Messages an den `Consumer Adapter`, der die eigentliche Provisionierung im Consumer durchführt.

Nach erfolgreicher Verarbeitung wird die Message gelöscht.

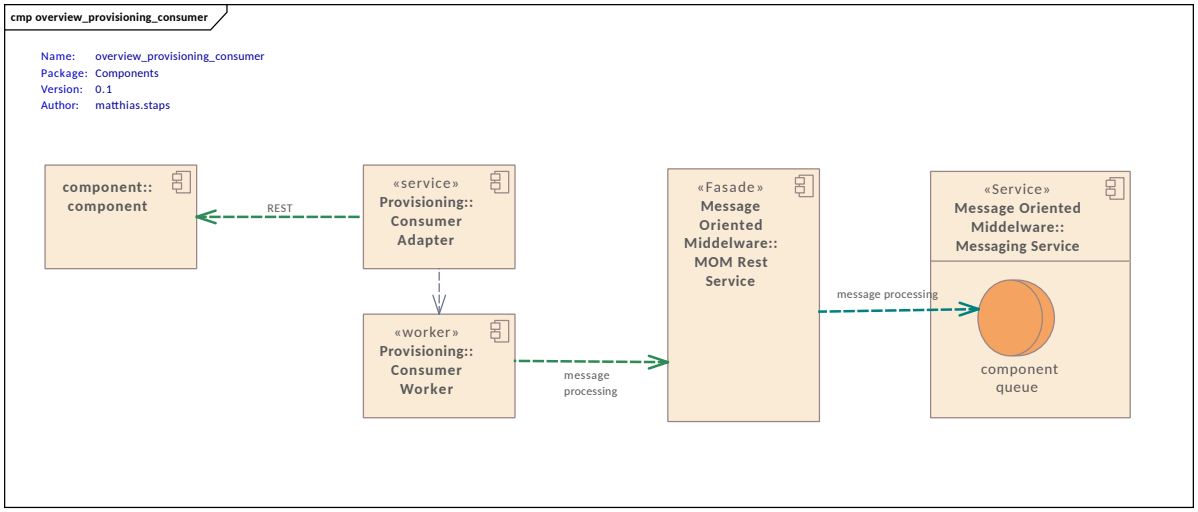

Abb. 5.3: **Figure** Lösungsdesign Message Processing

Umsetzung Anforderungen

Tab. 5.2: Umsetzung übergreifende Anforderungen

Anforderung ID	Umsetzung
INT_001 - Einsatz von standardisierten Protokollen für den Nachrichtenaustausch	Der Austausch von Nachrichten basiert über REST Services.
INT_002 - Einsatz von standardisierte Protokollen für den Datenaustausch	Als Kommunikationsprotokoll wird http/https eingesetzt.
INT_003 - Einsatz von standardisierten Datenformaten	Das eingesetzte Datenformat ist SCIM.
INT_004 - Austauschbarkeit von OSS Modulen	Durch die Standardisierung der Schnittstellen und Kapselung der Funktionalität ist die Austauschbarkeit sichergestellt.
INT_005 - Anpassung der Komponentenschnittstelle mit einem Adapter	Alle Zugriffe auf die Komponente sind mit Adaptern gekapselt.

5.4 Lösungsdesign Provisioning mit UDM

Zum besseren Verständnis wurden für die weitere Detaillierung Diagramme in UML Notation erstellt.

Ableitung Umsetzung in UCS/UDM

UCS ist die aktuelle Implementierung des IDP/IAM Stacks im SouvAP. Im Kontext der Provisionierung werden folgende Module durch UCS/UDM realisiert.

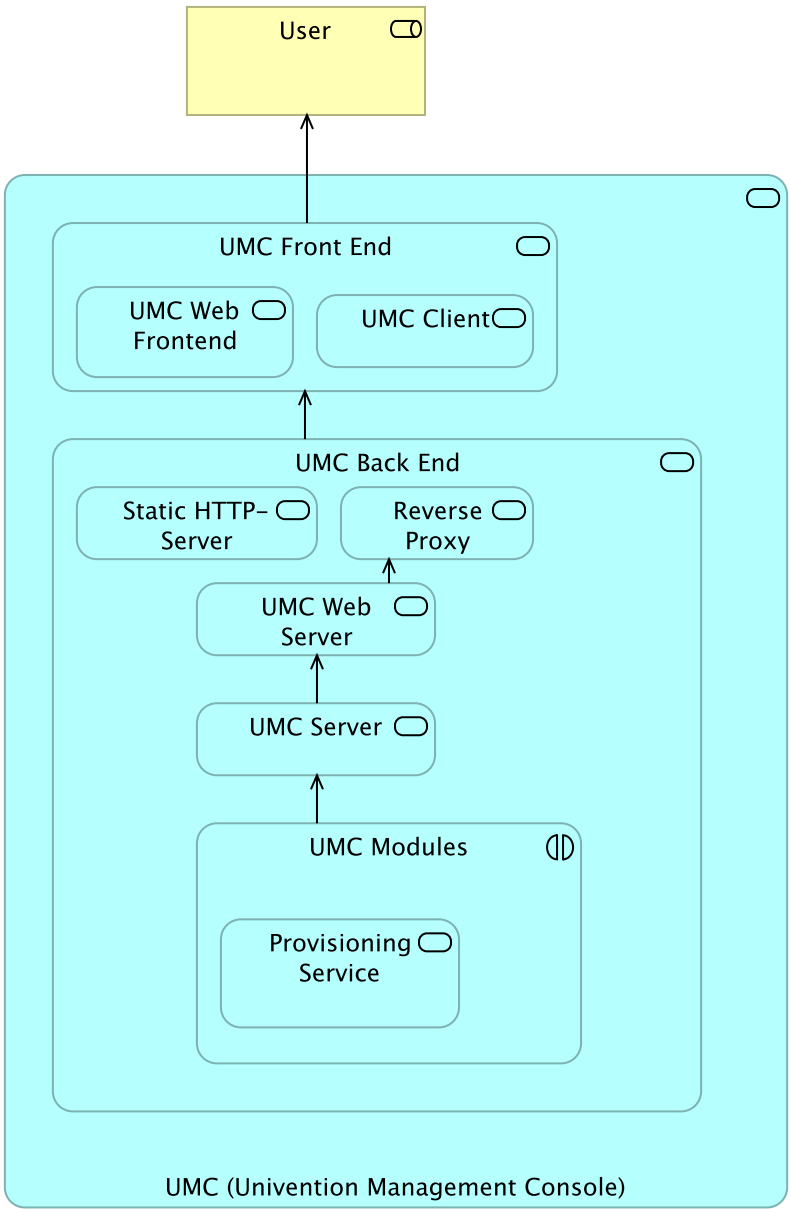

Abb. 5.4: **Figure** Einbettung Provisioning in UMC

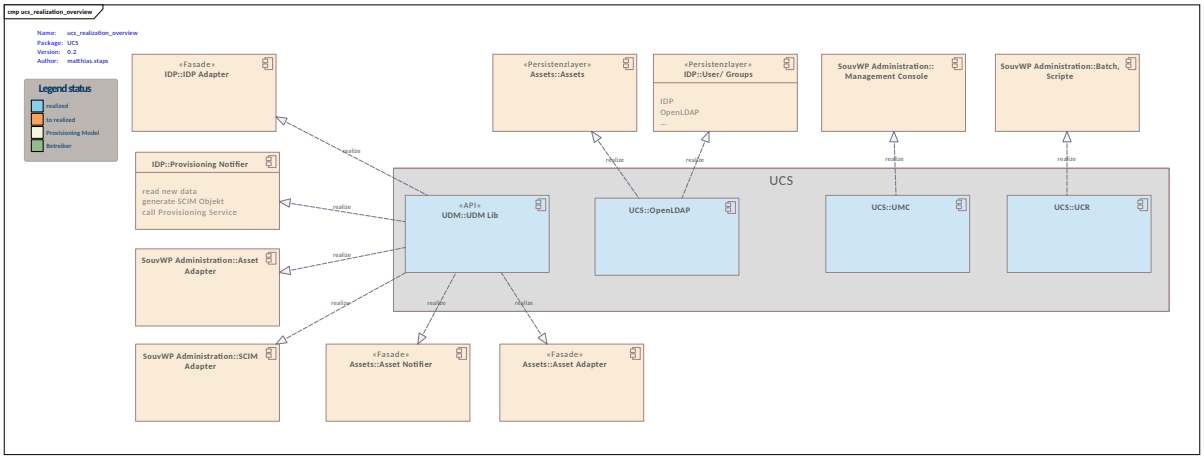

Abb. 5.5: **Figure** Übersicht Provisionierung in UCS

Erweiterung UDM Lib

Übersicht

UDM Lib kapselt die Zugriffe auf das OpenLDAP und stellt die dafür notwendigen Services bereit.

