Strategien für die Bereitstellung eines skalierbaren Audience-Response-Systems

Vom ARS-Router bis zum Cloud-Deployment

Daniel Gerhardt[1], Tom Käsler[2], Hermann Sutter[3] und Klaus Quibeldey-Cirkel[4]

Abstract: Audience-Response-Systeme (ARS) setzen ein Funknetz voraus, entweder ein eigenes oder ein öffentliches. Eigene Funknetze bieten eine Reihe von Vorteilen gegenüber dem Hörsaal-WLAN, vor allem Stabilität und Ablenkungsfreiheit. Allen gemeinsam ist die beschränkte Belastbarkeit des Funknetzes und des ARS-Servers durch die Anzahl gleichzeitiger User. Der Beitrag richtet sich an ARS-Entwicklerteams und -Betreiber. Er zeigt Strategien auf, wie ein Audience-Response-System bedarfsgerecht und zuverlässig bereitgestellt werden kann. Wenn sichergestellt werden soll, dass Studierende außer dem ARS-Dienst keine anderen Online-Dienste nutzen, sind Clicker die erste Wahl. Sollen Anschaffungskosten und Logistikaufwand der Clicker entfallen, sind die mobilen Endgeräte der Studierenden einzubinden: „Bring Your Own Device", BYOD. Soll die Verwendung des Smartphone-Browsers im WLAN auf den ARS-Dienst beschränkt sein, ist ein eigener mobiler WLAN-Router erforderlich. Ein ARS-Router ist nur begrenzt skalierbar; die Anzahl gleichzeitiger User ist aus Kostengründen auf wenige Hundert beschränkt. Der Beitrag skizziert eine Bereitstellungsstrategie für ein hochskalierbares ARS. Eine Microservice-Architektur, implementiert in Software-Containern, und die Orchestrierung der Container in einer Cloud ermöglichen einen performanten und hochverfügbaren Audience-Response-Dienst.

Keywords: Clicker, ARSnova, BYOD, ARS-Router, Microservices, Software-Container

1 ARS-Router oder „Bring Your Own Network"

Clicker und webbasierte Audience-Response-Systeme (ARS) bauen auf unterschiedlichen Netztechnologien auf. Während Clicker in einem eigenen Funknetz betrieben werden, nutzen webbasierte ARS das Hörsaal-WLAN. Wegen der Beschränkungen der verwendeten Netztechnologie und der mobilen Endgeräte – Handsender oder Smartphones – lassen sich die funktionalen und nicht-funktionalen Anforderungen an Audience-Response-Systeme nicht vollständig realisieren, siehe Tabelle 1. Die Strategie „Bring Your Own Device" (BYOD) löst das Kosten-, Verteilungs- und Wartungsproblem der Clicker. Die Strategie „Bring Your Own Network" (BYON) reduziert das Ablenkungsrisiko der browserbasierten ARS und bietet die Stabilität und Zuverlässigkeit eines Clicker-Funknetzes.

[1] TH Mittelhessen, THM, FB MNI, Wiesenstraße 14, 35390 Gießen, daniel.gerhardt@mni.thm.de
[2] TH Mittelhessen, THM, FB MNI, Wiesenstraße 14, 35390 Gießen, tom.kaesler@mni.thm.de
[3] TH Mittelhessen, THM, FB MNI, Wiesenstraße 14, 35390 Gießen, hermann.sutter@mni.thm.de
[4] TH Mittelhessen, THM, FB MNI, Wiesenstraße 14, 35390 Gießen, klaus.quibeldey-cirkel@mni.thm.de

Zielgröße: 200 User	Clicker-Funknetz	Hörsaal-WLAN	ARS-Router
Bereitstellungskosten	≈ 10.000 €	entfällt	≈ 1.000 €
Verteilungsaufwand	hoch (200 Clicker)	entfällt (BYOD)	entfällt (BYOD)
Wartungsaufwand	Batteriewechsel	entfällt	gering (Updates)
Netzstabilität	hoch	< 50 User pro AP stabil	hoch
Ablenkungsrisiko	nicht gegeben	hoch (WhatsApp & Co)	gering
Anzahl Frageformate	gering	abhängig vom ARS	abhängig vom ARS
Live-Rückkanal	nein	abhängig vom ARS	abhängig vom ARS
Fragetext auf Endgerät	nein	abhängig vom ARS	abhängig vom ARS
Datenschutz	hoch (Server intern)	kritisch (Server extern)	hoch (Server intern)

