Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

DHBW
Duale Hochschule
Baden-Württemberg

Zukunftsfrabrik Bodensee: Data Analytics

aus dem Studiengang IT-Security | IT Security | Mobile Computer Science

an der Cooperative State University Baden-Württemberg Ravensburg
Campus Friedrichshafen

von

Jonas Thiel

Nico Nasuti

Lars Zeppenfeld

10.07.2026

Matrikelnummer, Kurs:	3516503, TIS23 1234567, TIS23 1234567, TIM23
Unternehmen:	Boehringer Ingelheim Pharma GmbH & Co KG, 88397, Biberach, Germany Boehringer Ingelheim Pharma GmbH & Co KG, 88397, Biberach, Germany Boehringer Ingelheim Pharma GmbH & Co KG, 88397, Biberach, Germany
Betreuer im Unternehmen:	Prof. Thomas Dietmüller

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Sperrvermerk

Die vorliegende Arbeit mit dem Titel

Zukunftsfrabrik Bodensee: Data Analytics

enthält unternehmensinterne bzw. vertrauliche Informationen der Boehringer Ingelheim Pharma GmbH & Co KG, ist deshalb mit einem Sperrvermerk versehen und wird ausschließlich zu Prüfungszwecken im Studiengang IT-Security | IT Security | Mobile Computer Science der Cooperative State University Baden-Württemberg Ravensburg Campus Friedrichshafen vorgelegt.

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte (Boehringer Ingelheim Pharma GmbH & Co KG) vorliegt.

Biberach 10.07.2026

DocuSigned by:
Jonas Thiel
B0D324B59F60402...

Jonas Thiel

Signiert von:
Nico Nasuti
B302BD1E816F42B...

Nico Nasuti

DocuSigned by:
Lars Zeppenfeld
A2EBAD63E70840C...

Lars Zeppenfeld

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Selbstständigkeitserklärung

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden- Württemberg vom 29.09.2017. Wir versichern hiermit, dass wir unsere Arbeit mit dem Thema:

Zukunftsfrabrik Bodensee: Data Analytics

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass alle eingereichten Fassungen übereinstimmen.

Biberach 10.07.2026

DocuSigned by:
Jonas Thiel
B0D324B59F69402...

Jonas Thiel

Signiert von:
Nico Nasuti
B302BD1E816F42B...

Nico Nasuti

DocuSigned by:
Lars Zeppenfeld
A2EBAD83E70840C...

Lars Zeppenfeld

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Inhaltsverzeichnis

1 Einleitung 1
1.1 Problemstellung 1
1.2 Zielsetzung der Arbeit 2
1.3 Aufbau der Arbeit 2

2 Theoretische und technologische Grundlagen 3
2.1 Manufacturing Execution Systems (MES) und FORCAM FORCE 3
2.2 Moderne API-Architekturen: REST vs. GraphQL 6
2.3 Identity & Access Management (IAM): Keycloak, OAuth 2.0 und OIDC 6
2.4 Virtualisierungstechnologien: Hypervisor vs. Containerisierung 8
2.5 Moderne Webentwicklung und React 9
2.6 Das Next.js Framework und hybride Rendering-Strategien 9

3 Aktueller Systemaufbau (Ist-Zustand) 11
3.1 Gesamtüberblick und Datenfluss 11
3.2 Virtualisierungsebene mit Proxmox 12
3.3 Aufbau des Proxmox Web-Interfaces 14
3.4 Logische VM-Architektur und Rollenverteilung 15
3.5 VM 100 – Datenbankserver und Datenmodellierung (Das Archiv) 16
3.6 VM 102 – Datenerfassungsserver (Der Reporter) 17
3.7 VM 104 – Anwendungsserver (Die Zentrale) 18
3.8 VM 105 – Datenerfassung Labor (Der Hochschul-Reporter) 18
3.9 FORCAM Edge – Konnektivitätsschicht (Der Universal-Übersetzer) 19
3.10 FORCAM FORCE – Anwendungsschicht 21
3.11 Authentifizierung & Sicherheit mit Keycloak 21
3.12 Schnittstellen: REST-API und Token-Beschaffung 24
3.13 Aktuelle Herausforderungen und Workarounds 26

4 Systemintegration: Das Next.js Dashboard 29
4.1 Architektur und Technologie 29
4.2 Shopfloor UI/UX und Echtzeit-Streaming 30
4.3 Aktueller Status: Prototyp und nächste Implementierungsschritte 30

5 Monitoring & Wartungshinweise 32
5.1 Kritischer Ressourcen-Engpass – VM 102 32
5.2 Wartungsroutinen 33
5.3 Update-Strategie 34

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

6 Alternative MES-Architektur (Eigenentwicklung) 35
6.1 Motivation: Build-vs-Buy-Entscheidung 35
6.2 Architekturparadigma: Vom Monolithen zur Microservice-Architektur 36
6.3 Schichtenmodell des Eigenentwurfs 36
6.4 Deployment und Betrieb 39
6.5 Technologie-Stack im Überblick 39
6.6 Kritische Würdigung und Trade-offs 40

7 Anhang: Wichtige URLs & Referenzen 42
7.1 Verwendete Technologien & Versionen 42

Literatur ..... a

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Abbildungsverzeichnis

Abbildung 1 Technisches Architektur-Diagramm der FORCAM-Infrastruktur (Gesamtüberblick) 12