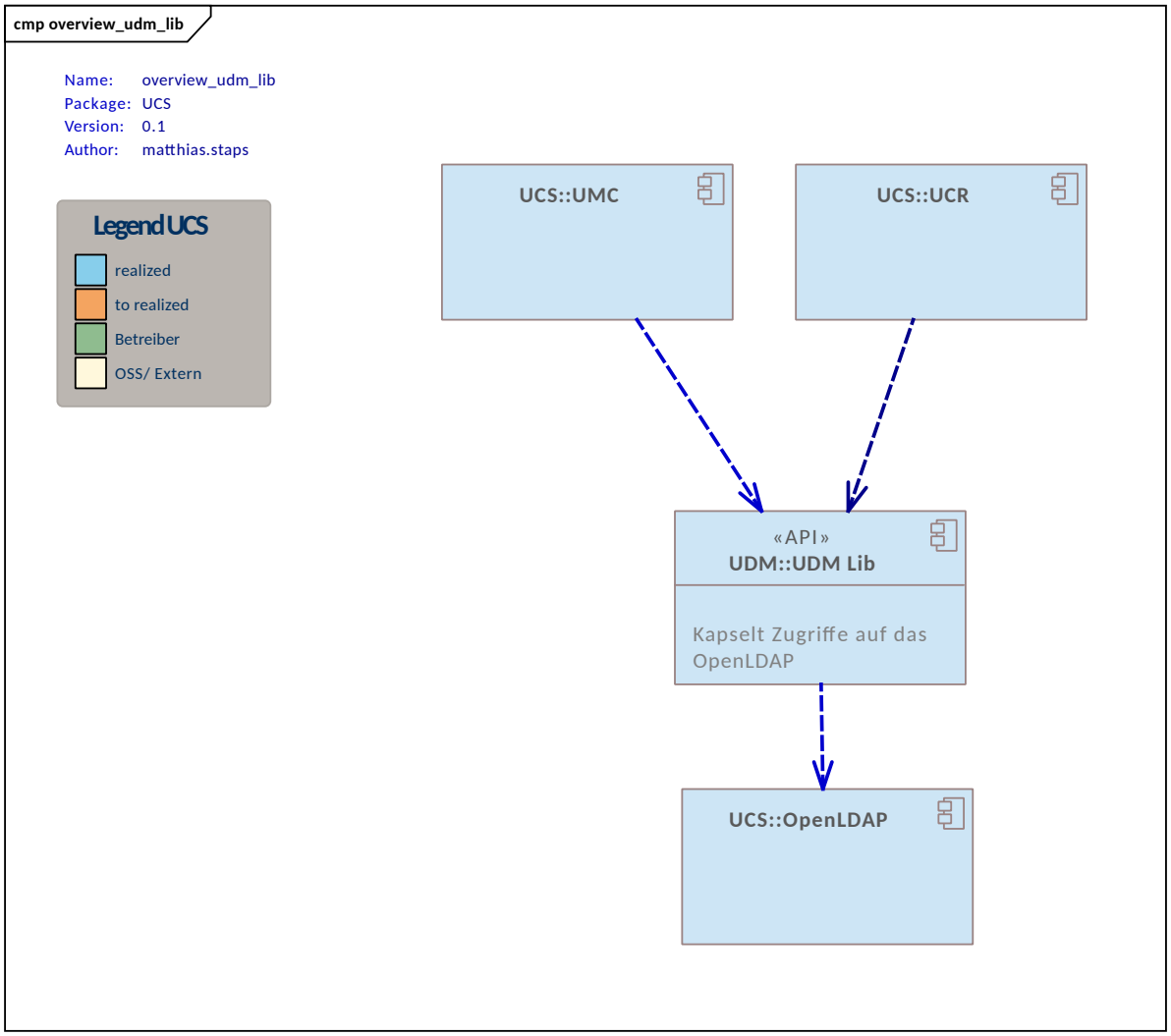

Abb. 5.6: **Figure** UDM Lib aktuell

Für die Provisionierung wird die UDM Lib um eine `Provisioning API` erweitert.

Provisioning API

Die `Provisioning API` stellt einmal die Funktionen für das Auslösen und die Verteilung von Änderungen bereit sowie ebenfalls die Funktionen und Services für die Registrierung von Komponenten am Provisioning Service.

Message Oriented Middleware

Der Messaging Service stellt die `provisioning incomming queue` bereit. Die Zugriffe auf die Queue werden durch den Service `MOM Rest Service` gekapselt.

OpenLDAP Im `OpenLDAP` werden die Änderungen persistiert.

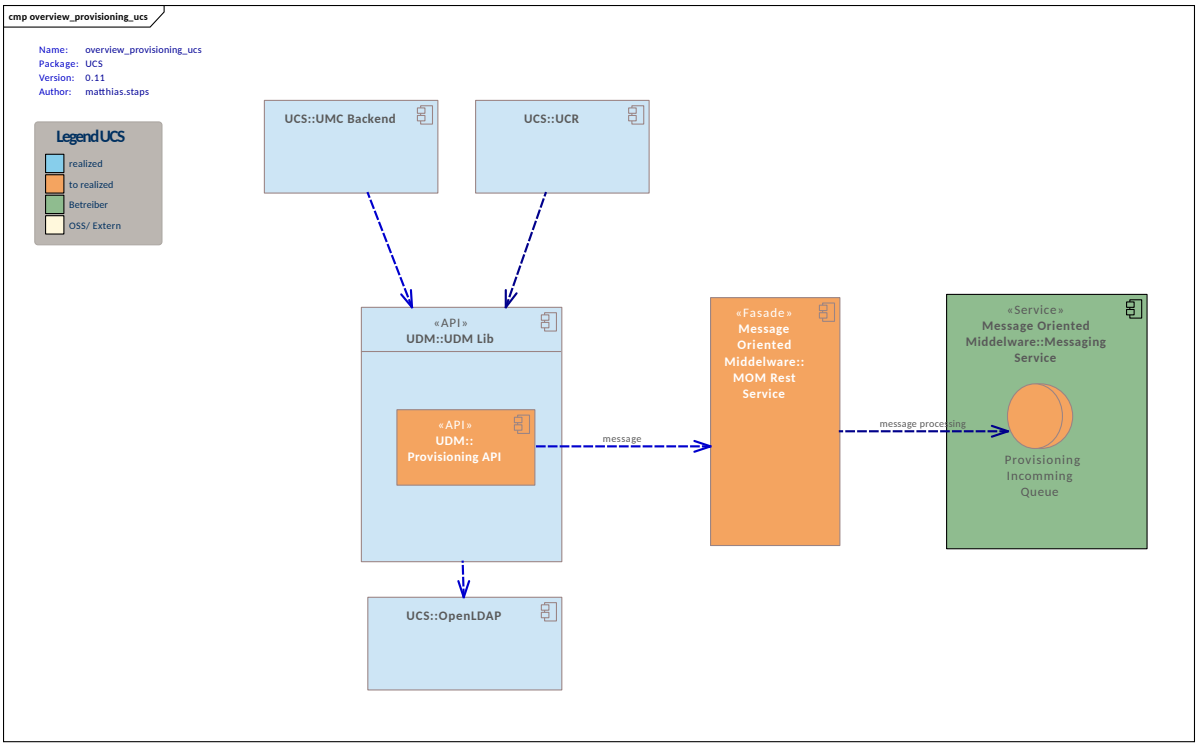

Abb. 5.7: **Figure** Komponenten Provisionierung

Übersicht Provisioning API

Provisioning Adapter Der `Provisioning Adapter` generiert die Message und speichert diesen in über die `MOM Rest API Fassade` in der `provisioning_incomming_queue`.

UDM Provisioning Message Dispatcher Verteilt die in der `provisioning_incomming_queue` eingestellten Messages auf die Komponenten Queues der registrierten Komponenten (Consumer).

UDM Consumer Register Service Service für die Registrierung/Deregistrierung von Komponenten (Consumer) bei deren Bereitstellung.

Provisioning Adapter

Der `Provisioning Adapter` implementiert die Funktionalitäten des `Provsioning Notifier` sowie des `Provisioning Services`.

Integration des `Provisioning Adapters` in die UDM Lib.

Der `Provisioning Adapter` besteht aus dem Modul `generateMessage` sowie dem `formatMessage` Adapter.

createMessage Generiert nach erfolgreichem Speichern im IDP (OpenLDAP) die Message. Die Message beinhaltet den alten Zustand des Objekte, den neuen Zustand des Objektes nach erfolgreichem Update sowie die ID des eingehenden Auftrages. Aus einem eingehenden Auftrag können ggf. mehrere Objekte im OpenLDAP geändert werden. Annahme: `greateMessage` klammert diese Änderungen und übergibt diese als ein Objekt an `formateMessage`.

formateMessage `formateMessage` formatiert die Message in SCIM und stellt diese in die `incomming_provisioning_queue`.

Der prinzipielle Ablauf ist wie folgt:

