Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät

Professur für Open-Source-Software

Projekt im Fach Softwarearchitektur

Software-Architektur-Projekt – DATEV Cloud Migration

von

Patrick Kaltenmaier, Gona Fatah, Violetta Tabirta, Vladimir Vulpe, Fabian Weilbrenner

16.07.2015

Erstprüfer
Prof. Dr. Dirk Riehle, M.B.A.

Zweitprüfer
Dr.-Ing. Martin Jung

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung und Ziele

Dieser Projektbericht dient der Dokumentation des Masterprojektes „AMOS Cloud Layer-Architektur“ an der Friedrich-Alexander Universität Erlangen-Nürnberg im Masterstudiengang Informatik im Sommersemester 2015.

> *"Wie erfolgreich wäre wohl ein Hausbau ohne die Planung von Statik, Grundriss und Haustechnik?!"*

Was im Bauwesen und der klassischen Bauarchitektur gang und gäbe ist, wurde in der Softwareentwicklung jahrelang vernachlässigt. Die „Strukturplanung einer Software“, getreu dem Motto:

- Brauchen wir nicht!
- Keine Zeit dafür, wir stehen unter Termindruck!
- Alle Projektbeteiligten kennen die Struktur des Systems, daher unnötig!

Doch immer höhere und umfangreichere Kundenanforderungen machen Softwaresysteme zunehmend komplexer. Die Systeme bleiben oftmals viele Jahre im Einsatz, werden kontinuierlich weiterentwickelt, das Personal kommt und geht und die Systemumgebung untersteht einem ständigen Wandel.

Um das tragische Ende einer Software zu verhindern, sie langfristig bezahlbar, wartbar, flexibel und verständlich zu konstruieren und um auf Risiken möglichst gut vorbereitet zu sein, ist die Strukturplanung einer Software, also ihrer Architektur, unablässig.

Im Rahmen einer Zusammenarbeit mit dem Unternehmen DATEV eG und einer weiteren Projektgruppe (AMOS-Team1) soll die Architektur einer Anwendung (Brutto-Netto Rechner inkl. Steuerklassen-Übersicht und Vergleich), die auf den drei u.g. Clouds laufen, erstellt werden:

- Google Cloud
- Amazon Webservices
- Microsoft Azure

Realisiert wird das Projekt mit der sog. **„AMOS Cloud Layer“**, eine einfach zu bedienende Web-Oberfläche, die als gemeinsame Schnittstelle für die Clouds dient. Aus diesem Layer werden auf die Datei-Speichersysteme von Amazon S3, Azure Storage und Google Cloud Storage zugegriffen. Dabei werden Daten, die über die AMOS Cloud Layer gespeichert werden, redundant auf allen drei Clouds gespeichert, um eine einfache Migration bzw. eine Wiederherstellung der Daten im Fehlerfall zu ermöglichen. Der Zugriff auf

die AMOS Cloud Layer wird anhand eines Benutzer-Authentifizierungssystems ermöglicht, welches eine Vielzahl an SSO-Diensten unterstützt. Hierdurch wird eine einfache Integration in bestehende Systeme erleichtert und die Datensicherheit gewährleistet.

Im Verlauf dieses Berichts werden vorab die Randbedingungen des Projektes (Kap. 2) beschrieben. Im Weiteren erfolgt die Architekturbeschreibung, die sich an dem 4+1 Sichtenmodell von Philippe Kruchten orientiert und UML als Modellierungsnotation nutzt. In der Architekturbeschreibung werden zunächst die Szenarien (Kap. 3), die sich an den gestellten Anforderungen im Product Backlog des AMOS-Teams anlehnen, erstellt. Diese Anforderungen bilden die Basis für die Entwicklung der externen und internen Szenarien im Projekt. Zur Vervollständigung der Architekturbeschreibung werden im nächsten Schritt die 4 weiteren Architektursichten beschrieben (Kap. 4). Hierbei handelt es sich um die logische-, dynamische-, Realisierungs- und Verteilungssicht. Während in der logischen Sicht der funktionale Aufbau des Systems anhand von Komponenten visualisiert wird, wird in der dynamischen Sicht dynamische Zusammenhänge des System während der Laufzeit aufgezeigt. Die Realisierungssicht beschäftigt sich mit den Entwicklungsergebnissen, die am Ende eines Implementierungsschritts resultieren und die Verteilungssicht zeigt welche Komponenten auf welchen physikalischen Knoten ausgeführt werden und wie diese miteinander verbunden sind.

Die beiden nachfolgenden Kapitel beschäftigen sich zum einen mit dem technischen Konzept (Kap. 5), in dem die Technologien vorgestellt werden, die für die Implementierung des Systems verwendet wurden, zum anderen mit der konkreten Vorgehensweise des Amos-Teams bei der Umsetzung des Projekts, in dem im Rahmen eines Interviews der Product Owner zur Projektkonkretisierung befragt wurde (Kap. 6).

Abschließend erfolgt ein weiterer relevanter Bereich der Softwarearchitektur „Metriken“ (Kap. 7). Zur Qualitätsbewertung ist es von großer Bedeutung Metriken festzulegen. Hierfür wurden Metriken nach dem Goal-Question-Metric-Verfahren aufgestellt und anhand des AMOS Cloud Layer´s gemessen.

Des Weiteren sollte besonders bei Anwendungen, die über ein/mehrere Clouds bereitgestellt werden, ein reibungsloser Betrieb sichergestellt werden. Ebenfalls darf eine angemessene Performance nicht außer Acht gelassen werden, selbst wenn die Anzahl der Benutzer, somit die Anzahl der Anfragen und die dabei entstehende Last nach oben getrieben werden. Um diese Qualitätsmerkmale anhand der Amos Cloud Layer bewerten zu können, wurden sog. Lasttests durchgeführt und analysiert (Kap. 7).

2 Randbedingungen

Zur Anwendung und Realisierung des Projekts wurde vom Auftraggeber Datev eG ein „Brutto-Netto Rechner“ zur Verfügung gestellt. Hierbei handelt es sich um eine vollständig in Java Script entwickelte, clientseitig ausgeführte Webapplikation. Des Weiteren wurden für die Migration der Anwendung in die Cloud folgende Rahmenbedingungen festgelegt:

Die bestehende Java Script-Logik bleibt weitestgehend unangetastet. Diese Einschränkung stellt die Wirtschaftlichkeit des Prototyps und letztlich die Relevanz der Studie sicher. Die hier gewonnenen Ergebnisse dienen als Machbarkeitsstudie und sollen bei der Aufwandsschätzung der Cloud-Migration weiterer Anwendungen helfen. Aus diesem Grund werden im Verlauf des Projekts ausschließlich Funktionalitäten hinzugefügt, die dabei helfen sollen, Erfahrungen in Bezug auf Migration von Anwendungen, in diesem Fall fehlender serverseitiger Programmlogik, zu sammeln und unterschiedliche Eigenschaften der Cloud-Umgebungen herauszuarbeiten.

Der komplette serverseitig eingesetzte Technologie- Stack muss, in vollem Funktionsumfang, mit der Google-, Amazon- und Microsoft Cloudumgebung kompatibel sein. Durch die Kompatibilität wird ein Vendor-Lock in einer der drei Cloudanbieter verhindert. Das bedeutet, dass jeder Zeit auf mögliche Preisänderungen oder Verfügbarkeitsproblemen eines Anbieters reagiert werden kann, ohne dabei Anpassungen am Source Code vornehmen zu müssen.

Es dürfen ausschließlich Open Source Komponenten, deren Lizenzmodell eine kommerzielle Nutzung erlaubt, verwendet werden. Dies ermöglicht die uneingeschränkte Nutzung von Quellcode ohne das Anfallen von zusätzlichen Lizenzgebühren. Gleichzeitig steigt die Relevanz der gewonnenen Ergebnisse, da keine Abhängigkeiten zu ClosedSource-Komponenten bestehen, deren interne Funktionsweise für die Außenwelt verborgen bleibt.

3 Szenarien

3.1 Externe Szenarien

3.1.1 User Management

UC1 - User registrieren

Beschreibung: Ein User hat die Möglichkeit sich in der WebApp zu registrieren

Beteiligte Aktoren: User, System

Vorbedingung: User ist nicht registriert

Auslöser: User möchte die WebApp nutzen und sich dafür registrieren

Nachbedingung: User ist registriert

Normaler Ablauf:

1. User öffnet die Webseite
2. User betätigt den Registrierung-Button
3. User gibt einen Usernamen ein
4. User gibt ein Passwort ein
5. User betätigt den Registrieren-Button
6. Das System legt einen neuen User mit den angegebenen Daten an

UC2 - Single Sign On (SSO)

Beschreibung: Ein User hat die Möglichkeit sich mittels eines Social-Media-Accounts (Google, Facebook) registrieren

Beteiligte Aktoren: User, System

Vorbedingung: User ist nicht registriert

Auslöser: User möchte sich registrieren, um den Brutto-Netto-Rechner zu verwenden, sich aber nicht noch einen zusätzlichen Online-Account anlegen möchte

Nachbedingung: User ist mit dem angegebenen Account registriert

Normaler Ablauf:

1. User öffnet die Webseite
2. User drückt den Registrieren-Öffnen-Button

3. User gibt seinen Social-Media-Account an, mit dem er sich registrieren möchte
4. Das System legt einen neuen User an und verknüpft diesen mit dem Social-Media-Account

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 3: Social-Media-Account konnte nicht gefunden werden*
  Das System teilt dem User mit, dass sein angegebener Account ungültig ist
- *In Schritt 4: E-Mail-Adresse des Accounts ist bereits bei einem angelegten User vorhanden*
  Das System verknüpft den angegebenen Account mit dem bereits existierenden User

UC3 - User einloggen

Beschreibung: Ein User hat die Möglichkeit sich mittels Usernamen und Passwort anzumelden

Beteiligte Aktoren: User, System

Vorbedingung: User ist registriert und nicht eingeloggt

Auslöser: User möchte den Brutto-Netto-Rechner verwenden

Nachbedingung: User ist eingeloggt

Normaler Ablauf:

1. User öffnet die Webseite
2. User gibt seinen Usernamen ein
3. User gibt sein Passwort ein
4. User drück den Einloggen-Button
5. Das System überprüft die Kombination von Usernamen und Passwort und loggt den User ein
6. User ist eingeloggt

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 5: Kombination aus Usernamen und Passwort ist dem System nicht bekannt:*
  Das System teilt dem User mit, dass seine eingegeben Daten falsch sind

UC4 - User ausloggen

Beschreibung: Ein User hat die Möglichkeit sich auszuloggen

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt

Auslöser: User möchte sich ausloggen, um die aktuelle Sitzung zu beenden.

Nachbedingung: User ist ausgeloggt

Normaler Ablauf:

1. User drückt den Ausloggen-Button
2. System loggt den User aus und beendet somit die aktuelle Sitzung

3.1.2 Dateiverwaltung

UC5 - Nettoeinkommen berechnen

Beschreibung: Ein User hat die Möglichkeit sein Nettoeinkommen zu berechnen.

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt

Auslöser: User möchte ein Nettoeinkommen berechnen

Nachbedingung: Das gewünschte Nettoeinkommen des Users wurde berechnet

Normaler Ablauf:

1. User drückt den Neu-Button, um eine neue Rechnung zu erstellen
2. User gibt die relevanten Daten für die Berechnung ein
3. Resultierend aus den eingegebenen Daten, wird die gewünschte Rechnung erstellt
4. User kann die erstellte Rechnung speichern

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 3: User füllt die Pflichtfelder unvollständig bzw. nicht dem festgelegten Format entsprechend aus:*
  Das System teilt dem User mit, dass seine eingegeben Daten unvollständig bzw. falsch sind

UC6 - Dateien-Management

Beschreibung: Ein User hat die Möglichkeit seine Dateien im Web-Interface zu managen (Erstellen, Öffnen, Lesen, Bearbeiten (Schreiben), Löschen)

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt, User hat min. eine hochgeladene Datei (falls eine bereits existierende Datei bearbeitet werden soll)

Auslöser: User möchte eine Dateien im Web-Interface einsehen bzw. bearbeiten

Nachbedingung: User kann die gewünschte Datei bearbeiten, öffnen, bearbeiten und anschließend schließen

Normaler Ablauf:

1. User wählt aus der Liste seine hoch-geladenen Dateien eine Datei aus
2. User führt die gewünschten Operation aus
3. User speichert evtl. Änderungen ab

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 1: User hat noch keine Datei erstellt:*
  UC5 oder einen fuer den User freigegebene Datei oeffen (UC11)
- *In Schritt 2: User A hat die aktuelle Datei gelöscht:*
  Schritt 3 entfällt

UC7 - Berechnungen speichern

Beschreibung: Ein User hat die Möglichkeit seine Berechnungen zu speichern

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt, User hat ein Nettoeinkommen berechnet

Auslöser: User möchte eine erstellte Rechnung speichern, um sie zu einem späteren Zeitpunkt wieder zu verwenden

Nachbedingung: Die erstellte Berechnung ist gespeichert

Normaler Ablauf:

1. User klickt auf den Datei-speichern-Button
2. User vergibt einen Dateinamen unter dem die Rechnung gespeichert werden soll

3. User bestätigt den Vorgang (Speichern OK)

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 3, 4: Dateiname existiert bereits:*
  Das System teilt dem User mit, dass eine Datei mit demselben Namen bereits existiert. Der User kann einen neuen Dateinamen eingegeben oder die bereits existierende Datei ersetzen.

UC8 - Berechnungen wiederherstellen

Beschreibung: Ein User hat die Möglichkeit seine Berechnungen wiederherzustellen

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt, User hat min. eine Berechnung gespeichert

Auslöser: User möchte eine erstellte Rechnung speichern, um sie zu einem späteren Zeitpunkt wieder zu verwenden

Nachbedingung: Die gespeicherte Berechnung ist geöffnet und die aus eine vorherigen Sitzung eingegebenen Daten des Users sind vorhanden

Normaler Ablauf:

1. User klickt auf den Öffnen-Button
2. User wählt aus der Liste seine hoch-geladenen Dateien eine Datei aus
3. Das System öffnet die ausgewählte Datei
4. User führt die gewünschten Operation aus (lesen, bearbeiten)

UC9 - Übersicht über gezahlte Steuern

Beschreibung: Ein User hat die Möglichkeit eine detaillierte Übersicht[1] über alle gesetzlichen Steuerabzüge einzusehen

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt

Auslöser: User möchte eine detaillierte Übersicht[1] über gesetzliche Steuerabzüge haben

Nachbedingung: User kann die Übersicht[1] über alle gesetzlichen Steuerabzüge einsehen

[1] Die Übersicht beinhaltet Steuername, Erklärung und der zu zahlende Prozentsatz

Normaler Ablauf:

1. User klickt auf ein Button, um die Übersicht einzusehen
2. Liste aller gesetzlichen Steuerabzüge wird angezeigt

UC10 - Steuerklassen-Vergleich

Beschreibung: Ein User hat die Möglichkeit verschiedene Steuerklassen-Kombinationen zu vergleichen

Beteiligte Aktoren: User, System

Vorbedingung: User ist eingeloggt

Auslöser: User möchte für seinen Lebenspartner und sich verschiedene Steuerklassen-Kombinationen vergleichen, um eine geeignete Kombination zu finden, die die geringsten Steuerabzüge zur Folge haben.

