Konzeption eines Softwaresystems zur Verwaltung von Hochschulsporteinrichtungen

Diplomarbeit von Dirk Beckmann

TECHNISCHE UNIVERSITÄT DARMSTADT

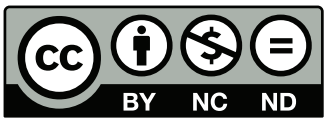
Fachbereich Humanwissenschaft
Institut für Sportwissenschaft

Konzeption eines Softwaresystems zur Verwaltung von Hochschulsporteinrichtungen

Vorgelegte Diplomarbeit von Dirk Beckmann

1. Gutachten: Prof. Dr. Joseph Wiemeyer
2. Gutachten: Dietbert Schöberl

Tag der Einreichung:



Inhaltsverzeichnis

1 Einleitung

2 Analyse der Fachdomäne Hochschulsport

Die Aufgaben und Anforderungen, die an deutsche Hochschulen gestellt werden sind vielfältig und erstrecken sich in viele verschiedene Bereiche. Neben den Kernaufgaben universitäre Lehre und wissenschaftliche Forschung gehört es auch zu den Aufgaben das studentische Leben in sozialen und gesellschaftlichen Belangen zu unterstützen und fördern. Neben Einrichtungen wie dem Studierendenwerk oder Allgemeinen Studierendenausschuss (AStA) ist auch der Hochschulsport eine Einrichtung die unterschiedlichste Aufgaben im Umfeld der Fachhochschulen und Universitäten übernimmt. Dieser Teil soll ein Verständnis für die Domäne Hochschulsport und deren Entwicklung, Organisationsformen, Aufgaben sowie Probleme und Herausforderungen vermitteln.

2.1 Historische Entwicklung des Hochschulsports

Das Sporttreiben an deutschen Hochschulen begann zwar nicht erst nach dem zweiten Weltkrieg, die historische Betrachtung soll sich hier jedoch auf die Zeit zwischen 1948 und 2009 beschränken, um den Rahmen der Arbeit nicht zu sprengen. Da die Organisation von Bildung in die Zuständigkeit der Bundesländer fällt, hat die Entwicklung durchaus unterschiedliche Wege genommen. Ich konzentriere mich in diesem Fall auf den geschichtlichen Rückblick mit dem Schwerpunkt NRW, da hier die umfangreichsten Dokumentationen verfügbar waren. Der 1999 von von 29 Bildungsministern initiierte Bologna-Prozess zur Schaffung eines einheitlichen Hochschulraums (vgl. QUELLE)

2.1.1 Hochschulsport heute

Durch die gesetzliche Verankerung im Hochschlrahmengesetz und die Festschreibung in den einzelnen Landeshochschulgesetzen hat sich eine vorerst gesicherte Grundlage zur Förderung des Sports an Hochschulen manifestiert. Die einzelnen Hochschulsporteinrichtungen besitzen

eine gefestigte Position im Hochschulkontext und müssen nicht mehr um ihre grundlegende Daseinsberechtigung kämpfen. Das Grundanliegen des HSP ist und bleibt jedoch die Förderung des Sporttreibens der Studierenden, die sich in speziellen Lebensphasen befinden und durch besondere Umstände gekennzeichnet sind. Das diese nicht generalisiert werden können, lässt sich leicht nachvollziehen, da unterschiedliche Studiengänge, Wohnumstände und Studiumsformen die Lebensphase in sehr heterogener Weise beeinflussen können. Der Einfluss auf den Hochschulsport und im speziellen auf die Akzeptanz und Teilnahme lässt sich an folgenden Kriterien festmachen, die speziell bei der Hochschulsportorganisation explizit berücksichtigt werden müssen:

1. Sportinteresse des Studierenden
2. Ort der Sporteinrichtungen
3. Soziale Kontakte
4. Belastungen und Entbehrungen

Somit ergibt sich für den Hochschulsport heute die besondere Herausforderung den Bedürfnissen der Studierenden gerecht zu werden und ihnen die Gelegenheit zu geben, gemäß ihrer Interessen und Vorstellungen Sport treiben zu können.
Der bereits beschriebene Bologna-Prozess führte allerdings zu zusätzlichen Herausforderungen im Verhältnis Hochschulsport-Studierende. Die Verkürzung der Studienzeit führte zu einem strenger vorgegebenen Studienplan und einer deutlichen Verknappung der eigenen Gestaltungs- und Freizeitmöglichkeiten. Diese Faktoren beeinflussen die Möglichkeiten der Studierenden an der Teilnahme im Hochschulsport erheblich, sodass sich des Hochschulsportprogramm diesen speziellen Anforderungen noch stärker durch mehr Flexibilität und neue Angebotsformen stellen muss.

Neben den Studierenden gehören aber zunehmend auch andere Personenkreise zur Zielgruppe des Hochschulsports. Besonders die Bediensteten der Hochschule werden zunehmend in den Programmen mit speziellen Programmen angesprochen. Speziell im Bereich Gesundheitssport haben sich hier eigene Programme an den Hochschulen oder auch durch übergeordnete Gremien, wie den allgemeinen deutschen Hochschulsportverband, etabliert. Das Angebot ist dabei

sehr unterschiedliche und stark abhängig von den verfügbaren Ressourcen, erstreckt sich aber von der Teilnahme am allgemeinen Programm über spezielle Bediensteten Kurse bis hin zur individuellen Betreuung am Arbeitsplatz im Büro. Die Nachfrage nach Gesundheitsfördernden Maßnahmen dieser Art erfreut sich derzeit einer hohen Nachfrage.
Neben den Bediensteten lassen sich aber auch Externe Teilnehmer als weitere Zielgruppe bestimmen. Die Einbindung dieser Personenkreise unterscheidet sich am stärksten in den Unterschiedlichen Einrichtungen und birgt großes Potenzial und Gefahr gleichzeitig. Je nach Organisationsform stehen dem Hochschulsport ganz unterschiedliche Möglichkeiten offen aber auch das regionale Umfeld muss hier genau in Betracht gezogen werden. Mit der Öffnung des Hochschulsports für externe Teilnehmer begibt man sich in direkte Konkurrenz mit kommerziellen Anbietern. Die Integrationsformen variieren von Kooperationen mit Vereinen, Firmen im Rahmen von Betriebssportangeboten, Rehakliniken und Gesundheitszentren bis hin zur freien Öffnung für Jedermann. Allgemein lässt sich jedoch beobachten, dass diese Gruppe vor allem dann angesprochen wird, wenn eine Auslastung durch Bedienstete und Studierende nicht möglich ist. Im Bereich Örganisationsformen"werde ich auf die Unterschiede noch detaillierter eingehen.

2.2 Typische Aufgaben

2.3 Organisationsformen

Die zentralen Einrichtungen
die – unabhängig von einem Fachbereich und direkt dem Senat unterstellt – den Hochschulsport eigenständig organisieren und verwalten

Die in das Aufgabenfeld der Institute für Sportwissenschaft integrierte Organisation des Hochschulsports
(diese enge Verbindung mit der Sportlehrer/innenausbildung kann sich auf das Hochschulsportangebot äußerst positiv auswirken, bringt aber auch durch notwendige „Prioritätensetzungen“ die Gefahr der Benachteiligung mit sich);

Der durch studentische Selbstverwaltung organisierten Hochschulsport

(diese Form hat sich vor allem an kleineren Hochschulen bzw. Fachhochschulen etabliert, wo keine hauptamtlichen Sportlehrer/innen zur Verfügung stehen) (vgl. Radde, 1996)

2.4 Befragung ausgewählter Standorte

2.5 Wichtige Geschäftsprozesse

1. Kunden anlegen
2. Kunden verifizieren
3. Kurse anlegen
4. Kurse finden
5. Kurse darstellen
6. Kurse buchen
7. Geld einziehen
8. Rechnungsstellung
9. Teilnehmer informieren
10. Übungsleiter abrechnen
11. Sportstätten vermieten
12. Verträge verwalten
13. Zutritt überprüfen
14.
15.

2.6 Probleme und Herausforderungen

3 Cloud Computing

Das letzte Kapitel hat deutlich gemacht, welche Anforderungen im Hochschulsport an ein Software System gestellt werden, sowie welchen Dynamiken und Besonderheiten das fachliche Umfeld unterliegt. In diesem Kapitel soll ein Verständnis für unterschiedliche Softwarearchitektur und Vertriebsmodelle geschaffen werden. Dabei soll vor allem auf das Thema Cloud Computing eingegangen werden, welches an Bedeutung gewinnt und Gegenstand vieler akademischen und wirtschaftlichen Beiträgen in der Fachpresse ist.

Für den Begriff Cloud Computing finden sich in der Literatur vielfältige Definitionen. [Beispiel] Die häufigste Verwendung findet jedoch die Definition des National Institute of Standards and Technology (NIST), dessen Definition sich auch die European Network and Information Security Agency (ENISA) angeschlossen hat, in der Form:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.*

(vgl. Mell & Grance, 2011, S.2)

Diese Definition des ist auch Grundlage für die Definition des Bundesamt für Sicherheit in der Informationstechnik (BSI) welches den Begriff "Cloud Computing"folgendermaßen festlegt:

> *Cloud Computing bezeichnet das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz. Angebot und Nutzung dieser Dienstleistungen erfolgen dabei ausschließlich über definierte technische Schnittstellen und Protokolle. Die Spannbreite der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software.*

(vgl. Bundesamt für Sicherheit in der Informationstechnik, o. J.)