Tab. 1: Funknetze für Clicker und Browser-ARS im Vergleich

Der Internetzugang im Hörsaal erweist sich oft als Nadelöhr, verstopft durch WhatsApp & Co. Das liegt nicht selten an der Installation von Access Points (AP), die eigentlich für Heimnetzwerke bestimmt und nur für etwa 50 gleichzeitige Verbindungen ausgelegt sind, es sei denn, die IT-Verantwortlichen wissen um diese Einschränkung und investieren in die Enterprise-Version der Access Points, die 500 bis 1.000 gleichzeitige Verbindungen ermöglichen, aber auch 10 bis 20 Mal teurer sind. Einsatz-Szenarien mit mehr als 200 gleichzeitigen Usern, deren Abstimmungen und Feedback in Echtzeit übertragen werden, sind die Ausnahme. Sie setzen die Optimierung der WLAN-Infrastruktur und ein Performance-Tuning des ARS-Servers durch IT-Experten voraus. Ein Erfolgsbeispiel für einen Hörsaal mit 800 Sitzplätzen zeigt Abb. 1. Hier wurden 7 Access Points mit einer Übertragungsrate von 450 Mbps (2,4 GHz) und 450 Mbps (5 GHz) installiert [Da17].

Abb. 1: Bergische Universität Wuppertal: 700 *ARSnova*-User in einem optimierten Hörsaal-WLAN

An der Justus-Liebig-Universität Gießen wurden in einer umfangreichen ARS-Vergleichs- und Einsatzstudie neben den technischen auch von Studierenden verursachten Störquellen vor Ort ausgemacht, die selbst ein leistungsfähiges Hörsaal-WLAN stark beeinträchtigen können: „Zu Problemen kann es im WLAN-Netz kommen, wenn beispielsweise mobile private WLAN-Hotspots eingerichtet werden. Dies kann unbeabsichtigt durch Unkenntnis der Studierenden passieren, kann aber auch absichtlich zur Störung des

WLAN-Netzes beitragen. Ebenfalls können große Downloads zu Störungen führen. Sollten sich mehrere Studierende im Hörsaal befinden, die beispielsweise im Hintergrund unbemerkt große Updates herunterladen, kann dies die Abstimmung erheblich stören.“ [FB15, S. 23]

Abb. 2: ARS-Ausstattung mit eigenem Funknetz: WLAN-Router & Mini-PC vs. Clicker-Koffer

Aufgrund mehrjähriger ARS-Erfahrungen [Qu17] in Hörsälen mit stark unterschiedlicher WLAN-Qualität wurde im Open-Source-Projekt *ARSnova*[5] der THM eine Deployment-Strategie entwickelt, die die Vorteile eines eigenen Funknetzes mit den Vorteilen browserbasierter ARS vereint: der *ARSnova-Router*.[6] Die im Abschnitt 2 beschriebenen Deployment-Verfahren werden auf einen mobilen Rechner mit WLAN-Router angewendet. Ein WLAN-Router für etwa 200 gleichzeitige Verbindungen, der mehrere Clicker-Koffer ersetzt, kostet ca. 250 €. Ein Mini-PC in Bierdeckelgröße, auf dem der ARS-Server installiert ist, kostet 50 € für einen Raspberry Pi oder 750 € für einen Intel® NUC (Abb. 2). Technisch einfache Systeme, z. B. die gamifizierte Quiz-App *arsnova.click*[7], passen in den Arbeitsspeicher des Raspberry Pi. Komplexe Systeme mit einem Java-Backend und Datenbankserver wie *arsnova.voting*[8] benötigen deutlich mehr RAM. In diesem Fall könnte der teurere Mini-PC auch als Präsentationsrechner fungieren und den Laptop der Lehrkraft ersetzen. arsnova.voting und die Ansteuerung des WLAN-Routers lassen sich auch auf jedem handelsüblichen Laptop mit einem GNU/Linux-Betriebssystem oder in einer Docker-Umgebung installieren.