Nachbedingung: Eine Vergleichsberechnung wurde erstellt

Normaler Ablauf:

1. User wählt (seine) Steuerklasse X
2. User wählt (seines Partners) Steuerklasse Y
3. Aus den beiden Steuerklassen werden die Steuerabzüge berechnet

UC11 - File Sharing

Beschreibung: Ein User hat die Möglichkeit seine hochgeladenen Dateien mit einem anderen User zu teilen

Beteiligte Aktoren: User A, User B, System

Vorbedingung: User A ist eingeloggt, User A hat min. eine hochgeladene Datei

Auslöser: User A möchte eine Datei mit User B teilen

Nachbedingung: User B kann die Datei von User A einsehen

Normaler Ablauf:

1. User A wählt eine Datei aus
2. User A drückt den Teilen-Button
3. User A wählt User B als Empfänger aus

3.2 Interne Szenarien

UC12 - Speichern der Userdaten

Beschreibung: Das System legt die Userdaten auf der Cloud-lokalen Datenbank ab

Beteiligte Aktoren: User, System

Vorbedingung: User hat sich registriert

Auslöser: User hat sich registriert, um den Brutto-Netto-Rechner zu verwenden

Nachbedingung: Userdaten sind gespeichert

Normaler Ablauf:

1. User klickt den Registrieren-Button
2. Das System speichert die angegebene Userdaten in der Datenbank ab

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 2: Speichervorgang schlägt fehl:*
  Das System teilt dem User mit, dass die Registrierung fehlgeschlagen ist.

UC13 - Repliziertes Speichern der Dateien

Beschreibung: Das System legt die Dateien repliziert auf allen zur Verfügung stehenden Cloud-Plattformen (AWS, Google, Azure) ab

Beteiligte Aktoren: User, ACL-Client, System

Vorbedingung: User hat eine Berechnung gespeichert

Auslöser: User möchte eine Berechnung zur späteren Wiederverwendung speichern

Nachbedingung: Datei ist repliziert auf allen Cloud-Plattformen gespeichert

Normaler Ablauf:

1. User klickt auf den Speichern Button (nach UC7)
2. Der ACL-Client verteilt die Datei redundant auf den Cloud-Plattformen

Alternativen oder Ausnahmenbehandlung:

- *In Schritt 2: Speichervorgang schlägt fehl:*
  Das System teilt dem User mit, dass der Speichervorgang misslungen ist.

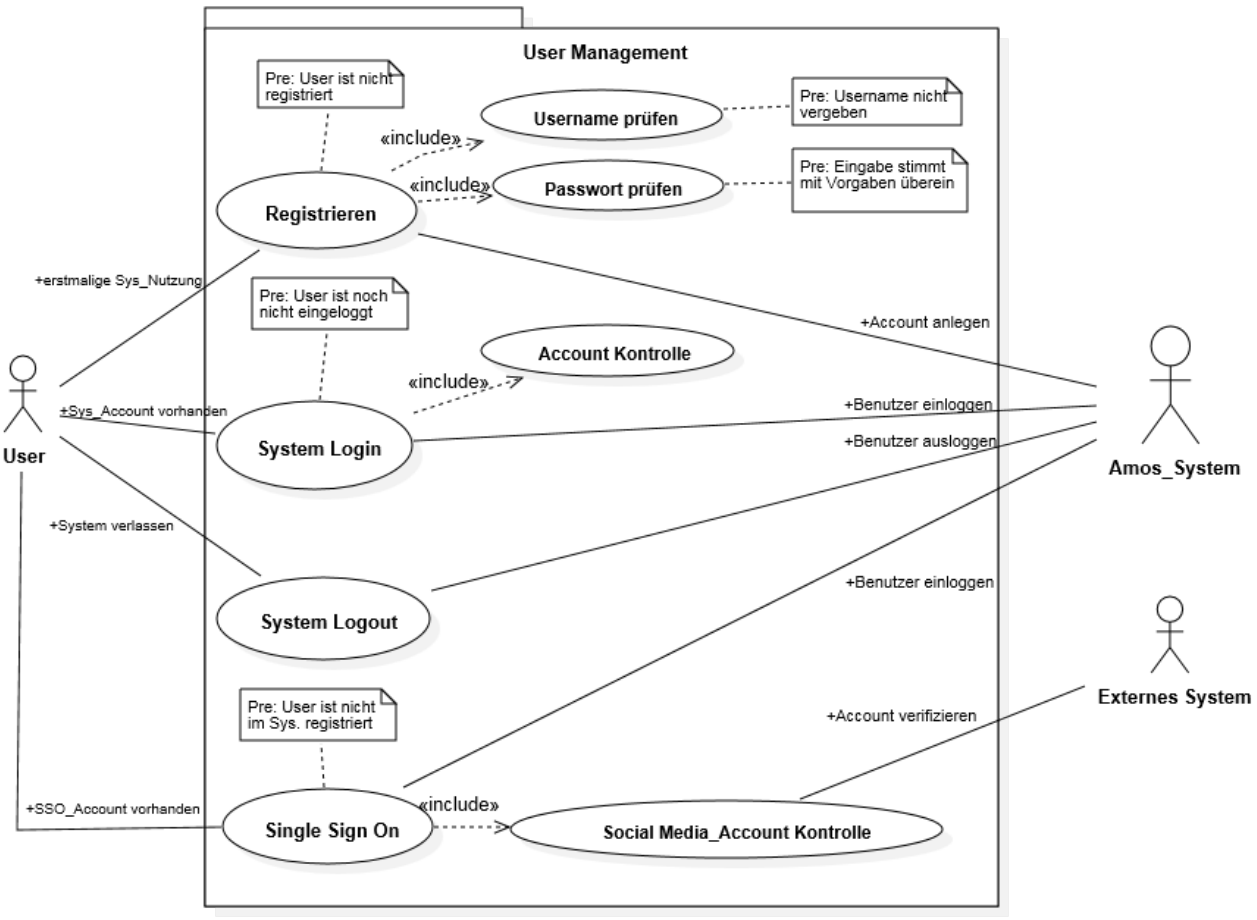


Abbildung 1: Use Cases – User Management

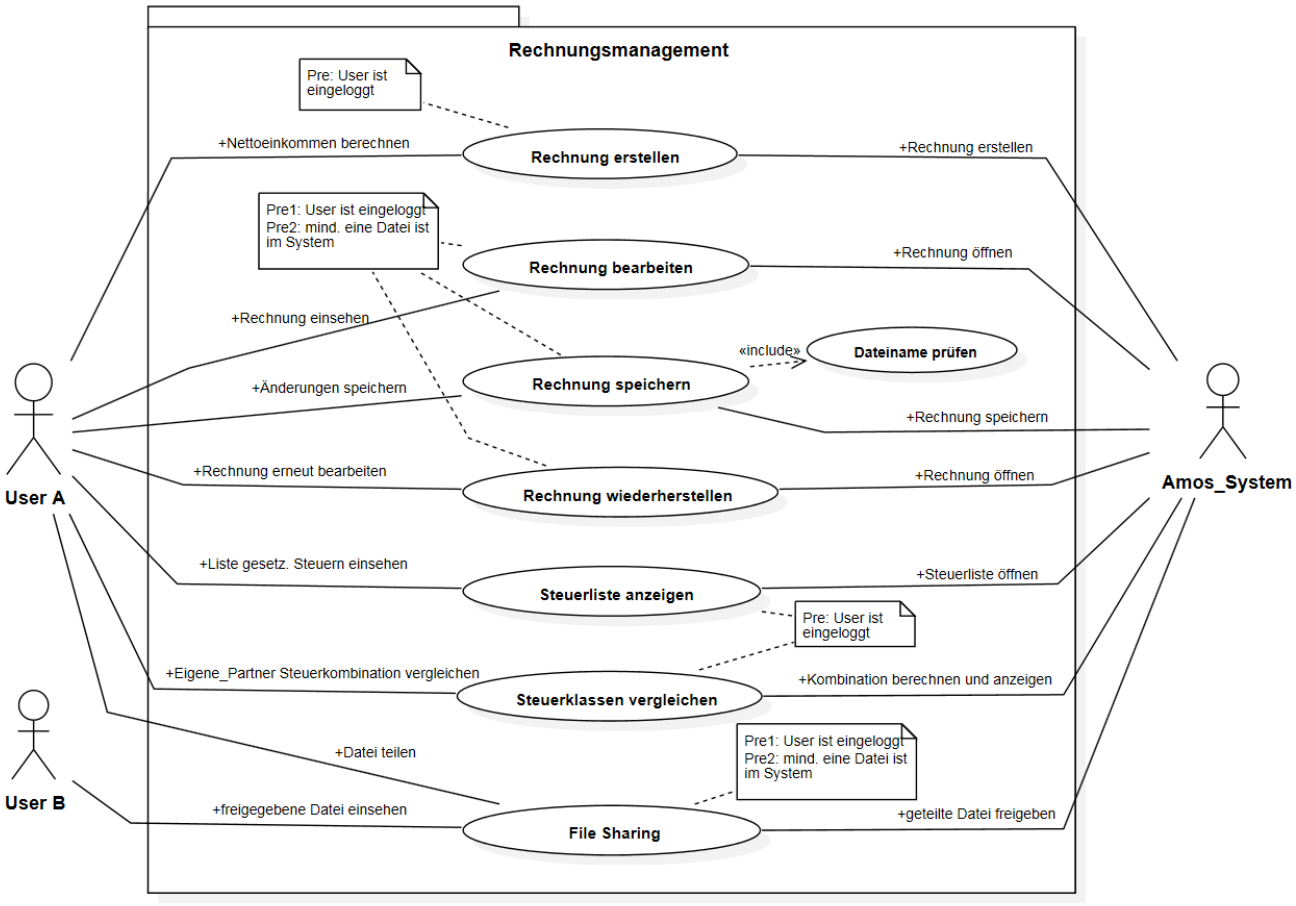


Abbildung 2: Use Cases – Rechnungsmanagement

4 Systemarchitektur

Die Systemarchitektur visualisiert die Umsetzung der externen- und internen Szenarien. Die Beschreibung erfolgt mit Hilfe der vier klassischen Sichten, bestehend aus logischer-, dynamischer, Realisierungs- und Verteilungssicht.

Ein klarer Architekturstil ist nicht erkennbar. Das System strebt eine REST-Architektur an, dabei werden serverseitig allerdings Zustände in Form von User Sessions gespeichert, was dem REST-Prinzip widerspricht.

Als Architekturmuster kommt Model-View-Controller zum Einsatz. Die Einhaltung des Musters wird durch die Verwendung des Flask Frameworks unterstützt.

4.1 Logische Sicht

4.1.1 Komponentenübersicht

Die Komponetenübersicht gibt einen Überblick über alle direkt zur AMOS Cloud Plattform gehörenden Komponenten (Hilfsprogramme zum Deployment ausgeschlossen).

Im Mittelpunkt steht die AMOS Cloud Layer-Komponente (ACL). Mit ihrer Hilfe werden alle zuvor beschriebenen Anwendungsszenarien (Kap. 3) umgesetzt. Die Clientanbindung erfolgt über eine leichtgewichtige REST Schnittstelle (ACL REST API). Die Synchronisation der verschiedenen AMOS Cloud Layer-Instanzen wird mittels des verteilten Key-Value Stores etcd[1] umgesetzt (siehe auch 4.4 Verteilungssicht). Der Zugriff auf etcd erfolgt über eine quelloffene python library namens pythen-etcd, visualisiert als PythonEtcd.

Die Benutzerprofile lokal registrierter User (Username, Passwort, Emailadresse etc.) werden redundant in den relationalen Datenbanksystemen der einzelnen Cloudanbieter gespeichert. Für die Anbindung der Datenbanken bzw. das OR-Mapping kommt SQLAlchemy[2] zum Einsatz.

Der Zugriff auf die Storage-Dienste erfolgt jeweils über die herstellerspezifische REST API.

[1] https://coreos.com/etcd/
[2] http://www.sqlalchemy.org/

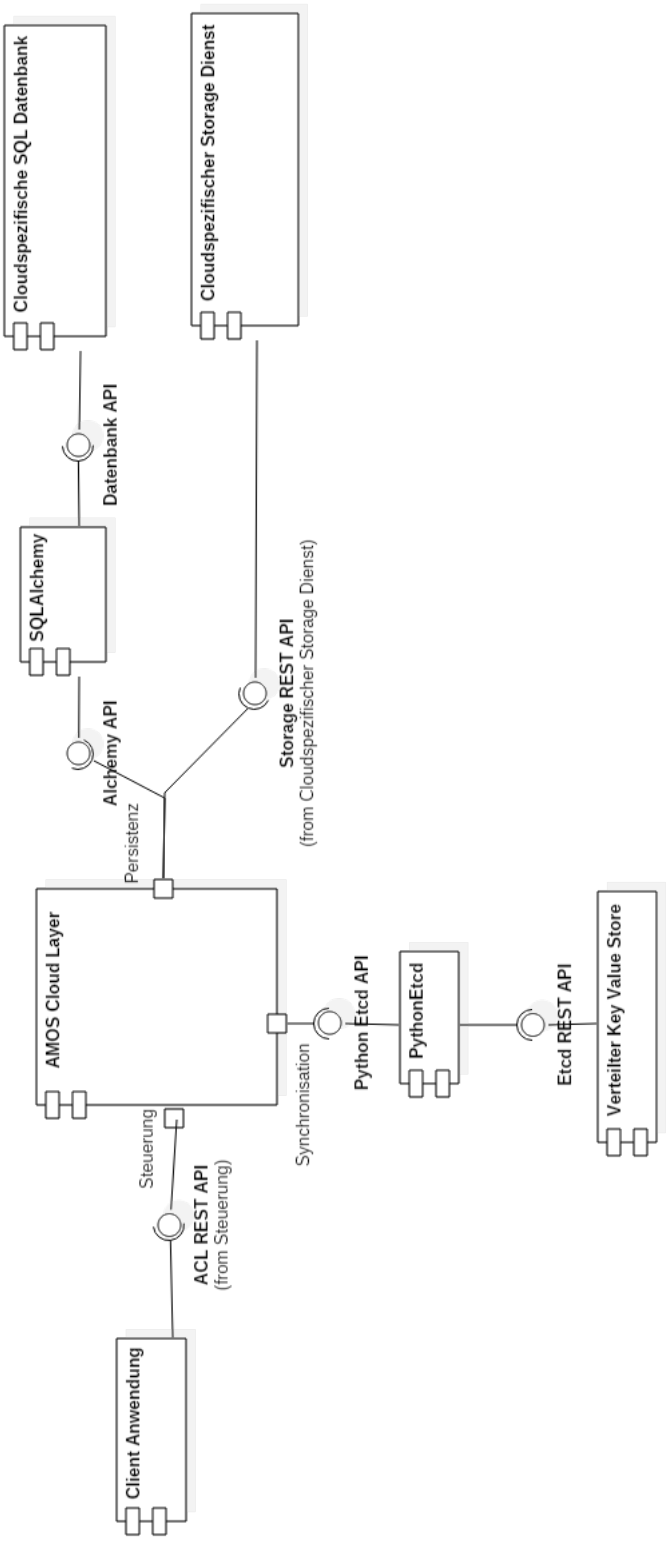


Abbildung 3: Logische Sicht – Komponentenübersicht

4.1.2 AMOS Cloud Layer

Der zentrale Bestandteil des Systems ist der AMOS Cloud Layer. Der Zugriff auf diesen Teil des Systems erfolgt über die grafische Oberfläche der WebApp oder die REST API der ACL. In beiden Fällen ist der zentrale Anlaufpunkt die Controller-Komponente, die für die Bearbeitung der Anfragen des Clients verantwortlich ist.

Die Controller-Komponente verwendet die Benutzerverwaltungs-Komponente und die StorageInterface-Komponente. Die Benutzerverwaltungs-Komponente kümmert sich um die Authentifizierung der Benutzer und die Verwaltung ihrer Session am lokalen System. Die StorageInterface-Komponente ist die Schnittstelle zu den cloudspezifischen Storage-Systemen und verwendet deren Schnittstellen, um Dateien zu speichern.