3.1 Merkmale

3.1.0.1 On-demand self-service

3.1.0.2 Broad network access

3.1.0.3 Resoruce pooling

3.1.0.4 Rapid elasticity

3.1.0.5 Measured service

(vgl. Mell & Grance, 2011, S.2)

3.2 Service Modell

In der Definition von (Mell & Grance, 2011, S.2) sind drei unterschiedliche Service Modelle beschreiben, die sich in der Cloud Community etabliert haben. Diese decken sich mit den Kategorisierungen von (Tharam, Chen & Elizabeth, 2010, S. 28) und (Jadeja & Modi, 2012, S. 878).

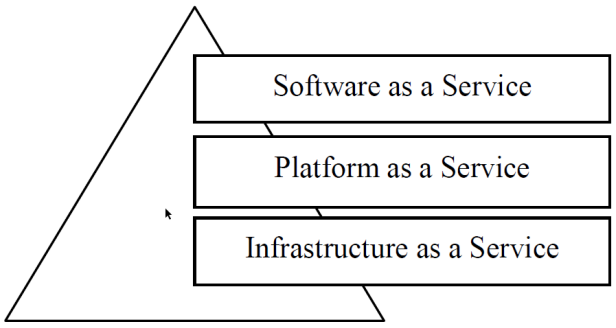


Abbildung 3.1: 3-Ebenen Modell von Cloud Computing

Software as a Service (SaaS) stellt eine Anwendung in der Cloud bereit, die von Nutzern verwendet werden kann, ohne eine eigene Installation vornehmen zu müssen. Der Nutzer hat dabei keinen Zugriff auf die Cloud Infrastruktur. Wichtige Merkmale einer SaaS Anwendungen sind Netzwerk basierter Zugriff, Nutzer können limitierte spezifische Konfigurationen vornehmen und sind meistens als Multi-Tenancy-System aufgebaut. Beispiele sind Anwendungen wie SalesForce.com, Google Mail, Google Docs, Office 365, Quicken Online, etc.

Platform as a Service (PaaS) bezeichnet Software und Entwicklungs Tools die der Nutzer verwenden kann, um eigene Anwendungen zu entwickeln und zu veröffentlichen. Der Nutzer hat dabei keinen Zugriff auf die darunter liegende Cloud Infrastruktur, kann jedoch im Vergleich zu SaaS die eigene Anwendung frei konfigurieren. Beispiele für PaaS Services ist Google AppEngine.

Infrastructure as a Service (IaaS) ist die unterste Ebene der Cloud Computing Pyramide. Dem Nutzer wird virtualisiert Hardware wie Datenspeicher, Virtuelle Maschinen, Netzwerke, etc. zur Verfügung gestellt, die frei verwendet werden können. IaaS Services werden gewöhnlich auf pay-per-use Basis bezahlt. Amazon EC2, VMWare und Windows Azure sind nur einige Beispiele.

In einigen Fällen werden diese drei Ebenen noch um zusätzliche Service Formen ergänzt. So verwendet (Tharam et al., 2010, S. 28) noch die Einteilung in storage as a Service (DaaS) während (Mahmood, 2011, S. 123) von öther Provision as Serviceßpricht und darunter Formen wie Storage as a Service, Database as a Service, Security as a Service, Communication as a Service, etc gruppiert. All diese Services können jedoch auch spezialisierte IaaS Services angesehen werden.

3.3 Bereitstellung Modell

Ein weiterer zentraler Punkt in der Definition von Cloud Computing ist das Bereitstellungsmodell, das primär die Art beschreibt, wer Zugriff auf die Daten und Anwendungen hat.

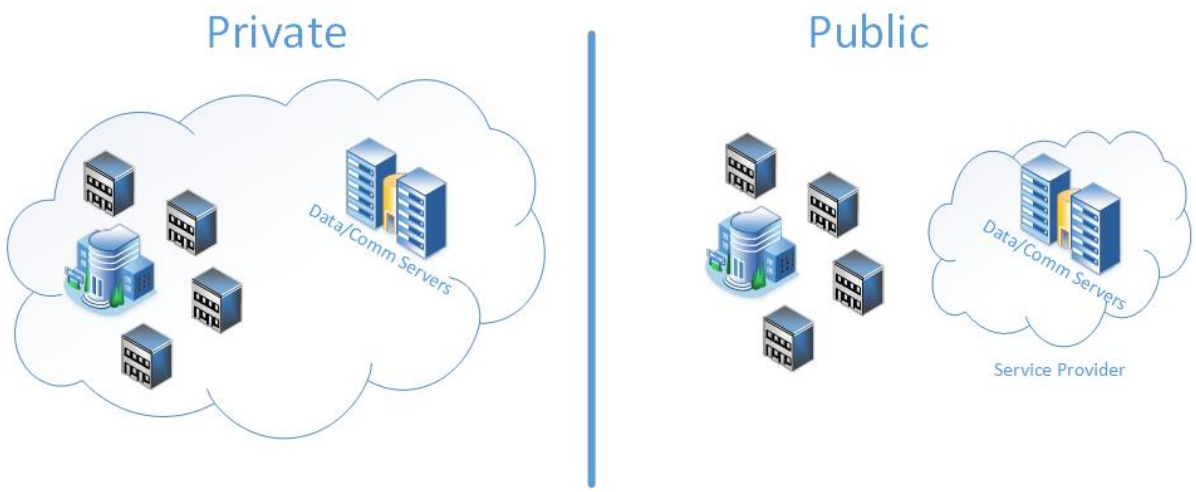


Abbildung 3.2: Public und Private Cloud Architektur

3.3.1 Private Cloud

Als Private Cloud wird das Szenario beschrieben, wenn die Infrastruktur nur von einem Kunden genutzt wird. Im Normalfall ist sie im Organisationseigenen Rechenzentrum beheimatet. Sicherheit, Datenschutz, Wartung sind so einfacher durch eigenes Personal oder externe Dienstleister zu organisieren. Die Private Cloud lässt sich daher mit einem Intranet vergleichen. (Jadeja & Modi, 2012, S. 879)
Tharam et al. (2010) nennen fünf Gründe, die für eine Private Cloud sprechen können:

1. Maximieren und optimieren bestehender, eigenen Ressourcen
2. Sicherheitsbedenken und Datenschutz
3. Datentransferraten
4. Vollständige Kontrolle über kritische Aktivitäten
5. Forschungs- und Ausbildungszwecke

3.3.2 Public Cloud

Dieses Modell ist derzeit am häufigsten verbreitet. Nutzer können die Cloud über ein Interface ansteuern und bezahlen für die Dauer der genutzten Services. Im Vergleich zu anderen Modellen

ist die Public Cloud weniger sicher, da sie offen für bösartige Angriffe ist. Bekannte Public Cloud Anbieter sind Amazon EC2, S3, Google AppEngine, Force.com und Microsoft Azure.
(vgl. Jadeja und Modi (2012) und Tharam et al. (2010))

3.3.3 Hybrid Cloud

Bei der Hybrid Cloud handelt es sich aus einer Kombination aus zwei oder mehr Cloud Arten, die für sich selbst existieren, aber miteinander verbunden sind. Dadurch lassen sich einfach eine Private und Public Cloud kombinieren, sodass Sicherheitsanforderungen gerecht geworden werden kann ein öutsourcing"weniger kritischer Teile in eine Public Cloud jedoch nichts im Wege steht.
Hybrid Clouds sind die Treiber zur Notwendigkeit einer Standardisierung und Cloud Interoperabilität.

