imc
part of Scheer

Technical Whitepaper

imc Learning Suite
Neue Systemarchitektur und Technologien

im-c.com

Technical Whitepaper

imc Learning Suite
Neue Systemarchtitektur und Technologien

Autor(en): Christoph Gast, Martin Mehlmann
Datum: 25.01.2021

Dokument	Beschreibung
Version	14.8.0 / 14.8.1
Status (Entwurf / Überprüfung / Finalisierung)	Finalisierung
Kontaktperson(en)	Christoph Gast

Historie	Status	Wer
20.11.2017	Entwurf	Christoph Gast
02.10.2020	Überprüfung	Christoph Gast
25.01.2021	Überprüfung	Christoph Gast, Martin Mehlmann
25.01.2021	Finalisierung	Dr. Peter Zönnchen

imc
information multimedia communication AG
Hauptsitz Saarbrücken
Scheer Tower, Uni-Campus Nord
D-66123 Saarbrücken
T. +49 681 9476-0 | Fax -530
info@im-c.com
im-c.com

Inhalt

1 Einleitung

In den letzten Jahren haben agile Entwicklung, Cloud-Hosting, DevOps, Continuous Integration und Continuous Deployment zunehmend an Bedeutung gewonnen. Infolgedessen hat sich eine bestimmte Softwarearchitektur namens "Microservices" als besonders geeignet für komplexe Webanwendungen herauskristallisiert. Durch den Einsatz einer Microservices-Architektur ist es möglich, alle wichtigen Anforderungen an ein modernes System wie Performance, Zuverlässigkeit, Sicherheit, Skalierbarkeit und Innovationsgeschwindigkeit gleichzeitig zu erfüllen. Aus diesem Grund hat sich die imc AG entschieden, ihr Learning Management System (LMS) ab dem Release der Version 14.8.0.0 auf Basis einer solchen Architektur zu entwickeln.

Das vorliegende Dokument beschreibt die verschiedenen Aspekte dieser Architektur und gibt einen Überblick über die zur Umsetzung verwendeten Technologien. Die Zielgruppe dieses Dokuments sind Entscheidungsträger und IT-Fachleute, die für die Evaluierung und Einrichtung eines LMS in ihrer Organisation verantwortlich sind. Der Schwerpunkt liegt darauf, die Vorteile einer solchen Architektur aus Sicht des Kunden aufzuzeigen, ohne dabei zu sehr ins technische Detail zu gehen.