Die EtcdListener-Komponente wird dazu verwendet, um die Speicherung von Usern, Dateien und deren Berechtigungen mit allen laufenden Cloud-Instanzen zu synchronisieren.

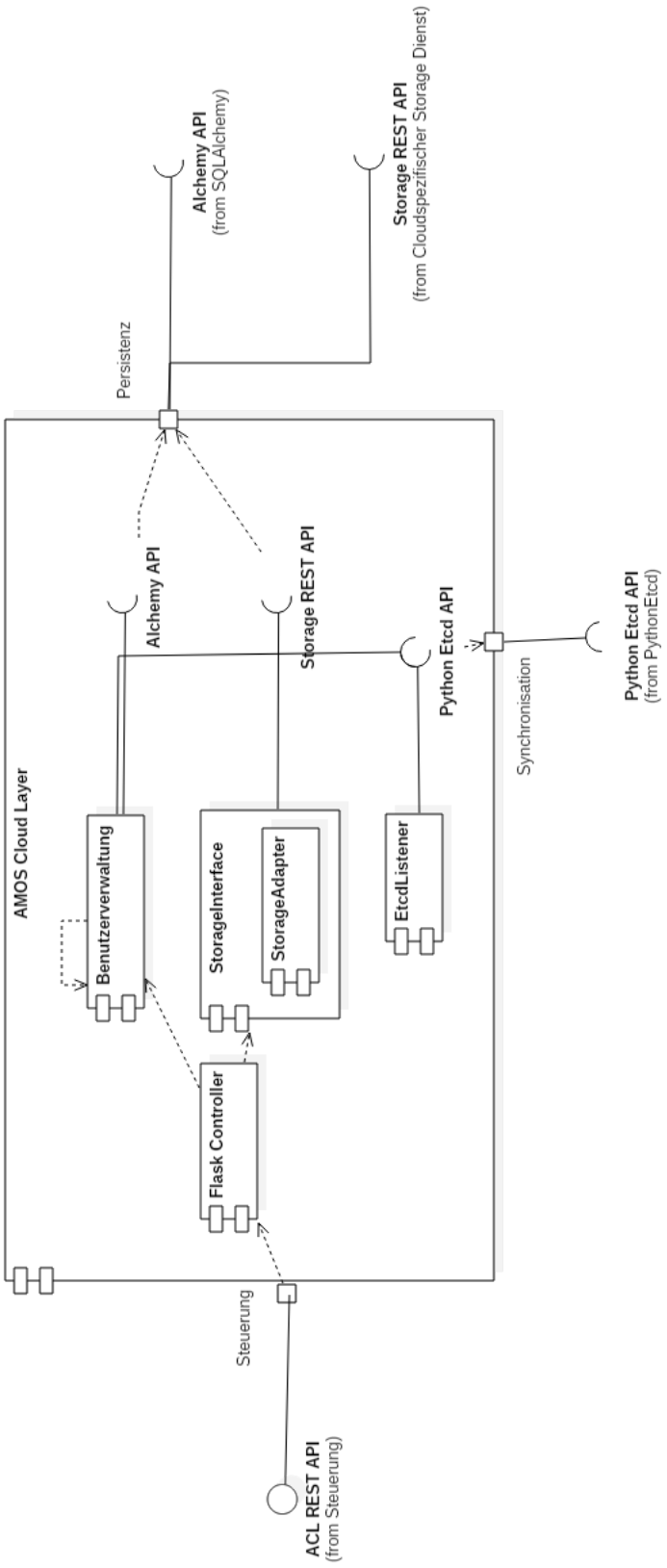


Abbildung 4: Logische Sicht – AMOS Cloud Layer

4.2 Dynamische Sicht

Die dynamische Sicht wird zur Visualisierung der Prozesse des Systems zur Laufzeit und deren gegenseitigen Interaktion verwendet. In Bezug auf die „AMOS Cloud Layer“ gehört die Verteilung der Daten auf den verschiedenen Cloud-Instanzen zu den wichtigsten Funktionen.

4.2.1 Synchronisation der Daten

Im Folgenden werden die Abläufe der o. g. internen Szenarien (Kap. 3.2), sowie der Verteilungsalgorithmus für externe Szenarien (Kap. 3.1) beschrieben.

Ablauf der internen Szenarien Für die Synchronisation der Daten wird das Etcd-System verwendet, ein verteilter Key-Value-Store. Die zugehörige Operation ist in Abbildung 5 visualisiert. Im ersten Schritt wird die Verfügbarkeit der Cloud-Instanzen geprüft, dies geschieht mit Hilfe der Operation, die in Abbildung 6 dargestellt ist. Das Ergebnis wird evaluiert und die eigentliche Verteilung für alle antwortenden Instanzen aufgerufen (Siehe Abbildung 13).

Verteilungsalgorithmus für externe Szenarien Der wiederholte Einsatz vom Verteilungsalgorithmus (siehe Abbildung 5) macht diese essenziell für die o. g. externen Szenarien (Kap. 3.1). Dabei besteht dieser Algorithmus aus 3-Schritten:

1) **Verfügbarkeit der Cloud-Instanzen prüfen**. Zunächst wird an alle bekannten Instanzen eine Nachricht gesendet. Erfolgt eine Antwort auf die Nachricht von den jeweiligen Instanzen, gelten diese als verfügbar, ansonsten als ausgefallen oder nicht erreichbar, falls keine Antwort erfolgt (Siehe Abbildung 6).

Für die **verfügbaren Instanzen** folgen Schritt 2 und 3 (siehe auch Abbildung 13).

2) **Nutzdaten versenden**. In diesem Schritt werden zunächst die Nutzdaten an die verfügbaren Instanzen geschickt. Der Empfang der Daten wird mit einer Bestätigungsnachricht quittiert.

3) **Commit-Nachricht**. Wird der Empfang der Nutzdaten bei der sendenden Instanz bestätigt, erfolgt eine sog. Commit-Nachricht. Die Empfänger-Instanzen dieser Commit-Nachricht schreiben dann die erhaltenen Nutzdaten in ihrer jeweiligen Datenbank und quittieren diese bei Erfolg mit einer bei der sendenden Instanz.

Hinweis: Als Koordinator wird immer der Etcd-Cluster verwendet.