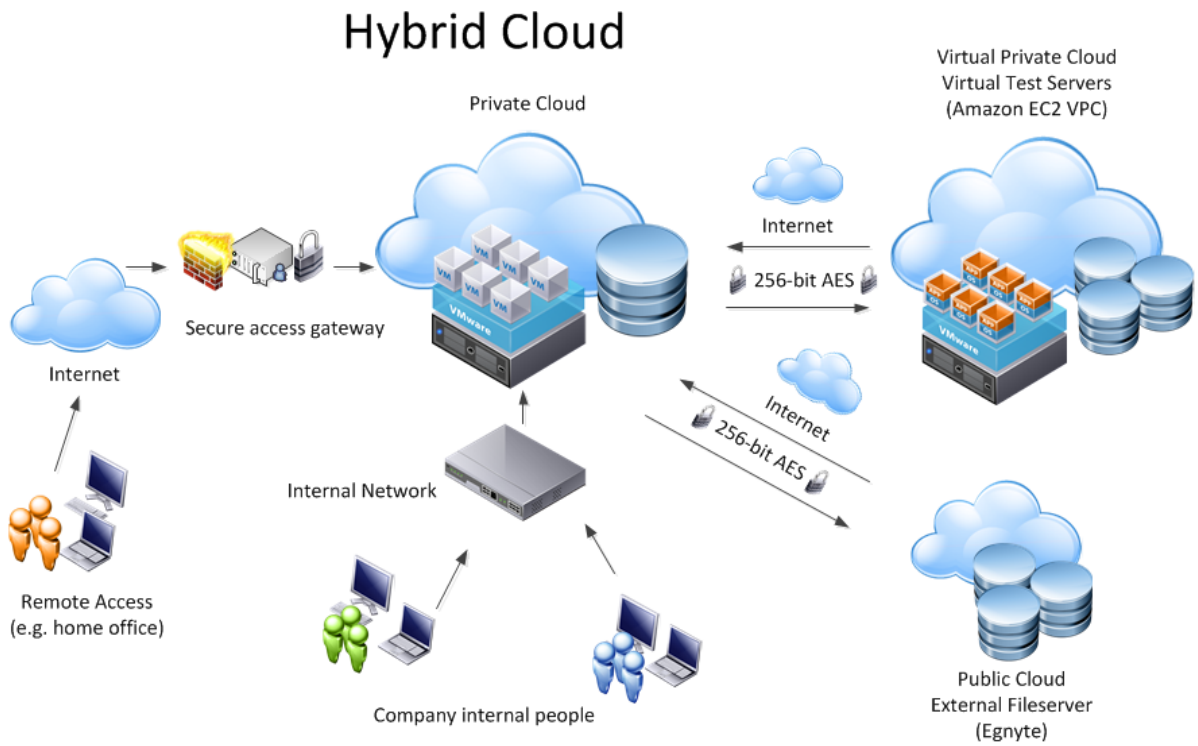


Abbildung 3.3: Hybrid Cloud Architektur

Community Cloud

3.4 Pro & Contra

Nachdem der Begriff Cloud Computing, sowie die beinhalteten Service und Bereitstellungs Modelle erläutert wurden stellt sich die Frage nach den Vor- und Nachteilen dieses neuen Modelles. Wie bei den meisten neuen Technologien und Entwicklungen ist die Entstehung getrieben aus den Unzulänglichkeiten bestehender Ansätze und so können die Merkmale, die diese Neuentwicklung definieren in den meisten Fällen auch als Vorteile angesehen werden. Nachdem einige Gründe der Notwendigkeit bereits in Abschnitt **??** erläutert wurden sollen hier die Vor- und Nachteile jedoch noch einmal explizit zusammen gefasst werden. Der Fokus liegt hierbei vor allem auf den Services IaaS und PaaS, die etwas anders betrachtet werden können als SaaS. Auch wenn viele der Vor- und Nachteile auch für SaaS Modelle gelten, werden diese im Abschnitt 3.5 noch einmal explizit betratet. Verglichen werden dabei Cloud Anbieter mit dem traditionellen Rechenzentrum.

3.4.1 Vorteile

3.4.1.1 Einfachheit

Cloud Angebote leben von ihrer einfachen Einrichtung. Speziell als Anwender bzw. Kunde genügt in den meisten Fällen ein Webbrowser, um sich auf der entsprechenden Oberfläche die benötigten Ressourcen anzufordern und automatisiert bereit gestellt zu bekommen. Es handelt sich hier um eine Art Outsourcing, sodass die Fachkenntnis auf Hardware- und Softwareebene nicht mehr in der Tiefe notwendig ist, wie sie bei einem eigenen Betrieb vorhanden sein muss, um ein gleichen Qualitätslevel zu erreichen.

3.4.1.2 Kostenersparnis

Der Faktor Kostenersparnis ist wohl einer der größten Vorteile, wobei die Berechnung der Kostenersparnisse für IT nicht all zu trivial, da sich je nach Modell diese ggf. erst nach Jahren rentieren. Die Kosten im eigener Verantwortung gegeben sich primär aus den Beschaffungskosten für Hardware und Software und zusätzlich anteilig an den Personal- oder Dienstleister

Kosten für Wartung und Instandhaltung. Mit einbezogen werden muss dabei jedoch auch, dass ggf. Neuanschaffungen und Erweiterungen über die Jahre notwendig sind.
Die Preismodelle von Cloud Anbietern wie Amazon Web Services (AWS), Microsoft Azure (Azure) oder Google AppEngine (AppEngine) unterscheiden sich dabei sehr von den Traditionellen Abrechnungsmodellen. Auch wenn nicht jeder Anbieter alle Modelle unterstützt, so haben sich sich doch Abrechnungen nach Datenvolumen (wie viel Speicherplatz wird bereit gestellt), Computer Ressourcen (wie viel Arbeitsspeicher und CPU wird beansprucht), Datentransfervolumen (wie viel Datenvolumen wird über das Netzwerk transferiert) und einige Andere in der Praxis etabliert. Der große Unterschied liegt jedoch im Zusammenhang mit der Skalierbarkeit. So wird im Gegensatz zum traditionellen Rechenzentrum nur das bezahlt, was wirklich benutzt wird und überschüssige Ressourcen können bei bedarf einfach zurück gegeben werden. In der Literatur gibt es für diese Abrechnungsform jedoch bisher keinen Einheitlichen Begriff und wird als utility-style pricing, pay-as-you-go oder pay-per-use bezeichnet. (vgl. Jadeja und Modi, S. 879; Achimugu, Oluwagbemi und Popoola, S. 313)
Wie groß die Kostenersparnisse im Detail sein können ist sehr abhängig von den Preisen, der Anwendungsarchitektur und der kalkulierten Laufzeit. In den letzten Jahren hat sich bei den Anbietern Amazon, Microsoft und Google ein richtiger Preiskrieg entwickelt, der zu stetig fallenden Preisen geführt hat und so die Kostenersparnisse weiter vergrößert hat.

3.4.1.3 Skalierbarkeit und Flexibilität

Elastizität und Skalierbarkeit ist ein weiter Hauptvorteil von Cloud Umgebungen. Datenspeicher und Computing Power steht in fast unbegrenzter Menge zur Verfügung und kann mit wenigen Handgriffen vom User über einen Self-Service dazu provisioniert werden. So wie die Erweiterung der Ressourcen einfach möglich ist, so lassen sich diese auch wieder zurück geben wenn sie nicht mehr benötigt werden. Zusammen mit einer pay-per-use Abrechnung führt dies dazu, dass zusätzlich Ressourcen ggf. nur für bestimmte Zeiträume in Anspruch genommen werden müssen.
In traditionellen Rechenzentren ist dies schwieriger. Werden neue Ressourcen benötigt, so stehen diese nur begrenzt zur Verfügung. Durch unterschiedliche Virtualisierung Umgebungen hat dies zwar auch zu mehr Flexibilität geführt, jedoch sind die Möglichkeiten begrenzt. Auch die

Zeit, die bis zur Bereitstellung neuer Ressourcen benötigt wird ist im Normalfall deutlich länger, da der Prozess weniger automatisiert abläuft.

3.4.1.4 Zukunfts- und Ausfallsicherheit

Ausfälle von Hardware ist in der IT ein Fakt der sich nicht vermeiden lässt und unterscheidet sich nicht in Cloud und eigenen Rechenzentren. Gerade in den großen Rechenzentren mit tausenden von Servern ist die Wahrscheinlichkeit für ein auftreten eines solchen Ausfalles natürlich deutlich höher. Mit Redundanz und das Vermeiden von single point of failures wird in Rechenzentren versucht den Hardware Ausfall für den Endkunden unsichtbar zu machen und den Betrieb ungestört aufrecht erhalten zu können.
Durch die reine Anzahl von Hardware und spezialisiertem Personal sind die Cloud Anbieter in ihren Rechenzentren hier deutlich im Vorteil einen störungsfreien Betrieb zu garantieren. Zudem ist durch den Austausch kaputter oder veralteter Hardware eine sukzessive Erneuerung der Hardware ohne weiter Kosten für den Endkunden gegeben.

3.4.1.5 Nachhaltigkeit

Green IT, der umweltschonende Betrieb und Herstellung von Informationstechnologie, ist in den letzten Jahren von immer größerer Bedeutung geworden. Cloud Computing kann in diesem Bereich einiges beitragen, da Energiekonzepte einfacher optimiert und Ressourcen besser genutzt werden können. Siegele geht von einer durchschnittlichen Auslastung von Hardware Ressourcen mit 5%-20% in traditionellen Rechenzentren aus. Was auf den ersten Blick sehr wenig erscheint begründet sich darin, das in Lastspitzen die benötigten Ressourcen um ein 2 bis 10-faches steigen. Um diesen Anforderungen gerecht zu werden, wird in vielen Fällen eine Provisionierung auf Basis des maximalen Workloads vorgenommen und führt zur Ressourcenverschwendung in weniger aktiven Zeiten.
Durch das einfache hinzufügen von Ressourcen und die interne Ressourcen Verwaltung modernen Cloud Rechenzentren kann dies die Auslastung der einzelnen Systeme auch bei geringer oder moderater Belastung verbessern und so seinerseits zur Umweltschonung beitragen.

3.4.1.6 Beispiel

Diese Vorteile lassen sich an einem kleinen Beispiel im Hochschulsport mit fiktiven Zahlen verdeutlichen. Wie bereits in Abschnitt **??** beschrieben kommt es zu einem enormen zusätzlichen Performance Bedarf für das Buchungssystem zum Buchungsstart. Als Lastspitze wird ein Bedarf von 100 Servern zu Grunde gelegt. Diese Lastspitzen treten etwa 10 mal im Jahr auf und halten für 24 Stunden an. Den Rest der Zeit besteht ein Workload für etwa 20 Server. Daraus ergibt sich über das Jahr ein durchschnittlicher Bedarf an Serverstunden von 532,6 je Tag. Für eine Veranschaulichung des Kostenfaktors wird eine Serverstunde mit 50 Cent veranschlagt.

Serverstunden je Tag: $\frac{((355 * 20) + (10 * 100)) * 24}{365} = \mathbf{532{,}6}$

Kosten je Tag: $\frac{532{,}6 * 50}{100} = \mathbf{266{,}3€}$

Um den Kunden auch in den Lastspitzen die benötigte Performance zusichern zu können, so müsste in einer traditionellem Rechenzentrum die Bemessung des Bedarfs am Peak Load erfolgen und somit für 100 Server. Gemessen am durchschnittlichen Bedarf über das Jahr stellt dies jedoch eine deutliche Überprovisionierung da (3.4).