[5] https://arsnova.thm.de/blog/
[6] https://github.com/thm-projects/arsnova-router/
[7] https://arsnova.click/
[8] https://arsnova.voting/

Ein Clicker-Funknetz oder ein ARS-Router sind deutlich besser vor unabsichtlichen und mutwilligen Störungen im Hörsaal gefeit, da nur der lokale Audience-Response-Dienst verfügbar ist. Die Ablenkung der Studierenden durch Nutzung vorlesungsfremder Online-Dienste ist weitgehend ausgeschlossen.[9] Beide Funknetz-Implementierungen machen den ARS-Einsatz unabhängig von der Verfügbarkeit eines externen ARS-Dienstes.

2 Server-Deployment: Servlet- und Docker-Container

Das ARSnova-System arsnova.voting besteht aus zwei obligatorischen Softwarekomponenten: dem Java-Backend und einem Webclient. Eine lose Kopplung zwischen den Komponenten wurde über die Definition einer REST-API erreicht. Zusätzlich zur REST-API wird das Websocket-Protokoll für Echtzeitkommunikation eingesetzt. Die öffentliche Schnittstelle ermöglicht die unabhängige Entwicklung weiterer Clients, beispielsweise des PowerPoint-Add-ins[10] oder des CRAN-Package *exams*[11]. Umgekehrt ist es auch möglich, eine alternative Backend-Implementierung mit dem Webclient zu verwenden [Kä17].

Das Backend von arsnova.voting wurde als Java-Servlet-Applikation implementiert und wird als Webarchiv (WAR-Datei) ausgeliefert.[12] Das Webarchiv kann in Servlet-Containern, welche die Servlet-3.0-Spezifikation erfüllen, z. B. Apache Tomcat, deployt werden. Um ein einheitliches Deployment-Verfahren zu ermöglichen, ist der Webclient ebenfalls als Webarchiv verfügbar.[13] Sofern eine Hochschule spezifische Anpassungen benötigt, kann sie eigene Webarchive aus dem Open-Source-Code erzeugen. Die hierfür notwendigen Build-Definitionen und -Skripte sind auf GitHub verfügbar.[14] [15]

Neben dem Servlet-Container benötigt arsnova.voting für die Datenhaltung das dokumentenbasierte Datenbanksystem Apache CouchDB. Darüber hinaus muss ein Webserver, z. B. Nginx, als Reverse-Proxy eingerichtet werden, der die Aufnahme von HTTP- und Websocket-Verbindungen unter einem einzigen Port ermöglicht und Verbindungen mittels TLS absichert. Abb. 3 zeigt diese Funktionen eines Reverse-Proxys am Beispiel der Meteor-App arsnova.click.

Eine vorkonfigurierte Serverumgebung wird in Form von Docker-Container-Images[16] bereitgestellt. Diese vereinfachen erheblich die Installation und Upgrades von ARSnova-Produkten, da die benötigten Serverdienste bereits aufeinander abgestimmt sind.

[9] Das Risiko der Ablenkung durch Nutzung anderer Netze (Mobilfunk oder WLAN) bleibt bestehen. Prinzipiell ist dem Ablenkungsrisiko *didaktisch* zu begegnen [Qu17], z. B. durch Gruppenarbeit bei der Abstimmung, wie es die *Peer-Instruction*-Methode vorsieht. Dieser Beitrag beschränkt sich aber auf die technischen Optionen.
[10] https://github.com/thm-projects/arsnova-ppt-integration
[11] https://cran.r-project.org/web/packages/exams/exams.pdf
[12] https://github.com/thm-projects/arsnova-backend/releases
[13] https://github.com/thm-projects/arsnova-mobile/releases
[14] https://github.com/thm-projects/arsnova-mobile/blob/master/src/site/markdown/development.md
[15] https://github.com/thm-projects/arsnova-backend/blob/master/src/site/markdown/development.md
[16] https://hub.docker.com/u/arsnova/