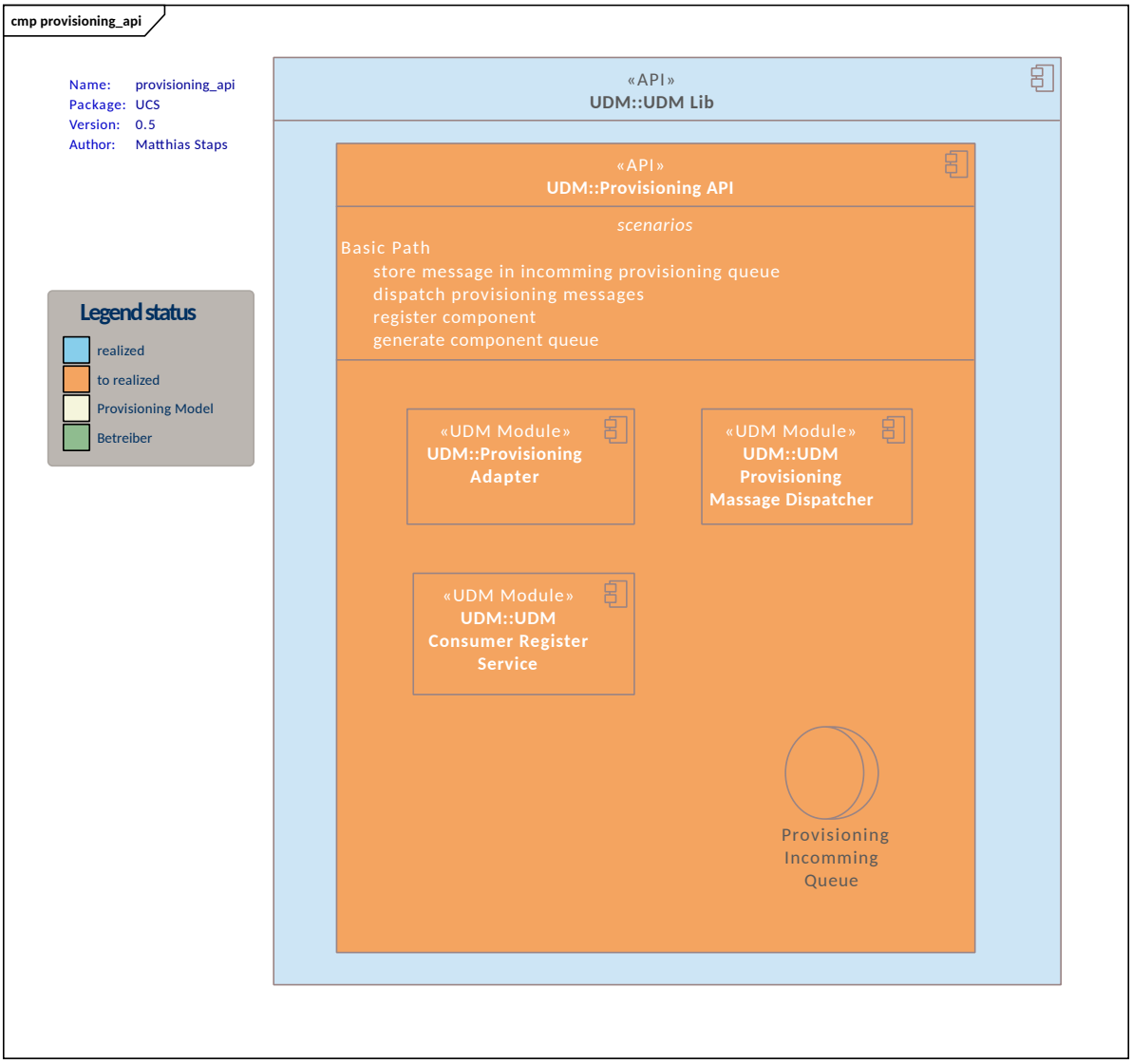

Abb. 5.8: **Figure** Aufbau Provisioning API

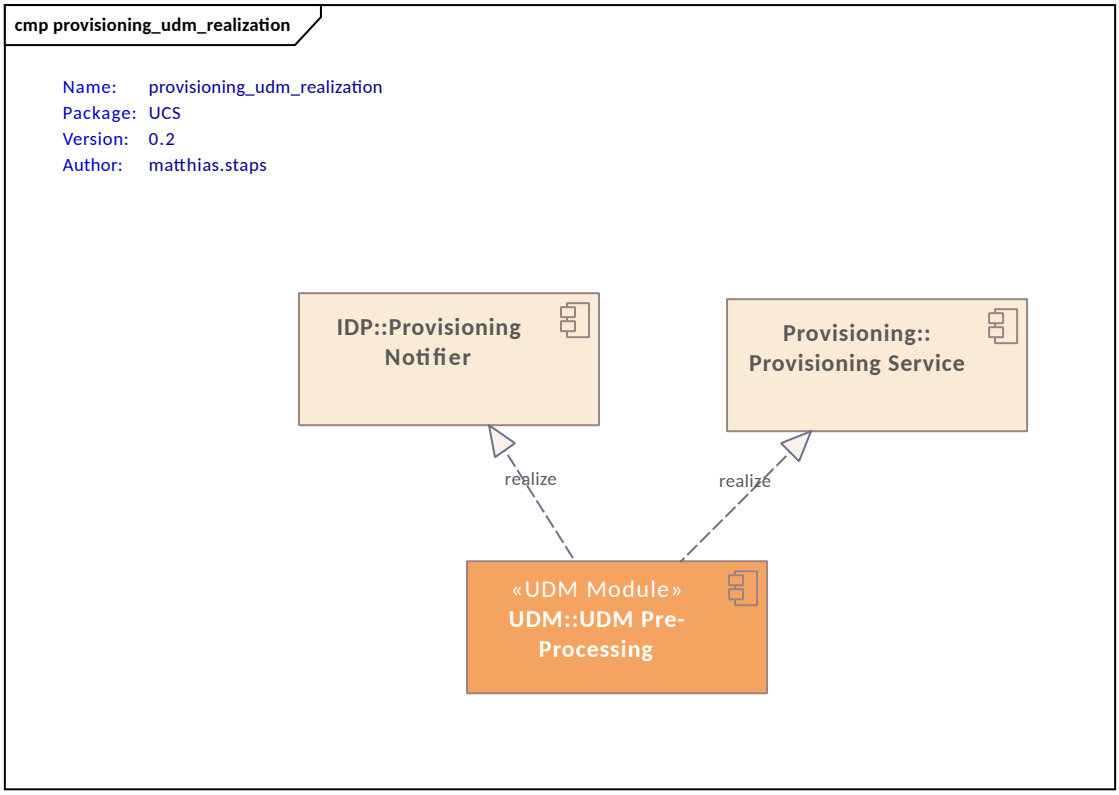

Abb. 5.9: **Figure** Realisierung Anforderungen Lösungsdesign

Provisioning Message Dispatcher

Lösungsdesign siehe *Dispatching* (Seite 19).

Die Implementierung erfolgt mit folgenden Modulen:

Eingesetzte Open Source Software

Für die aktuelle Implementierung wurde als NATS (nats.io) ausgewählt. Die Austauschbarkeit des MOM, z.B. bei Bereitstellung einer MOM Infrastruktur durch den Betreiber, wird durch Fassaden gewährleistet.