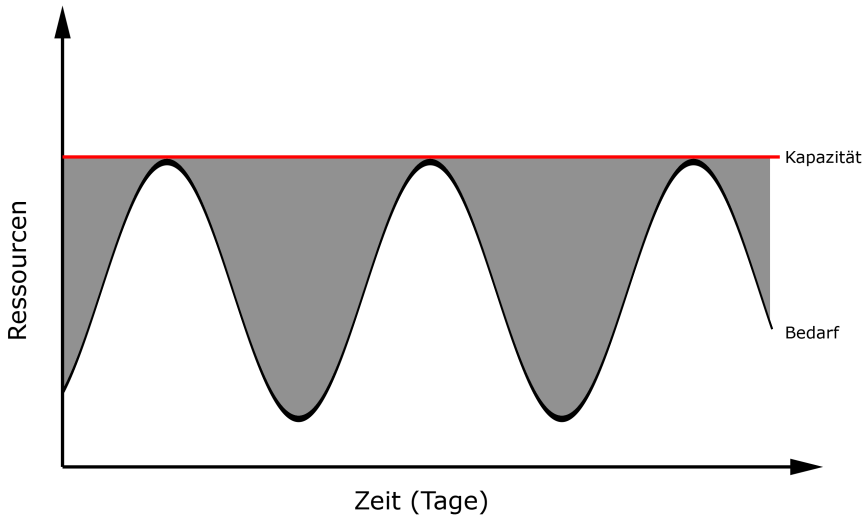


Abbildung 3.4: Provisionierung für Lastspitzen

Serverstunden je Tag: $100 * 24 = \mathbf{2400}$

Kosten je Tag: $\frac{2400 * 50}{100} = \mathbf{1200€}$

Dies entspricht einem erhöhten Preis um Faktor 4,5.
Ein alternativer Ansatz wäre eine moderatere Bemessung des Bedarfes möglich mit z.B. 50 Servern und somit einer Unterprovisionierung. Dies würde dazu führen, das die Lastspitzen jedoch nicht mehr entsprechend versorgt werden könnten und es zu Einschränkungen und sogar Ausfällen des System kommen könnte (3.5). Mit diesem Ansatz ließen sich deutliche Einspa-

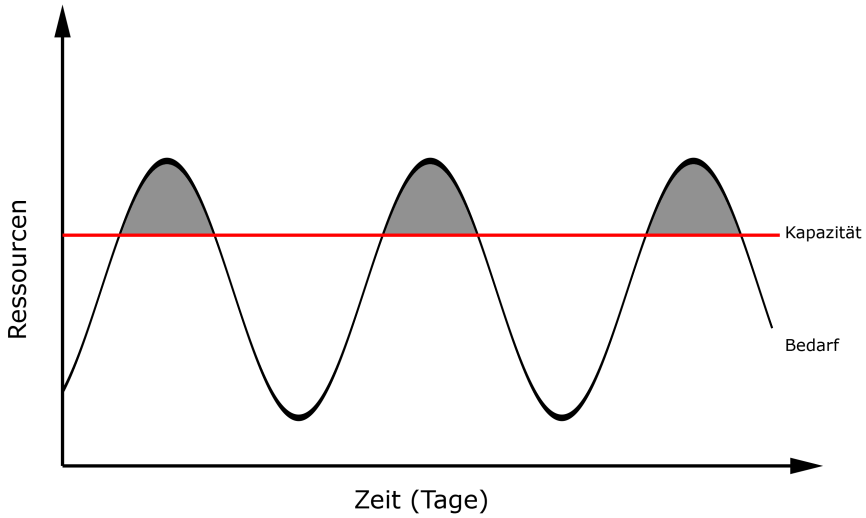


Abbildung 3.5: Unterprovisionierung

rungen erzielen, jedoch muss mit betrachtet werden, dass es zu deutlichen Einnahme Einbußen kommen könnte, da nicht allen Kunden eine Buchung möglich wäre oder sich Kunden sogar vollständig vom diesem Dienst entfernen.

Serverstunden je Tag: $50 * 24 = \mathbf{1200}$

Kosten je Tag: $\frac{1200 * 50}{100} = \mathbf{600€}$

Dieses auf fiktiven Zahlen basierende Beispiel ist in der realen Welt häufig anzutreffen und es müssen mit dem traditionellen Modell Kompromisse eingegangen werden, die unterschied-

lich stark ausfallen können. Kosten-Nutzen-Optimierungen sind immer notwendig, bekommen mit dem Cloud Computing aber neue Möglichkeiten. Eine Umsetzung im Cloud Umfeld könnte wie folgt aussehen. Um einen reibungslosen Betrieb in den Phasen mit weniger Aktivität zu gewährleisten und einen gewissen Puffer zu integrieren, werden für das Jahr 25 Server provisioniert. Für die Anmelde Zeiträume werden jeweils für 48 Stunden weiter 75 Server dazu geschaltet. Daraus ergäbe sich folgende Rechnung:

Serverstunden je Tag: $\frac{((355 * 25) + (20 * 100)) * 24}{365} = \mathbf{715{,}07}$

Kosten je Tag: $\frac{715{,}07 * 50}{100} = \mathbf{357{,}53€}$

Dies entspräche dem 1,34-fachem des durchschnittlichen Bedarfes und ist in erster Linie dem Puffer geschuldet. Im Vergleich zur Überprovisionierung betragen die Kosten aber nur das 0,3-fache und im Vergleich zur Unterprovisionierung das 0,6-fache ohne die einhergehenden Geschäftlichen Verluste. Dieses vereinfachte Beispiel macht so klar, welche Vorteile vor allem die Punkte Kosteneinsparung, Skalierbarkeit und Flexibilität in Cloud Umgebungen bringen.

3.4.2 Nachteile

Neben den bereits genannten Vorteilen haben sich neue Technologien und Modelle meist mit einigen Problemen und Herausforderungen auseinander zu setzen, die hier ebenfalls vorgestellt werden sollen.

3.4.2.1 Sicherheit und Datenschutz

3.4.2.2 Abhängigkeit

3.4.2.3 Internetzugang

Da alle Cloud Dienste über das Internet angebunden werden, ergibt sich eine zusätzliche Internet Abhängigkeit, die ein Arbeiten bei gestörten Verbindungen unmöglich macht. Ob die

Störung dabei beim Kunden oder beim Anbieter liegt ist dabei unerheblich. Im Vergleich zu den traditionellen eigenen Rechenzentren erhöht sich damit die Komplexität und die Anforderungen an die Internet-Bandbreite, die in lokalen Netzwerken leichter bereit gestellt werden konnte.

3.4.2.4 Verfügbarkeit

Die Zuverlässigkeit eines Anbieters gilt weiter als Nachteil, da mehr Schnittpunkte existieren. Ausfälle bei Amazon, Microsoft oder Google können zum Ausfall aller Systeme führen. Es muss dabei aber gesagt werden, das diese Ausfälle auch in lokalen Rechenzentren jederzeit passieren und passieren können. Die Ausfallzeiten der etablierten Anbieter beschränkte sich in den letzten Jahren auf ein Minimum (QUELLE?)

3.4.2.5 Interoperabilität

Viele Cloud-Anbieter bieten je nach Service viele eigenen Techniken, Produkte und Services mit denen gearbeitet wird. Prinzipiell ist dabei ein Umzug zu einem anderen Cloud Anbieter nicht vorgesehen, da es zu Inkompatibilitäten führen kann. Dies kann zu großen Problemen führen, wenn diese Anbieter ggf. insolvent gehen oder der Kunde aus anderen Gründen den Anbieter wechseln möchte.

3.5 Software-as-a-Service Anwendungen (SaaS)

Wie bereits in den vorherigen Abschnitten beschrieben, bietet das Cloud Computing viele neue Möglichkeiten. In Abschnitt 3.2 wurde der Begriff Software-as-a-Service erläutert und gilt heute als einer der bekanntesten Begriffe aus dem Cloud Computing Service Katalog. Da sich diese Arbeit mit der Konzeption eines Softwaresystems für Hochschulsporteinrichtungen beschäftigt soll dieses Modell in diesem Abschnitt noch einmal genauer betrachtet werden.
Per Definition gibt es für SaaS-Anwendungen keine speziellen Einschränkungen, wie diese aufgebaut sein muss. Aus Anbietersicht lohnt sich eine SaaS-Anwendung jedoch nur in den seltensten Fällen, sodass das Ziel meist ist, mit einer Anwendung möglichst viele Kunden gleichzeitig versorgen zu könne, ohne spezielle Anpassungen, Installationen etc. machen zu müssen. Damit wird das Multi-Tenancy Konzept zu einem zentralen neuen Bestandteil der meisten Anwendun-

gen. Betrachtet man die Anwendungsmerkmal genauer, so lassen sich weitere Charakteristika erkennen, die im Normalfall erfüllt sein müssen:

- Generalisiert und anpassbar, sodass ein breite Nutzerbasis angesprochen werden kann
- Intuitive, einfache Bedienung und Navigation
- Modular und Service-orientiert aufgebaut
- Integrierte Messung und Monitoring für Nutzungsbasierte Abrechnung
- Integrierter Rechnungsstellung
- Konstante Weiterentwicklung mit neuen Features
- Gewährleistung von Datenschutzrichtlinien
- Unterstützung von mehreren gleichzeitigen Kunden (Multi-Tenancy)

3.5.1 Vorteile

Betrachtet man die Vorteile, die auf Kundenseite durch SaaS-Anwendungen entstehen, so lässt sich leicht feststellen, dass diese zum Teil auf den Vorteilen von Cloud Computing basieren.