2 Systemarchitektur

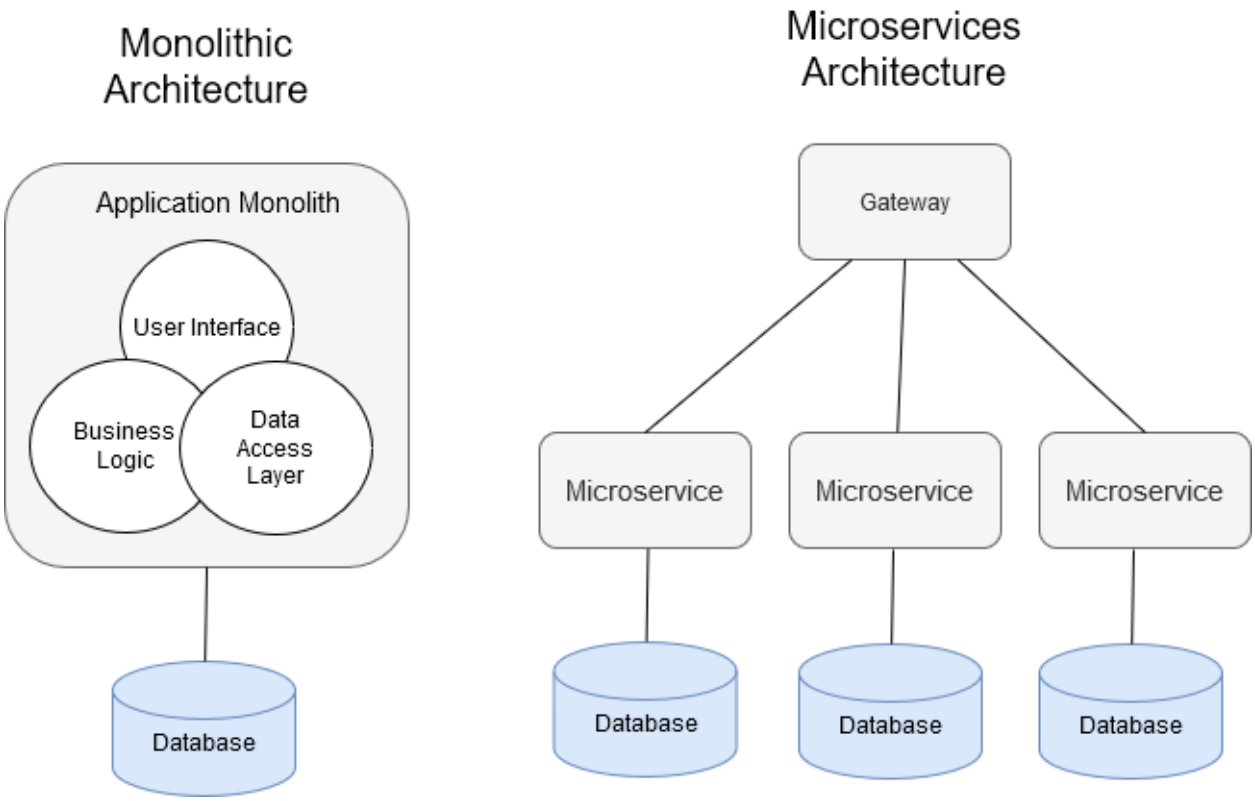

Abb. 2.1: Systemarchitektur

2.1 Microservices

Bis zur Version 14.8.0.0 war das LMS als monolithische Anwendung implementiert. Eine monolithische Anwendung ist in sich geschlossen und unabhängig von anderen Anwendungen. Sie stellt die gesamte Funktionalität des Systems zur Verfügung und führt alle erforderlichen Aufgaben selbständig aus. Der Vorteil einer monolithischen Anwendung ist ihre Unabhängigkeit, d. h. sie kann als eine einzige Einheit eingesetzt werden.

Mit zunehmender Größe und Komplexität der Anwendung erweist sich diese Unabhängigkeit jedoch eher als Hindernis denn als Hilfe. Der Programmcode, der aus Hunderttausenden von Zeilen bestehen kann, befindet sich oft in einem einzigen Repository und ist schwer zu verwalten. Es ist eine große Disziplin erforderlich, um eine modulare Struktur zu gewährleisten. Selbst die kleinste Änderung an einem Teil der Anwendung erfordert, dass die gesamte Anwendung neu erstellt und eingesetzt wird. Dies ist fehleranfällig, mühsam und zeitaufwendig. Darüber hinaus kann eine monolithische Architektur dazu führen, dass ein Release verschoben wird, weil die

Implementierung an einem bestimmten Teil der Anwendung noch nicht abgeschlossen ist, alle anderen Teile aber schon.

Diese Tatsachen verlangsamen die Innovationsgeschwindigkeit und führen zu einem Zustand, in dem das System mit der Zeit immer fehleranfälliger wird. Aus diesem Grund hat sich die imc AG entschieden, ihre Architektur auf eine moderne Microservices-Architektur umzustellen. Auch wenn dies eine große Investition bedeutet, sind wir davon überzeugt, dass wir unseren Kunden damit die bestmögliche Softwarelösung für ihre Bedürfnisse bieten.

Microservices sind ein Architekturmuster, bei dem das Gesamtsystem aus einer Menge von unabhängigen Diensten besteht, die über sprachunabhängige Programmierschnittstellen miteinander kommunizieren. Die Dienste sind weitgehend entkoppelt und jeder Dienst führt eine wohldefinierte kleine Aufgabe aus. Der Grundgedanke eines Microservices besteht darin, eine Sache zu tun und diese eine Sache gut zu machen. Auf diese Weise ermöglichen Microservices einen modularen Aufbau von Anwendungssoftware. Eine Microservices-Architektur bringt eine Reihe von deutlichen Vorteilen mit sich:

- Die Komplexität eines einzelnen Dienstes ist gering und leicht überschaubar. Dies reduziert die Wahrscheinlichkeit von Implementierungsfehlern und gewährleistet eine hohe Qualität der Software.
- Jeder Dienst wird unabhängig von allen anderen Diensten entwickelt. Da die Abhängigkeiten zu anderen Diensten gering gehalten werden, kann ein Dienst in der Regel freigegeben werden, sobald er fertiggestellt ist. Dies verbessert die Innovationsgeschwindigkeit deutlich.
- Ein Dienst kann unabhängig bereitgestellt werden. Sobald eine neue Version eines Dienstes verfügbar ist, kann sie eine ältere Version desselben Dienstes ersetzen, auch in einem laufenden System. Dies führt zu schnelleren Releases, die mit weniger oder sogar ohne Ausfallzeit bereitgestellt werden können.
- Die Schnittstellen der Services basieren auf bewährten Technologien wie REST und asynchronem Message Passing. Bei der imc AG werden diese Schnittstellen von einem engagierten Team erfahrener Experten definiert, um eine zuverlässige und performante Kommunikation zu gewährleisten und unerwünschte Abhängigkeiten zu vermeiden, die bei monolithischen Anwendungen oft entstehen.
- Microservices können automatisch, dynamisch und unabhängig voneinander skaliert werden. Durch das Einrichten von Replikaten, also mehreren Instanzen desselben Dienstes, ist es möglich, die Last auszugleichen und das System robust zu halten, auch wenn eine Instanz eines Dienstes ausfällt. Dies führt zu einer besseren Leistung und Fehlertoleranz.
- Durch den Einsatz von Containern und Systemen zur Container-Orchestrierung, wie z. B. Kubernetes, wird die Nutzung der Ressourcen jederzeit automatisch an die jeweiligen Gegebenheiten angepasst. Dies führt zu einer optimalen Ressourcenauslastung und Kosteneinsparungen.

- Die imc AG bedient vor allem große Kunden mit mehreren zehntausend Nutzern. Durch den Einsatz von Microservices ist es nun möglich, aus der Menge der verfügbaren Services ein System individuell zusammenzustellen, wobei jeder dieser Services eine bestimmte Funktionsdomäne implementiert. Dies ermöglicht neue Preismodelle, da die Services unabhängig voneinander verkauft werden können und das Basissystem zu einem niedrigeren Preis angeboten werden kann, was es auch für kleinere Kunden attraktiv und erschwinglich macht.

Natürlich gibt es auch Nachteile bei der Verwendung einer Microservices-Architektur:

- Da Microservices eine verteilte Architektur sind, weisen sie alle Probleme auf, die mit verteilten Architekturen im Allgemeinen einhergehen, wie z. B. erhöhte Komplexität, Datenkonsistenz und die Notwendigkeit einer ausgefeilten Fehlerbehandlung. Da unsere Softwareentwickler entsprechend geschult sind, sind wir davon überzeugt, dass wir die erhöhte Komplexität sicher handhaben können.
- Eine Microservices-Architektur benötigt in der Regel mehr Ressourcen als eine monolithische Anwendung, insbesondere wenn Container eingesetzt werden. Allerdings ist die Hardware heutzutage meist nicht mehr der limitierende Faktor. Zudem sorgen Systeme zur Container-Orchestrierung dafür, dass nur die Ressourcen genutzt werden, die tatsächlich benötigt werden.
- Das Deployment einer Microservices-Architektur ist anspruchsvoller als das einer monolithischen Anwendung. Aus diesem Grund stellt die imc AG ihren On-Premise-Kunden Deployment-Pakete für verschiedene Zielplattformen zur Verfügung, die weitgehend in sich geschlossen sind. Das Hosting von Cloud-Systemen wird bei imc von einem erfahrenen Team von Ingenieuren durchgeführt.