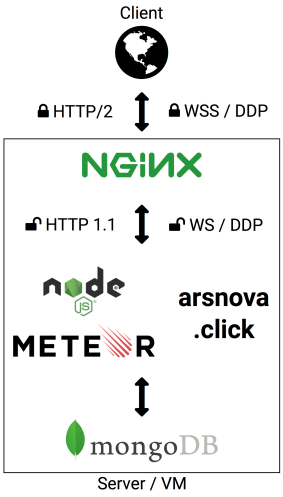

Abb. 3: Typische Server-Infrastruktur für die *Full-Stack-JavaScript*-Implementierung eines ARS

3 Cloud-Deployment: Microservices und Container-Orchestrierung

Soll ein Audience-Response-System hochschulweit zum Einsatz kommen, so ist die vorgestellte *monolithische* Lösung gut geeignet. Soll aber ein Response-System hochschulübergreifend installiert werden, müssen Software-Architektur und Server-Infrastruktur des ARS auf die gleichzeitige Nutzung durch zehntausende Studierende ausgelegt sein. Das Lastprofil einer flächendeckenden ARS-Installation wird während der Vorlesungszeiten extreme Schwankungen aufweisen und in der vorlesungsfreien Zeit gegen Null gehen. Aus Kostengründen müssen das ARS-Backend und die Server-Infrastruktur flexibel und eigenständig Lastspitzen bewältigen und in Ruhezeiten Hardware-Ressourcen wieder freigeben können.

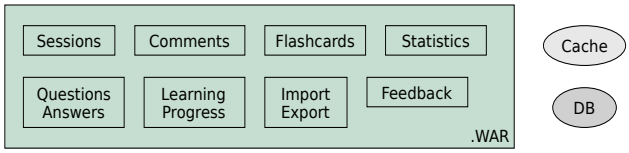

Abb. 4: Fachliche Bestandteile des aktuell monolithischen Backends von arsnova.voting

Bei einer monolithischen Architektur (Abb. 4) wird die gesamte Anwendungslogik gebündelt und durch eine einzige Serverinstanz ausgeliefert. Entsteht nun ein hohes Nutzungsaufkommen, kann die Anwendung *vertikal skaliert* werden, indem man dem Server mehr Ressourcen zur Verfügung stellt [Be14]. Der limitierende Faktor ist hierbei die Hardware. Des Weiteren steigt bei einem monolithischen Backend mit der Funktionalität auch die Komplexität der Software, sodass die Entwicklung träge wird: Der Programmcode

wird schwer verständlich und ist nur noch aufwändig erweiterbar. Bonér vergleicht solche Legacy Software mit einem Kartenhaus und verdeutlicht damit die Fragilität einer monolithischen Architektur [Bo16]. Neue Funktionalität kann nicht losgelöst von der bestehenden entwickelt, sondern muss in diese eingewoben werden.

Eine *Microservice-Architektur* reduziert diese Schwierigkeiten erheblich. Jeder Microservice implementiert eine einzige Geschäftsfunktion und entspricht einem *Bounded Context* im Domain-Driven Design [Fo14]. Microservices sind zudem voneinander isoliert, daher verwaltet jeder Service seine Daten selbst und stellt eine API bereit, über die andere Services auf diese zugreifen können. Es entsteht ein Netz von in sich abgeschlossenen, vergleichsweise schlanken Services.

Für eine auf Microservices basierende Architektur müssen die *Bounded Contexts* bestimmt und isoliert werden. Im Falle der Funktionalität von arsnova.voting lassen sich neun Kontexte abgrenzen (Tab. 2). Durch die Kopplung von Datenhaltung und Service kann je nach Anforderungen durch die Anwendungslogik eine geeignete Datenspeicherung eingesetzt werden. So kommen für Ressourcen wie Sessions, Fragen und Antworten relationale Datenbanken zum Einsatz. Für Services wie den Feedback Service, die lediglich temporäre Daten verwalten, kann auf eine eigene Datenhaltung verzichtet werden; für Services, die Daten von anderen Services aggregieren, ist ebenfalls keine eigene Datenhaltung notwendig. Auch kann bei Bedarf ein Cache zum Einsatz kommen. Daraus ergibt sich eine Architektur, wie sie Abb. 5 zeigt.