3.5.1.1 Kostenersparnis

Die Kostenersparnisse setzen sich aus mehreren Faktoren zusammen, um die SaaS Variante als günstiger bezeichnen zu können. Betrachtet man die Kosten für die Anwendungsnutzung isoliert, so sind SaaS-Anwendungen in vielen Fällen sogar teurer als traditionelle on-premise Anwendungen. Bezieht man die Kosten für IT-Infrastruktur, Skalierungsmöglichkeiten und Wartung mit ein, so ändert sich das Verhältnis zu Gunsten der SaaS-Lösung.

3.5.1.2 Zeitersparnis

Da die Software bereits beim Cloud Anbieter installiert ist, entfällt die Zeit für Installation und Einrichtung der Software und beteiligter IT-Infrastruktur.

3.5.1.3 Fokus auf Geschäftsaufgaben anstatt Infrastruktur

Der Kunde wird von kostenintensiven, zeitaufwändigen Unterstützungsaufgaben für den Betrieb der IT-Infrastruktur befreit und kann sich auf die Kernaufgaben konzentrieren.

- Beschaffung und Wartung der hauseigenen IT-Infrastruktur, um die Software vor Ort zu betreiben
- Sicherstellung von Sicherheit, Reliabilität und Skalierbarkeit mit redundanter Hardware
- Aufrechterhalten eines Update und Upgrade-Prozesses

3.5.1.4 Direkter Zugriff auf Neuerungen

In herkömmlichen Anwendungen musste bis zu neuen Release gewartet werden, um neue Features nutzen oder Fehlerbehebungen einspielen zu können. Der Update-Aufwand und die Release Frequenz können diesen Prozess oft in die Länge ziehen. In SaaS-Anwendungen wir das Update vom Cloud Anbieter eingespielt und steht sofort zur Verfügung. Dem Anbieter ist im Rahmen des Wettbewerbsvorteils meist sehr daran gelegen, neue Features zeitnah zu Verfügung zu stellen.

3.5.2 Herausforderungen

Viele Herausforderungen, die für herkömmliche Anwendungen gelten, sind auch bei SaaS-Anwendungen anzutreffen. Hinzu kommen jedoch weiter, neue Herausforderungen, die bisher nicht so häufig zu Tage traten jetzt aber eine höhere Relevanz bekommen. Eine der zentralen Herausforderungen umfasst das Thema Security. Daten und Anwendung werden an den Dienstanbieter weiter gegeben und unterliegt deren Verantwortlichkeit. Für viele Firmen, Organisationen und Anwender ist die Sicherheit und der Datenschutz zu erheblichen Bedenken, die von Seiten der Anbieter beseitigt werden müssen. Eine weitere Herausforderung ergibt sich in diesem Zusammenhang mit der Multi-Tenancy-Architektur. Da SaaS-Anwendungen meist für mehrere Kunden ausgelegt ist, muss sichergestellt werden, das jeder Kunde auch nur auf seine eigenen Daten zugreifen kann. Durch das Multi-Tenancy Prinzp muss die Anwendung mit einer hohen Useranzahl und sehr unterschiedlichen Kundenwünschen zurecht kommen können. Dies

stellt spezielle Anforderungen an das Architekturdesign und die Implementierung. Da SaaS-Anwendungen als ein spezieller Service des Cloud Computing zu sehen sind, beanspruchen sie für sich die Vorteile für sich am besten Nutzbar machen zu können. Damit werden besondere Ansprüche an die Kernpunkte Skalierbarkeit und Flexibilität gestellt. Diese drei Kernherausforderungen sollen im Anschluss betrachtet werden.

3.5.2.1 Architektur und Design

Software Architektur und Design sind komplexe Konstrukte, die von vielen Seiteneffekten **BENENNEN** beeinflusst werden. Ein passendes Konzept für die jeweilige Anwendung zu finden ist daher äußerst schwierig und kann sich im Lebenszyklus einer Anwendung durchaus ändern. Auch gibt es für SaaS-Anwendungen nicht die richtige oder falsche Architektur jedoch lassen sich anhand der Anforderungen geeignete und ungeeignete Modelle identifizieren.
Monolithische Software Systeme sind in der Software-Entwicklung eine der am häufigsten Anzutreffenden

3.5.2.2 Multi-Tenant Application (MTA)

Bevor die MTA im Detail erklärt werden kann, ist es wichtig ein gemeinsames Verständnis von Tenant (dt. Mieter) und Multi-Tenancy (dt. Mehrfach Mieter) im Kontext von Software-Anwendungen zu schaffen.

Die primäre Herausforderung in MTA liegt daher die saubere Trennung von Abläufen und Daten zwischen den einzelnen Tenants. Chong, Carraro und Wolter zeigen in ihrem Artikel drei unterschiedliche Möglichkeiten, wie Multi-Tenant Daten verwaltet werden können.

Separierte Datenbanken
In diesem Ansatz werden die Daten für jeden Tenant in einer eigenen Datenbank gespeichert (vgl. Abbildung 3.6). Die Hardware Ressourcen und der Anwendungscode werden dabei geteilt. Innerhalb des Datenbank Management Systems (DBMS) bekommt jeder Tenant jedoch eine eigene Datenbank und auf Datenbank Zugriffsebene eine Trennung zu anderen Nutzern hergestellt. Dieser Ansatz macht es einfach das Datenmodell individuell zu erweitern und Daten aus

Tenant	Bezeichnet eine Gruppe von Benutzern, die die gleiche Sicht auf eine Software teilen. Dies bezieht sich auf die Punkte wie zugreifbare Daten, Konfiguration, User Management und Funktionalität. Ein Tenant kann zum Beispiel eine Hochschulsporteinrichtung sein.
Multi-Tenancy	Beschreibt ein Modell bei dem eine Anwendungsinstanz von mehreren Tenants genutzt wird aber jeder einen eingeschränkten, isolierten Zugriff erhält. Dabei werden Performance und Datenschutz Kriterien je Tenant behandelt.
Multi-Tenant Application	Teilt eine Anwendungsinstanz unter mehreren Tenants, um zusätzliche Kosten zu reduzieren.

Tabelle 3.1: Begriffserklärung Tenant (vgl. Krebs, Momm & Kounev, 2012, S.2)

einem Backup wiederherzustellen. Anderseits führt es zu höheren Wartungskosten, da mehrere Datenbanken verwaltet werden müssen und das Backup Prozedere komplizierter wird. Zusätzlich ist in einigen Systemen die Anzahl der Datenbanken beschränkt, sodass bei einer hohen Tenant Anzahl zusätzliche DBMS benötigt werden.

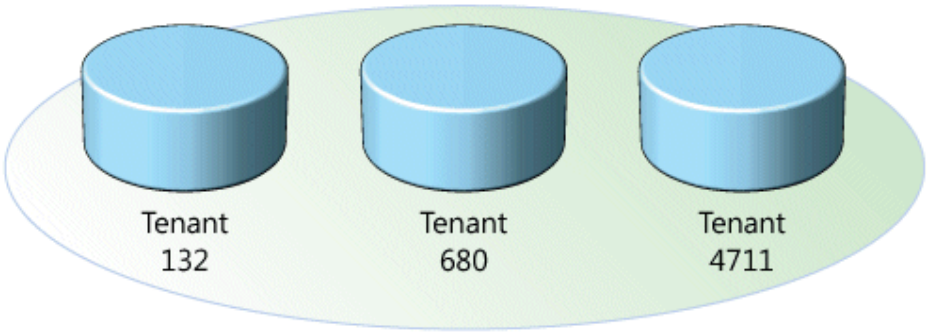


Abbildung 3.6: Separierte Datenbanken

vgl.

Geteilte Datenbank, separiertes Schema

Ein alternativer Ansatz ist es, alle Tenant Daten innerhalb einer Datenbank zu speichern, wobei jeder ein eigenes Schema und Set von Tabellen mit entsprechenden Zugriffsrechten bekommt.

Die Implementierung ist relativ einfach und erreicht ein gemäßigtes Maß an logischer Isolation. Im Gegensatz zum Ansatz mit separierten Datenbanken lassen sich so jedoch deutlich mehr Tenants unterstützen (vgl. Abbildung 3.7). Das wiederherstellen von Tenant spezifischen Daten aus einem Backup gestaltet sich in diesem Ansatz deutlich schwieriger. Da ein Datenbank Backup nicht einfach zurück gespielt werden kann, müssen zuerst alle Daten, aller Tenants aus einem Backup in einen Temporären Server zurück gespielt werden, um sie anschließend aus zurück in das Produktiv System spielen zu können. Diese Vorgehensweise führt zu einem erheblichen Mehraufwand im Falle von Datenverlust.

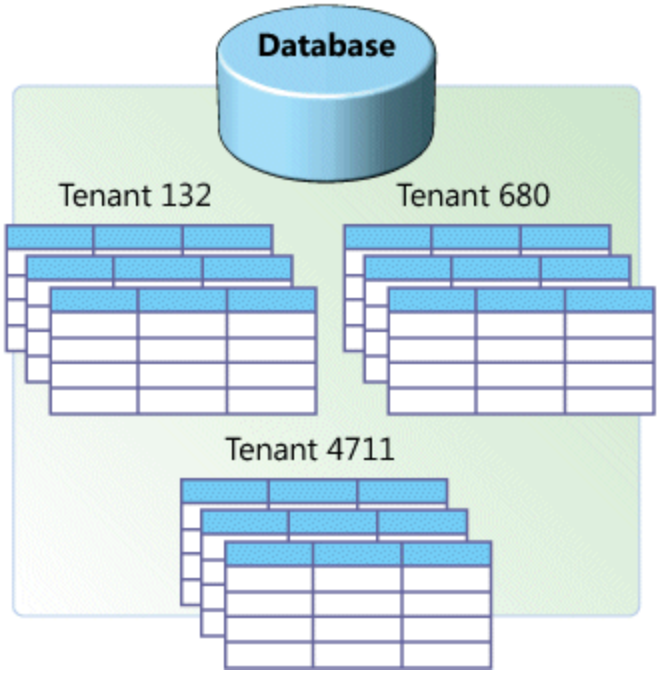


Abbildung 3.7: Geteilte Datenbank und separierte Schemata

vgl.