Abbildung 2 Software-Architektur – Schichtenmodell der FORCAM-Umgebung . 12

Abbildung 3 Virtualisierungsebene – Proxmox-Host mit VM-Übersicht 14

Abbildung 4 Netzwerk-Topologie – Maschinen → Edge → Applikation 19

Abbildung 5 Authentifizierungsfluss – OAuth2 / OpenID Connect (SSO) 22

Abbildung 6 Aktueller Entwicklungsstand des Next.js-Dashboard-Prototyps 31

Abbildung 7 Arbeitsspeicher-Auslastung der VMs (Schätzwerte) 32

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Tabellenverzeichnis

Tabelle 1 Infrastrukturparameter 13

Tabelle 2 Server-Rollen & VM-Übersicht 16

Tabelle 3 Administration & Zugriffsrollen 20

Tabelle 4 Keycloak-Konfiguration 23

Tabelle 5 Swagger UI 24

Tabelle 6 REST-API Technische Spezifikation 25

Tabelle 7 Netzwerk-Routing und Port-Mapping 27

Tabelle 8 Empfohlene Maßnahmen 33

Tabelle 9 Gegenüberstellung der Technologie-Stacks 40

Tabelle 10 Abwägung: Standardsoftware (Buy) vs. Eigenentwicklung (Build) 41

Tabelle 11 Wichtige URLs & Referenzen 42

Tabelle 12 Verwendete Technologien & Versionen 43

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
ANSI	American National Standards Institute
API	Application Programming Interface
CNC	Computerized Numerical Control
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DBMS	Database Management System
DHBW	Duale Hochschule Baden-Württemberg
DNS	Domain Name System
DOM	Document Object Model
ERP	Enterprise Resource Planning
EoL	End-of-Life
EoS	End-of-Support
FQDN	Fully Qualified Domain Name
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IAM	Identity & Access Management
IIoT	Industrial Internet of Things
ISA	International Society of Automation
ISO	International Organization for Standardization
ISR	Incremental Static Regeneration
IT	Informationstechnik
IWT	Institut für Werkzeug- und Fertigungstechnik
IoT	Internet of Things

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
LVM	Logical Volume Manager
MDE	Maschinendatenerfassung
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
MSSQL	Microsoft SQL Server
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
MitM	Man-in-the-Middle
OAuth	Open Authorization
ODS	Operational Data Store
OEE	Overall Equipment Effectiveness
OIDC	OpenID Connect
OOM	Out of Memory
OPC UA	Open Platform Communications Unified Architecture
OS	Operating System
PKI	Public Key Infrastructure
PLC	Programmable Logic Controller
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comments
RSA	Rivest-Shamir-Adleman
RSC	React Server Components
SAML	Security Assertion Markup Language
SAN	Subject Alternative Name
SHA	Secure Hash Algorithm

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

SPA	Single-Page-Application
SPS	Speicherprogrammierbare Steuerung
SQL	Structured Query Language
SSD	Solid State Drive
SSG	Static Site Generation
SSL	Secure Sockets Layer
SSO	Single Sign-On
SSR	Server-Side Rendering
SWR	Stale-While-Revalidate
TCO	Total Cost of Ownership
TCP/IP	Transmission Control Protocol / Internet Protocol
TEEP	Total Effective Equipment Performance
TLS	Transport Layer Security
TPM	Total Productive Maintenance
TSDB	Time-Series Database
UI	User Interface
UNS	Unified Namespace
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
VE	Virtual Environment
VLAN	Virtual Local Area Network
VM	Virtual Machine (Virtuelle Maschine)
VNC	Virtual Network Computing
XSS	Cross-Site Scripting

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

1 Einleitung

Die globale Fertigungsindustrie durchläuft aktuell eine der gravierendsten Transformationsphasen ihrer Geschichte, die allgemein unter dem Begriff **Industrie 4.0** subsumiert wird [1]. Im Kern dieses Wandels steht die Evolution von der rein automatisierten, mechatronischen Produktion hin zur datengetriebenen, hochvernetzten „Smart Factory“. Während in der dritten industriellen Revolution speicherprogrammierbare Steuerungen (SPS) isolierte Maschinen automatisierten, fordert der moderne Paradigmenwechsel eine nahtlose, horizontale und vertikale Integration aller Systeme – vom physischen Sensor in der Werkshalle (Shopfloor) bis hinauf in die kaufmännischen Cloud-Systeme der Unternehmensebene (Topfloor).

Der entscheidende Wettbewerbsvorteil liegt heute nicht mehr allein in der mechanischen Fräs- oder Drehgeschwindigkeit einer CNC-Maschine, sondern in der Fähigkeit, die großen Mengen anfallender Prozessdaten (Big Data) in Echtzeit zu erfassen, zu harmonisieren und in verwertbare Geschäftsentscheidungen (Actionable Insights) umzuwandeln. In der Praxis scheitert diese Vision jedoch häufig an heterogenen Maschinenparks, proprietären Herstellerprotokollen und veralteten, monolithischen IT-Infrastrukturen (Legacy Systems), die sogenannte „Datensilos“ bilden.

1.1 Problemstellung

Ein solches Datensilo stellt oftmals das Manufacturing Execution System (MES) dar. Zwar aggregieren MES-Lösungen die Maschinen-Telemetrie und berechnen wichtige Leistungskennzahlen, jedoch sind die mitgelieferten Benutzeroberflächen häufig starr, wenig anpassbar und primär für stationäre Desktop-PCs im Büro konzipiert. Für den Werker direkt an der Maschine, der Tablets oder Industrie-Displays nutzt, fehlen oftmals responsive, hochkontrastige und auf das Wesentliche reduzierte Dashboards.

Verschärft wird diese Situation in diesem spezifischen Projekt durch den Lebenszyklus-Status der eingesetzten Software: Das zugrundeliegende System **FORCAM FORCE** wird durch den Hersteller offiziell nicht mehr mit Updates und Support versorgt (End-of-Life / End-of-Support). Dies führt zu schwerwiegenden administrativen Hürden, insbesondere im Bereich der API-Authentifizierung (Swagger/Keycloak), da bei fehlenden Administrator-Zugangsda-

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

ten kein Herstellersupport mehr zur Verfügung steht. Ein nativer, unkomplizierter Datenabruf über die vorgesehenen Schnittstellen ist somit blockiert.

1.2 Zielsetzung der Arbeit

Das primäre Ziel dieses Projekts ist es, diesen architektonischen Flaschenhals zu durchbrechen. Zunächst erfolgt die initiale Konfiguration und Analyse der bestehenden, virtualisierten FORCAM-Infrastruktur (Data Lake), um die genauen Datenstrukturen und Netzwerkwege der Maschinendaten zu verstehen.