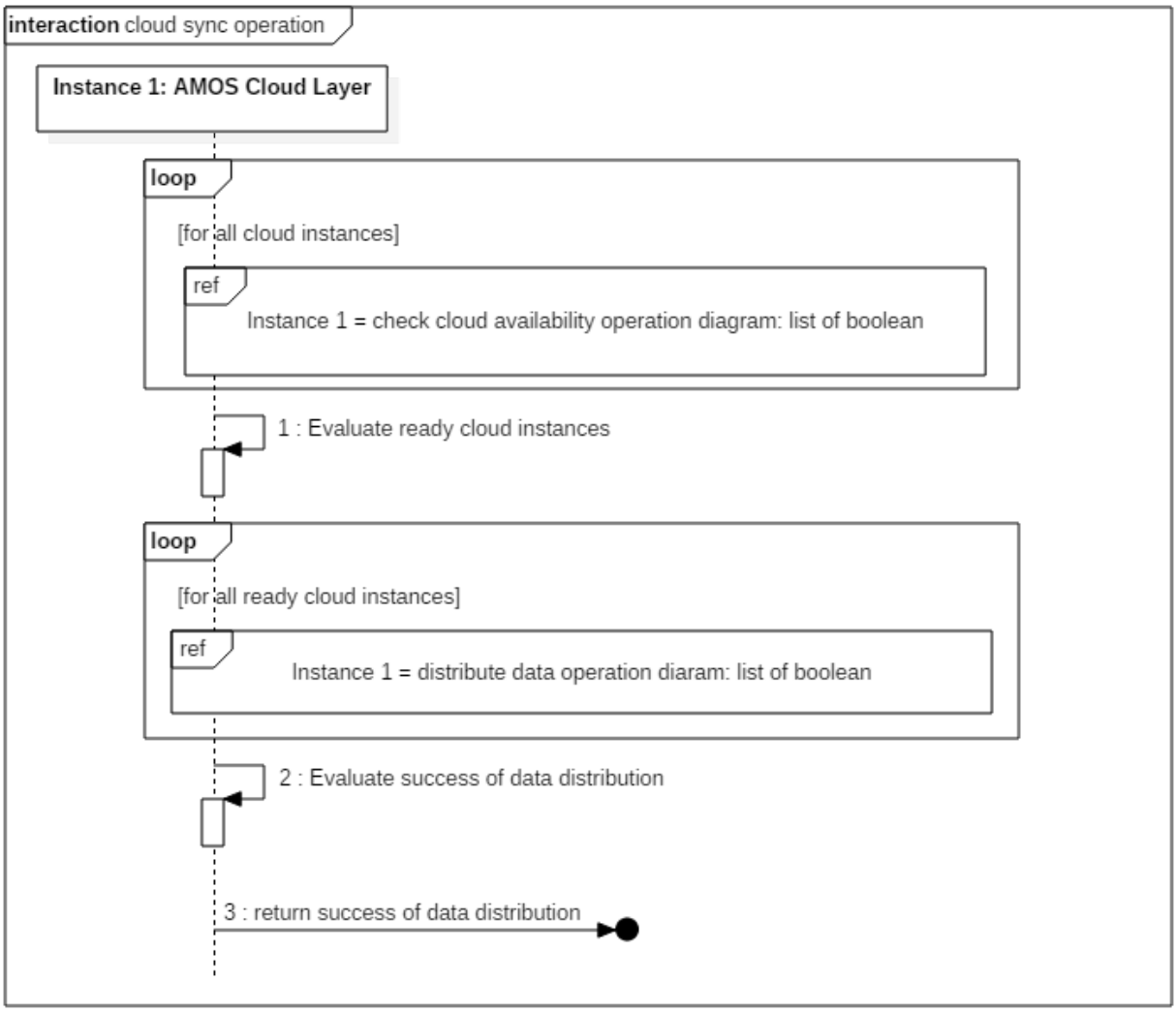


Abbildung 5: Dynamische Sicht – Cloud Sync Operation

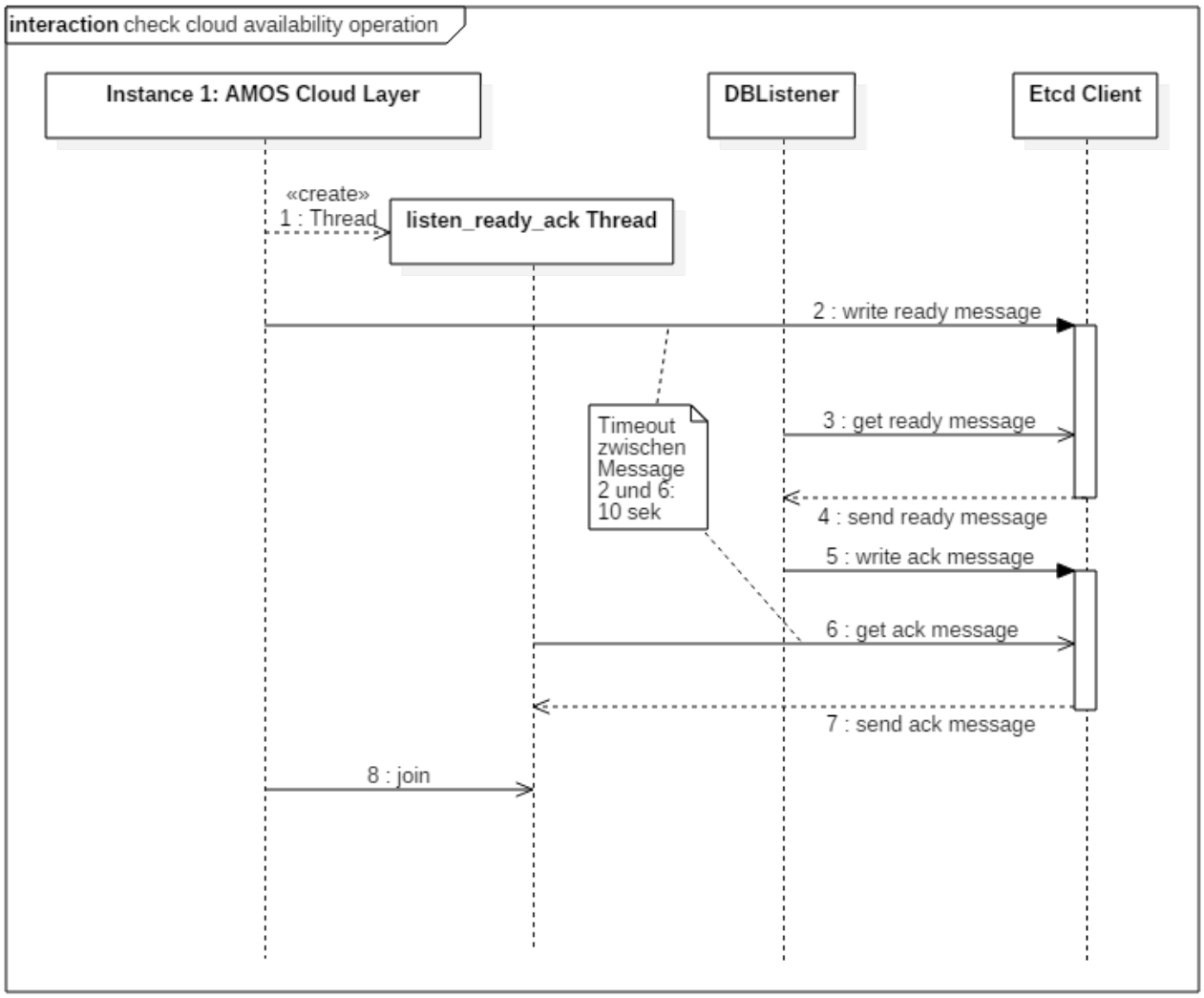


Abbildung 6: Dynamische Sicht – Check Cloud Availability Operation

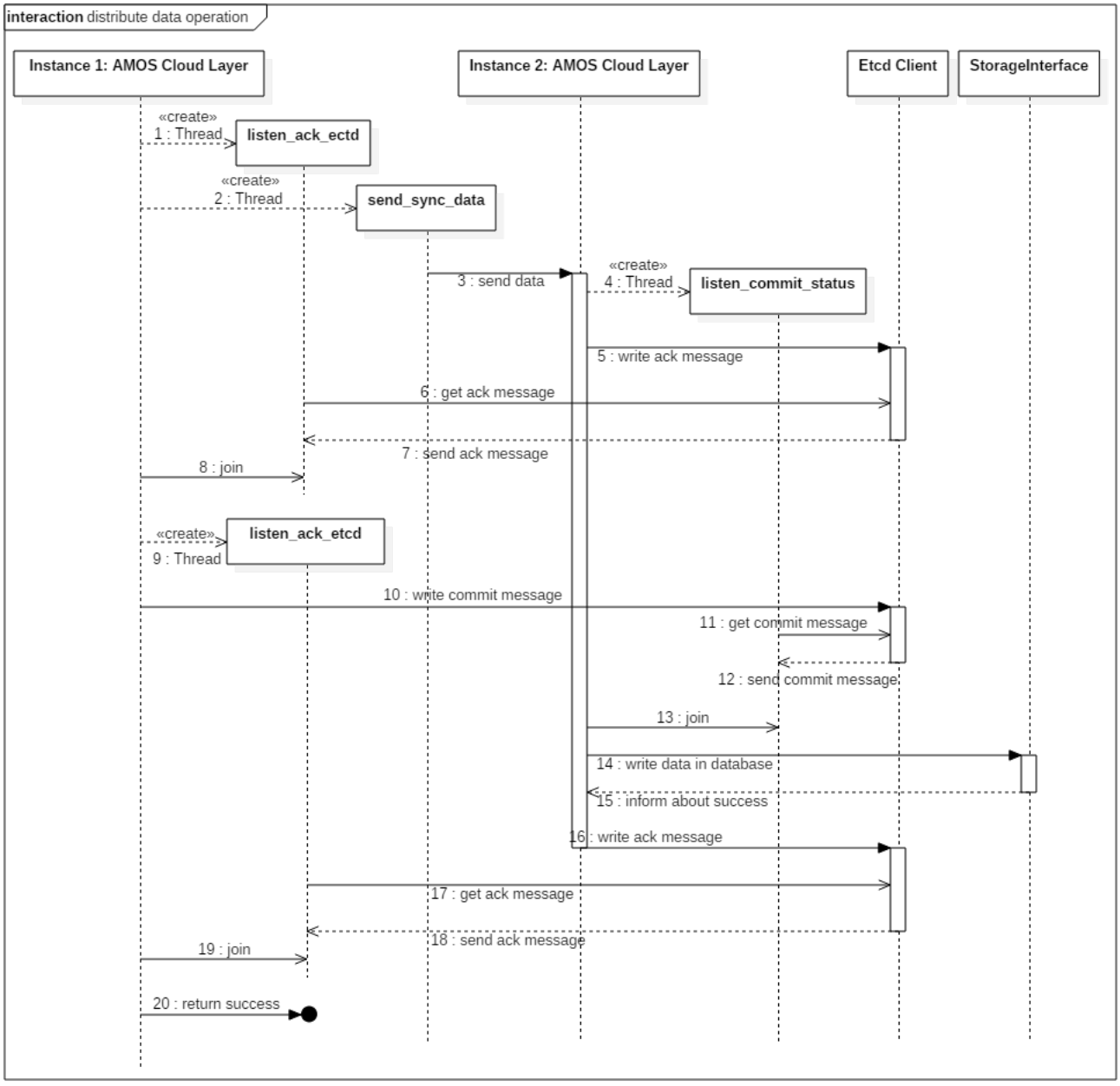


Abbildung 7: Dynamische Sicht – Distribute Data Operation

4.2.2 Dynamische Sicht der externen Use Cases

Ein Anwendungsbeispiel für Verteilungsalgorithmen befindet sich im (Kap. 3.1.1-UC1) in dem das Registrieren eines Users beschrieben wird. Das zugehörige Sequenzdiagramm zu dem o. g. Anwendungsfall wird in der folgenden Abbildung 8 beschrieben.

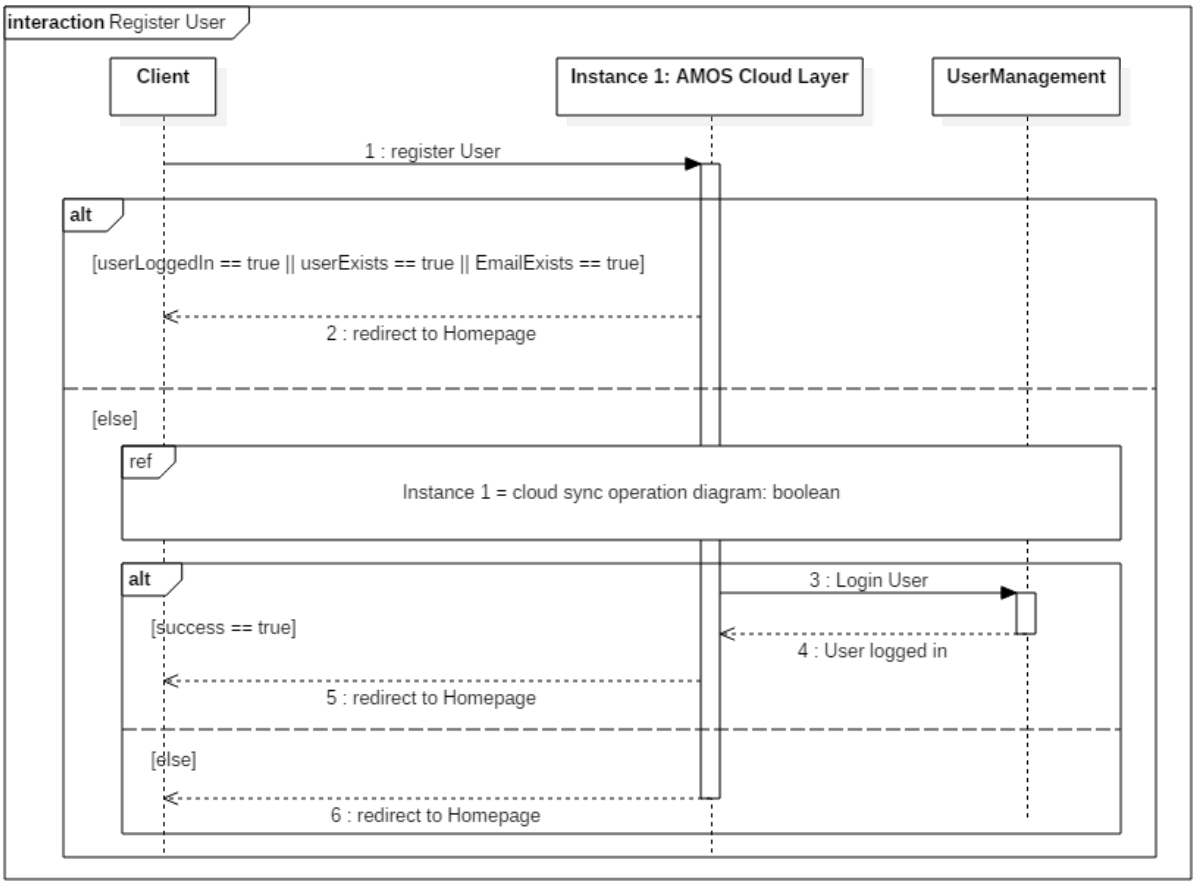


Abbildung 8: Dynamische Sicht – Register User

Bzgl. der Nutzung der vorhandenen Datenbanken, wird zwischen schreibende und lesende Operationen unterschieden. Daten werden nur dann auf die Cloud-Instanzen verteilt, wenn es sich um schreibende Operationen handelt. Bei Lesenden Anfragen hingegen wird nur die jeweilige lokale Datenbank bemüht. Der Anwendungsfall 3- Login eines Users (Kap. 3.1.1-UC1) dient hierfür als Beispiel. Wie in Abbildung 9 erkennbar ist. Dabei werden keinerlei Synchronisations-Operationen aufgerufen und somit finden die Aktivitäten nur innerhalb der lokalen Instanz statt.

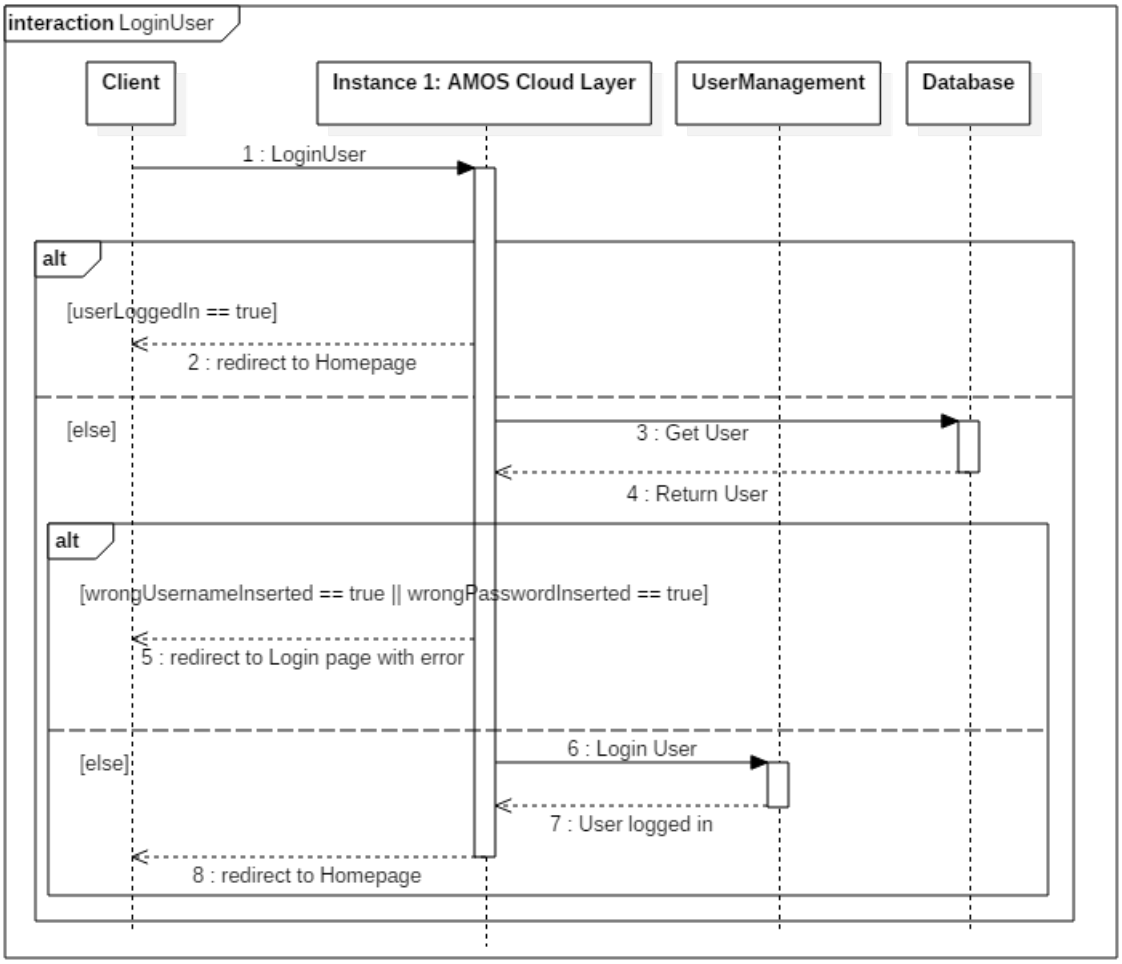


Abbildung 9: Dynamische Sicht – Login User

Die restlichen o. g. Anwendungsfälle (Kap 3.1) laufen äquivalent ab. Handelt es sich um schreibende Operationen, wird der bereits beschriebene Verteilungsalgorithmus (siehe UC 1 Abbildung 8) angewandt. Bei lesenden Operationen wird hingegen nur die betroffene lokale Datenbank befragt (Siehe UC 3 Abbildung 9).

4.3 Realisierungssicht

Die Realisierungssicht beschreibt die Entwicklungsergebnisse, die am Ende jedes Entwicklungsschritts entstehen. Dabei werden Komponenten konkrete Entwicklungsartefakte zugeordnet und Abhängigkeiten visualisiert.

4.3.1 Client Anwendung

Der Zugriff auf dem AMOS Cloud Layer erfolgt exemplarisch über eine webgestützte Client Anwendung. Den Einstiegspunkt in die Anwendung bildet die index.html. Die Registrierung neuer Nutzer erfolgt über das mittels register.html umgesetzte Registrierungsformular. Bestehende Nutzer melden sich über die login.html an.

Der Zugriff auf die Storage-Funktionalität erfolgt über eine angepasste Variante des Datev Brutto-Netto Rechners. Der Brutto-Netto Rechner wird in dieser Architekturbeschreibung als Black Box-Komponente betrachtet.

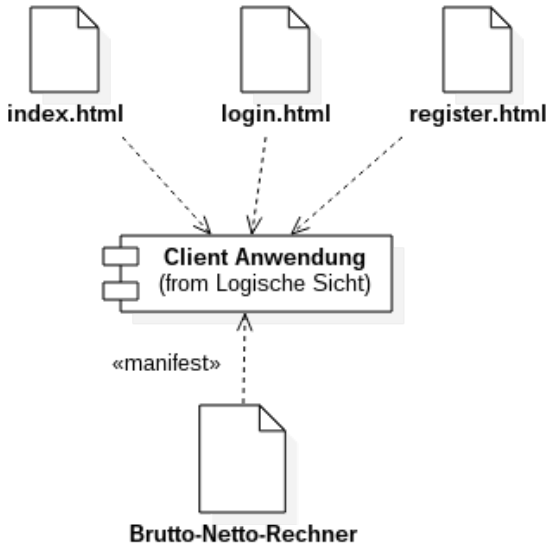


Abbildung 10: Realisierungssicht – Client Anwendung

4.3.2 AMOS Cloud Layer

Die AMOS Cloud Layer bildet das Herzstück der Anwendung, daher werden ihm die meisten Entwicklungsartefakte zugeordnet.

Models.py In der models.py befinden sich die Klassendefinitionen der Benutzer, die Modellierung der Dateien und die Klassenrepräsentation der Zugriffsrechte. Die dabei

definierten Klassen sind zusätzlich mittels SQLAlchemy persistierbar. Die models.py beschreibt somit Datenmodelle. Zum einen das Benutzermodell, das von der Benutzerverwaltung benötigt wird und zum anderen das vom StorageInterface genutzte Datenmodell der Dateien und Zugriffsrechte.

Views.py Die views.py setzt die ACL REST API um. Sie realisiert die Benutzerverwaltung und bildet die Schnittstelle zum StorageInterface. Zusätzlich fungiert sie als Controller (Flask Controller) und liefert je nach aufgerufener URI Daten oder eine View aus.

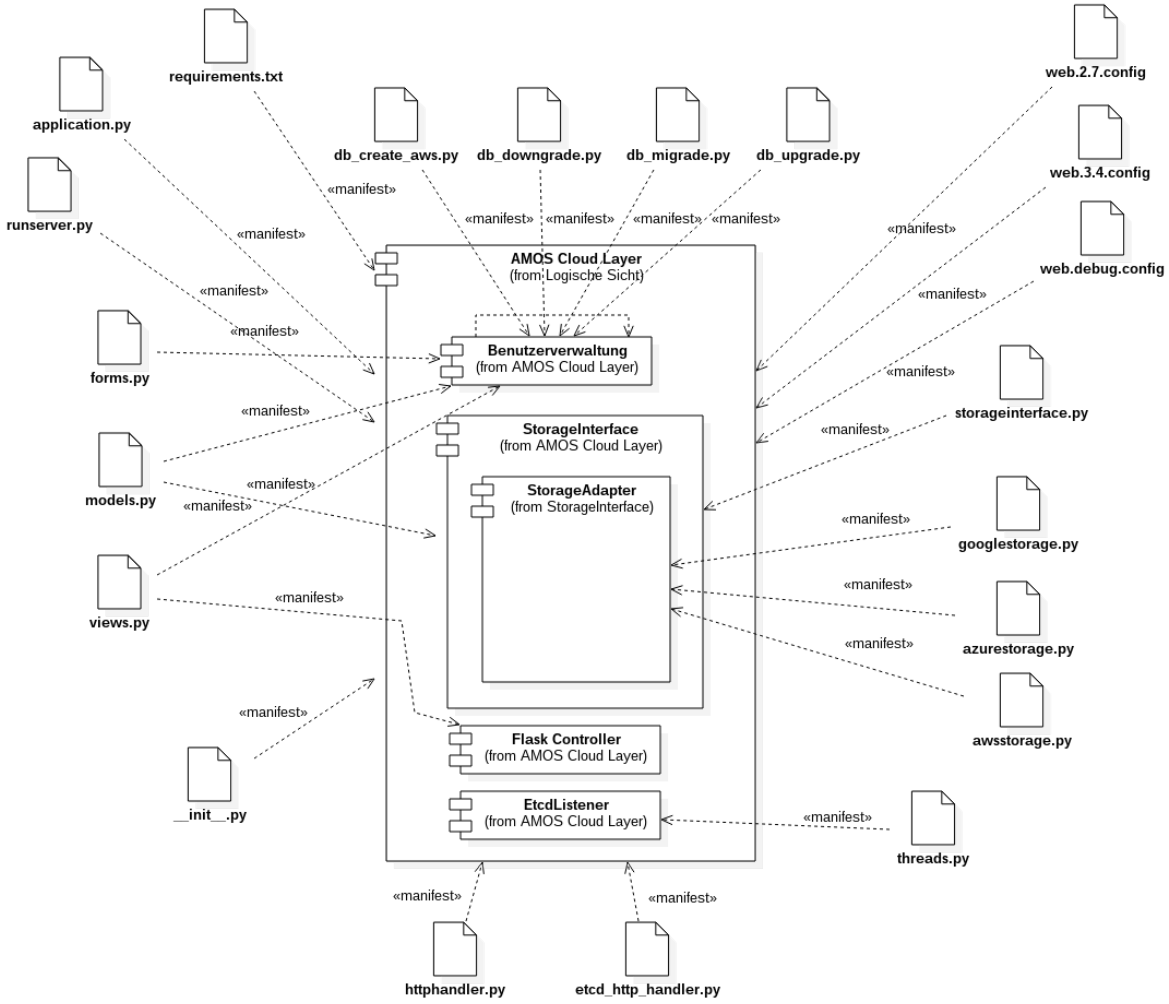


Abbildung 11: Realisierungssicht – AMOS Cloud Layer

Storageinterface.py Die storageinterface.py enthält die Definition des StorageInterfaces und abstrahiert bzw. vereinheitlicht den Zugriff auf die Storage-Dienste der einzelnen

Cloudanbieter.