Geteilte Datenbank, geteiltes Schema

Dieser Ansatz basiert auf einer Datenbank und einem Set von Tabellen. Die Trennung der Daten erfolgt über das mitführen einer Tenant-ID in jedem Datensatz wie in Abbildung 3.8 zu sehen. Diese Lösungsstrategie erlaubt die Versorgung einer großen Tenant-Anzahl per Datenbank und erfordert daher die geringsten Hardware und Backup Kosten. Gleichzeitig erfordert es auch den höchsten Aufwand die Trennung Anwendungsseitig zu sicherzustellen, um alle Sicherheits- und Datenschutzbedingungen zu erfüllen. Die Probleme beim Wiederherstellen von Backups verhält

sich wie beim Ansatz mit separiertem Schema. vgl.

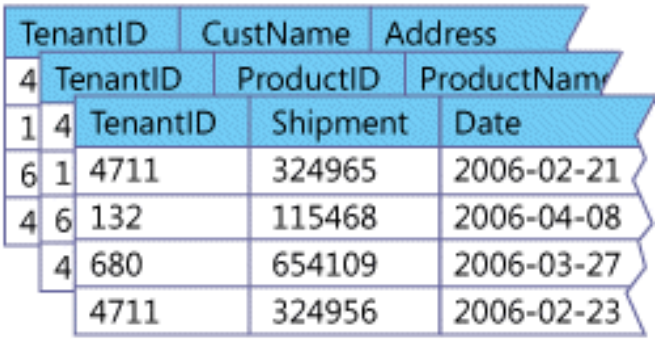


Abbildung 3.8: Geteilte Datenbank und Schema

adfkadf

3.5.2.3 Security

3.5.3 Softwarearchitektur

Die Definitionen, was Softwarearchitektur ist und wie dieser Term zu definieren ist, unterscheidet sich bei einzelnen Autoren. Bass, Clements und Kazman (2013, S. 4) verwenden für mich eine sehr passende Definition des Begriffes:

> *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*

Vor allem die Zusammenhänge sind hier von Bedeutung, denn auch wenn das Design der Architektur nicht endgültig sein muss und sich im Laufe der Entwicklung noch Veränderungen und Anpassungen ergeben könne, so ist es doch von Anfang an wichtig eine Vorstellung über die geplanten Zusammenhänge im gesamten Team zu entwickeln.
Für das Verständnis, welche Prinzipien sich für eine SaaS-Anwendung eignen, will drei grundlegende Architekturen darstellen, die sich in den Zusammenhängen unterscheiden. Jede diese Architekturen lässt sich optimieren, anpassen und so besser nutzbar machen und oftmals gibt es auch unterschiedliche Kombinationen die zum gewünschten Erfolg führen.

3.5.3.1 Monolitisch

Eine der am weitest verbreiteten Software Architekturen ist die monolithische Architektur. Gerade zentrale Server Anwendungen sind häufig nach diesem Prinzip aufgebaut. Die Stärken dieser Architektur liegen in:

- Einfach zu entwickeln - Das Ziel bestehender Entwicklungstools ist es die Erstellung monolithischer Programme zu unterstützen
- Einfach auszuliefern - Nur die Anwendung als Ganzes muss ausgeliefert werden
- Einfach zu skalieren - Hinter einem Lastverteiler können einfach mehrere Kopien der Anwendung laufen

Durch diese Vorteile hat sich dieses Architektur Prinzip weit verbreitet und ist jedem Entwickler bekannt. Gerade für kleine bis mittelgroße Projekte sind die genannten Faktoren entscheidend. Der interne Aufbau der Anwendung kann dabei jedoch sehr unterschiedlich gestaltet sein. In den meisten Fällen ist allerdings ein Schicht-Aufbau zu erkennen, der unterschiedliche Komponenten intern trennt. Gängige Schichten in einer Webanwendung sind folgende:

- Präsentation - Darstellung und Verwalten von HTTP Anfragen und Ergebnissen in HTML oder JSON/XML Format
- Geschäftslogik - Interne Logik für Geschäftsabläufe
- Datenbank Zugriff - Zuständig für alle Zugriffe auf die Datenbank
- Anwendungsintegration - Nachrichten Transfer, Email etc.

Richardson (2014) sieht jedoch auch eine Reihe von Nachteilen dieser Architektur, die mit zunehmender Größe der Anwendung und wachsender Teamgröße immer signifikanter werden.

- Die große monolithische Codebasis kann neue Entwickler schnell einschüchtern. Änderungen und Verständnis können schwierig sein und die Entwicklung verlangsamen. Durch fehlende Modulgrenzen geht im laufe der Zeit die Modularität verloren.
- Mit wachsender Größe er Codebasis werden die IDEs (Integrierte Entwicklungsumgeben) langsamer und bremsen die Produktivität

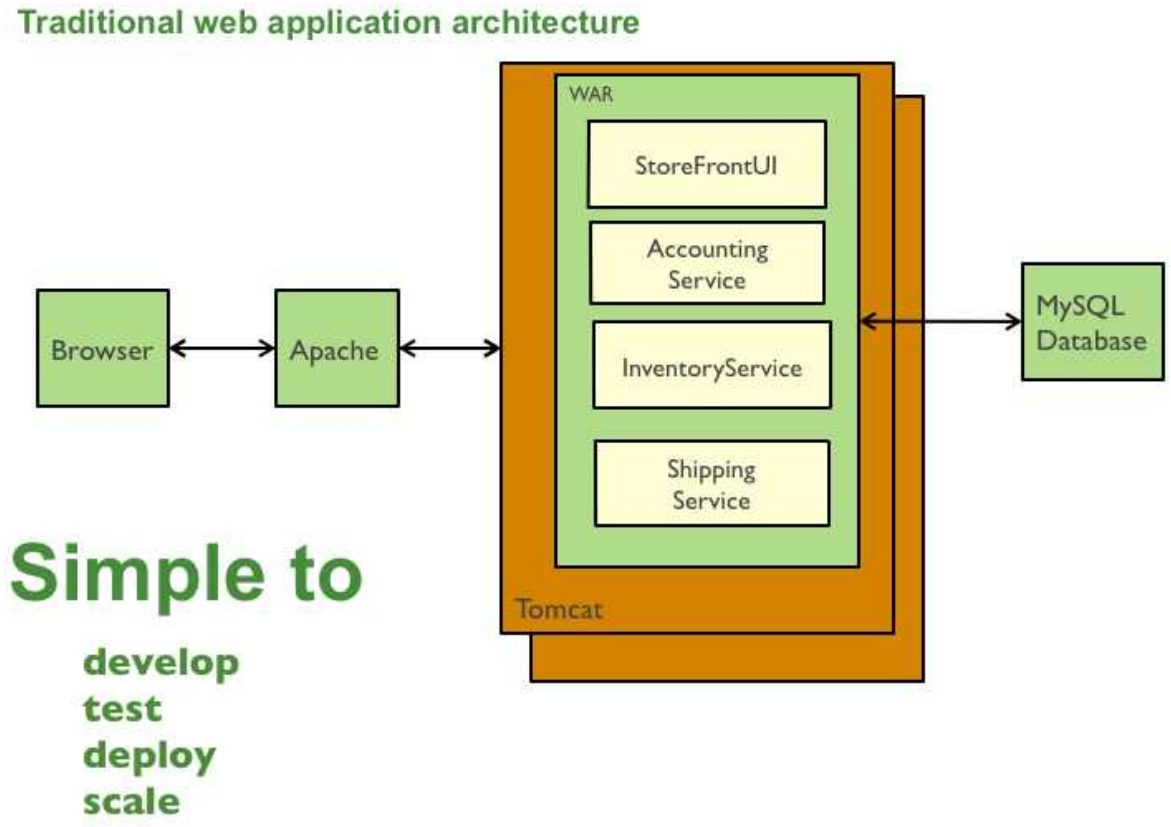


Abbildung 3.9: Traditionelle Webanwendungs Architektur

- Anwendungsstart wird zunehmend länger und hat einen großen Effekt auf die Produktivität der Entwickler und die Auslieferung. Wartezeit ist verschwendete Zeit.

- **Continuous deployment is difficult**

- Das Skalieren der Anwendung kann schwierig werden, da es nur in eine Dimension skalieren kann. Mit wachsenden Transaktionen kann das System durch zusätzlich Instanzen skaliert werden. Hingegen kann bei einem wachsenden Datenvolumen diese Architektur nicht gut skalieren. Jede Instanz greift auf alle Daten zurück und führt zu wenig effektivem Caching, höherem Speicherverbrauch sowie zusätzlichem I/O traffic. Außerdem können unterschiedliche Anwendungsbereiche verschiedene Anforderungen z.B. CPU haben, das skalieren von einzelnen Komponenten ist jedoch unmöglich.