Darauf aufbauend wird im Rahmen der Praxisphase ein maßgeschneidertes, modernes Shopfloor-Dashboard konzipiert und prototypisch entwickelt. Das Dashboard wird als Fullstack-Anwendung in **Next.js** programmiert und muss industriellen Standards für Benutzeroberflächen (z.B. Dark Mode, hohe Touch-Target-Sizes) genügen. Eine zentrale Forschungsfrage dieses Projekts lautet dabei: **Wie können trotz der restriktiven Legacy-Schnittstellen und des fehlenden Supports die aggregierten Echtzeit-Maschinendaten performant und ausfallsicher in das neu entwickelte Web-Frontend integriert werden?** Hierzu müssen alternative API-Architekturen (z.B. der Bau einer autarken Middleware) evaluiert werden.

1.3 Aufbau der Arbeit

Die Arbeit folgt einem durchgehenden roten Faden, der von den theoretischen Grundlagen über den konkreten Ist-Zustand bis zur entwickelten Lösung und einem strategischen Ausblick reicht. Kapitel 2 legt zunächst die theoretischen und technologischen Fundamente aus den Bereichen MES, industrielle Maschinenkommunikation (OPC UA/MQTT), Identity Management sowie moderne Webentwicklung. Kapitel 3 überführt diese Konzepte anschließend in die Praxis und beschreibt den vollständigen **aktuellen Systemaufbau (Ist-Zustand)** – von der Virtualisierungsebene über die einzelnen virtuellen Maschinen und Software-Schichten bis hin zum Sicherheitskonzept, den Schnittstellen und den aktuell bestehenden Zugangshürden. Kapitel 4 stellt die darauf aufbauende Eigenentwicklung des Next.js-Dashboards vor. Kapitel 5 widmet sich den Monitoring- und Wartungsstrategien für den laufenden Betrieb. Kapitel 6 skizziert als Ausblick eine alternative, quelloffene MES-Architektur als Eigenentwicklung. Der Anhang fasst abschließend die zentralen URLs und die eingesetzten Technologien zusammen.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

2 Theoretische und technologische Grundlagen

Vor der detaillierten Betrachtung der IT-Infrastruktur und der Dashboard-Implementierung werden in diesem Kapitel die wesentlichen technologischen Konzepte erläutert, die das theoretische Fundament dieses spezifischen Praxisprojekts bilden. Die Reihenfolge folgt dabei dem späteren Systemaufbau: von der Rolle eines MES in der Fabrik über die Schnittstellen- und Sicherheitstechnologien bis hin zur Virtualisierung und den Web-Frameworks, mit denen das Dashboard umgesetzt wird.

2.1 Manufacturing Execution Systems (MES) und FORCAM FORCE

Den Ausgangspunkt bildet die Frage, an welcher Stelle ein MES wie das im Projekt eingesetzte FORCAM FORCE überhaupt in der IT-Landschaft einer Fabrik verortet ist und welche Aufgaben es übernimmt. Um diese Einordnung nachvollziehbar zu machen, wird zunächst das normierte Schichtenmodell der Fabrik-IT vorgestellt, bevor die konkreten Kommunikationsprotokolle und Kennzahlen erläutert werden.

Die Automatisierungspyramide (ANSI/ISA-95) Um das informationstechnische Zusammenspiel in einer Fabrik zu standardisieren, definiert die internationale Leit-Norm **ANSI/ISA-95** die sogenannte Automatisierungspyramide [2]. Diese strikte hierarchische Trennung teilt die Fabrik-IT in fünf logische Ebenen auf:

- **Ebene 0 (Physischer Prozess):** Die tatsächlichen mechanischen und elektrischen Prozesse (Motoren, Ventile, Pumpen).
- **Ebene 1 (Sensoren & Aktoren):** Die Datenerfassung und physische Manipulation direkt an der Maschine.
- **Ebene 2 (Steuerungsebene / Shopfloor):** Speicherprogrammierbare Steuerungen (SPS bzw. PLC), welche die Millisekunden-genaue Logik der Maschinen ausführen (z.B. Siemens S7).
- **Ebene 3 (Manufacturing Operations Management):** Das **Manufacturing Execution System (MES)**. Es übernimmt die Feinplanung, Qualitätskontrolle und Maschinendatenerfassung (MDE) in Echtzeit über die gesamte Fabrik hinweg.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

- **Ebene 4 (Unternehmensebene / Topfloor):** Das Enterprise Resource Planning (ERP), wie z.B. SAP, welches sich rein um kaufmännische Prozesse (Bestellwesen, Rechnungen) im Rahmen von Wochen- oder Monatszyklen kümmert.

Ein MES wie FORCAM FORCE agiert demnach nativ auf Ebene 3. Es schließt die informationstechnische Lücke zwischen den trägen, datenbankzentrierten ERP-Systemen und den sehr schnellen, rohen Binärsignalen der Maschinensteuerungen auf dem Shopfloor.

Industrielle Kommunikation (OPC UA und MQTT) Damit ein MES-System (Ebene 3) mit den Maschinen (Ebene 2) sprechen kann, werden standardisierte Netzwerkprotokolle benötigt. Historisch bedingt war der Shopfloor geprägt von hunderten proprietären Feldbussen (wie Profibus oder CAN-Bus), die nicht miteinander kompatibel waren. In modernen Industrie-4.0-Architekturen haben sich primär zwei IP-basierte Protokolle durchgesetzt:

1. **OPC UA (Open Platform Communications Unified Architecture):** Ein plattformunabhängiger, serviceorientierter Standard, der nicht nur Rohdaten, sondern deren semantische Bedeutung überträgt. Eine Maschine überträgt nicht nur den nackten Wert „45“, sondern signalisiert explizit im **OPC UA Address Space**: „Temperatur des Bohrkopfs in Grad Celsius = 45“. OPC UA arbeitet klassischerweise im Client/Server-Modell, wobei der Server registrierte Wertänderungen über sogenannte Subscriptions (Monitored Items) aktiv an den Client meldet, statt permanent abgefragt zu werden. Authentifizierung und Verschlüsselung sind über X.509-Zertifikate auf Protokollebene integriert.
2. **MQTT (Message Queuing Telemetry Transport):** Ein sehr leichtgewichtiges Publish/Subscribe-Protokoll. Maschinen (als Publisher) senden ihre Datenpakete hierbei ereignisgesteuert (Event-Driven) an einen zentralen Broker. Das MES (als Subscriber) abonniert lediglich die für OEE-Berechnungen relevanten Datenkanäle („Topics“). MQTT zeichnet sich durch seinen geringen Protokoll-Overhead aus und ist für massive Skalierung bei begrenzter Bandbreite konzipiert.