2.2 Containerisierung

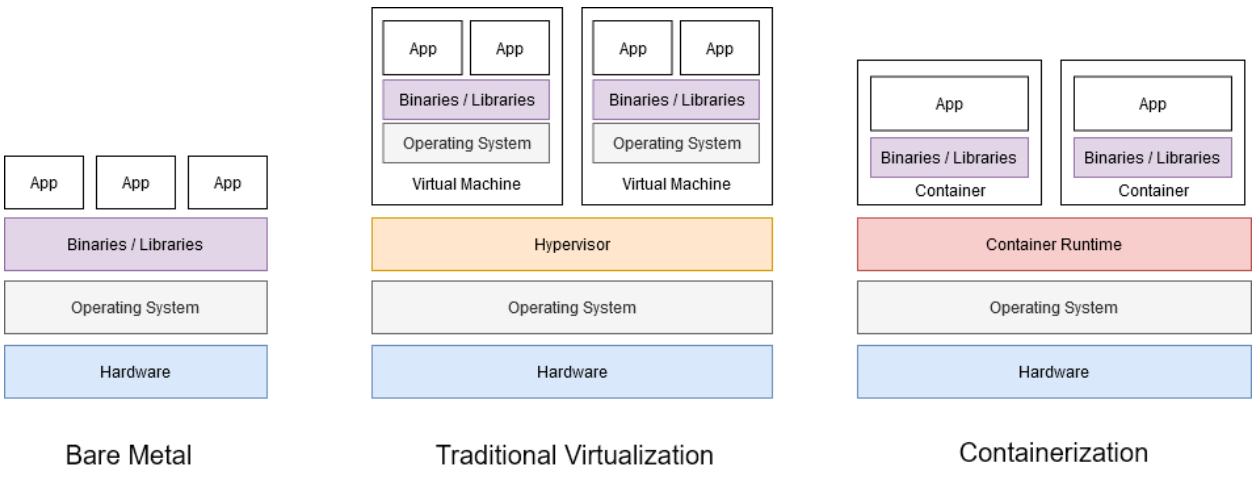

Abb. 2.2: Containerisierung

Containerisierung ist eine Form der Betriebssystemvirtualisierung, bei der Anwendungen in isolierten Softwareumgebungen, den sogenannten "Containern", ausgeführt werden. Ein Container ist im Wesentlichen eine vollständig verpackte Datenverarbeitungsumgebung, die die Anwendung, ihre Abhängigkeiten und ihre Konfiguration in einem einzigen "Container-Image" bündelt. Mit Hilfe einer Containerisierungssoftware wie Docker können mehrere Container auf demselben gemeinsamen Betriebssystem ausgeführt werden.

Der Container selbst ist vom Betriebssystem abstrahiert und hat nur begrenzten Zugriff auf die darunter liegenden Ressourcen. Dadurch kann die containerisierte Anwendung in verschiedenen Infrastrukturen betrieben werden, auf Bare Metal, in virtuellen Maschinen und in der Cloud - ohne dass sie für jede Umgebung angepasst werden muss.

Seit dem Release 14.8.0.0 kann das LMS als ein Satz von Container-Images bereitgestellt werden. Diese Container-Images werden zusammen gestartet, um ein isoliertes Netzwerk von Containern zu bilden, in dem ein definierter Satz von Ports für das Host-System zugänglich gemacht werden kann. Container eignen sich besonders gut für eine Microservices-Architektur, bei der jeder Dienst und alle seine Abhängigkeiten in einem einzigen Container-Image gebündelt sind.
Die Containerisierung bietet eine Menge von Vorteilen:

- Portabilität zwischen verschiedenen Plattformen. Docker-Container können fast überall ausgeführt werden, sowohl auf virtualisierten Infrastrukturen als auch auf Bare-Metal-Servern. Sie können in der Cloud oder auf jeder selbst gehosteten Maschine mit Linux oder Microsoft Windows eingesetzt werden.
- Verbesserte Sicherheit durch Isolierung der Anwendungen vom Hostsystem und voneinander.
- Schnelle und einfache Installations-, Upgrade- und Rollback-Prozesse mit einer Container-Orchestrierungssoftware wie Kubernetes.
- Skalierbarkeit und Replikation auf Container-/Microservices-Ebene. Dies ermöglicht performante und hochverfügbare Systemimplementierungen.
- Flexibles Routing zwischen Diensten, die nativ von Containerisierungsplattformen unterstützt werden.

Aufgrund dieser Vorteile empfehlen wir, das LMS nach Möglichkeit in einer containerisierten Umgebung zu betreiben. Natürlich unterstützen wir weiterhin die Option, die neue Microservices-Architektur in einer nicht containerisierten Umgebung einzusetzen. Beginnend mit 14.8.0.0 stellen wir mit jeder Version ein Microsoft Windows Deployment-Paket zur Verfügung, das vollständig in sich geschlossen ist. Neben den WAR-Dateien, die für die Ausführung der Dienste erforderlich sind, enthält es einen Tomcat-Servlet-Container, eine Java-Laufzeitumgebung und Wartungs-skripte zur Installation und Wartung der Bereitstellung als Satz von Microsoft Windows-Diensten.