Google-, azure- und awsstorage.py Jedes dieser drei Artefakte realisiert einen der drei cloudspezifischen StorageAdapter.

Threads.py Die Threads.py enthält die Definition des EtcdListeners. Jede ACL-Instanz startet einen, in einem unabhängigen Thread laufenden, EtcdListener. Hierüber werden die Instanzen über Änderungen an dem in etcd gespeicherten Datenbestand informiert.

Die restlichen Artefakte sind entweder flask- oder pythonspezifisch (___init___.py bzw. requirements.txt), sie dienen der Synchronisation von Datenmodell und Datenbank (db_*.py) oder konfigurieren die Anwendung abhängig von der jeweils vorhandenen Pythonversion (web.*.config). Die verbleibenden Artefakte sind für den Start der Applikation (runserver.py bzw. application.py) bzw.fürs Generieren der Debugging Outputs (httphandler.py bzw. etc_http_handler.py) zuständig.

4.3.3 Gemeinsame und plattformspezifische Artefakte

Ziel des Systems ist eine möglichst große, gemeinsame Code-Basis, die auf jeder Cloud-Umgebung verwenden werden kann. Allerdings ist die Verwendung von cloudspezifische APIs unerlässlich, die daraus resultierenden Folgen sind, plattformunabhängige bzw. plattformspezifische Artefakte. Eine Aufteilung der entstandenen Artefakte werden in Abbildung 12 geliefert.

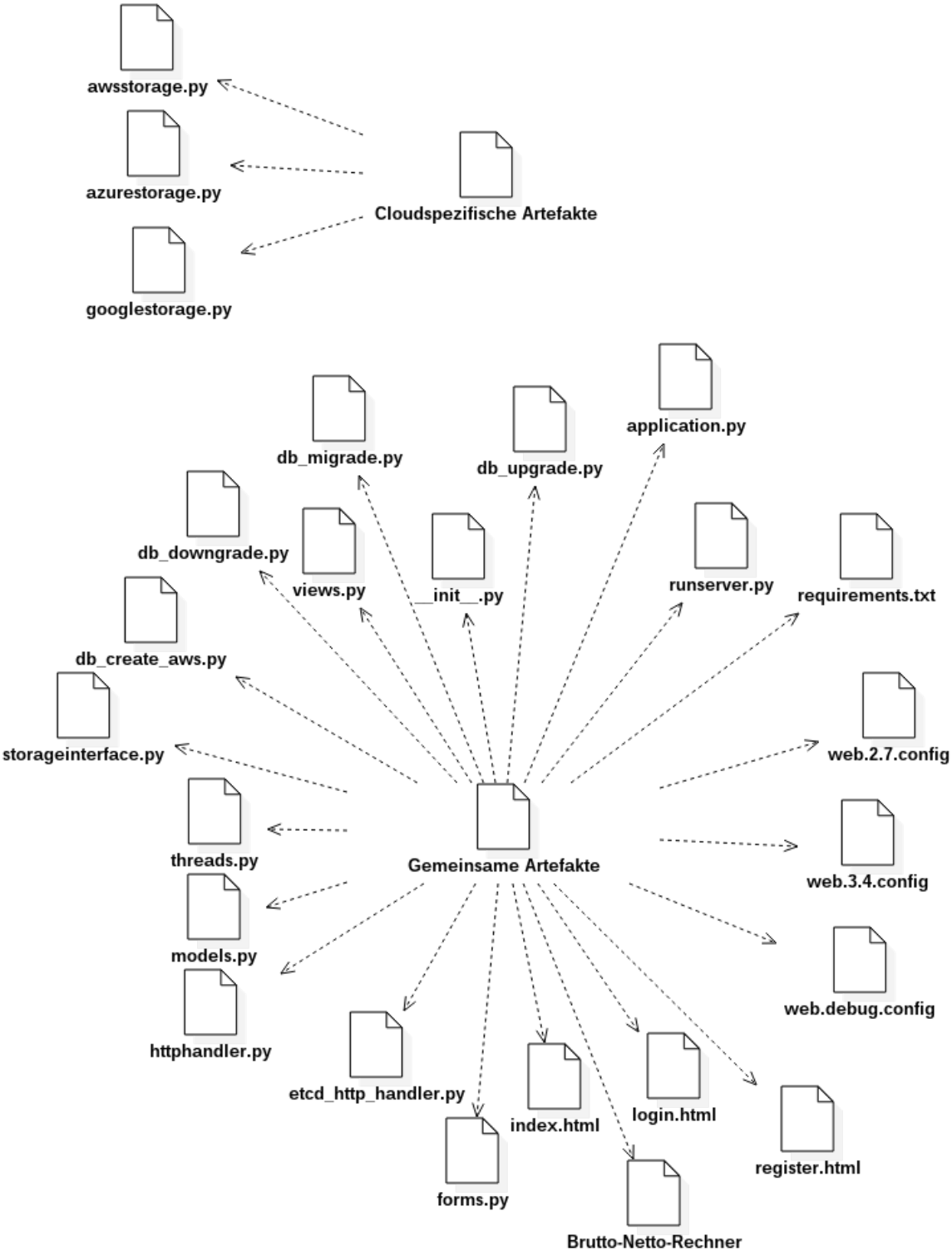


Abbildung 12: Realisierungssicht – Virtuelle Pakete

4.4 Verteilungssicht

Die Verteilung des Systems erstreckt sich über mehrere Cloud-Umgebungen. Im konkreten Fall des AMOS Cloud Layers (ACL) handelt es sich um die Amazon WebServices (AWS), die Google AppEngine und die Azure-Cloud von Microsoft. Auf jede dieser Umgebungen werden plattformspezifische Artefakte des ACLs aufgespielt, die die spezifischen APIs der jeweiligen Cloud als Adapter beinhaltet. Zudem gibt es gemeinsame Artefakte, die auf allen Cloud-Instanzen gleich sind.

Des Weiteren laufen zusätzlich in jeder Cloud die eigenen Datenbank- und Storagedienste. Die genaue Vorgehensweise der Verbindung zu diesen Diensten kann nicht konkret evaluiert werden, da zum einen hier u. a. die physikalische Distanz der Instanzen eine bedeutende Rolle spielt (Ortsabhängigkeit des Rechenzentrums) und zum anderen die Auslastung der Netzwerkkomponenten die Performance beeinflussen. Die Auslastung der Netzwerkkomponenten ist dabei an die Gesamtauslastung des Datenzentrums gekoppelt, was vollständig unkalkulierbar ist.

Für die konkrete Implementierung des ACLs spielt AWS eine besondere Rolle. Anfragen eines Clients landen zentral bei einem Load Balancer des AWS, um sie dann umgehend unter den laufenden Instanzen aufzuteilen. Außerdem läuft in einer AWS-Instanz der Etcd-Cluster, der eine wichtige Rolle bei der Synchronisation von Daten spielt (siehe Dynamische Sicht).

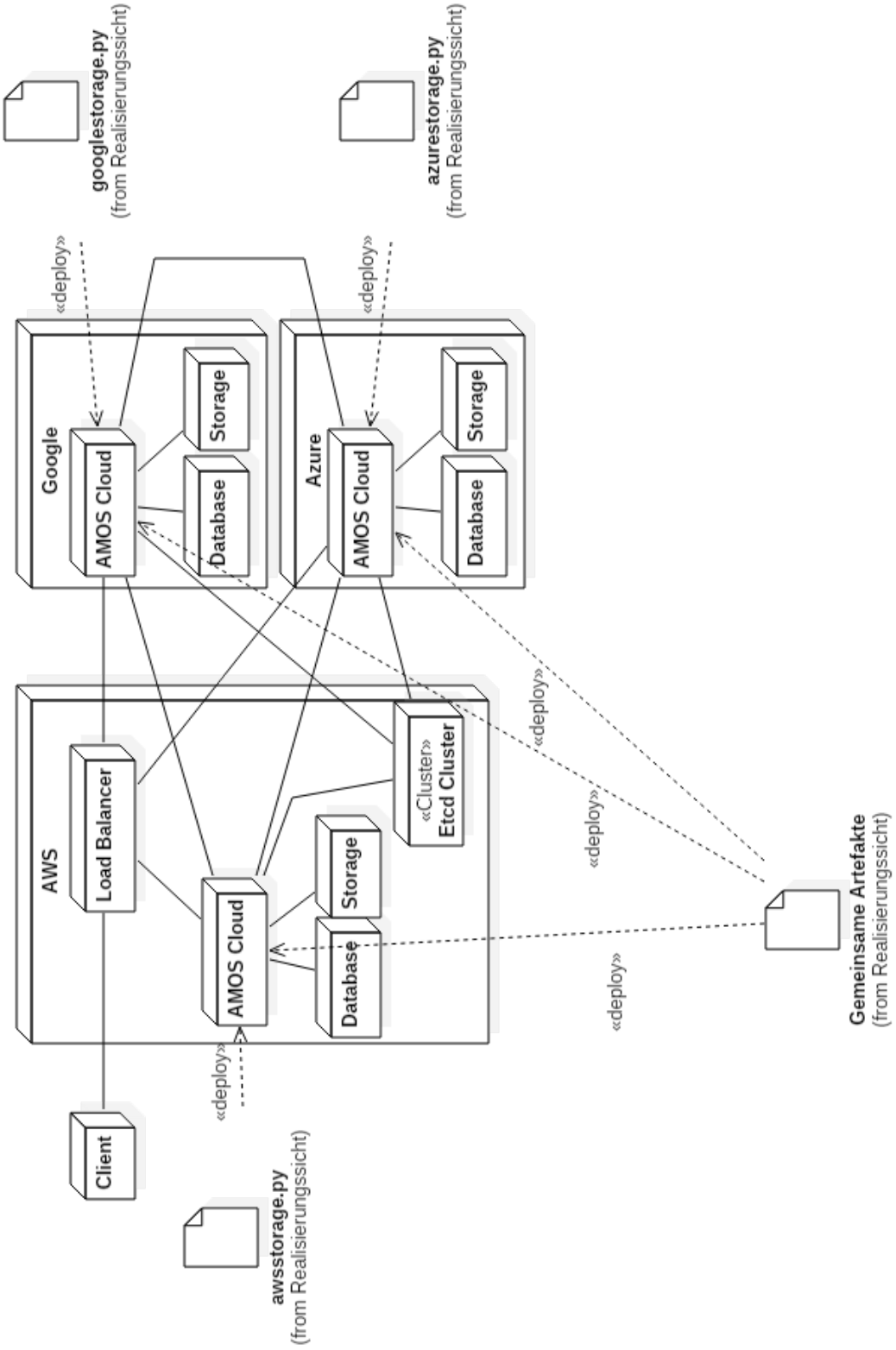


Abbildung 13: Verteilungssicht – Knoten

5 Entwurfsentscheidungen

Der Brutto-Netto-Rechner der DATEV eG ist in Javascript implementiert, wird also clientseitig im Browser ausgeführt. Dies stellt somit keine Herausforderung für die Cloud-Plattformen dar.

Für die Entwicklung wurde Python als Programmiersprache ausgewählt, da diese den Vorteil hat, dass sie von allen Clouds nativ unterstützt wird. Des Weiteren wird das MVC-Framework Flask eingesetzt. Dieses bietet eine Vielzahl an Modulen an, z.B. für das User Management, womit die Entwicklung einzelner Komponenten erleichtert wird und somit der Fokus auf die Cloud-Synchronisation gelenkt wird.

Die Entwurfsentscheidungen des AMOS-Teams bezüglich der Multi-Cloud-Synchronisation wurden im Rahmen eines Interviews mit dem Product Owner ermittelt. Eine Zusammenfassung des Interviews folgt im unteren Abschnitt:

1. Welche Daten werden synchronisiert?
    - User Accounts
    - Sessions
    - Nutzdaten
    - Metadaten
2. Wie wird die Gruppenkommunikation funktionieren?
    - Es wird das Open-Source-System Etcd als Locking-System und zur verteilten Synchronisation eingesetzt.
3. Wie sind die Konsistenz-Eigenschaften des Systems?
    - Sie sind stark, da die Client-Anwendung nicht weiter angepasst werden muss, es müssen lediglich die missglückten Schreib- oder Lesezugriffe bzw. die Timeouts umgangen werden können.
4. Welche Besonderheiten sind bei der Synchronisation zu beachten?
    - Die Daten werden bei Schreibzugriffen vorab in ein temoräres Verzeichnis geschrieben. Falls dabei etwas fehlschlägt, wird ein Fehler an die Anwendung zurückgegeben. Erst mit dem Erhalt der Commit-Nachricht werden die Daten produktiv geschrieben.
    - Etcd wird verwendet, um locks auf bestimmte Dateien zu erhalten bzw. eine totale Ordnung auf Schreiboperationen zu erzwingen.

6 Qualitätsbetrachtungen

6.1 Metriken

Um die Qualität der entstandenen Architektur bewerten zu können, werden Metriken benötigt. Diese werden an dieser Stelle mittels Goal-Question-Metric-Verfahren (GQM) definiert. Eine Übersicht der möglichen Metriken in diesem Kontext wird in der Abbildung 14 visualisiert. Es ist zu empfehlen, nicht allzu viele Metriken festzulegen, damit vorrangig der Fokus auf aussagekräftige Metriken gelegt wird.