Das in diesem Projekt verwendete System **FORCAM FORCE** ist eine modulare Enterprise-MES-Lösung, welche der feingranularen Echtzeiterfassung, Visualisierung und Auswertung von Maschinendaten dient. Durch die Anbindung sogenannter Edge-Gateways werden direkte Hardware-Sensordaten aggregiert, in einheitliche Protokolle übersetzt und digitalisiert. Diese horizontale Integration ermöglicht die vollautomatisierte Erhebung zentraler Produktionskennzahlen.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Die primär entscheidende Metrik hierbei ist die Gesamtanlageneffektivität (OEE – Overall Equipment Effectiveness), ein von Seiichi Nakajima im Rahmen des Total Productive Maintenance (TPM) entwickelter Kennzahlrahmen [3]. Die OEE setzt sich multiplikativ aus den Schlüsselfaktoren Verfügbarkeit, Leistung und Qualität zusammen. Mathematisch lässt sich dies wie folgt ausdrücken:

$$\text{OEE} = \text{Verfügbarkeitsfaktor} \times \text{Leistungsfaktor} \times \text{Qualitätsfaktor} \tag{1}$$

Dabei ergibt sich der **Verfügbarkeitsfaktor** aus dem Verhältnis der tatsächlichen Laufzeit (geplante Belegzeit abzüglich ungeplanter Stillstände) zur geplanten Belegzeit. Er berücksichtigt primär ungeplante Stillstände (Maschinenausfälle), die häufig über Kennzahlen wie **MTBF (Mean Time Between Failures)** und **MTTR (Mean Time To Repair)** statistisch erfasst werden. Der **Leistungsfaktor** beschreibt das Verhältnis von theoretischer zu realer Taktzeit. Hierbei werden Performance-Verluste durch kurze Mikro-Stopps (z.B. Materialstaus) oder reduzierte Maschinengeschwindigkeiten identifiziert. Der **Qualitätsfaktor** entspricht dem Anteil der Gutteile an der Gesamtproduktionsmenge und spiegelt somit die Ausschuss- und Nacharbeitsrate (Rework) wider.

Abgrenzung zur TEEP (Total Effective Equipment Performance): Im akademischen Kontext des Produktionsmanagements muss die OEE von der TEEP abgegrenzt werden. Während die OEE lediglich die **geplante Belegzeit** als 100%-Basis heranzieht (z.B. eine 8-Stunden-Schicht von Montag bis Freitag), setzt die TEEP das theoretische Maximum von 24 Stunden an 365 Tagen im Jahr an (Kalenderzeit). Eine Anlage kann folglich eine hervorragende OEE von 95 % aufweisen (während der Schicht läuft sie fehlerfrei), aber gleichzeitig eine niedrige TEEP von 20 % besitzen (weil sie nachts und am Wochenende komplett stillsteht). Für die operative Analyse des Shopfloors durch den Werker ist daher primär die OEE relevant, für das strategische Management zur Planung von Neuanschaffungen hingegen die TEEP.

Der im System implementierte **Data Lake** fungiert bei FORCAM als zentraler, hochperformanter Datenspeicher auf Zeitreihen-Basis. In diesem Speicherbecken werden die eingehenden Ströme von Maschinenwerten unveränderlich historisiert. Technisch wird hier oft eine In-Memory-Datenbank oder eine spezialisierte Time-Series Database (TSDB) eingesetzt, um die sehr hohen Schreib-Frequenzen (High-Throughput Ingestion) abzudecken, bevor sie

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

über REST-Schnittstellen für externe Big-Data-Analysen oder spezialisierte Frontends bereitgestellt werden.

2.2 Moderne API-Architekturen: REST vs. GraphQL

Nachdem geklärt ist, wie die Maschinendaten in das MES gelangen, stellt sich die Frage, wie externe Anwendungen – wie das geplante Dashboard – diese Daten anschließend wieder abrufen. Damit autarke Softwarearchitekturen interoperabel über Netzwerkgrenzen hinweg kommunizieren können, werden APIs eingesetzt. Historisch betrachtet hat sich der **REST-Architekturstil** (Representational State Transfer) als industrieller De-facto-Standard durchgesetzt [4]. Eine **RESTful API** unterliegt strikten Design-Paradigmen, primär der Zustandslosigkeit (Statelessness). Ressourcen (wie angeschlossene Maschinen oder Fehleralarme) werden über eindeutige URLs adressiert und mittels standardisierter HTTP-Verben (`GET`, `POST`, `PUT`, `DELETE`) manipuliert.

In modernen Web-Dashboards offenbart die klassische REST-Architektur jedoch oft deutliche Schwächen, insbesondere das sogenannte **Overfetching** (der Server liefert hunderte Datenfelder, obwohl das Dashboard nur den OEE-Wert benötigt) und das **Underfetching** (das Dashboard muss nacheinander fünf verschiedene REST-Endpunkte abfragen, um alle nötigen KPIs für eine einzige Ansicht zu sammeln). Aus diesem Grund evaluiert die moderne Webentwicklung zunehmend Alternativen wie **GraphQL**. Bei GraphQL existiert nur ein einziger, flexibler API-Endpunkt. Das Frontend (Next.js) sendet dabei genau die Struktur der gewünschten Daten als Abfrage (Query), und das Backend antwortet exakt und ausschließlich mit diesen Feldern. Für das hier konzipierte MES-Dashboard würde eine GraphQL-Middleware die Netzwerklast im Shopfloor reduzieren. Die Übermittlung der Nutzdaten erfolgt bei beiden Technologien nahezu ausschließlich im leichtgewichtigen **JSON-Format** (JavaScript Object Notation).