2.3 Kommunikation

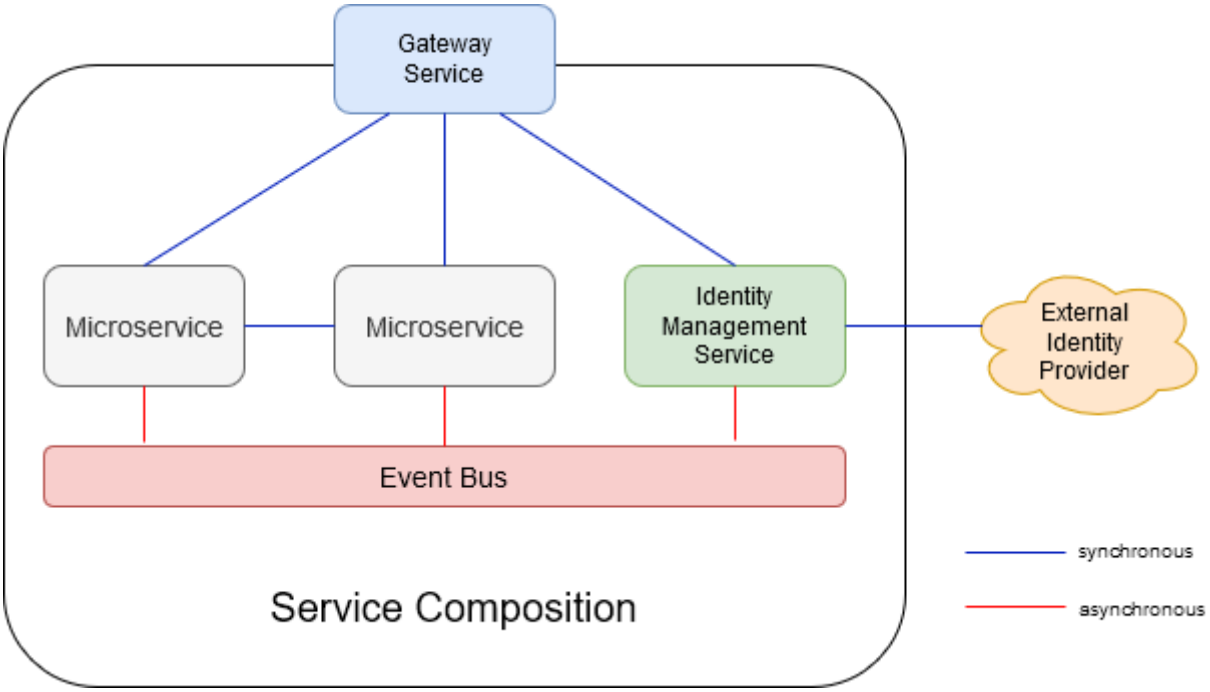

Abb. 2.3: Kommunikation.

In Bezug auf die Kommunikation wird zwischen externer und interner Kommunikation unterschieden. Innerhalb des Systems kommunizieren die Microservices entweder synchron über HTTP oder durch asynchrone Nachrichten über einen Message Bus. Der Event-Bus basiert auf Active MQ Artemis. Ob synchrone oder asynchrone Kommunikation verwendet wird, hängt vom jeweiligen Anwendungsfall ab.

Von außen ist das System standardmäßig nur über einen einzigen HTTPS-Port zu erreichen. Alle eintreffenden Anfragen werden von einem dedizierten Microservice, dem Gateway-Dienst, bearbeitet. Bei jeder Anfrage prüft und verifiziert das Gateway ein JWT, das zur Authentifizierung des Benutzers dient. Anschließend leitet er die eingehenden Anfragen an den entsprechenden Dienst weiter, wobei er eine Reihe von Routing-Regeln verwendet und somit als Reverse-Proxy fungiert. Das Gateway basiert auf Netflix Zuul.

Falls das Gateway eine Anfrage ohne gültiges JWT erhält, leitet es diese Anfrage an den Identity Management Service (IDM) weiter. Der IDM unterstützt verschiedene Authentifizierungsmethoden, um den Benutzer zu authentifizieren. Bei Erfolg gibt er ein JWT aus, das einige grundlegende Informationen über den Benutzer als Nutzdaten enthält. Für browserbasierte Clients wird das JWT als Cookie gespeichert, so dass jede weitere Anfrage das Gateway passiert und die Dienste innerhalb der Komposition erreicht. Das LMS stellt eine umfangreiche REST-API zur Verfügung, die von außen über das Gateway zugänglich ist. Der Zugriff auf die meisten Endpunkte erfordert jedoch eine Authentifizierung.
Gateway-Dienst und IDM sind beides Kerndienste, die Teil jeder Bereitstellung sind.