- Die monolithische Architektur hat Grenzen wie die Entwicklung skalieren kann. Bei einer entsprechenden Anwendungsgröße werden Teams gebildet, die für spezielle Bereiche ver-

antwortlich sind z.B. UI Team, Buchhaltungs Team etc. Die enge Verzahnung der einzelnen Bereiche erschwert jedoch die unabhängige Arbeit.

- Mit dem Beginn der Entwicklung muss sich auf eine Technologie und ggf. eine Version festgelegt werden, die es sehr schwer macht neue Technologien zu adaptieren. Eine inkrementelle Umstellung gestaltet sich als schwierig, wobei eine vollständige Ersetzung der aktuellen Technologie sehr kostenintensiv ist und das Problem im Kern nicht behebt.

3.5.3.2 Client/Server-Architekturmodell

Im Gegensatz zur monolitischen Architektur handelt es sich beim Client/Server Modell um ein verteiltes System. Das Client/Server-Architekturmodel ist ein Systemmodell, das aus einer Anzahl von zugehörigen Diensten und Servern besteht, sowie aus Clients, die diese Dienste nutzen und auf sie zugreifen. Das einfachste Modell ist dabei das zweischichtige Client/Server Modell, die zusammen ein komplettes System bilden mit unterschiedlichen Zuständigkeiten.
Dabei lassen sich drei logische Hauptbestandteile unterschieden.

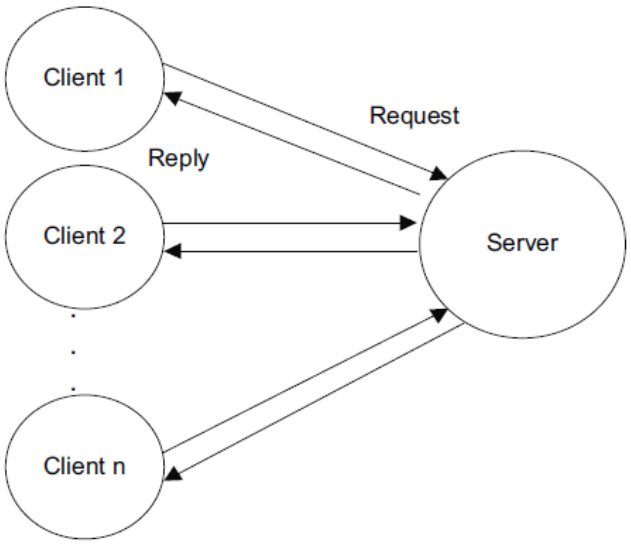


Abbildung 3.10: Client Server Model

3.5.3.2.1 Server

Der Server ist die zentrale Einheit der Architektur. Sie stellt Services und Daten anderen Einheiten zur Verfügung. Der Server muss nicht zwingend wissen, welche und wie viele Einheiten die von ihm zur Verfügung gestellten Dienste und Daten nutzen.

3.5.3.2.2 Client

Clients können in beliebig großer Anzahl existieren. Sie nehmen die Daten oder Services, die vom Server bereitgestellt werden in Anspruch. Um mit den Diensten interagieren zu können, muss ein Client von der Existenz und dessen Adresse wissen. Der Client sendet eine Anfrage an den Server und bekommt ein Resultat zurück geliefert. Sommerville (2007, S. 302) unterscheidet folgende zwei Client-Modelle:

Thin-Client-Modell

Bei diesem Modell werden die gesamten Anwendungsprozesse und Datenhaltung auf dem Server erledigt. Der Client ist nur für die Darstellung verantwortlich. Daraus ergibt sich der Vorteil, dass sehr wenig Ressourcen für den Client benötigt werden und dieser sich auch auf kleinen Systemen (z.B. Terminals) implementieren lässt. Der Nachteil dieses Modelles liegt jedoch darin, dass so sämtliche Arbeit Serverseitig geleistet werden muss. Gerade bei Anwendungen mit großer Client Anzahl oder rechenintensiven Operationen kann so der Server schnell zur kritischen Komponente werden und das System vor Skalierungsprobleme stellen. Zusätzlich führt der Thin-Client zu einer erhöhten Netzwerkbelastung, da alle Daten vom Server zum Client transportiert werden müssen.

Fat-Client-Modell

Dieses Modell verlagert die Anwendungslogik vollständig oder in großen Teilen vom Server direkt in den Client. Dadurch kann der Server deutlich entlastet werden und die Rechenleistung des Client Rechners zusätzlich genutzt werden. Dies kann sich in speziellen Situationen allerdings auch als Nachteile erweisen, wenn die Client Computer die notwendige Rechenleistung nicht selbstständig bereitstellen könne. Der Server fungiert dabei nur noch als Daten Service und die Netzwerklast wird reduziert. Die Verlagerung der Anwendungslogik auf den Client bringt

aber auch Nachteile mit sich. Müssen hier Änderungen vorgenommen werden, so müssen alle Clients auf die neue Version aktualisiert werden, was zu einem komplexen Systemmanagement führen kann. Der Aufwand dafür steigt mit zunehmender Zahl der Clients.

3.5.3.2.3 Netzwerk (optional)

In der Literatur wird das Netzwerk häufig noch ein dritter Bestandteil aufgeführt, der jedoch nicht zwingend notwendig ist da prinzipiell Server und Client auf einem Computer ausgeführt werden können und so kein Netzwerk zur Kommunikation benötigen. Da dieses Modell aber in der Praxis nahezu ausschließlich als verteiltes System angewendet wird, ist in diesen Fällen das Netzwerk grundlegende Kommunikationsgrundlage.

Generell hat das zweischichtige Client/Server Modell das Problem, dass die drei logischen Schichten auf zwei unterschiedlichen Computersystemen abgebildet werden müssen.

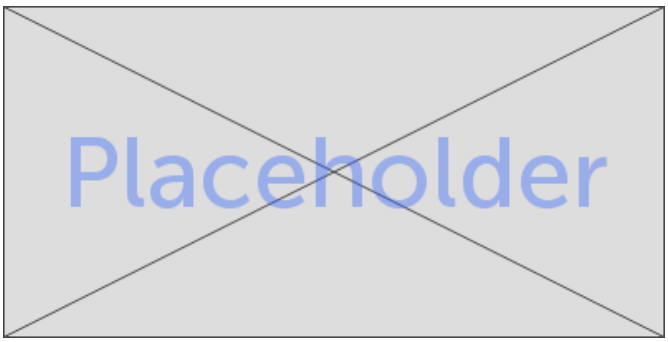


Abbildung 3.11: Logische drei Schichten

Als Alternativansatz hat sich dafür die dreischichtige System-Architektur Entwickelt. Sie stellt für jede logische Schicht ein separates Computersystem zur Verfügung. Die Vorteile liegen dabei in der besseren Skalierbarkeit, da jedes System individuell skaliert werden kann, geringere Netzwerklast im Vergleich zum Thin-Client-Modell und Anwendungsprozesse sind zentralisiert und lassen sich dort leichter anpassen und aktualisieren.

Sommerville gibt in seinem Buch Software Engineering Beispiele, wann die Anwendung der jeweiligen Architektur sinnvoll ist.

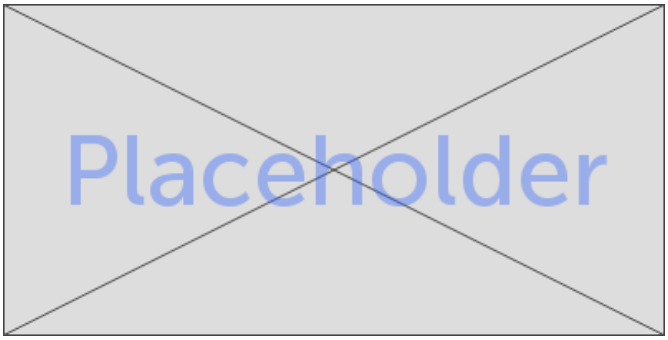


Abbildung 3.12: Dreischichtige Architektur

3.5.3.3 Dienstbasiert

Neben den beiden oben genannten Architekturen hat sich eine weitere Form verbreitet. Bei der dienstbasierten Architektur spricht man von einem Architekturmuster aus dem Bereich der verteilten Systeme. Der Schwerpunkt wird dabei auf eigenständige Dienste als primäre Komponente gelegt, auf die über eine Art Remote-Protokoll zugegriffen werden kann. Die Art des Protokolls ist dabei nicht festgelegt oder auf eines beschränkt wie z.B. Representational State Transfer (REST), Simple Object Access Protocol (SOAP), Advanced Message Queuing Protokol (AMQP) oder andere geschehen. Dienstbasierten Architekturen werden signifikante Vorteile im Bezug auf Skalierbarkeit, Entkopplung, Kontrolle über die Entwicklung, Testing und Deployment zugeschrieben. Verteilte Systeme tendieren zudem dazu weniger abhängig und besser modular aufgebaut zu sein. Im Zusammenhang mit dienstabasierten Architekturen bedeutet das, das jeder Dienst eine in sich geschlossen Einheit bildet, sie selbst designed, entwickelt, getestet und ausgeliefert wird. Abhängigkeiten zu anderen Komponenten sollten dabei nicht, oder nur minimal bestehen.
Diese Vorteile bringen jedoch auch einige Nachteile mit sich, so zählen erhöhte Komplexität und höhere initiale Kosten zu den meist genannten Argumenten.