2.3 Identity & Access Management (IAM): Keycloak, OAuth 2.0 und OIDC

Ein API-Zugriff auf sensible Produktionsdaten setzt eine zuverlässige Zugriffskontrolle voraus. Zum Schutz kritischer Maschinendaten vor unautorisierten Zugriffen ist in der FORCAM-Infrastruktur daher ein zentrales IAM-System auf Basis der Open-Source-Software **Keycloak** etabliert. In der Theorie der Cybersicherheit muss hierbei zwischen **Authentifizierung** („Wer

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

ist dieser Nutzer?“) und **Autorisierung** („Darf dieser Nutzer die OEE-Daten sehen?“) unterschieden werden.

Architekturmuster von Keycloak (Identity Brokering)
Keycloak fungiert in der Infrastruktur als sogenannter „Identity Broker“. Die Architektur unterteilt sich in logische Sicherheitsbereiche, die **Realms**. Innerhalb eines Realms (hier: der `FORCAM`-Realm) werden **Users** (die menschlichen Maschinenbediener), **Roles** (wie Schichtleiter oder Admin) und sogenannte **Clients** verwaltet. Ein Client repräsentiert in Keycloak eine externe Applikation (z.B. das geplante Next.js-Dashboard oder die Swagger-UI), die eine Authentifizierung anfordert.

Die fehlenden administrativen FORCAM-Zugangsdaten (Username/Passwort) stellen in diesem architektonischen Muster einen kritischen „Single Point of Failure“ dar. Ohne Administratorrechte auf den Realm ist es technisch unmöglich, neue **Clients** zu registrieren, Redirect-URIs für das Dashboard freizugeben oder **Service Accounts** für automatisierte Maschinen-Skripte anzulegen. Da die FORCAM-REST-API serverseitig alle Anfragen abweist, die nicht von einem sauber konfigurierten Keycloak-Client mit gültigem Token stammen, blockiert der fehlende Admin-Zugang naturgemäß die gesamte nachgelagerte Schnittstellen-Entwicklung.

Protokolle: OAuth 2.0 und OIDC
Um die Zugriffe der registrierten Clients zu steuern, implementiert Keycloak zwei branchenübliche Protokolle:

1. **OAuth 2.0:** Ist ein reines Autorisierungs-Framework [5]. Es erlaubt dem Next.js-Dashboard, im Namen des Nutzers auf die REST-API zuzugreifen, ohne dass das Dashboard jemals das Passwort des Nutzers zu Gesicht bekommt (Delegated Access).
2. **OpenID Connect (OIDC):** Ist eine Identitätsschicht, die direkt **auf** OAuth 2.0 aufbaut. Sie liefert dem Dashboard ein standardisiertes Identitäts-Token (ID-Token), um den Nutzer sicher anzumelden.

Technisch werden diese Berechtigungen meist als asymmetrisch signiertes **JSON Web Token (JWT)** übertragen. Ein JWT besteht aus drei Base64-URL-codierten Blöcken:

- **Header:** Definiert den Signaturalgorithmus (häufig RS256 – eine RSA-Signatur mit SHA-256).

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

- **Payload (Claims):** Enthält die tatsächlichen Nutzerdaten, Rollen und den Ablaufzeitpunkt (`exp`).
- **Signature:** Ein kryptografischer Hash, der mit dem geheimen privaten Schlüssel des Keycloak-Servers signiert wurde. Das Next.js-Backend kann dieses Token offline über den öffentlichen Schlüssel verifizieren, was eine unbemerkte Manipulation der Payload mathematisch ausschließt.

2.4 Virtualisierungstechnologien: Hypervisor vs. Containerisierung

Sämtliche zuvor beschriebenen Dienste laufen nicht auf eigener Hardware, sondern virtualisiert – ein Konzept, das für das Verständnis des späteren Ist-Zustands zentral ist. Die Bereitstellung komplexer Serverlandschaften (wie FORCAM) erfolgt heute nicht mehr auf physischen Einzelrechnern (Bare-Metal), sondern hochgradig virtualisiert. Die Virtualisierung entkoppelt das Betriebssystem von der zugrundeliegenden Hardware, wodurch Ressourcen effizienter genutzt werden können. Technologisch wird primär zwischen zwei Paradigmen unterschieden:

1. **Hardware-Level-Virtualisierung (Hypervisor):** Ein Hypervisor (Virtual Machine Monitor) teilt physische Hardwarekomponenten wie CPU und RAM in vollständig isolierte Virtuelle Maschinen (VMs) auf. Jede VM betreibt dabei einen vollständigen, eigenen Kernel und ein komplettes Gast-Betriebssystem. Man unterscheidet Typ-2-Hypervisoren (installiert auf einem normalen Desktop-OS, z.B. VirtualBox) von Typ-1-Hypervisoren (Bare-Metal). Ein **Typ-1-Hypervisor** wie das in diesem Projekt genutzte **Proxmox VE** basiert auf KVM (Kernel-based Virtual Machine) und wird direkt auf die Server-Hardware installiert. Dies bietet hohe I/O-Performance und eine starke Sicherheitsisolation für kritische Datenbanken.
2. **OS-Level-Virtualisierung (Containerisierung):** Bei Technologien wie **Docker** teilen sich mehrere isolierte Umgebungen (Container) denselben Linux-Kernel des Host-Betriebssystems. Container verpacken lediglich die Applikation samt ihrer direkten Abhängigkeiten (Libraries). Da kein gesamtes Gast-OS simuliert werden muss, starten Container in Millisekunden und benötigen nur einen Bruchteil des Arbeitsspeichers einer VM.

In modernen Architekturen werden diese Konzepte kombiniert: Der schwere, monolithische FORCAM Data Lake läuft aus Sicherheits- und Stabilitätsgründen in isolierten Proxmox-VMs,

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

während leichtgewichtige Microservices (wie das Next.js-Dashboard) in der Zukunft idealerweise als skalierbare Docker-Container orchestriert werden.