Auth	Authentifizierung und User-Verwaltung
Session	Erstellung, Bearbeitung, Freigabe und Sperrung von Sitzungen
Question Answer	Verwaltung der Fragen, Antwortoptionen und Antworten
Comment	Zwischenfragen der Studierenden
Flashcard	Lernkarten für das Selbststudium
Feedback	Echtzeit-Feedback der Studierenden
Import Export	Import und Export von Sessions, Fragen und Antworten
Statistics	Statistiken über die Sitzung, z. B. Anzahl Antworten und Zwischenfragen
Learning Progress	Lernstandsberechnung individuell und für die Gruppe

Tab. 2: *Bounded Contexts* von arsnova.voting

Eine Stärke der Aufteilung in Microservices liegt in der Erweiterbarkeit: Wenn ein neues Feature, z. B. global verfügbare Pools mit fachbezogenen Konzeptfragen für die Lehrmethode *Peer Instruction,* implementiert werden soll, wird dieses Feature in einem neuen *Bounded Context* mit eventuell eigener Datenhaltung umgesetzt. Es ist nicht erforderlich, andere Services anzupassen, sodass sich die Entwicklung auf die neue Funktionalität fokussiert. Auch bleiben die einzelnen Bestandteile übersichtlicher als in einem monolithischen Ansatz.

Ein weiterer wichtiger Aspekt bei der Umsetzung mit Microservices ist das Deployment. Jeder Service wird mit seiner Datenhaltung in einem replizierbaren Docker-Container ge-

bündelt und hat eine eigene interne IP-Adresse (Abb. 5). Zusammen mit einer Orchestrierungslösung wie *Kubernetes* ist das Backend für die *horizontale Skalierung* in der Cloud vorbereitet. Neue Instanzen von Microservices können bei hoher Last flexibel zugeschaltet werden; verebbt die Auslastung, werden diese automatisch wieder entfernt.