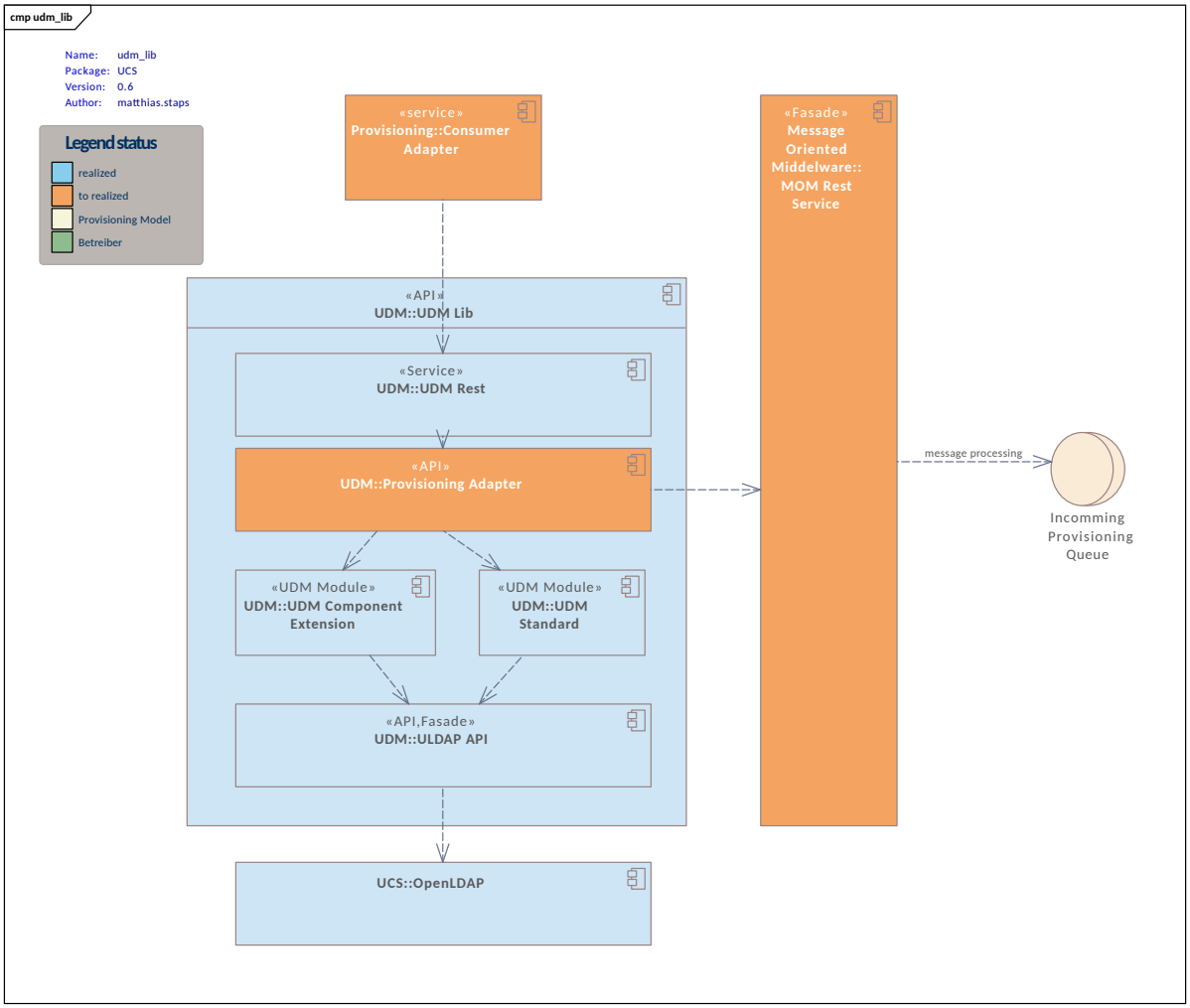

Abb. 5.10: **Figure** Integration Provisioning Adapter in UDM Lib

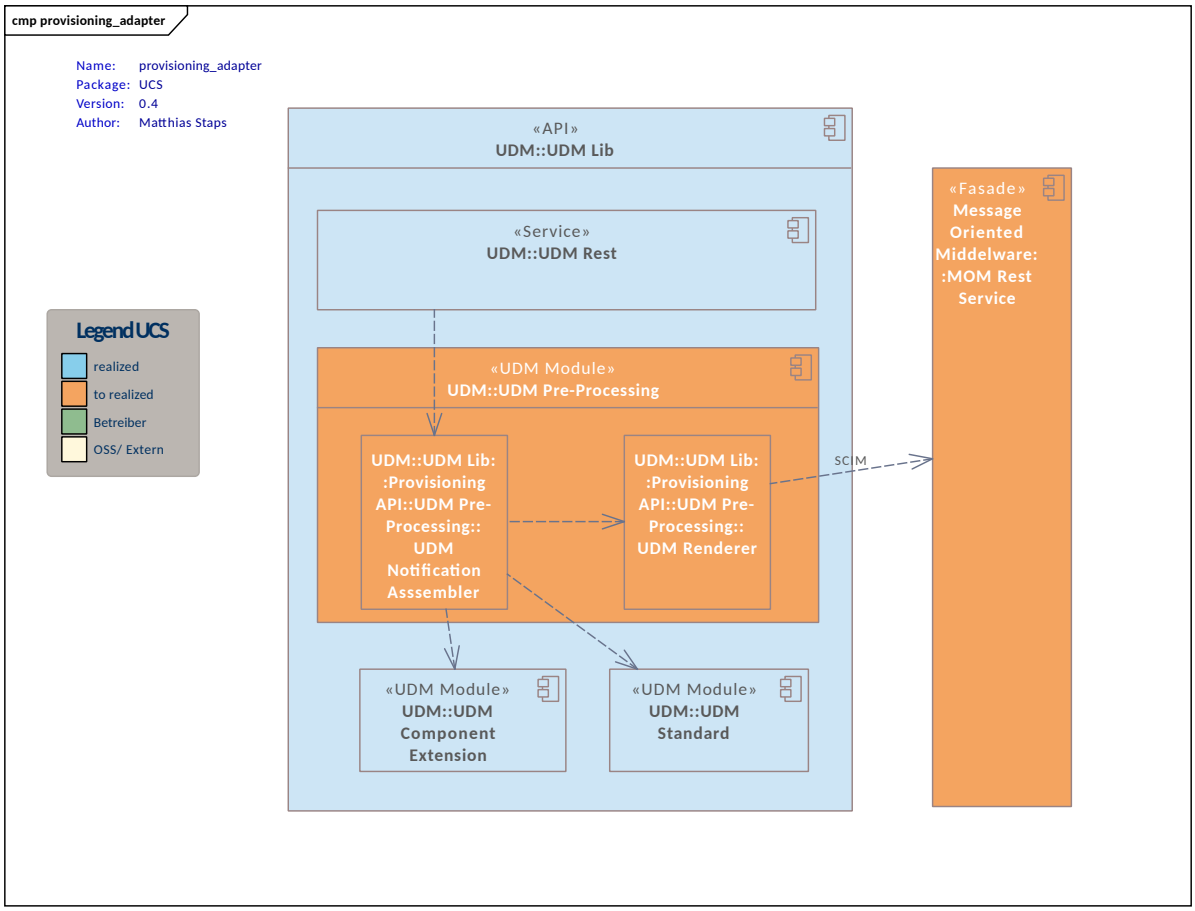

Abb. 5.11: **Figure** Provisioning Adapter

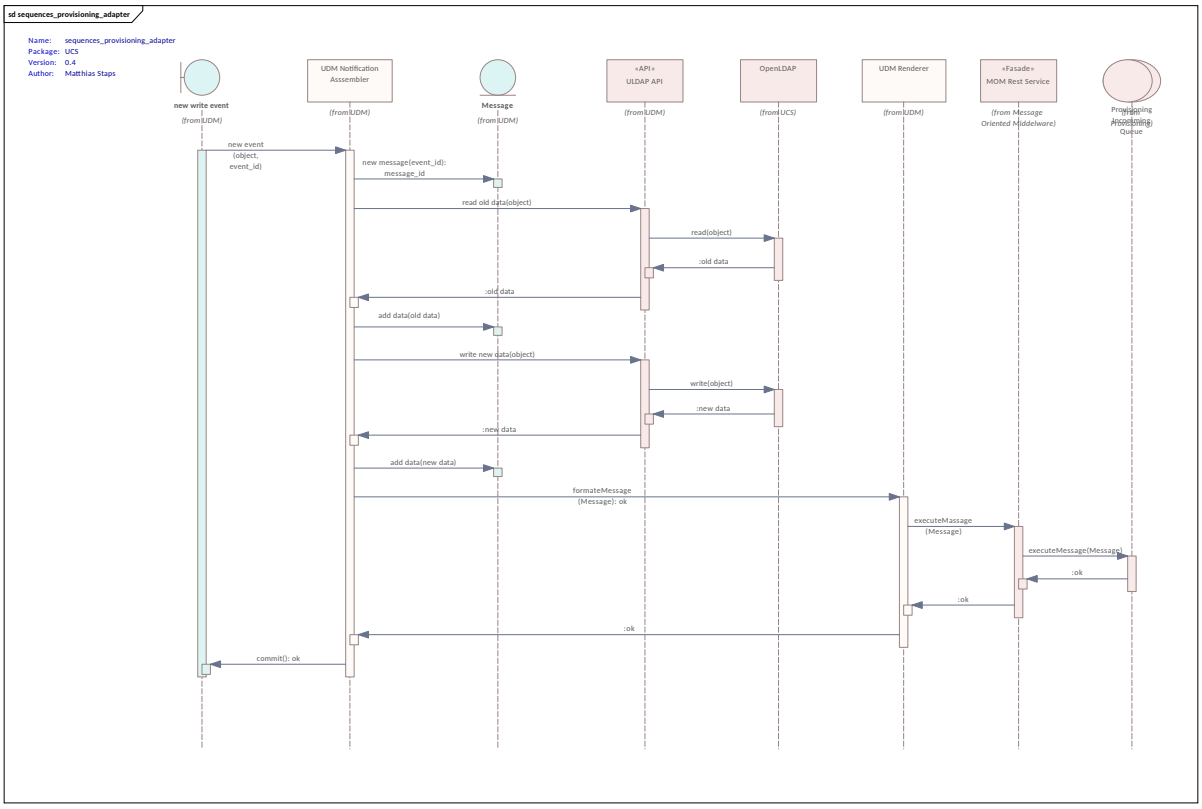

Abb. 5.12: **Figure** Ablauf Generierung Message

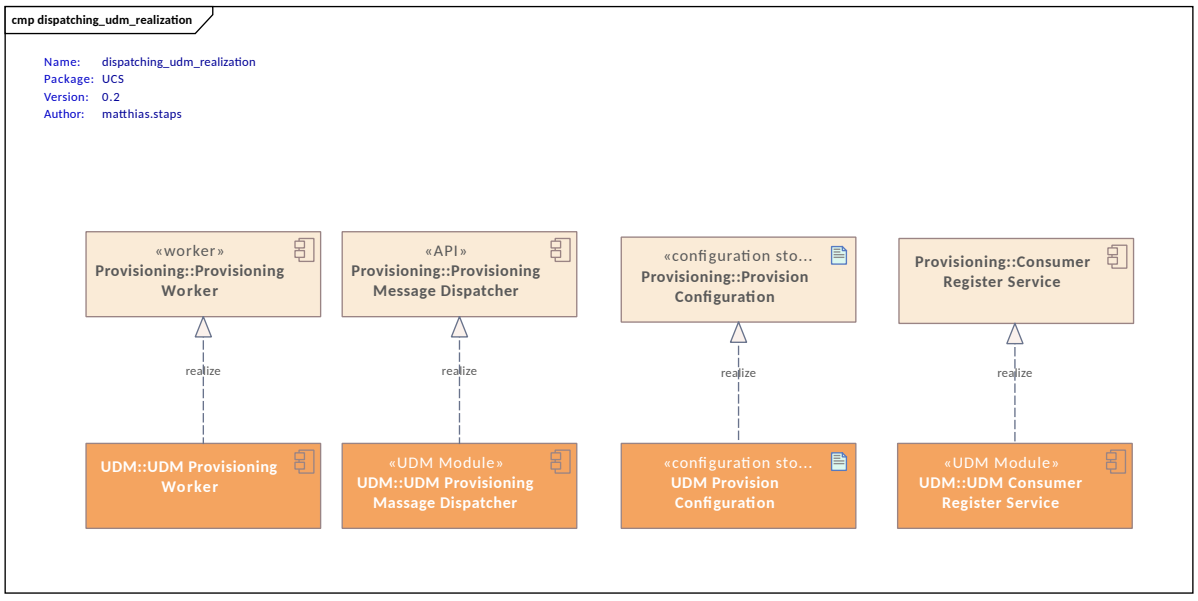

Abb. 5.13: **Figure** Umsetzung Message Dispatcher

5.5 Aktueller Stand Umsetzung

Die Planung und Umsetzung erfolgt über die **D-05-UV-104_Provisioning_infrastructure_implementation** ((see also *Ref. ID 004* (Seite 2)):) Details sind bitte der dort referenzierten Dokumentation zu entnehmen.

Zum besseren Verständnis wurden für die weitere Detaillierung Diagramme in UML Notation erstellt.