2.5 Moderne Webentwicklung und React

Die Präsentationsschicht der zu entwickelnden Lösung baut auf modernen Web-Technologien auf, deren Grundprinzipien dieser Abschnitt einführt. Die konventionelle Webentwicklung verzeichnete in der letzten Dekade einen deutlichen Paradigmenwechsel. In traditionellen Anwendungen wurden HTML-Ansichten für jeden Interaktions-Klick statisch auf dem Backend-Server generiert und an den Client-Browser ausgeliefert (klassischer Seitenumbruch). Der moderne Ansatz hingegen sind sogenannte **Single-Page-Applications (SPA)**. Hierbei wird ein einziges, schlankes HTML-Gerüst initial vom Server geladen; sämtliche anschließende Updates, Routings und Auslieferungen neuer UI-Bausteine erfolgen dynamisch durch das Ausführen von JavaScript-Code direkt im Browser des Endnutzers.

Dieses Praxisprojekt nutzt als programmiertechnische Basis die Open-Source-Bibliothek **React**, welche durch das Unternehmen Meta (Facebook) entwickelt wurde. React erzeugt komplexe Benutzeroberflächen über eine strikt deklarative, **komponentengetriebene Architektur**. Umfangreiche Bildschirmseiten werden dafür in kleine, hierarchische und autarke UI-Einheiten (Components) aufgeteilt, die auch in anderen Projekten wiederverwendbar sind. Ein Kernkonzept von React ist der sogenannte **Virtuelle DOM** (Document Object Model) inklusive des „Reconciliation"- bzw. „Diffing"-Algorithmus [6]. Ändern sich die zugrundeliegenden Daten in der Anwendung (beispielsweise eine im Millisekundentakt eingehende OEE-Ziffer), berechnet React die grafischen Auswirkungen zunächst abstrakt im Arbeitsspeicher auf dem eigenen virtuellen DOM. Der Algorithmus vergleicht (diffing) anschließend diesen neuen Zustand mit dem zuvor angezeigten Layout. Am Ende wendet React nur die minimal erforderlichen Änderungen auf den „echten", berechnungsintensiven Browser-DOM an. Dies sorgt auch bei hochfrequenten Datenaktualisierungen für eine flüssige, flackerfreie Darstellung.

2.6 Das Next.js Framework und hybride Rendering-Strategien

Aufbauend auf React ergänzt das gewählte Framework die für ein industrielles Dashboard nötigen serverseitigen Fähigkeiten. Obwohl React die Grundlage für das Rendern interaktiver Benutzeroberflächen bildet, fehlen dem Kernsystem eigenständige Lösungen für komplexe

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

serverseitige Infrastrukturen, ordnerbasiertes Routing oder tiefergehendes API-Management. An diesem Punkt übernimmt das durch Vercel publizierte Meta-Framework **Next.js**, das React um genau diese Funktionen ergänzt und sich als verbreiteter Standard für Web-Applikationen auf Unternehmensebene etabliert hat [7].

Next.js bettet reine React-Komponenten in das serverseitige Node.js-Ökosystem ein und ermöglicht dadurch hybride Rendering-Strategien. Diese haben das Hauptziel, Applikationen resistent gegen langsame Endgeräte zu machen:

- **Server-Side Rendering (SSR):** Im Gegensatz zum puren React (welches ausschließlich clientseitig aufgebaut wird), instruiert Next.js den Webserver im Rechenzentrum, die komplexe DOM-Struktur und die anfänglichen API-Abfragen bereits vor der ersten Auslieferung zusammenzustellen. Der Browser erhält dadurch bereits vollständig aufgebautes HTML, was die Ladezeit beim Erstaufruf (First Load) deutlich reduziert.
- **Static Site Generation (SSG) & Incremental Static Regeneration (ISR):** Next.js kann Seitenansichten alternativ komplett zum Build-Zeitpunkt generieren (SSG). Das zusätzliche Feature ISR erlaubt es zudem, in festgelegten Zeitintervallen im Hintergrund einzelne Produktionsdaten wie eine Temperatur-Anzeige direkt auf dem Server neu zu **fetchen** und nahtlos auszutauschen, ohne dass ein Nutzer seinen Browser aktualisieren müsste.
- **Integrierter Router (App Router):** Next.js setzt auf ein konsequent dateibasiertes Entwicklungs-Format. Anstatt Routen manuell in Konfigurations-Dateien einzutragen, definieren die Ordner-Namen im Code-Verzeichnis unmittelbar die URL-Adressen im finalen Browser (`/pages/dashboard/analytics`).
- **Fullstack-Autonomie:** Über sogenannte „Server Actions“ oder API-Routen verbindet Next.js das Frontend direkt mit dem eigenen Backend, wodurch separate Dritt-Server für Datenbankabfragen entfallen können.

Für die Visualisierung des MES-Data-Lakes fiel die Entscheidung der Dashboard-Technologie auf **Next.js**. Die Begründung hierfür liegt in der Performance-Priorität: Durch das Verlagern der OEE-Berechnungslogik sowie des Parsens der rohen JSON-Responses direkt in den Node.js-Backend-Layer des Frameworks bleiben die Anzeige-Tablets an den Produktionsmaschinen unbelastet. Die Thin-Clients laden lediglich die final aufbereiteten Visualisierungen, wodurch auch ältere Hardware in der Werkshalle die Web-Applikation flüssig betreiben kann.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

3 Aktueller Systemaufbau (Ist-Zustand)

Nachdem das vorangegangene Kapitel die theoretischen Grundlagen gelegt hat, überführt dieses Kapitel die abstrakten Konzepte in die konkret vorgefundene Infrastruktur des Projekts. Ziel ist eine vollständige und nachvollziehbare Bestandsaufnahme (Ist-Zustand) der FORCAM-Umgebung, wie sie zu Beginn der Praxisphase konfiguriert und analysiert wurde. Die Darstellung folgt bewusst dem physikalischen Datenfluss: von der untersten Hardware- und Virtualisierungsebene über die einzelnen virtuellen Maschinen und die darauf laufenden Software-Schichten bis hinauf zum Sicherheits- und Schnittstellenkonzept. Den Abschluss bilden die aktuell bestehenden Zugangshürden, welche die Motivation für die in den Folgekapiteln beschriebene Dashboard-Entwicklung und die alternative Architektur liefern.