2.4 Konfigurationsmanagement

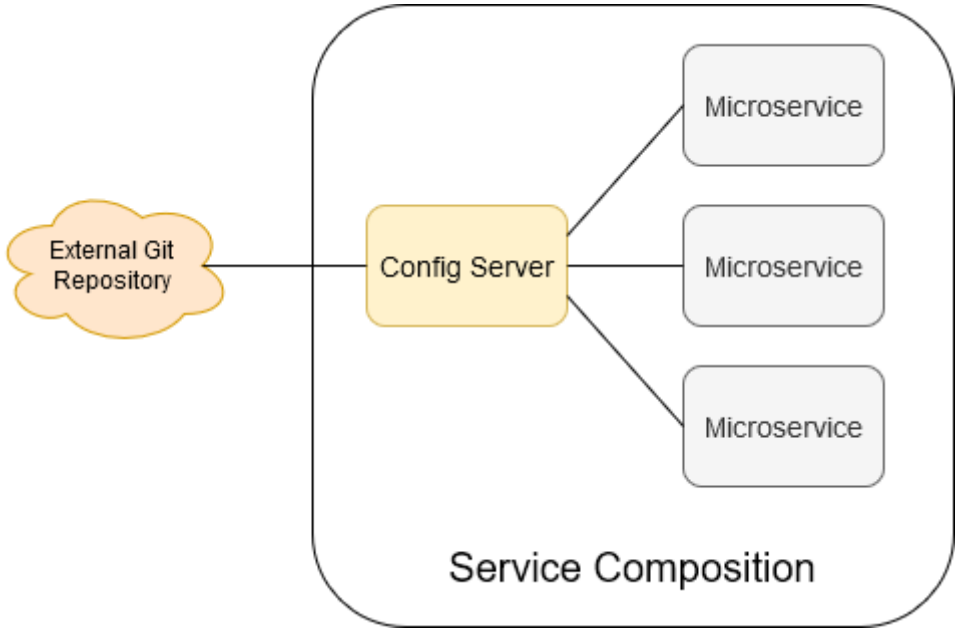

Abb. 2.4: Startkonfiguration.

Die Startkonfiguration umfasst alles, was notwendig ist, damit das System ordnungsgemäß im gewünschten anfänglichen Systemzustand hochfährt. Dieser Teil der Konfiguration wird beim Systemstart verarbeitet und ändert sich nicht zur Laufzeit. Falls die Startup-Konfiguration geändert werden muss, ist ein Neustart des Systems erforderlich. Die Startup-Konfiguration erfolgt über Konfigurationsdateien mit einem externen, zentralisierten Ansatz.

Extern bedeutet, dass die Konfiguration nicht Teil der Build-Artefakte ist, sondern außerhalb von ihnen lebt. Durch diese Trennung ist es möglich, die Konfiguration eines Dienstes zu ändern, ohne dessen Binärdateien neu erstellen zu müssen.

Zentralisiert bedeutet, dass es einen dedizierten Dienst gibt, den Config Server, der Teil jedes Deployments ist und die Konfiguration für alle anderen Dienste über eine REST-API bereitstellt. Bitte beachten Sie, dass dies zu einer Startabhängigkeit führt. Beim Starten des Systems warten alle Dienste darauf, dass der Config Server verfügbar ist, um sich vor dem Start zu konfigurieren. Der Config Server basiert auf dem Spring Cloud Config Server. Die eigentlichen Konfigurationsdateien können in verschiedenen Backends liegen, z. B. in einem Git-Repository, auf einem Webserver oder auf einem lokalen oder einem gemounteten Dateisystem.
Die Laufzeitkonfiguration hingegen umfasst alles andere, also alle Konfigurationseinstellungen, die beim Systemstart nicht verfügbar sein müssen. Die Laufzeitkonfiguration wird über die Benutzeroberfläche des Config-Managers im ILS-Dienst vorgenommen. Die dort zur Laufzeit vorgenommenen Änderungen werden in der ILS-Datenbank gespeichert. Alle anderen Dienste verwenden einen internen REST-API-Endpunkt, um ihre Laufzeitkonfiguration in regelmäßigen kurzen Abständen abzufragen, um sie anzuwenden.

Bitte beachten Sie, dass die oben beschriebene Trennung in Startup- und Runtime-Konfiguration weitgehend, aber noch nicht vollständig umgesetzt ist. Das bedeutet, dass es immer noch Werte in den Konfigurationsdateien gibt, die besser in der Datenbank gehalten werden sollten, um sie ändern zu können, ohne dass ein Systemneustart erforderlich ist. Dennoch werden wir diese Aufteilung in naher Zukunft vollständig implementiert haben.

2.5 Systembereitstellung

Für On-Premise-Kunden kann das LMS als Microservices auf verschiedenen Plattformen installiert werden. Die beste Option aus unserer Sicht ist die Bereitstellung in einer Container-Umgebung aufgrund der oben genannten Vorteile. Wir bieten auch Deployment-Pakete für die Installation auf einer Bare-Metal-Maschine mit einem Microsoft Windows-Betriebssystem an. Ein Deployment-Paket für ein Bare-Metal-Linux stellen wir derzeit nicht zur Verfügung.