Nachfolgend sind genauere Beschreibungen der definierten Metriken aufgelistet:

G.1 - Erhöhte Wartbarkeit des Codes

G.1.1 - Verbesserung der Komplexität

Blickwinkel: Entwickler, Tester

Zweck: Optimierung

Beschreibung des Qualitätsaspekts:

- Aufwand zum Verstehen und Warten des Codes für Entwickler
- Aufwand zum Erstellen von Tests

Quantifizierung:

- Ermittleln des McCabe-Wertes jeder Funktion (Ziel: McCabe < 10)
- Ermitteln der max. Schachtelungstiefe jeder Funktion (Ziel: Tiefe < 4)

G.1.2 - Erhöhte Lesbarkeit

Blickwinkel: Entwickler, Tester

Zweck: Optimierung

Beschreibung des Qualitätsaspekts:

- Aufwand zum Verstehen des Codes für Entwickler
- Aufwand zum Erstellen von Tests

Quantifizierung:

- Ermitteln der max. Schachtelungstiefe jeder Funktion (Ziel: Tiefe < 4)
- Ermitteln des Kommentaranteils jeder Funktion (Ziel: 30% Kommentaranteil)
- Ermitteln der Anzahl an Zeilen jeder Funktion (Ziel: LOC < 50, LLOC < 20)

G.2 - Erhöhte Wiederverwendbarkeit des Codes

Blickwinkel: Entwickler

Zweck: Optimierung

Beschreibung des Qualitätsaspekts:

- Aufwand zum Verstehen des Codes für Entwickler, um den Code korrekt wiederverwenden zu können
- Hohe lokale Kohäsion, damit Querverbindungen zwischen Funktionen/Modulen unterbunden werden

Quantifizierung:

- Ermitteln des Kommentaranteils jeder Funktion (Ziel: 30% Kommentaranteil)
- Ermitteln, ob Kommentare zur Verwendung einer Funktion vorhanden sind (Ziel: jede Funktion weißt diese auf)
- Ermitteln der Anzahl an Verbindungen jeder Funktion zu anderen Modulen (Ziel: Verbindungen < 10)
- Ermitteln der Anzahl an Zeilen jeder Funktion (Ziel: LLOC < 20)

G.3 - Verbessertes Deployment

Blickwinkel: Entwickler, Kunde

Zweck: Optimierung

Beschreibung des Qualitätsaspekts:

- Bei hoher Last sind neue Instanzen nötig, um die Last besser verteilen zu können

Quantifizierung:

- Ermitteln der Dauer, bis eine neue Instanz einsatzbereit ist (Ziel: Zeit < 1 min)

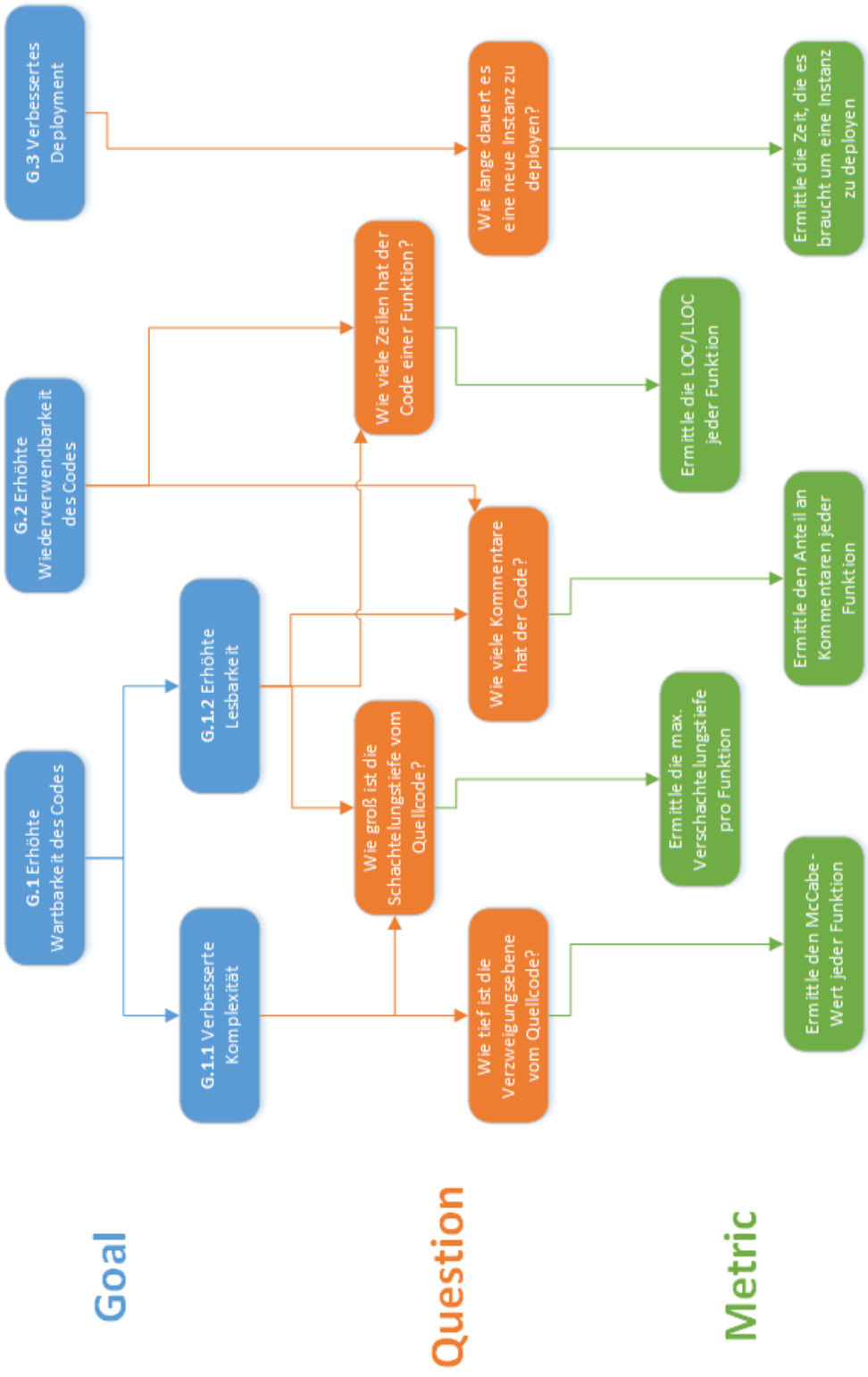


Abbildung 14: Übersicht über das GQM-Verfahren

6.2 Last- und Stresstest von Webanwendungen

Die wichtigsten Aspekte für den Erfolg eines Softwareprodukts sind u. a. Performance und Ausfallsicherheit. Auch im Rahmen dieses Projekts spielen diese genannten Aspekte eine bedeutende Rolle. Es ist entscheidend zu wissen, wie sich eine Applikation auf eine jeweilige Cloud während des Betriebs und Benutzerzugriffs verhält. Daher ist es erforderlich, die Anwendung unter Last effektiv zu testen, bevor sie produktiv eingesetzt wird. Der Mehrwert dabei ist, dass mögliche Performance-Probleme bereits im Vorfeld aufgespürt und wenn möglich beseitigt werden können.

Einen wichtigen Teil hierzu tragen Last- und Stresstests bei. Diese gehören zu den nicht funktionalen Tests und berechnen bei Ihrer Ausführung Kennzahlen über verbrauchte Kapazitäten und Ressourcen von Anwendungen in eine Cloud. Der Lasttest prüft das Verhalten eines Systems (Hier das Cloud-Verhalten) bei steigendem Systemlast. Das bedeutet, dass das Systemverhalten bei kontinuierlich steigender Last, innerhalb eines erwarteten Normalbetriebs, getestet wird. Somit können Engpässe, die evtl. im späteren Live-Betrieb auftreten können besser eingeplant und einen zuverlässigeren Einsatz ermöglicht werden. Im Gegensatz zum Lasttest handelt es sich bei dem Stresstest um eine Testmethode, die versucht die Last bis über die Grenzen hinaus zu steigern um das Systemverhalten bei Überlastung und in Extremfällen zu testen. Mit dieser Testmethode werden sowohl die Stabilität, als auch die Grenzen der Performance der Clouds getestet.

Entscheidend für dieses Projekt ist die Analyse des Last- und Stressverhaltens sowie die Überprüfung des Zeit- und Verbrauchsverhaltens der Clouds unter gegebenen Umständen. Dabei werden Latenzzeiten sowie der Verbrauch von Speicher gemessen und analysiert.

6.2.1 Wie wird gemessen?

Für die Performancemessung können verschiedene Modelle eingesetzt werden, die drei grundlegendsten Modellarten sind: Messung, Simulation und Modellierung.

Die hier vorgestellten Performancemessungen basieren auf das Model „Simulation“. Hierfür wird ein vereinfachtes Model des Systems verwendet. Grundlegende Funktionalitäten sowie das Verhalten der Benutzer bleiben erhalten. Durch Simulation von tausenden Benutzern in Echtzeit, die gleichzeitig http-Anfragen an den Server senden, wird das System einer außerordentlichen Last ausgesetzt wodurch typische Problembereiche geprüft werden können. Während dieser Ausführungen werden die Daten aufgezeichnet, analysiert und ausgewertet.

Die Ergebnisse einer Performancemessung durch Simulation sind in erste Linie Schätzungen, die allerdings sehr genaue und zuverlässige Werte über das zukünftige Verhalten des Systems vorhersagen. Die Analyse des Lastverhaltens und die Darstellung der Er-

gebnisse geschehen mittels einer Zeitmessung.

$$Gesamtzeit = Zeit_{Hinweganfrage} + Zeit_{Verarbeitung} + Zeit_{Zurück} \quad (1)$$

Die gemessene Zeit (Gesamtzeit) wird aus der Transportzeit(die Latenz der Verbindung) und der Bearbeitungszeit, der für die vollständige Bearbeitung der enthaltenen Anfragen benötigt wird, errechnet.

Die Basis für die Kommunikation in Webanwendungen ist das HTTP-Protokoll. Bekannterweise basiert es auf einem einfachen Request/Response-Mechanismus. So ist die Latenzzeit durch Round-Trip-Time definiert und stellt die Zeit dar, die benötigt wird, um ein Paket von Einem zum anderen Ende einer Verbindung hin und zurück zu schicken. Diese kann allerdings je nach Entfernung, Verbindungsart und verwendete Pakete sich unterscheiden. Die Latenzzeit wird dann mithilfe von Ping-Test ziemlich genau bestimmt und gemessen. Um eine möglichst genaue und gleichzeitig vielseitige Messung durchführen zu können und um aussagekräftige Ergebnisse zu erhalten, werden die Latenzzeiten von 4 unterschiedlichen Standorte aus gemessen: Amsterdam, Paris, Zürich und Nürnberg. Die Web-Anwendung befindet sich in Amsterdam. Gemessen wird somit die Transportzeit zwischen Amsterdam und die der aufgelisteten vier Standorten. Die nachfolgende Abbildung 15 veranschaulicht die einzelnen Standorte sowie die Struktur der Messung.

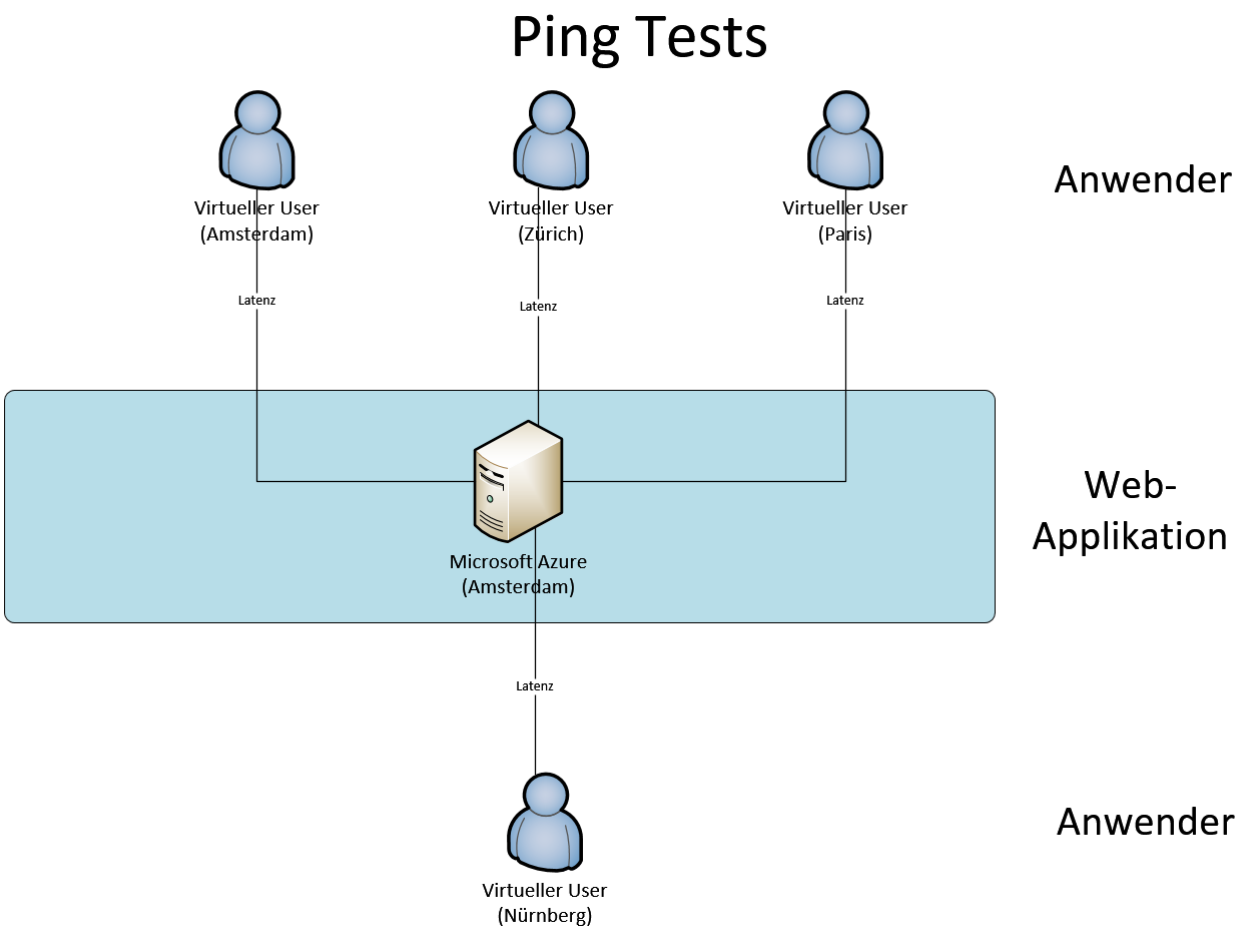


Abbildung 15: Übersicht über alle Ping Tests

Zum besseren Verständnis werden im folgenden Abschnitt wichtige Begriffe zu Lasttests definiert:

Server response time: Zeit zwischen dem Empfang einer HTTP-Anforderung und dem Abschluss des Sendevorgangs der Antwort. Diese Metrik ist ein Durchschnitt. Jeder Punkt im Diagramm steht für die Daten von 1 Minute.

Anzahl der Requests: Anzahl der abgeschlossenen HTTP-Anforderungen. Diese Metrik ist eine Anzahl. Jeder Punkt im Diagramm steht für die Daten von 1 Minute.

Send request time: Zeit zwischen der Netzverbindung und dem Empfang des ersten Bytes. Diese Metrik ist ein Durchschnitt. Jeder Punkt im Diagramm steht für die Daten von 1 Minute.

Dauer der Abhängigkeiten: Dauer der von der Serveranwendung an externe Ressourcen ausgeführten Aufrufe. Diese Metrik ist ein Durchschnitt. Jeder Punkt im Diagramm steht für die Daten von 1 Minute.

Es ist zu beachten, dass die inländischen Verbindungen nicht vernachlässigt und somit bei der Durchführung der Messungen berücksichtigt und ausgewertet werden sollten. Die Latenzzeiten für diese Verbindungen sollten relativ stabil sein, es dürfen im Idealfall nur leichte Latenzzeit-Unterschiede vorhanden sein.

6.2.2 Verwendete Performancetools

Da das im vorherigen Kapitel beschriebene Szenario (PingTests) sehr kostenintensiv und mit viel Aufwand verbunden ist, ist an der Stelle der Einsatz eines Tools zur Unterstützung mehr als sinnvoll.