3.1 Gesamtüberblick und Datenfluss

Bevor die einzelnen Komponenten im Detail beleuchtet werden, ordnet dieser Abschnitt sie in das Gesamtbild ein und spannt damit den roten Faden für das restliche Kapitel auf. Die gesamte FORCAM-Umgebung läuft in einem Rechenzentrum auf einem einzigen physischen Server, dessen Leistung über die Virtualisierungssoftware Proxmox in mehrere isolierte virtuelle Maschinen (VMs) aufgeteilt wird. Diese VMs bilden gemeinsam ein mehrschichtiges System, das den Weg der Maschinendaten von der Werkshalle bis zum Endgerät abbildet.

Der Datenfluss verläuft dabei entlang klar getrennter Schichten: Die Edge-Server (VM 102 für die Produktion und VM 105 für das Labor) erfassen die Rohsignale direkt an den Produktionsmaschinen und übersetzen die heterogenen Maschinenprotokolle in ein einheitliches JSON-Format. Der Anwendungsserver (VM 104) führt diese Datenströme im zentralen „Data Lake“ zusammen, berechnet daraus die Kennzahlen (OEE) und steuert über den Keycloak-Dienst zentral sämtliche Zugriffsrechte. Der Datenbankserver (VM 100) historisiert die Werte schließlich dauerhaft. Die folgenden Abschnitte beschreiben diese Ebenen in eben dieser Reihenfolge des Datenflusses – beginnend mit der Virtualisierungsebene, gefolgt von den einzelnen VMs, den logischen Software-Schichten, dem Sicherheitskonzept, den Schnittstellen und den derzeitigen Herausforderungen.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

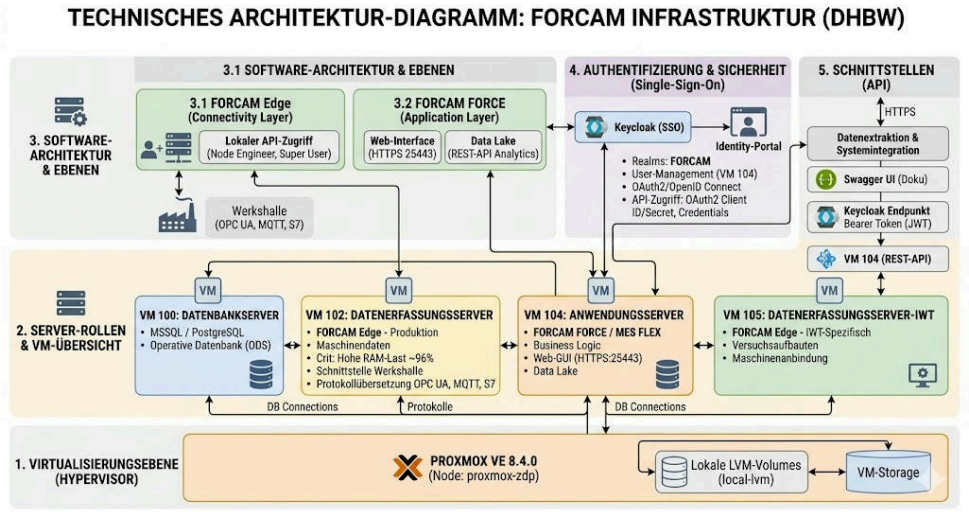


Abbildung 1 — Technisches Architektur-Diagramm der FORCAM-Infrastruktur (Gesamtüberblick)

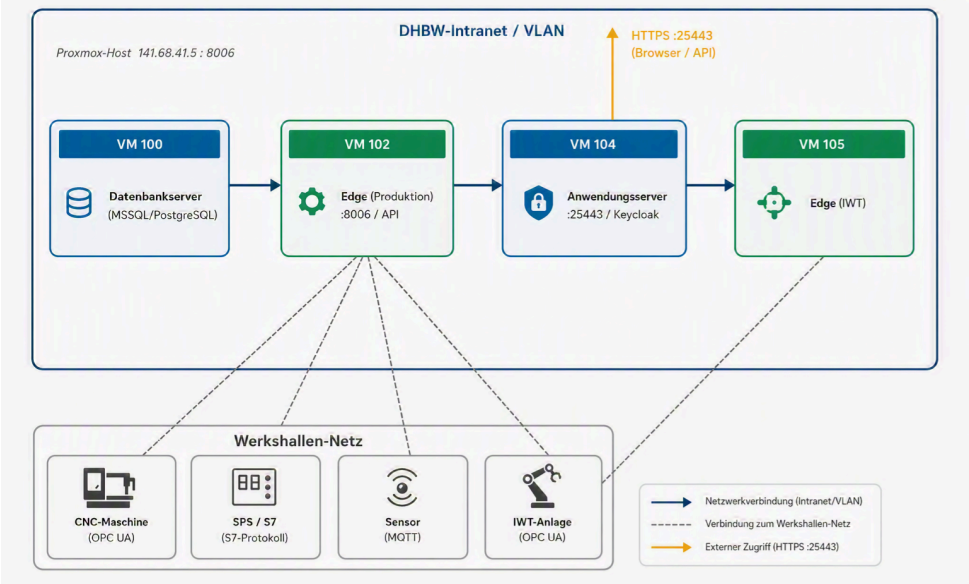


Abbildung 2 — Software-Architektur – Schichtenmodell der FORCAM-Umgebung

3.2 Virtualisierungsebene mit Proxmox

Die unterste Ebene des Systemaufbaus bildet die Virtualisierung, die aus einem einzelnen physischen Server die gesamte Server-Landschaft ableitet. Anstatt für jeden Software-Dienst (wie die Datenbank oder die Datenerfassung) einen eigenen, separaten Computer aufzustellen, kommt die Virtualisierungs-Software **Proxmox Virtual Environment** (Version 8.4.0) zum Einsatz.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Man kann sich Proxmox wie ein gut strukturiertes Bürogebäude vorstellen: Es nimmt die Gesamtleistung der physischen Server-Hardware (CPU, Arbeitsspeicher) und teilt sie softwareseitig so präzise auf, als gäbe es mehrere räumlich isolierte Büros (sogenannte Virtuelle Maschinen, oder VMs). Technisch basiert Proxmox hierbei auf der **KVM-Hypervisor-Architektur** (Kernel-based Virtual Machine), bei der der Linux-Kernel selbst als Virtualisierungsschicht fungiert [8]. Diese Virtualisierung stellt sicher, dass selbst bei einem Software-Fehler in einer VM (z.B. durch einen Systemabsturz) die Arbeit in der Nachbar-VM (z.B. der Datenbank) ungestört und sicher weiterläuft. Das sichert die Stabilität der gesamten IT-Landschaft.