Unsere kontinuierliche Pipeline erstellt automatisch Artefakte für jeden Service, den wir anbieten. Dies sind in der Regel WAR-Dateien für Java-Backend-Dienste, die auf Spring boot basieren und entweder in einem Tomcat Servlet Container oder als eigenständige Anwendungen ausgeführt werden können. Für Frontend-Dienste sind die Artefakte minimierte, komprimierte Archive mit JavaScript-Quellen und anderen Assets, die auf einem einfachen Webserver extrahiert werden können. Neben den Artefakten baut die Pipeline auch Docker-Images für jeden Dienst. Diese werden in unserer internen Docker-Registry gehostet und im Falle eines neuen Releases in die AWS ECR oder eine andere Container-Registry gepusht, um sie den Kunden zur Verfügung zu stellen.

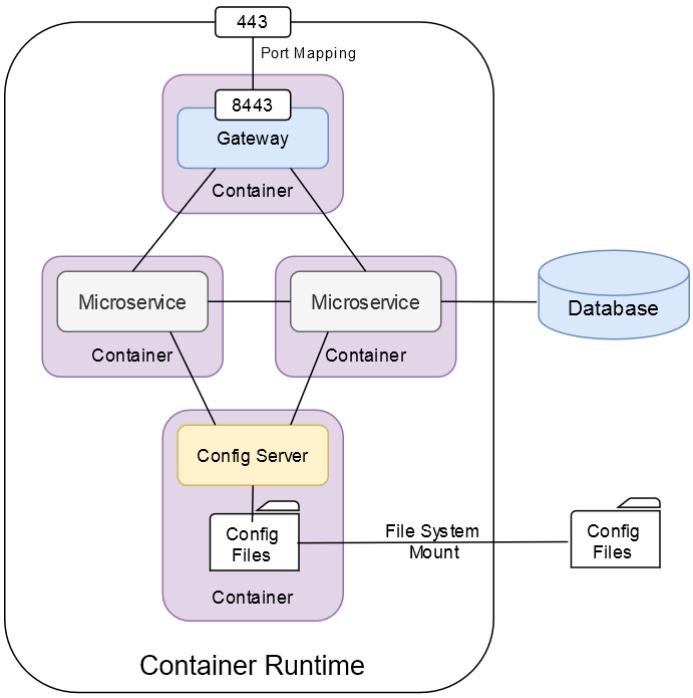

Abb. 2.5: Container-Bereitstellung.

Für eine containerisierte Bereitstellung stellen wir Docker-Images in kundenspezifischen AWS ECR-Namespaces bereit. Bitte beachten Sie, dass es auch mehrere Images für einen einzelnen Service geben kann, z. B. für Backend, Frontend und sogar Datenbank. Diese Images können von einem Kunden nach erfolgreicher Anmeldung gezogen werden. Zusätzlich liefern wir die Startkonfigurationsdateien als komprimiertes Archiv zusammen mit einer docker-compose.yml-Datei, die zeigt, wie die Dienste zusammengesetzt werden. Natürlich ist auch eine ausführliche Dokumentation im Paket enthalten.

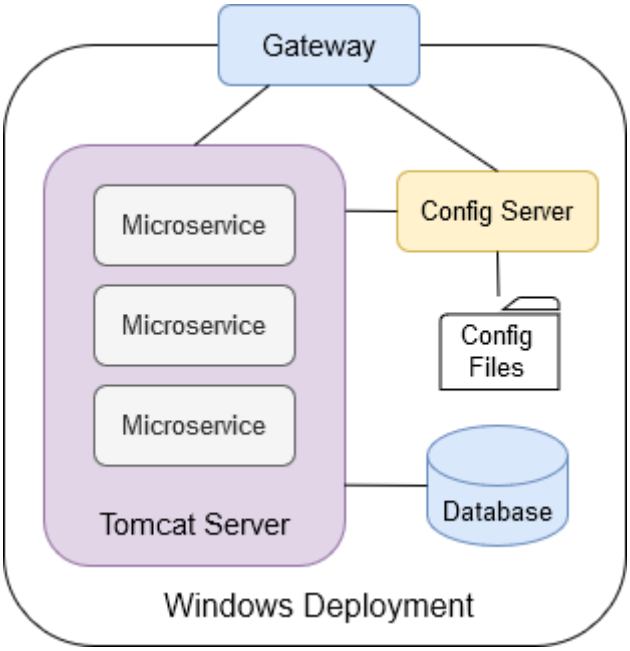

Abb. 2.6: Windows-Bereitstellung.