Die Loadtests sollten mit mehreren tausend virtuellen Usern, über einen Zeitraum von einer Woche, durchgeführt werden. Durch die enorme Bedeutung von Performanz-Tests existieren viele bereits fertiggestellte Werkzeuge mit deren Hilfe Web-Anwendungen zuverlässig und mit angemessenem Aufwand getestet werden können. Solche Werkzeuge sind im Stande sowohl Last- als auch Stresstests durchzuführen. Abhängig davon, ob zielgerichtet punktuelle Test, was einem Stresstest entsprechen würde, durchführt oder kontinuierliche Tests ausführt, wobei die Belastung auf das System sukzessive erhöht wird. Bleibt dabei die Belastung innerhalb der Systemgrenzen, wurde somit ein Lasttest ausgeführt.

Für die Durchführung der Tests in diesem Projekt, wird Microsoft Studio Application Insights verwendet. Dabei wird die Infrastruktur von Windows Azure verwendet, diese ermöglicht eine barrierefreie Ausführung und Auswertung in der Cloud. Erfahrungsbasiert hat sich der o. g. Tool als sehr mächtig erwiesen, da sie über viele und umfangreiche Funktionen (u.a. anpassbare Dashboards, Diagnostizieren von Leistung) verfügt, die die Durchführung der gewünschten Tests enorm erleichtern.

Hinweis: Eine ausführliche Beschreibung des Tools erfolgt im Rahmen dieses Berichts nicht. Weitere Informationen erhalten Sie auf der Hersteller-Homepage.

6.2.3 Ermittlung der Ergebnisse

Die gestellten Anforderungen für die Tests seitens des Industriepartners waren die Messung von Performanz und Latenzzeit der verschiedenen Clouds, insbesondere im Bereich Storage. Für unseren Fall wurde ein Testszenario entwickelt, um möglichst realitätsnahe Testmesswerte zu entsprechen. (Siehe Abbildung 16).

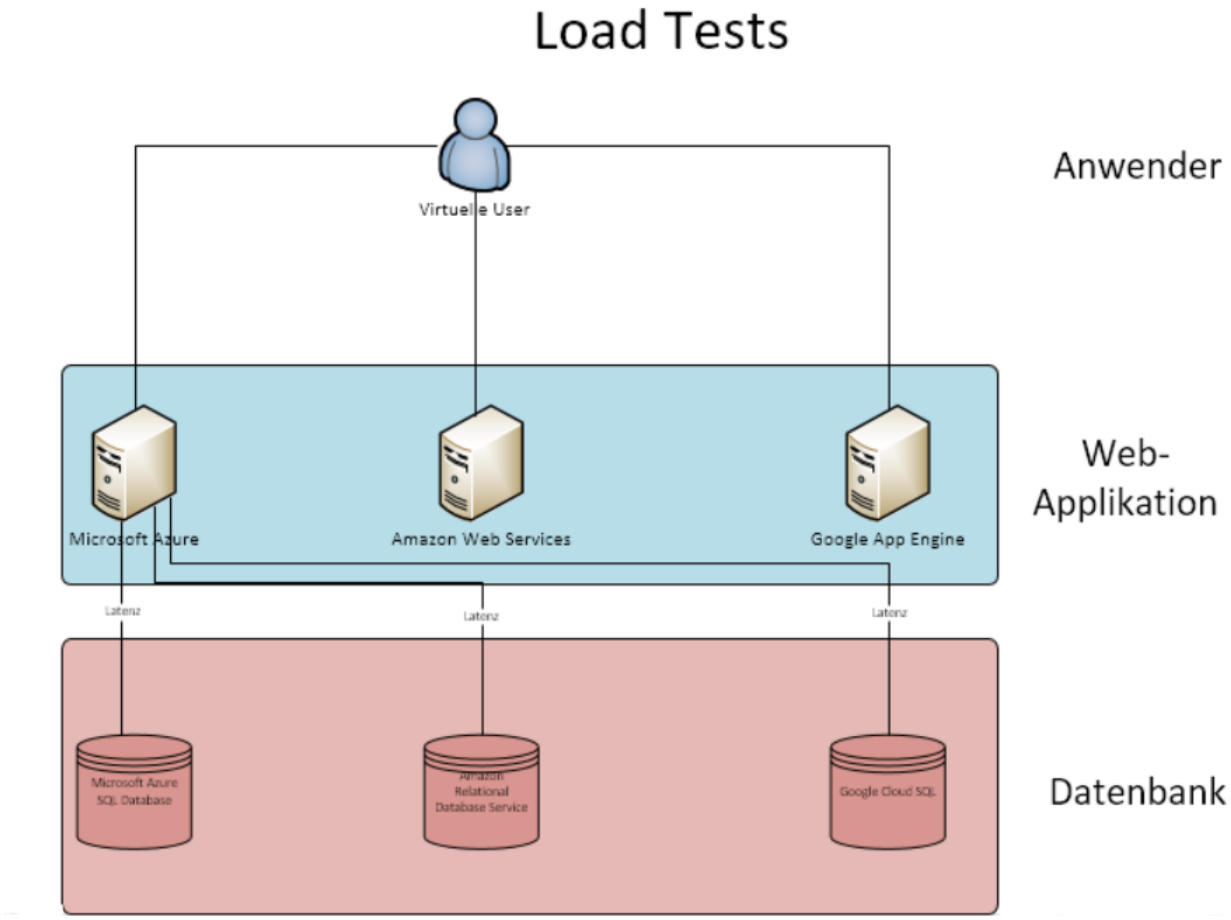


Abbildung 16: LastTests Szenario

In dem voran gegangenen Kapitel wurden die grundlegenden Daten, die bei der Analyse des Performance-Verhaltens eines Systems eine wichtige Rolle spielen, vorgestellt. In den nachfolgenden Abbildungen werden mehrere Webtests bzw. Pingtests, die mit Hilfe der Azure Infrastruktur, den Microsoft Tools Visual Studio sowie Microsoft Studio Application Insights generiert wurden, dargestellt.

Die Pingtests wurden über mehrere Tage, sowohl Werktags als auch am Wochenende, durchgeführt.Diese Vorgehensweise hat den Vorteil, dass genauere Aussagen über die Latenzzeiten der westeuropäischen Netze getroffen werden können.

Für die Tests befindet sich die Anwendung in der Azure Cloud am Standort Amsterdam. Die Pingtests werden aus drei Standorten, Amsterdam, Paris sowie Zürich, initialisiert. Zum Vergleich werden Pingtests über einen externen Anbieter aus Nürnberg durchgeführt. Neben der feingranularen graphischen Darstellung, über die Ausführungsdauer jedes Tests, wird auch die durchschnittliche Dauer aller durchgeführten Tests berechnet. Auf dieser Weise ist eine sinnvolle Aussage über die Latenzzeiten einfacher zu treffen

und diese zu vergleichen.

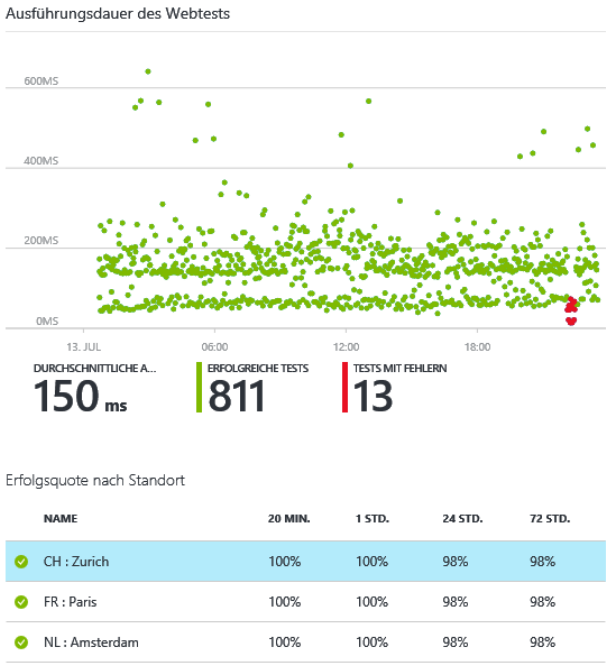


Abbildung 17: Ping Tests Amsterdam-Amsterdam-Zürich-Paris

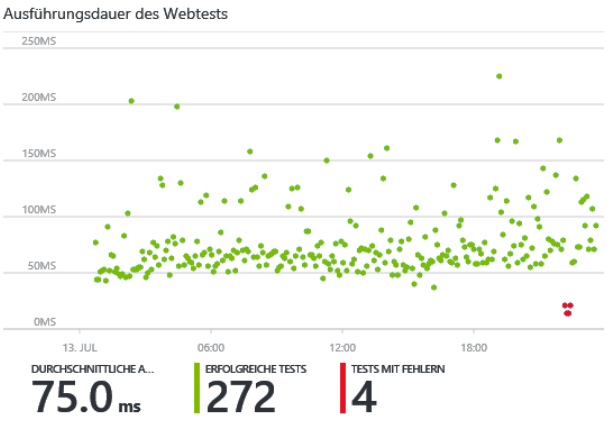


Abbildung 18: Ping Tests Amsterdam-Amsterdam

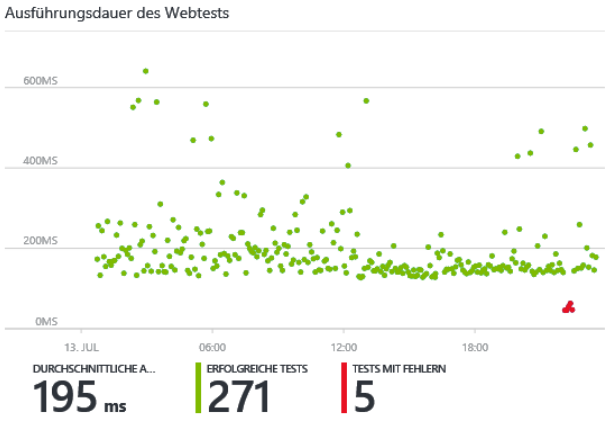


Abbildung 19: Ping Tests Amsterdam-Paris

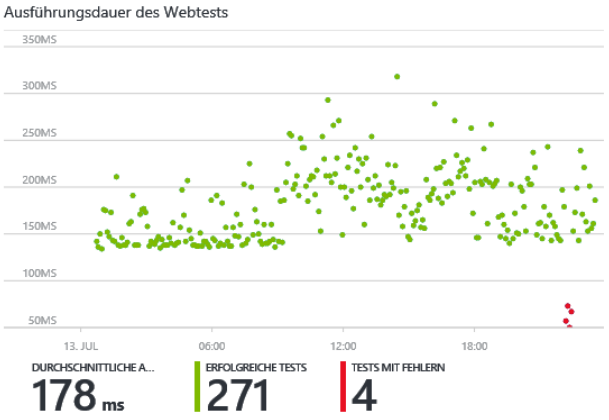


Abbildung 20: Ping Tests Amsterdam-Zürich

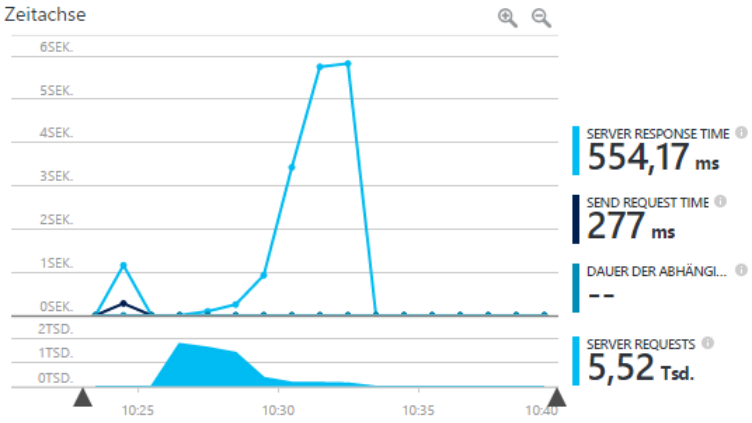
Insgesamt lässt sich feststellen, dass die inländischen Verbindungen, die besten und kleinsten Werte liefern. Im Durchschnitt beträgt die Latenz etwa 75ms. Bei unseren Messungen benötigte die Verbindung zwischen Amsterdam-Paris die längste Reaktionszeit. Somit lässt sich leicht ein Zusammenhang zwischen der Entfernung der Standorte und die dabei entstehende Latenzzeit feststellen. Aus diesem Grund lässt sich, die im Rahmen dieser Arbeit größte gemessene Durchschnitt-Latenzzeit auf die große Entfernung zurückführen.

Anhand der Abbildungen erkennt man auch Latenzzeiten, die große Abweichungen von der Durchschnittszeit aufweisen. Über die Entstehungsgründe lässt es sich nur spekulieren. Eine mögliche Erklärung dafür könnte sein, dass die Verbindung unter Umständen auf den verschiedenen Webknoten geleitet wird, was zu diesen (erstmal) unerwarteten Schwankungen führen kann.

Die Aussage, ob die getesteten Systeme als performant bezeichnet werden können, ist natürlich abhängig von den definierten Erwartungswerten. Sollte auf diesen Systemen, die ausgeübte Last nicht die getestete Last überschreiten, führt dies zu der Aussage, dass das Ergebnis mehr als zufriedenstellend betrachtet werden kann und der Performance-Test somit als bestanden gilt.

Im Folgenden werden die Ergebnisse, nach dem Szenario aus der Abbildung 16, präsentiert. Als Webanwendung wurde eine ASP.NET 4.5 WebApi Anwendung ausgewählt, die CRUD-Operationen unterstützt. Zu erwähnen ist auch, dass die Tests punktuell durchgeführt wurden. Punktuell bedeutet in diesem Kontext jeweils Vormittags und Nachts, um verschiedene Lastzeiten zu testen. Die Anwendung wurde über die gesamte Testlaufzeit nicht mehr angepasst bzw. verändert mit dem Ziel möglichst genaue und echte Vergleiche der Performance und Latenzeiten zwischen der einzelnen Clouds durchzuführen.

In den folgenden Abbildungen werden die Ergebnisse der Performancemessungen abgebildet. Auffällig sind insbesondere die Ergebnisse Applikation (Amsterdam) - Datenbank (Dublin), wo die Server Requests Time bei nur 41,95ms liegt. Dieser Wert ist deutlich niedriger verglichen mit den aus den anderen Testszenarien. Über die Ursache solcher Ergebnisse kann man nur schätzen. In diesem Fall empfiehlt es sich, mehrere Lasttests über längere Zeiten durchzuführen. Eine weitere Auffälligkeit bei den Messergebnissen waren die sehr hohen Standardabweichungen, die zwei bis vier Mal größer waren als die eigentlichen Messergebnisse. Die tatsächliche Ursache der Messungenauigkeit kann aus den nicht ausreichenden Anzahl an Server Requests zurück zuführen sein. Die oberen Ergebnisse sind vor allem Durchschnittswerte. Application Insights erlaubt jedoch die genaue Messung einzelner CRUD-Operationen. Auffällig sind die langen Ladezeiten des GET Home/Index-Aufrufs, die charakteristisch für alle bisherigen Messungen sind.