3.5.3.3.1 Representalional State Transfer (REST)

REST wurde erstmals von 2000 Roy Thomas Fielding in seiner Dissertation beschrieben. Im Gegensatz zu bekannten Protokollen wie Simple Object Access Protokol (SOAP) oder XML-RPC handelt es sich bei REST um einen Architekturstil, wie Anwendungen basierend auf dem HTTP

Architektur	Anwendung
Zweischichtige C/S-Architektur mit Thin-Clients	Anwendungen mit Legacy-Systemen, bei denen die Trennung von Anwendungsprozessen und Datenmanagement nicht praktikabel ist Rechenintensive Anwendungen wie zum Beispiel Compiler mit geringem oder keinem Datenmanagement -Datenintensive Anwendungen (Browser oder Abfragesysteme) mit wenig oder keinen Anwendungsprozessen
Zweischichtige C/S-Architektur mit Fat-Clients	Anwendungen, bei denen die Prozesse durch Standardanwendungen (z.b. Microsoft Excel) auf dem Client verarbeitet werden Anwendungen mit rechenintensiver Datenverarbeitung (z.B. Datenvisualisierung) Anwendungen mit relativ stabiler Endbenutzerfunktionalität, die in einer Umgebung mit gut strukturiertem Systemmanagement angeordnet sind
Drei- oder mehrschichtige C/S-Architektur	Anwendungen großen Umfangs mit Hunderten oder Tausenden von Clients Anwendungen, bei denen sowohl die Daten als auch die Anwendung selbst ständigen Veränderungen unterliegen Anwendungen, bei denen Daten aus vielfältigen Quellen verarbeitet werden

Tabelle 3.2: Verwendungsbeispiele von mehrschichtigen Architekturen Sommerville (2007)

Protokoll miteinander kommunizieren können. Unterstützt eine Anwendung die REST-Stil, so bezeichnet man sie als RESTful. (vgl. Melzer, 2010)
REST basiert auf einem Konzept von Ressourcen, über die der Service Kenntnis hat. Der Server erstellt dabei verschiedene Darstellungen, wobei diese entkoppelt ist von der internen Speicherung. HTTP und HTTPS sind die primär benutzten Protokolle bei der Verwendung von REST. HTTP Caching Proxies, Load-Balancers und Monitoring Tools besitzen einen tiefe Unterstützung für HTTP von Haus aus. Den HTTP Verben GET, POST, PUT und DELETE lösen in RESTful Anwendungen spezielle Aktionen aus.

Der große Vorteil und der Grund für die schnelle Verbreitung von REST Architekturen liegt in der Einfachheit. Methoden für die Nutzung des HTTP Protokoll sind in allen modernen Pro-

HTTP Verb	REST Aktion
GET	Abfragen einer Ressource ohne Änderungen vorzunehmen
POST	Erstellt eine neue Ressource. Der Link zur neuen Ressource wird zurück gegeben
PUT	Eine Ressource wird angelegt. Wenn die Ressource bereits existiert, wird sie geändert.
PATCH	Ein Teil der Ressource wird geändert
DELETE	Löscht eine Ressource

Tabelle 3.3: Rest Operationen

grammiersprachen enthalten und können einfach genutzt werden. Das Ressourcen Konzept ist leicht anwendbar und bietet umfangreiche Möglichkeiten der eigenen Anpassung.

Die Bandbreite der Möglichkeiten, die mit der REST Architektur umgesetzt werden können zeigt Leonard Richardson in seinem ´´Richardson Maturity Model``indem er vier REST Level beschreibt.

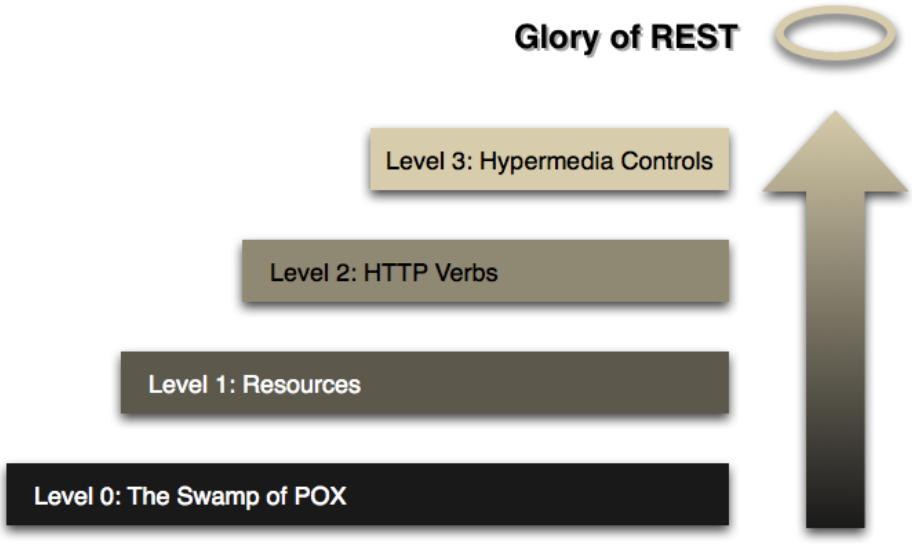


Abbildung 3.13: Richardson Maturity Model

(vgl. Fowler, 2010)

4 System Konzeption

Nachdem in den ersten Kapiteln dieser Arbeit die grundlegenden fachspezifischen Anforderungen der Hochschulsporteinrichtungen an ein Softwaresystem sowie einige wichtige Konzepte der Systementwicklung vorgestellt wurden, soll im vierten Teil ein Konzept erstellt werden, wie eine Umsetzung unter Betrachtung der Anforderungen aussehen kann. Des Weitern sollen in dieses Konzept Anforderungen aus Sicht des Software Erstellers mit berücksichtigt werden, sodass ein einstehendes Konzept theoretisch auch in der Praxis umgesetzt werden kann.
Der Aufbau der Konzeption erfolgt dabei in Anlehnung an Balzert (1996) und beinhaltet die Phasen:

1. Planungsphase
2. Definitionsphase
3. Entwurfsphase

4.1 Planungsphase

- Lastenheft
- Nutzeranforderungen
- Systemanforderungen
- Multi-Tenantcy
- PaaS, SaaS
- Glossar

- WAS und nicht WIE
- Priorisierung?

4.2 Definitionsphase

- Pflichtenheft
-

4.3 Entwurfsphase

- WIE
- Entwerfen einer Software-Architektur
- Zerlegung des definierten Systems in Systemkomponenten
- Strukturierung des Systems durch geeignete Anordnung der Systemkomponenten
- Beschreibung der Beziehungen zwischen den Systemkomponenten
- Festlegung der Schnittstellen, über die die Systemkomponenten miteinander kommunizieren.

Microservice Diagram

https://dzone.com/articles/building-microservices-inter-process-communication-1

https://insidethecpu.com/2015/07/17/microservices-in-c-part-1-building-and-testing

5 Diskussion, Zusammenfassung und Ausblick

Literaturverzeichnis

Achimugu, P., Oluwagbemi, O. & Popoola, V. (2012). Architecting the Cloud: Prospects and Problems. *International Journal of Information*, *2* (3).

Balzert, H. (1996). *Lehrbuch der Software-Technik - Software-Entwicklung*. Heidelberg [u.a.]: Spektrum Akadem. Verl.

Bass, L., Clements, P. & Kazman, R. (2013). *Software architecture in practice* (3rd ed Aufl.). Upper Saddle River and NJ: Addison-Wesley.

Bundesamt für Sicherheit in der Informationstechnik. (o. J.). *Cloud Computing Grundlagen.* Zugriff am 27.08.2016 auf `https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/CloudComputing/Grundlagen/CloudComputing-Grundlagen.html`

Chong, F., Carraro, G. & Wolter, R. (2006). *Multi-Tenant Data Architecture.* Zugriff am 17.08.2016 auf `https://msdn.microsoft.com/en-us/library/aa479086.aspx`

Fowler, M. (2010). *Richardson Maturity Model.* Zugriff am 06.07.2016 auf `http://martinfowler.com/articles/richardsonMaturityModel.html`

Jadeja, Y. & Modi, K. (2012). Cloud computing - concepts, architecture and challenges. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on* (S. 877–880). doi: 10.1109/ICCEET.2012.6203873

Krebs, R., Momm, C. & Kounev, S. (2012). *Architectural Concerns in Multi-Tenant SaaS Applications.* At Setubal and Portugal.

Mahmood, Z. (2011). Cloud Computing: Characteristics and Deployment Approaches. In *11th International Conference on Computer and Information Technology (CIT)* (S. 121–126). doi: 10.1109/CIT.2011.75

Mell, P. M. & Grance, T. (2011). *SP 800-145. The NIST Definition of Cloud Computing.* Gaithersburg and MD and United States: National Institute of Standards & Technology.

Melzer, I. (2010). *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis* (4. Aufl.). Heidelberg: Spektrum Akademischer Verlag.

Radde, G. (1996). Hochschulsporteinrichtungen - Eine vergleichende Betrach-

tung. *dvs-Informationen*, *11* (3), 15–17. Zugriff am 05.07.2016 auf http://www.sportwissenschaft.de/fileadmin/pdf/dvs-Info/1996/1996_3_radde.pdf

Richardson, C. (2014). *Pattern: Monolithic Architecture.* Zugriff am 05.07.2016 auf http://microservices.io/patterns/monolithic.html

Siegele, L. (2008). Let it rise: A special report on corporate IT. *Economist Newspaper* (8603).

Sommerville, I. (2007). *Software-Engineering* (8. Aufl.). München and Boston [u.a.]: Pearson Studium.

Tharam, D., Chen, W. & Elizabeth, C. (2010). Cloud Computing: Issues and Challenges. In *24th IEEE International Conference on Advanced Information Networking and Applications* (S. 27–33). Zugriff auf http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5474674 doi: 10.1109/AINA.2010.187