Grundlegend erfolgt die Umsetzung wie folgt:

Umsetzung in UCS/UDM

Das Design der aktuellen Implementierung stellt die Kompatibilität zum Standard UCS von Univention sicher.

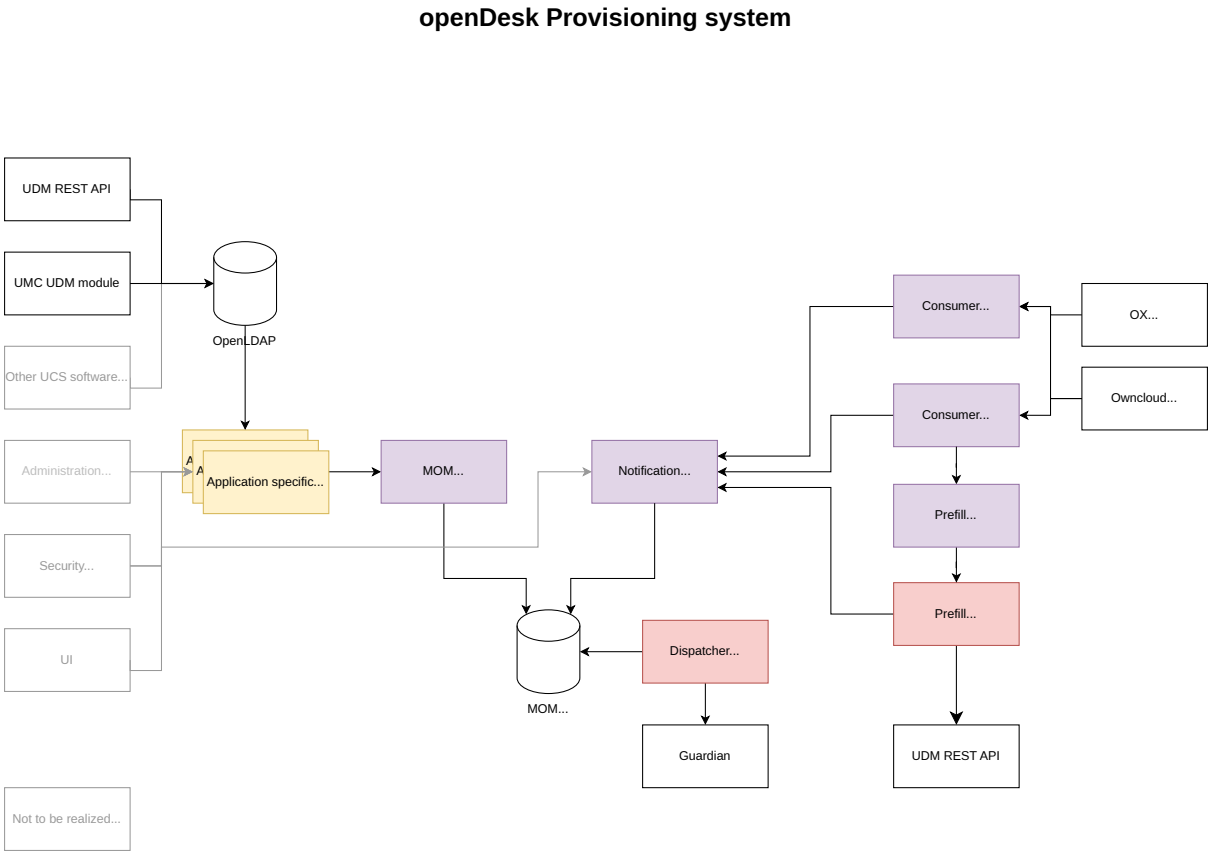

Abb. 5.14: **Figure** Aktueller Stand Provisionierung in UCS

Details Umsetzung in UCS/UDM

Umsetzung in UCS/UDM - MVP 1. Ausbaustufe

6 Kapitel 3: Deployment

Das Kapitel 3 beschreibt den aktuellen Stand der Bereitstellung des Provisioning Services.

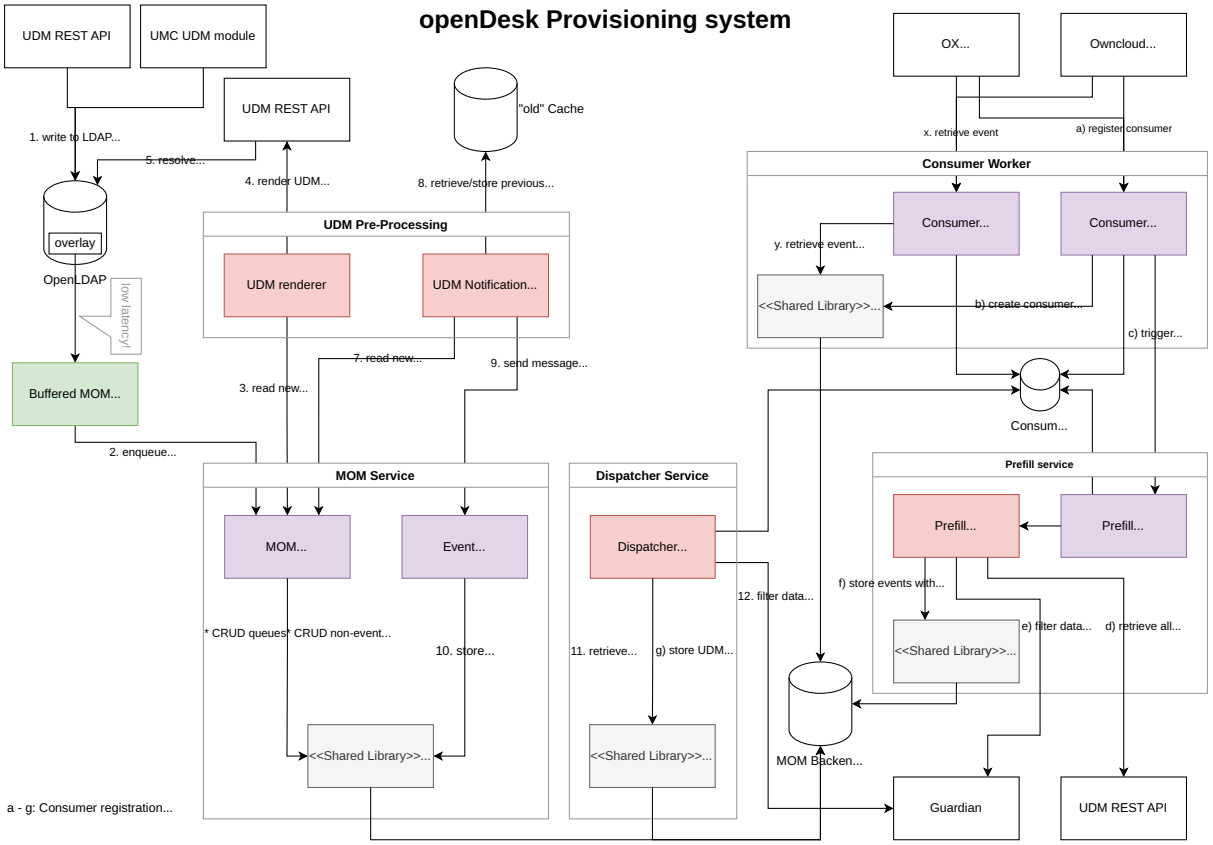

Abb. 5.15: **Figure** Provisionierung in UCS - Details

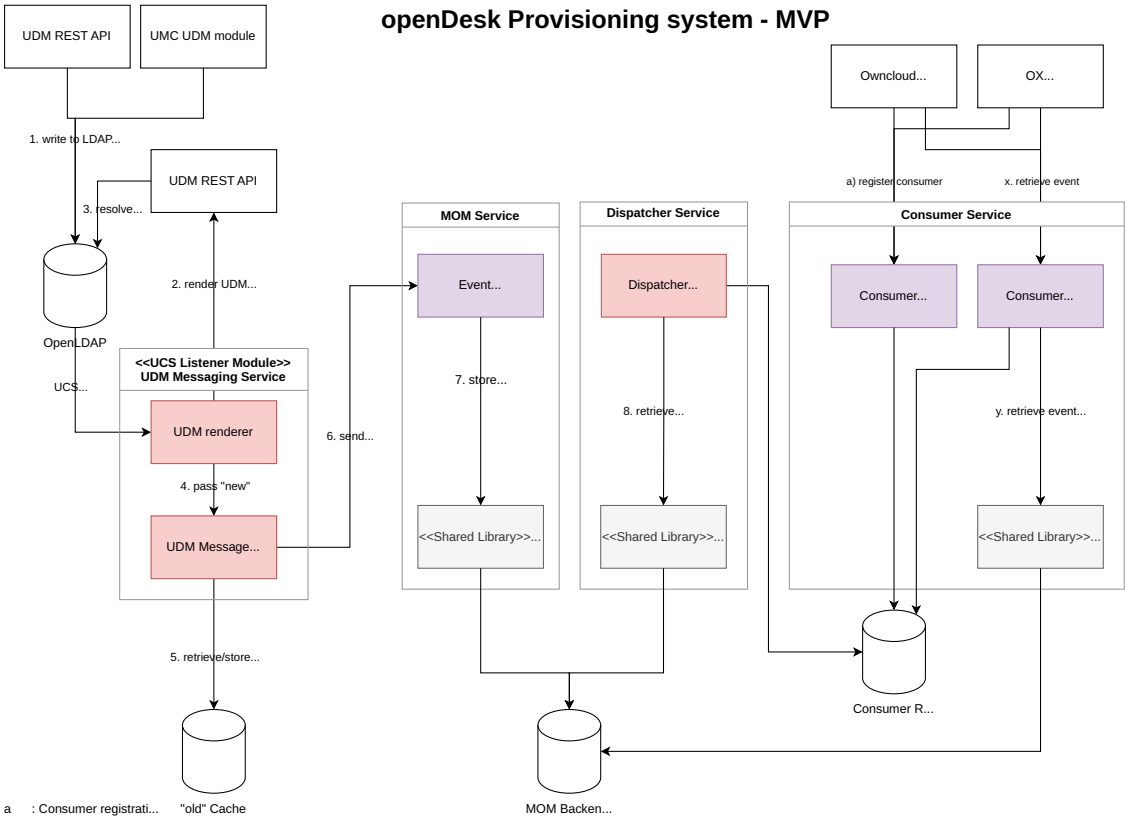

Abb. 5.16: **Figure** Provisionierung in UCS - MVP

6.1 Containerisierung

Übergreifendes Modell

Die Implementierung des Provisioning erfolgt auf der Basis des definierten **Standard Pods** für die Integration von Applikationen.