Durchschnitt von server response time nach operation name

OPERATION NAME	DURCHSCHNITT	ANZAHL	STANDAR...
GET Home/Index	1,62 Sek.	1.8 K	4,25 Sek.
POST Home/Create	78,44 ms	863	157,18 ms
GET bundles/jquery	12,99 ms	250	181,3 ms
GET Home/Create	12,45 ms	929	58,9 ms
GET bundles/jqueryval	1,9 ms	891	4,7 ms
GET Content/css	1,47 ms	250	2,77 ms
GET bundles/bootstrap	1,46 ms	250	2,01 ms
GET bundles/modernizr	1,33 ms	250	1,83 ms

Abbildung 21: LoadTests Azure Applikation Amsterdam SQL-Datenbank AWS Frankfurt ca. 10:30Uhr

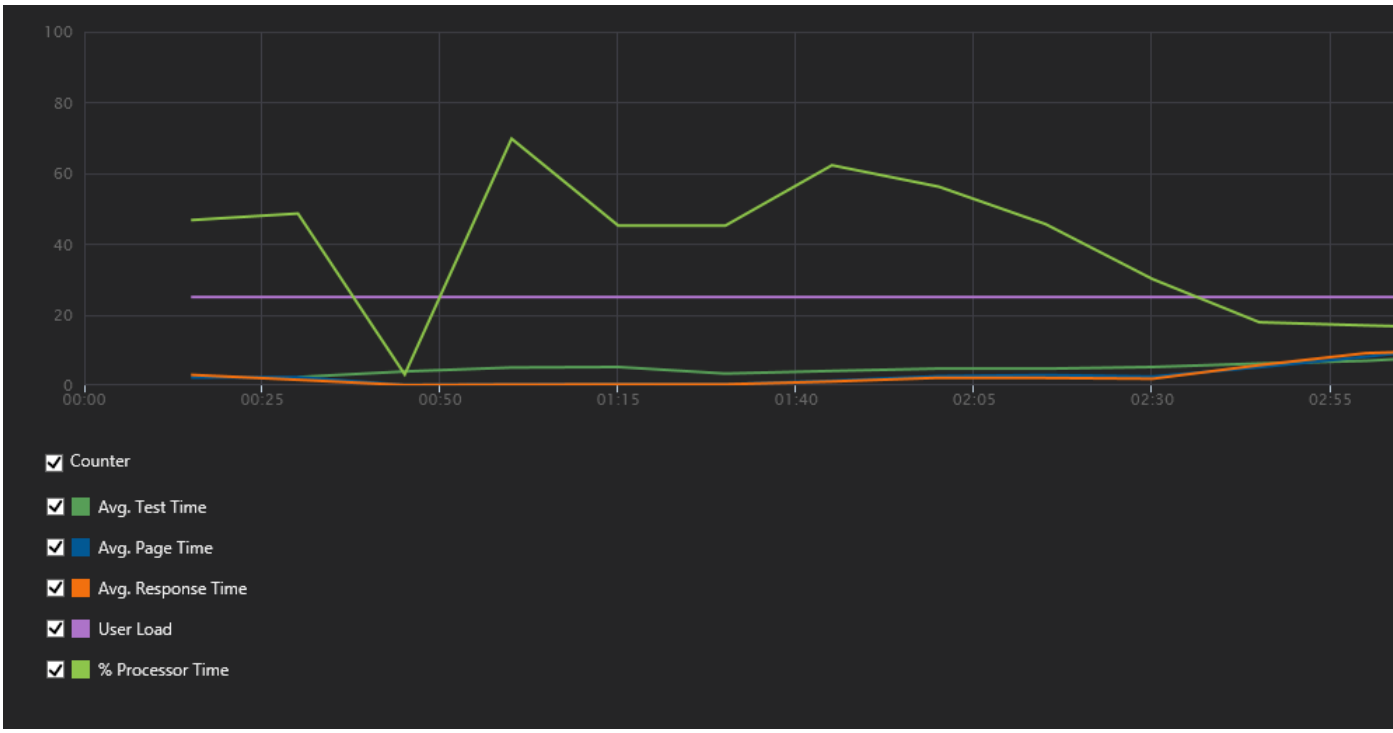
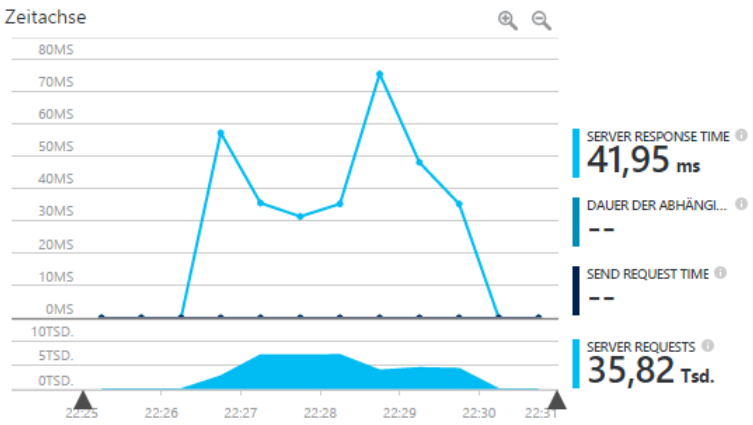


Abbildung 22: LoadTests Azure Applikation Amsterdam SQL-Datenbank AWS Frankfurt ca. 10:30Uhr



Durchschnitt von server response time nach operation name

OPERATION NAME	DURCHSCHNITT		ANZAHL	STANDAR...
GET Home/Index		42,82 ms	35 K	157,36 ms
GET bundles/modernizr		5,77 ms	200	24,93 ms
GET Content/css		4,69 ms	200	16,67 ms
GET bundles/jquery		2,69 ms	200	6,75 ms
GET bundles/bootstrap		1,88 ms	200	5,48 ms

Abbildung 23: LoadTests Azure Applikation Amsterdam SQL-Datenbank Azure Dublin ca. 23:00Uhr

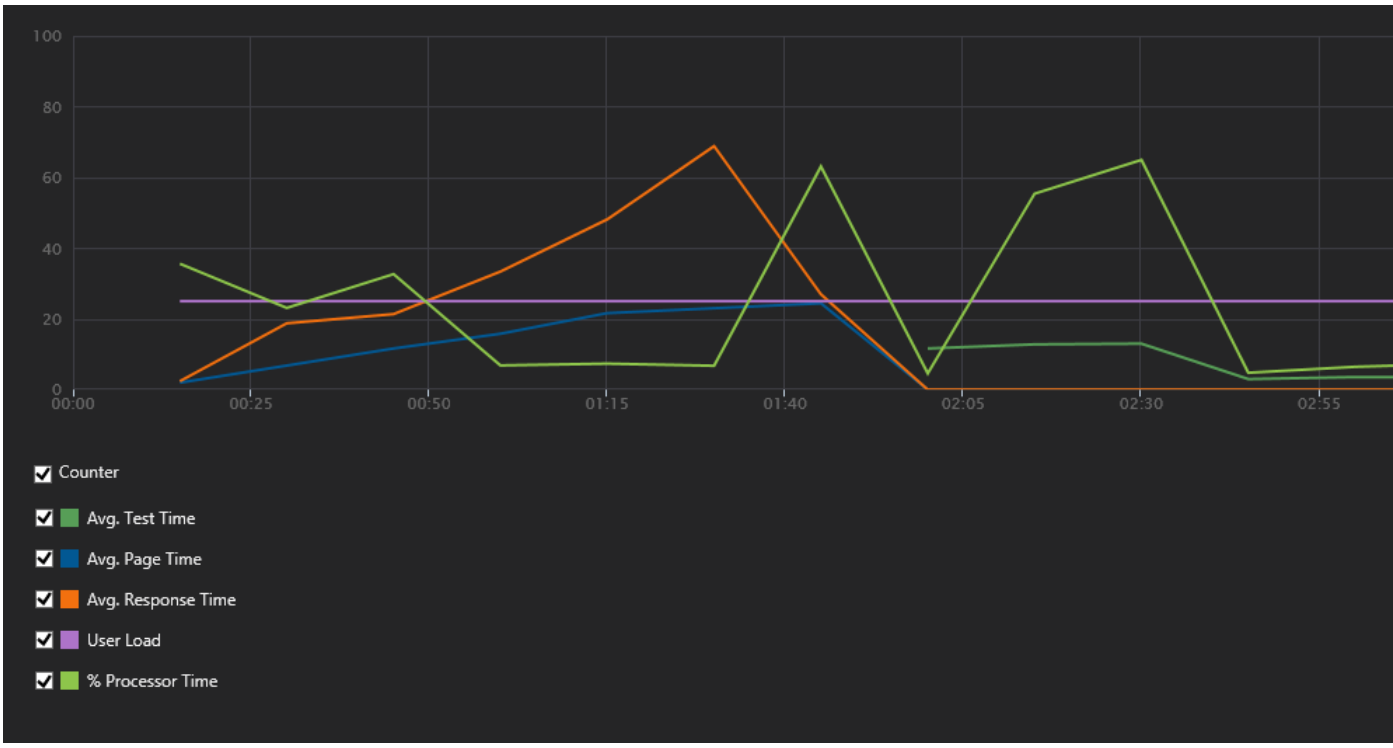
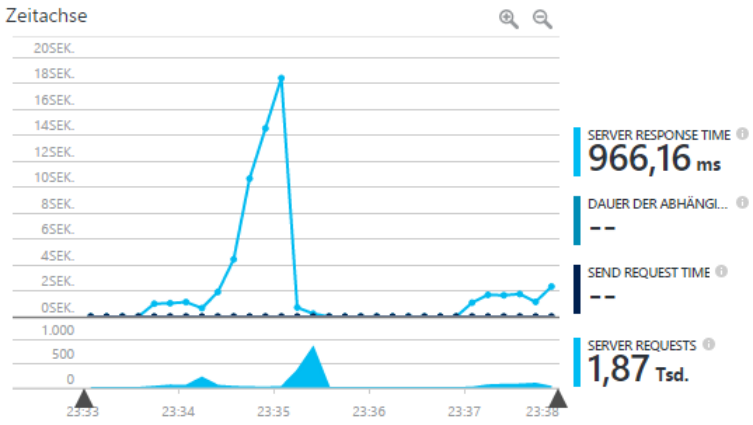


Abbildung 24: LoadTests Azure Applikation Amsterdam SQL-Datenbank AWS Frankfurt ca. 23:00Uhr



Durchschnitt von server response time nach operation name

OPERATION NAME	DURCHSCHNITT	ANZAHL	STANDAR...
GET Home/Index	4,09 Sek.	419	5,36 Sek.
GET Home/Create	143,82 ms	247	400,18 ms
GET bundles/jqueryval	57,68 ms	204	243,27 ms
GET bundles/jquery	53,88 ms	250	349,96 ms
GET Content/css	52,4 ms	250	424,79 ms
GET bundles/bootstrap	37,03 ms	250	293,32 ms
GET bundles/modernizr	33,27 ms	250	267,52 ms

Abbildung 25: LoadTests Azure Applikation Amsterdam SQL-Datenbank AWS Frankfurt ca. 23:00Uhr

7 Schlusswort

In Kooperation mit dem Unternehmen Datev eG als Auftraggeber und dem AMOS-Team1 als Umsetzungspartner, sollte eine multi-cloudfähige Architektur bzw. Anwendung entstehen. Im Rahmen dieser Projektarbeit wurde die Architektur aufgebaut.

Im ersten Schritt wurden zunächst die Randbedingungen festgelegt, die gestellten Anforderungen analysiert und Szenarien abgeleitet. Basierend auf den abgeleiteten Szenarien wurden neue Entwicklungsartefakte produziert und ein Architekturmodell im Stil des 4+1 Sichtenmodells von Phillipe Kruchten entwickelt. Im Laufe der Bearbeitungszeit bildeten die entstandenen Artefakte iterativ die logische, dynamische, Realisierungs- und Verteilungssicht. Auftretende Änderungen seitens der Implementierung wurden in der Architektur entsprechend modifiziert.

Die ursprüngliche Produktvision, eines cloud-unabhängigen, multi-tenancy-fähigen Systems zur Speicherung von Daten in verschiedenen Cloud-Diensten zu implementieren, wurde erreicht. Angefangen bei der Umsetzung einer AWS-konformen Schnittstelle, deren Struktur als Grundlage der eigenen generischen Storage-Schnittstelle diente, erfolgte die konkrete Umsetzung mit Hilfe von Python-Frameworks (Flask). Die Lösung, die bei der anfänglichen Umsetzung der AWS-konformen Schnittstelle erfolgte, wurde im weiteren Verlauf auch auf alle anderen Cloud-Plattformen migriert.

Im Laufe des Projektes wurden einige Probleme festgestellt. Beispielsweise wurde die hohe Komplexität der Datenverteilung unterschätzt. Während ursprünglich u. a. die Synchronisation der User-Sessions geplant war, musste aufgrund des engen Zeitplans diese Anforderung verworfen werden. Des Weiteren wurden Aspekte wie Zustandstransfer und Wiederherstellung, die in Betracht gezogen werden sollten im Hinblick auf den produktiven Einsatz der Software, außer Acht gelassen. Diese könnten z. B. den Ausfall einer Instanz tolerieren und durch das Starten einer neuen Instanz den Ausfall kompensieren.

Bei Betrachtung der Cloud-Unabhängigkeit ist die komplette Abstraktion von allen Cloud-Plattformen schwer bis gar nicht zu ermöglichen. Jeder Anbieter entwickelt an möglichen Standards vorbei, um sich einen möglichen Wettbewerbsvorteil am Markt zu sichern. Sollte z.B. die Amazon Web Services (Referenz-Plattform des AMOS-Teams) eine Funktionalität anbietet, die kein anderen Anbieter zur Verfügung stellt, so muss an der eigenen Schnittstelle vorbei programmiert werden, um die anderen gewünschten Plattformen einbeziehen zu können.

Auch Qualitätsanforderungen werden immer wichtiger. Mit dem kontinuierlichen Wachstum der Benutzeranzahl im Web steigt auch das Bedürfnis nach möglichst schnellen, fehlerfreien und perfomanten Systemen, die zuverlässig und ausfallsicher für den Benutzer da sind. Aus diesem Grund sind die performancespezifischen Tests von enormer Bedeutung.

Die Tests wurden anhand eines konkreten Beispiels bzw. einer vom Industriepartner vor-

gegebenen Anwendung (Brutto-Netto-Rechner) auf die zu testenden Clouds ausgeführt. Zwischen den zahlreichen Performancetools, die alle ihre Schwächen und Stärken aufweisen, wurde letztlich Microsoft Studio Application Insights eingesetzt, der sich für den Einsatzzweck bestens geeignet hat. Entstandene Testdaten wurden zur Analyse gesammelt und ausgewertet. Die Auswertungsergebnisse wurden in Rahmen dieses Berichts ausführlich beschrieben.

8 Glossar

Begriff	**Erklärung**
ACL	Abkürzung- AMOS Cloud Layer
ACL Storage Interface	Generische Schnittstelle, die verwendet wird, um mit den cloudspezifischen Schnittstellen zu kommunizieren.
AMOS Cloud Layer	Das Produkt, das die Architektur in diesem Dokument beschreibt. Es bietet eine Plattform zur Datenspeicherung in einer Multi-Cloud-Umgebung an.
Artefakt	Physische Datei, die beim Entwicklungsprozess erstellt wird.
AWS	Abkürzung-Amazon Web Services
Etcd	Verteilter Key-Value-Store, der zur Koordination der Synchronisierung verwendet wird.
Goal-Question-Metric	Ein Verfahren zur Erfassung von Metriken, indem man ein Ziel aufstellt, das erreicht werden soll. Dazu erstellt man Fragen, dessen Antwort das Ziel erreichen soll und definiert dazu passende Metriken
Kruchtens 4+1 Sichtenmodell	Modell zur Beschreibung von Softwarearchitekturen mittels verschiedener Sichten.
Microsoft Azure	Cloud-Plattform von Microsoft
Multi-Cloud-System	Ein System, das in der Lage ist mit verschiedenen Cloud-Umgebungen zu arbeiten ("Cloud of Clouds").
UML	Unified Modeling Language - ist eine Modellierungssprache zur grafischen Darstellung von Spezifikationen und zur Dokumentationen von Softwaresystemen.