Für Bare-Metal-Implementierungen unter Microsoft Windows stellen wir ein Paket bereit, das alles enthält, was zur Installation und Wartung des Systems erforderlich ist. Dazu gehören ein abgehärteter Tomcat-Servlet-Container, Artefakte aller Dienste, eine Java-Laufzeitumgebung, Konfigurationsdateien, Wartungsskripte und eine ausführliche Dokumentation. Die meisten Service-Artefakte werden in Tomcat bereitgestellt, es gibt jedoch zwei Ausnahmen, das Gateway und der Config Server werden als eigenständige Spring-Boot-Anwendungen bereitgestellt. Tomcat, Gateway und Config Server werden bei der Installation des Systems als Microsoft Windows-Dienste registriert, um einen automatischen Start beim Hochfahren von Microsoft Windows zu ermöglichen und die Möglichkeit zu haben, diese Dienste über den Microsoft Windows Services Manager zu stoppen und zu starten.
Für Cloud-Kunden bieten wir eine Vielzahl von Bereitstellungsoptionen an. Bitte kontaktieren Sie unsere Hosting-Experten, damit sie Ihre Anforderungen einschätzen und die bestmögliche Lösung für Sie finden können. Insbesondere die Bereitstellung auf Kubenetes ermöglicht eine Vielzahl von Optionen zur Anpassung und Feinabstimmung.

2.6 Skalierbarkeit und Lastausgleich

Wenn es um den Lastausgleich und den ausfallsicheren Betrieb einer containerisierten Bereitstellung geht, bietet Kubernetes mehrere Möglichkeiten, dies zu erreichen, z. B. durch das Einrichten von Replikaten von Containern und durch die Verwendung des Kubernetes-Ingress-Controllers. Neben dem in Kubernetes integrierten Lastausgleichs bieten viele Cloud-Provider Container-Lastausgleich-Dienste mit providerabhängigen Fähigkeiten an. Wir verweisen den Leser auf die Dokumentation der Container-Orchestrierungsplattform und des Cloud-Anbieters für weitere Informationen. Im Allgemeinen gibt es für containerisierte Bereitstellungen eine große Auswahl an Lastausgleich-Lösungen auf dem Markt. Der Hauptvorteil einer containerisierten Bereitstellung in Bezug auf Lastausgleich ist die Tatsache, dass es möglich ist, einzelne Container und damit Dienste dynamisch und automatisch zu skalieren. Das führt zu hochperformanten Systemen, die sich je nach Ressourcenbedarf anpassen.

Bei einem Windows-Einsatz wird die Skalierung einzelner Dienste, die als Kontexte im Tomcat-Servlet-Container laufen, nicht unterstützt. Stattdessen kann das System nur skaliert werden, indem Replikate des gesamten Systems eingerichtet werden. Das heißt, es werden zwei Bereitstellungen parallel betrieben und ihnen ein Server vorgeschaltet, der Lastausgleich unterstützt, z. B. Microsoft IIS. Diese Form der Skalierung ist fix und passt sich nicht an, solange keine zusätzliche Skalierungssoftware im Spiel ist.

Soll das LMS in einem Cluster-Modus mit mehreren Replikaten betrieben werden, sind einige Besonderheiten zu beachten:

- Das LMS ist nicht zustandslos und verwendet Web-Sessions, um den aktuellen Benutzer zu identifizieren. Daher muss auf dem Load Balancer die Session-Affinität aktiviert werden.
- Das LMS speichert Dateien nicht als binäre Blobs in der Datenbank, sondern als reguläre Dateien in einem eigenen Verzeichnis "data" im Dateisystem. Dieses Verzeichnis muss von allen Replikaten gemeinsam genutzt werden, indem Netzlaufwerke für Bare-Metal-Bereitstellungen von Microsoft Windows oder gemeinsam genutzte Cloud-Dateispeicherdienste wie Azure Files oder AWS EFS für containerisierte Bereitstellungen verwendet werden.

2.7 Zentralisierte Protokollierung

Bei Bare-Metal-Bereitstellungen von Microsoft Windows befinden sich alle Protokolldateien in einem Unterordner des Installationsordners.

Bei containerisierten Bereitstellungen erfolgt die Protokollierung innerhalb der einzelnen Container. Sobald ein Container z. B. von der Orchestrierungssoftware nicht mehr bereitgestellt wird, sind die Protokolldateien für diesen Container nicht mehr verfügbar. Um die Persistenz der Logfiles zu ermöglichen, bietet unsere Architektur die Möglichkeit, die Konsolenausgabe und alle Logfiles jedes Dienstes an einem zentralen Ort zu sammeln. Diese Aufgabe wird von einem dedizierten Dienst, dem Logging Service, übernommen. Der Logging Service beinhaltet eine Elasticsearch -Instanz zur Aggregation der Logdateien und eine Kibana-Instanz zur Visualisierung und Analyse der Logdateien. Zusammen mit Logstash, einer kleinen Anwendung, die Teil jedes Service-Images ist, wird das zentralisierte Logging realisiert.