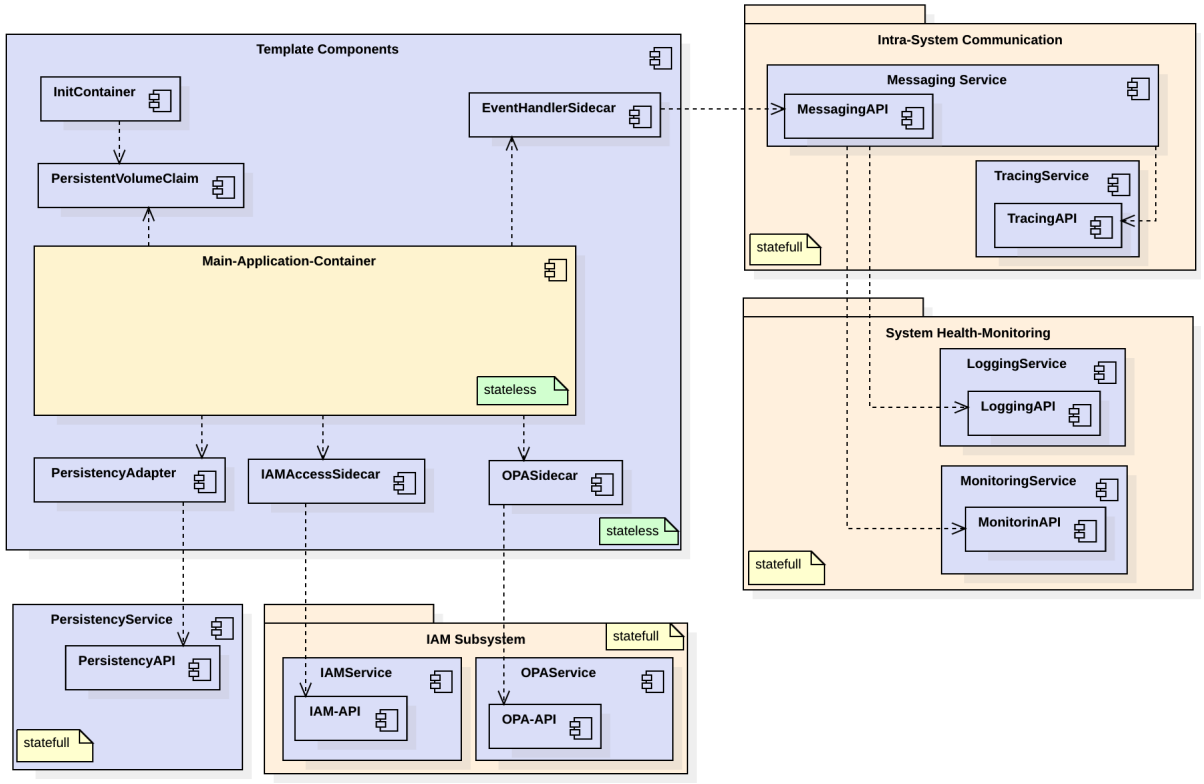

Abb. 6.1: **Figure** Standard Pod

Standard Pod für Komponenten

Die Anbindung des Provisioning Services erfolgt über einen Container, der als Sidecar bereitgestellt wird.

Init Container - UDM Init Container - Komponenten Init Container

Sidecar Container - Komponenten spezifischer UDM Sidecar Container - Provisioning Sidecar

Aktueller Stand der Containerisierung

6.2 Bereitstellung einer neuen Komponente mit Provisioning (in Arbeit)

Annahmen

1. Die Komponente benötigt eine Schema Erweiterung, da sie eigene Objekte (Assets etc.) benötigen.

Ablauf

1. **Implementierung**
    1. Modellierung und Erstellung Skript für die Schema Erweiterung
    2. Implementierung der komponentenspezifischen UDM Module

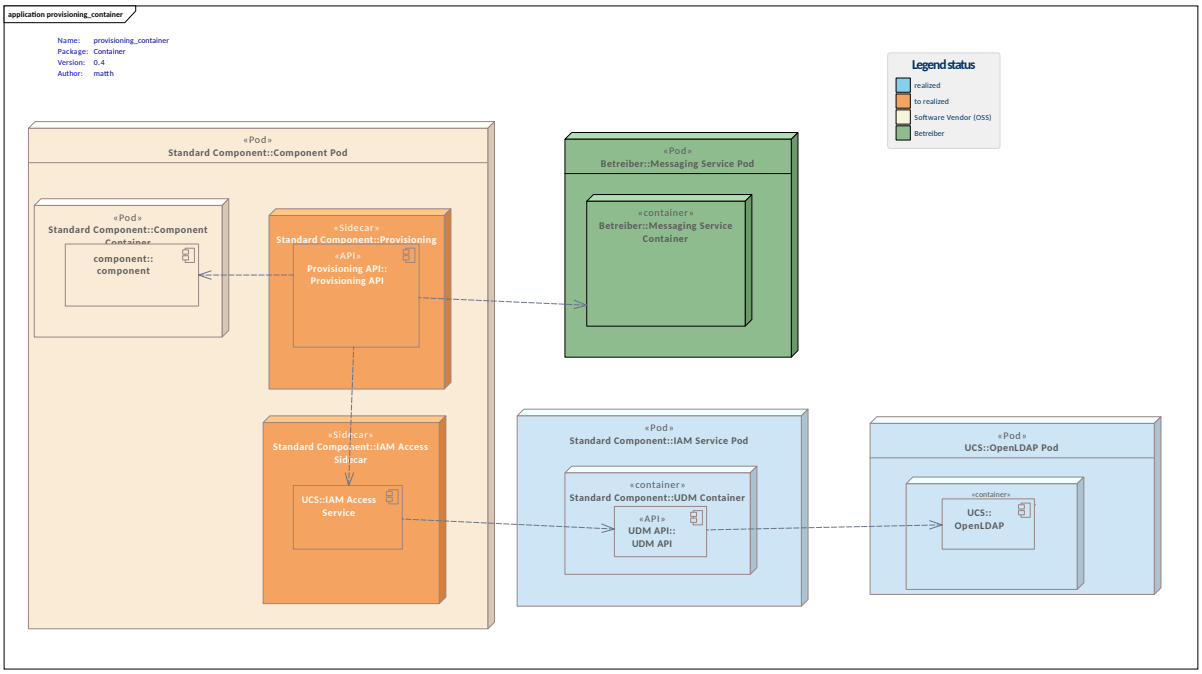

Abb. 6.2: **Figure** Standard Pod für die Anbindung Provisioning

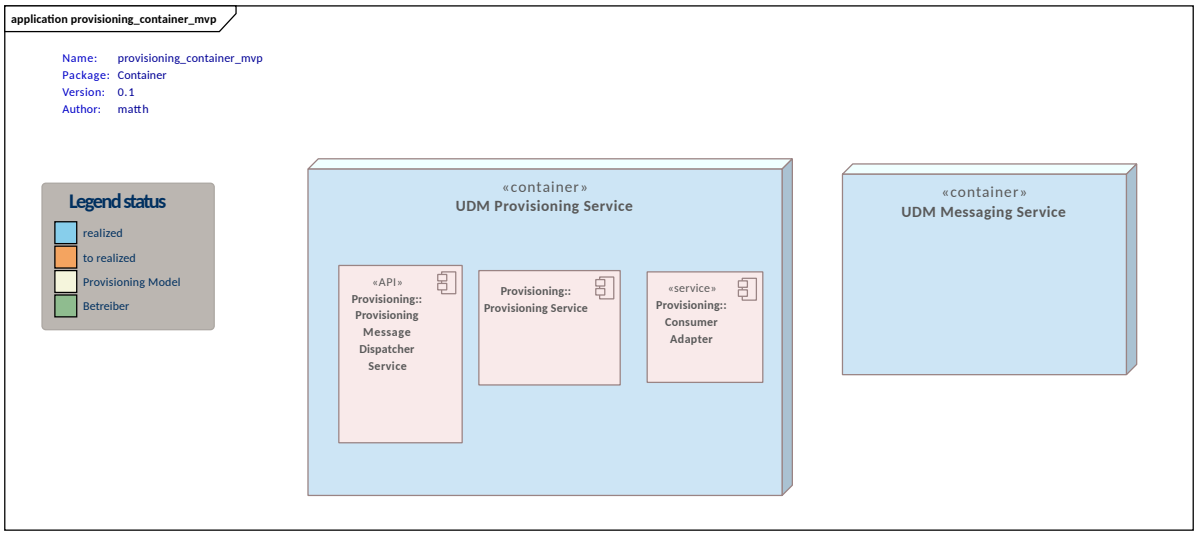

Abb. 6.3: **Figure** Container Provisioning MVP

3. Erweiterung der Provisioning API mit komponentenspezifischen Funktionen (`componenten adapter`)
4. Anpassungen Init Container für den Komponenten Pod und IAM Container