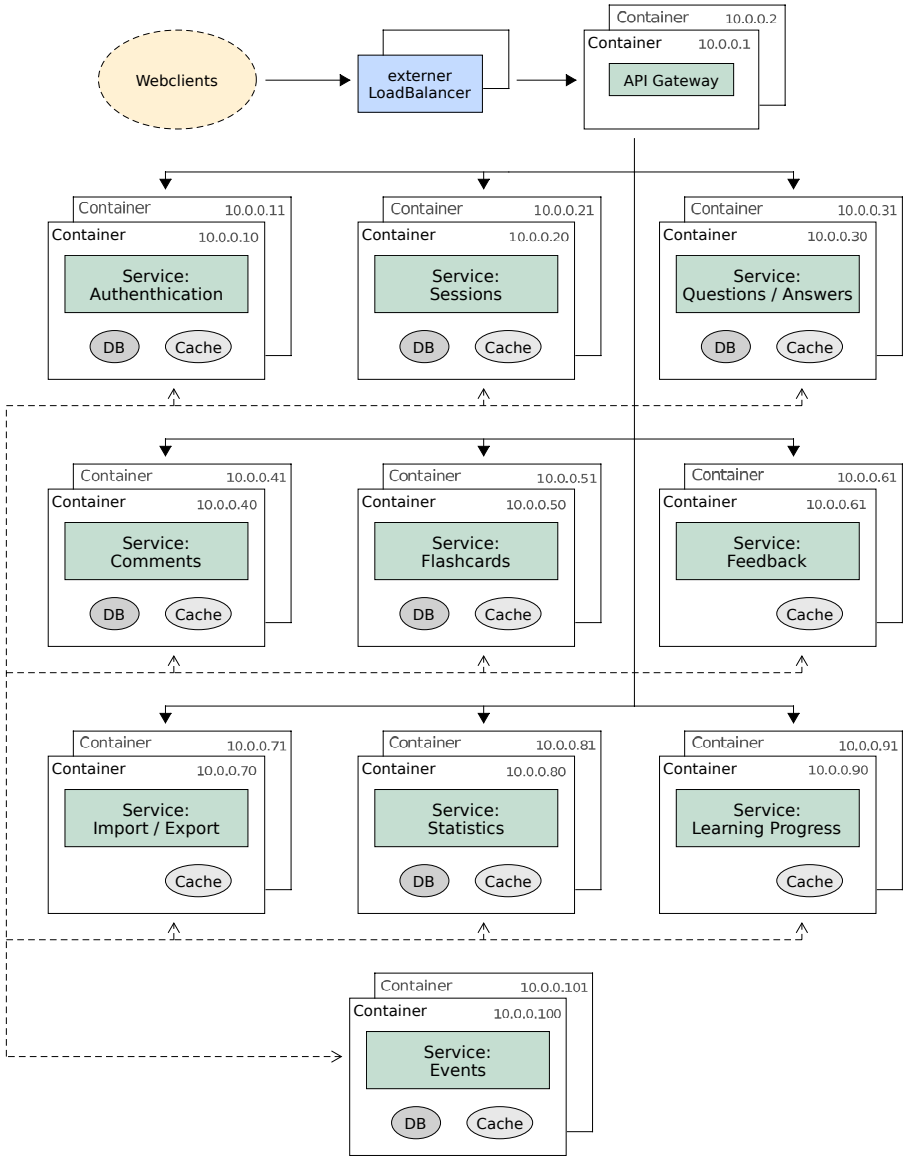

Abb. 5: Das zukünftige Backend von arsnova.voting, aufgeteilt in Microservices

4 Zusammenfassung und Ausblick

Der Beitrag beschreibt drei Bereitstellungsstrategien für Audience-Response-Systeme: (1) den mobilen ARS-Router, (2) das Server-Deployment in Servlet- oder Docker-Containern und (3) ein Cloud-Deployment mit Container-Orchestrierung. Damit wird ein Spektrum aufgespannt von einer mobilen Vor-Ort-Lösung im Hörsaal über einen dezentralen hochschuleigenen Dienst bis zu einer zentralen hochschulübergreifenden Lösung. Im Open-Source-Projekt *ARSnova* der THM wurden die ersten beiden Bereitstellungsstrategien realisiert: Der *ARSnova-Router* befindet sich in der praktischen Erprobung. Das Server-Deployment von *arsnova.voting* wird an mehreren Universitäten und Hochschulen praktiziert, flächendeckend in Sachsen-Anhalt[17] und Hessen[18]. Die Implementierung als hochskalierbare Microservice-Architektur soll zum ARS-Workshop der DeLFI 2017 als Prototyp[19] verfügbar sein.

Literaturverzeichnis

[Be14] Beaumont, D.: How to explain vertical and horizontal scaling in the cloud. 9. April 2014, https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud/.

[Bo16] Bonér, J.: Reactive Microservices Architecture. O'Reilly Media, Boston, 2016.

[Da17] von Danwitz, F.: Erfahrungen mit ARSnova – Live Feedback. 20. April 2017, https://zimblog.uni-wuppertal.de/2017/erfahrungen-mit-arsnova-live-feedback/.

[FB15] Frenger, R.; Bernhardt, S.: Audience-Response-Systeme an der JLU. Abschlussbericht, Justus-Liebig-Universität Gießen, 2015, http://www.uni-giessen.de/fbz/svc/hrz/org/mitarb/abt/3/Archiv/projekte/mob-abst-projekt-bericht.

[Fo14] Fowler, M.: BoundedContext. 15. Januar 2014, https://martinfowler.com/bliki/BoundedContext.html.

[Kä17] Käsler, T.: Redesign und Implementierung eines Legacy-Backends mit Scala-Technologie. Bachelor-Thesis, Technische Hochschule Mittelhessen, Gießen, 28. Februar 2017.

[Qu17] Quibeldey-Cirkel, K.: Lehren und Lernen mit Audience-Response-Systemen. In (de Witt, C.; Gloerfeld, C., Hrsg.): Handbuch Mobile Learning. Springer, Wiesbaden, in Druck, 2017.

Alle Links in den Fußnoten und im Literaturverzeichnis wurden am 20.07.2017 abgerufen.

[17] https://arsnova.uni-halle.de/
[18] https://arsnova.hessen.de/
[19] https://github.com/thm-projects/arsnova-microservices