2. Erstellung UDM Sidecar mit den benötigten komponentenspezifischen UDM Modulen
3. Erstellung und Signierung UDM Sidecar
4. Erstellung Provisioning Sidecar mit der erweiterten komponentenspezifischen Provisioning API
5. Erstellung und Signierung Provisioning Sidecar
6. Anpassung HELM Charts
7. Deployment UDM Pod und Komponenten Pod

Genreller Ablauf

1. Beim Deployment einer neue Komponente mit dem `component adapter`, meldet dieser sich beim Provisioning (Dispatcher) an. Es wird einen Queue für die Komponente erstellt und konfiguriert.
2. Durch die UDM API's wird bei jedem schreibenden Zugriff auf das OpenLDAP ein Event ausgelöst und in eine Queue gestellt (UDM Queue). Der Event ist typisiert. Ein Dispatcher liest diese Queue aus und generiert für jede Komponente, die sich registriert hat, einen Event und kopiert diesen in die konfigurierte `component_queue`.
3. Der `component adapter` der Komponente liest die Events aus der Queue, interpretiert bzw. mappt diese entsprechend den implementierten Regeln und löst über die veröffentlichten Standard Services der Komponente die entsprechenden Aktionen aus.

6.3 Beispiel Anbindung OX

Exemplarische Beschreibung der Anbindung an das Provisioning am Beispiel von OX.

Komponente

Komponente mit eigene Datenhaushalt.

Message Oriented Middleware

Im **UCS-MOM-Server** ist die Queue `provisioning_communication_ox` konfiguriert. Die entsprechenden Worker zum Zugriff auf die Queue sind gestartet.

UDM API

In der UDM Rest API wird bei jedem schreibenden Zugriff auf das OpenLDAP ein Event generiert und über den Dispatcher in die Queue `provisioning_communication_ox` gestellt.

Provisioning API

Der `component_adapter provisioning_communication_ox``pollt über die Fassade die Queue ``provisioning_communication_ox` und ruft die OX spezifischen Services zur Weiterverarbeitung auf.

OX

OX mit eigene Datenhaushalt.

Beispiel Deployment OX

Im Beispiel von OX würde das Deployment wie folgt erfolgen:

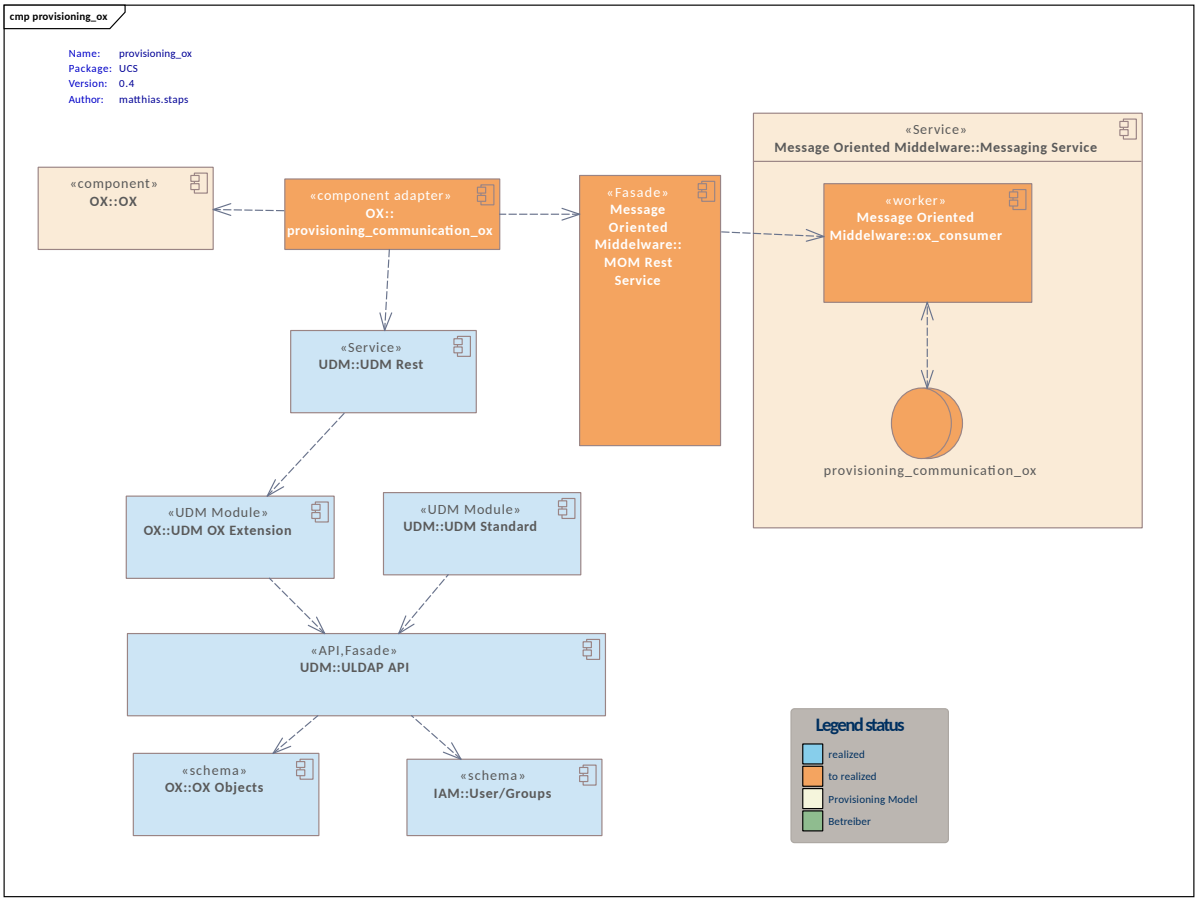

Abb. 6.4: **Figure** Komponenten Provisionierung

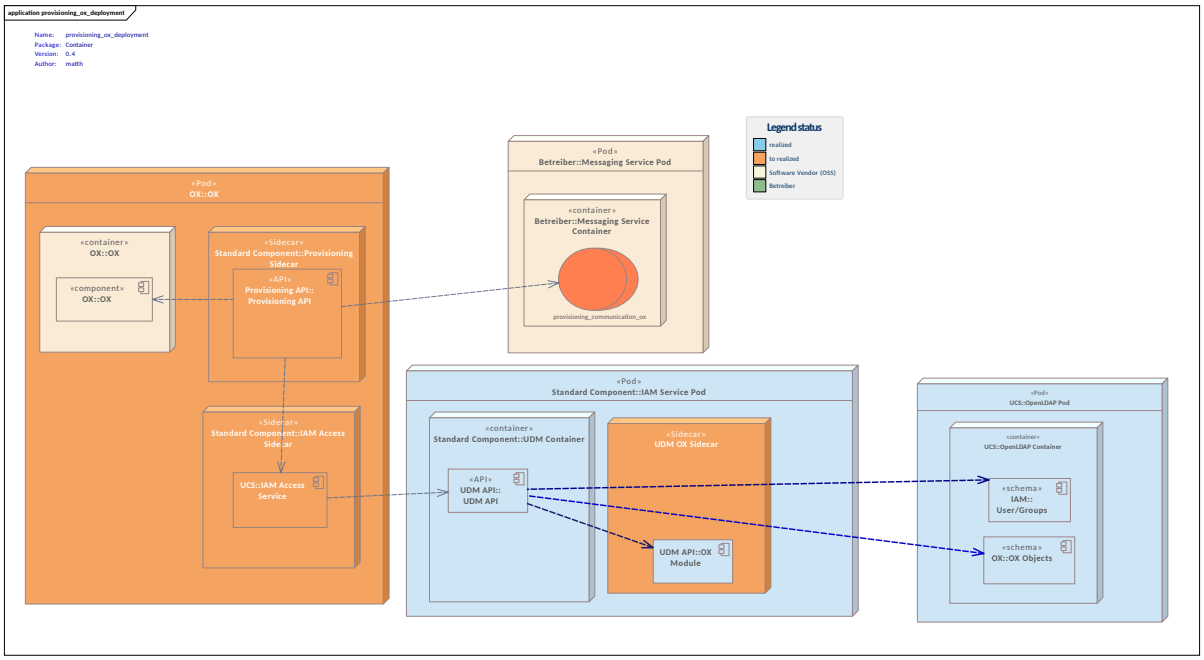

Abb. 6.5: **Figure** Deployment OX

7 Anlagen

7.1 Eingearbeitete Anmerkungen

Tab. 7.1: Eingearbeitete Anmerkungen

ID	Beschreibung	Quelle	Status
009	*Punkt 2: sollte besprochenFähigkeiten werden - Fokus in diesem Dokument ist die Anbindung von UDM und damit dessen IAM-Fähigkeiten. Nicht betrachtet wird z.B. Keycloak, und damit nicht alle Module des gesamten IAM*	Dataport Anmerkung V0.5	erledigt
010	*Punkt 3: Authentifizierung und Autorisierung am Provisioning sollen identisch zu anderen APIs erfolgen, das sollte im Dokument besser dargestellt werden. Das Dokument ist aber nicht der Ort, diese Vorgaben zu machen (da sie nicht spezifisch für das Provisioning sein sollen)*	Dataport Anmerkung V0.5	erledigt
011	*Punkt 4: gemeint ist: das Provisioning soll unabhängig von den Datenmodellen arbeiten, also auch zukünftige Änderungen verarbeiten können, Daher sind Datenmodelle nicht Teil dieses Dokuments.*	Dataport Anmerkung V0.5	erledigt, eingearbeitet in Version 0.8
012	*Punkt 5: ein direktes Schreiben in das LDAP ist grundsätzlich nicht vorgesehen, schreibende Zugriffe erfolgen immer über den UDM (sonst ist keine Datenkonsistenz gesichert)*	Dataport Anmerkung V0.5	erledigt und implementiert
013	*Relevante Behörden und Firmen: BWI sollte mit aufgenommen werden, Hessische Datenzentrale für Datenverarbeitung*	Dataport Anmerkung V0.5	erledigt in Version 0.6 aufgenommen
014	*Das Konzept macht wegen der Fokussierung des Scopes bewusst diese Annahme. Wir sollten hier explizit aufnehmen, das die Implementierung mittelfristig auch Events anderer Quellen verarbeiten können sollte.*	Dataport Anmerkung V0.5	Umgesetzt und in Version 0.8 aufgenommen
015	*Module und Services sollten in einem ``Registry/Service Catalog`` registriert sein und konfiguriert sein (Capabilities) sodass nicht für jedes neue Modul neuer Notifier erstellt oder angepasst werden muss, Ich glaube das wird eine andere Abstraktionsebene, die wir gerne als Ausblick berücksichtigen können. Voraussetzung dazu sind neben der Service Registry auch eine Standardisierung der Schnittstellen – stand heute sind die Plugins servicespezifsch (z.B. spezifisch für die OX APIs). Wenn Standards definiert sind kann es auch Standard-„Notifier“ geben die auf Basis von Einträgen einer Service Registry ihre Arbeit aufnehmen.*	Dataport Anmerkung V0.5	erledigt siehe generieren Message, Version 0.7 aufgenommen
015	*Lösungsarchitektur: Die Provisionierungs dient nicht der Protokollierung, stellt aber eine solche für die Nachvollziehbarkeit bereit.Der (Provisionierung) Zustand der Identität über den gesamten SouvAP kann aus der Schicht ausgelesen werden. Das sind zwei Punkte, richtig? Zur Protokollierung: Ich glaube wir meinen das Gleiche. Der Provisioning Stack selber muss auditierbar sein, d.H. er stellt z.B. Logmeldungen mit entsprechendem Informationsumfang bereit.Zum Zustand: Ich würde das als „operatives Monitoring“ verstehen, d.H. der Betreiber kann erkennen welche Events zu einer Identität verarbeitet werden müssen. Wir sollten das im Konzept explizit machen, aber auch hier schauen ob das realistisch in einer ersten Version umgesetzt werden kann.*	Dataport Anmerkung V0.5	erledigt, nicht im Scope, da Aufgabe des Betreibers
016	*Abrufen von Events aus der Queue als Transaktion beschreiben*	Univention DEV 18.07.23	erledigt in Version 0.7

7.2 Relevante ADR's Univention

ADR_001 Message Queue Backend Evaluation: RabbitMQ, Kafka, Redis

- **status:** proposed
- **date:** 2023-10-16
- **author:** acaceres
- **deciders:** tkintscher, skoenig, jlohmer
- **source:** [https://git.knut.univention.de/univention/customers/dataport/upx/provisioning-api/-/issues/15{]}(https://git.knut.univention.de/univention/customers/dataport/upx/provisioning-api/-/issues/15)

Context and Problem Statement

We are developing a publish-subscribe backend system where one party notifies us about changes, and other parties expect to receive a notification when this happens. The challenge is to select a suitable message queue system that can handle this communication efficiently, ensuring scalability, reliability, and ease of management.

Decision Drivers

Update 23.10: right now we are choosing between NATS and RabbitMQ Main drivers for a decision are specified in [SPIKE: Backend evaluation for message queue](https://git.knut.univention.de/univention/customers/dataport/upx/provisioning-api/-/issues/15), and some came out of the discussion with the team.

Evaluation criteria:

1. can be clustered
2. provides persistence: messages are not lost when the broker is restarted
3. manageable (provides monitoring)
4. has a suitable API: Python bindings, docs, etc
5. provides features that otherwise would need to be implemented
6. resource consumption: how hungry for memory/CPU, if needs to keep everything in memory or can use disk
7. performance: throughput and latency considerations
8. developer experience: familiarity of the team/department with the technology
9. support: availability of documentation and community resources
10. encryption: ensures secure transmission and storage of data.
11. authentication: ensures that only authorized parties can publish or subscribe to messages.

Considered Options

- Kafka
- RabbitMQ
- Redis
- [NATS](https://nats.io/)
- [Pulsar](https://pulsar.apache.org/)
-

Pros and Cons of the Options

Kafka

Kafka is a distributed streaming platform designed for high throughput.

- Good, because of its performance and scalability (1, 7)
- Good, because it provides strong durability with message persistence (2)
- Good, because it allows different usage scenarios, like not-deleting-messages and rereading them for analysis or deduplication. Thus allowing the backend to be used in other projects. (5)
- Good, because a wide range of professional tooling exists to maintain and monitor it. (3)
- Good, because can be integrated with Open Policy Agent ([OPA docs](https://www.openpolicyagent.org/docs/latest/kafka-authorization/)). (11)
- Neutral, because the official Python client does not integrate with asyncio but a 3rd party lib *aiokafka* does. (4)
- Bad, because compared to other solutions has higher resource consumption, as it runs in a JVM. (Though not _that_ bad: I can run it incl. zookeeper with no problems on my notebook.) (6)
- Bad, because it either requires an additional component (Zookeeper) to run, or uses KRaft protocol, which is in early stage. (8)
- Bad, because of its complexity in setup and management. (3, 8)

Example Python code for publishing a message:

```
producer = KafkaProducer(bootstrap_servers=['host:1234'])
producer.send('provisioning-channel', key=b'key', value=b'value')
producer.flush()``
```

RabbitMQ

RabbitMQ is a message broker that supports multiple messaging protocols.

- Good, because of its ease of setup and management. (3, 8)
- Good, because it supports a variety of messaging patterns, including pub/sub. (5)
- Good, because of low resource consumption. (6)
- Good, because can be integrated with Open Policy Agent (RabbitMQ blog). (11)
- Neutral, because while it can handle a significant number of messages, it does not match Kafka's performance on max load. (1, 7)
- Neutral, because official Python client despite being asynchronous is very callback-oriented. However, a 3rd party libs aio-pika and aio-rabbit integrates with asyncio. (4)
- Bad, because horizontal scaling is not as straightforward as Kafka. (1)
- Bad, because of comparatively rather basic management and monitoring tools. (3)

Encryption (10): RabbitMQ supports TLS/SSL for encrypted connections. It allows for configuring the required level of encryption and only accepting connections that meet these requirements. Additionally, RabbitMQ supports encrypting data using disk encryption methods provided by the operating system.

Authentication (11): RabbitMQ many authentication and authorization mechanisms. It supports many authentication backends, including LDAP, allows to define fine-grained permissions for users, including specifying which operations a user can perform on queues and exchanges. RabbitMQ also supports SASL for secure authentication.

Example Python code for publishing a message:

```
connection = pika.BlockingConnection(pika.ConnectionParameters('host:1234'))
channel = connection.channel()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
channel.basic_publish(exchange='', routing_key='provisioning-channel', body='value
↪')
connection.close()``
```

Redis Redis is an in-memory data structure store that supports various data structures and has a pub/sub capability.

- Good, because every developer knows it (8)
- Good, because it's easy to set up (3)
- Good, because it can be a DB for other information, not just pub/sub messages (5)
- Bad, because it's primarily an in-memory datastore, leading to potential data loss if not persisted properly, and requiring proportional RAM growth (2, 6)

Example Python code for publishing a message:

```
r = redis.Redis(host='host', port=6379, db=0)
r.publish('provisioning-channel', 'value')``
```

NATS

NATS is a lightweight and high-performance messaging system.

- Good, because it's lightweight and fast, providing high throughput and low latency (7)
- Good, because it is easy to set up and manage (3)
- Good, because official python client supports AsynCIO (4)
- Good, because it supports various messaging patterns, including pub/sub, request/reply, and point-to-point. (5)
- Neutral, because while it supports clustering, it might not scale as horizontally as Kafka. (1)
- Bad, because it might not provide as extensive monitoring and management tools out of the box (not yet evaluated) (3)

Encryption (10): NATS supports TLS for encrypted connections. Also NATS provides the option for end-to-end encryption, ensuring that messages are encrypted during transit and only decrypted by the intended recipient.

Authentication (11): NATS provides many authentication mechanisms, including token-based authentication, username and password, and decentralized JWT-based authentication. This flexibility allows administrators to choose the authentication method that best suits their security requirements. Additionally, NATS supports Authorization, which allows defining permissions at a fine-grained level, specifying what subjects (topics) a user can publish or subscribe to.

Example Python code for publishing a message using the nats client:

```
import asyncio
import nats.aio.client
    async def run():``
    nc = nats.aio.client.Client()
    await nc.connect("nats://localhost:4222")
    await nc.publish("provisioning-channel", b'value')
    await nc.close()``

loop = asyncio.get_event_loop()
loop.run_until_complete(run())``
```

Decision Outcome

Chosen option:

To be determined based on further evaluation and specific project needs, because both Kafka and RabbitMQ have their strengths and are capable of meeting the requirements. Redis, while a powerful tool for certain use cases, may not be suitable for our needs due to its in-memory nature and potential data persistence challenges.

Risks

As listed MQ backends are totally different in nature, have different APIs and design concepts, it will be difficult to migrate to another backend later on.