Zur besseren Nachvollziehbarkeit fasst die folgende Tabelle die zentralen Infrastrukturparameter der vorgefundenen Umgebung zusammen:

Parameter	**Wert**
Hypervisor	Proxmox Virtual Environment 8.4.0
Node-Name	proxmox-zdp
Verwaltungs-URL	https://141.68.41.5:8006
Speichertypen	local-lvm (LVM Thin Pool) · VM-Storage (ISO/Templates)
Virtualisierungstechn.	KVM (Kernel-based Virtual Machine)
Netzwerkintegration	DHBW-Intranet / VLAN

Tabelle 1 — Infrastrukturparameter

Proxmox VE ermöglicht die zentralisierte Verwaltung aller virtuellen Maschinen über eine einzige Oberfläche. Funktionen wie Live-Migration, Snapshots und automatisierte Backups sind integraler Bestandteil der Plattform und sichern den Betrieb der FORCAM-Umgebung ab.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

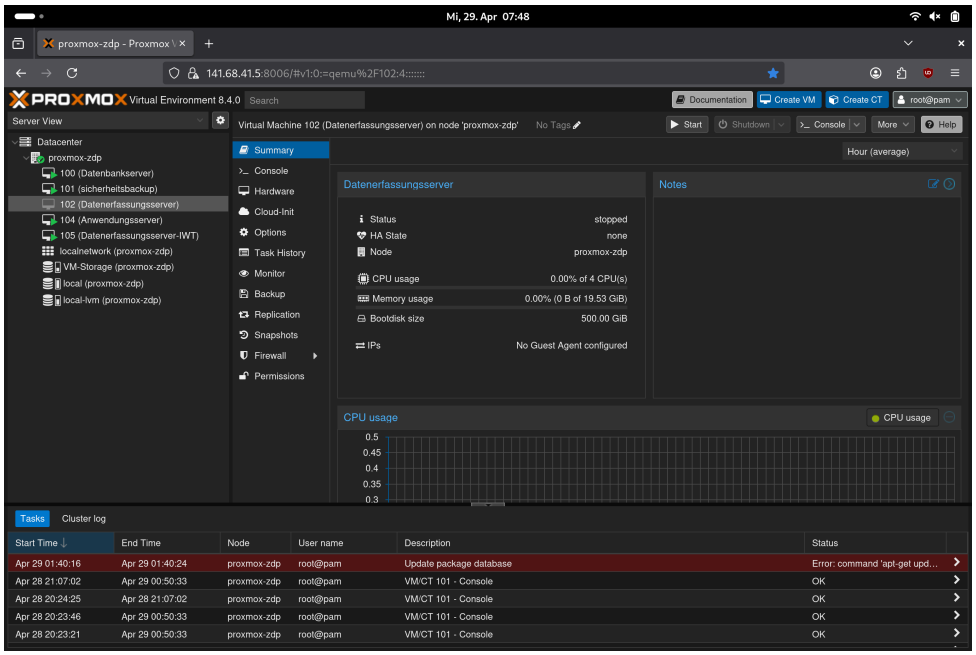


Abbildung 3 — Virtualisierungsebene – Proxmox-Host mit VM-Übersicht

3.3 Aufbau des Proxmox Web-Interfaces

Um die im vorigen Abschnitt beschriebene Virtualisierungsebene tatsächlich administrieren zu können, wird ein zentrales Verwaltungswerkzeug benötigt. Die Verwaltung der kompletten Virtualisierungsebene erfolgt über das webbasierte Graphical User Interface (GUI) von Proxmox VE (siehe Abbildung 3). Die Benutzeroberfläche ist logisch in drei Hauptbereiche strukturiert, welche eine effiziente Administration der Hardware-Umgebung ermöglichen:

1. **Server View (Linke Navigationsleiste):** Diese hierarchische Baumstruktur visualisiert das gesamte Rechenzentrum („Datacenter“) und den darunterliegenden physischen Host-Knoten (`proxmox-zdp`). Unterhalb dieses Knotens sind alle instanziierten virtuellen Maschinen mit ihren eindeutigen IDs (z.B. `102` für den Datenerfassungsserver) sowie die angebundenen Speicher-Pools (`local`, `local-lvm`, `VM-Storage`) inventarisiert. Auffällig in dieser Ansicht ist die dedizierte VM 101 (`sicherheitsbackup`), die als autarke Einheit für Sicherheitskopien im Systemkontext fungiert.
2. **Ressourcen- und Konfigurationsansicht (Zentraler Bereich):** Wählt man eine spezifische VM an (im Bildbeispiel VM 102), öffnet sich das zugehörige Detail-Dashboard. Der „Summary“-Tab liefert essenzielle Telemetriedaten auf einen Blick: den aktuellen Laufzeitstatus (hier im Zustand `stopped`), die provisionierten Hardware-Ressourcen (4 vCPUs, ca.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

20 GB virtueller Arbeitsspeicher, 500 GiB Bootdisk) sowie Echtzeit-Graphen zur Systemauslastung. Die tiefere vertikale Navigation erlaubt den direkten Zugriff auf die virtuelle Konsole (`NoVNC`), Hardware-Zuweisungen und das Snapshot-Management.

3. **Task- und Cluster-Log (Unterer Bereich):** Diese historische Konsole dient der Nachverfolgung aller asynchronen Systemereignisse und Administratoren-Aktionen (Audit-Log). Die grafische Ausgabe im Screenshot protokolliert kürzlich ausgeführte Konsolen-Verbindungen (Status `OK`) sowie einen fehlgeschlagenen Hintergrund-Prozess (`Update package database`), was auf einen Paket-Manager-Fehler (`apt-get`) des zugrundeliegenden Debian-Hostsystems hinweist.

3.4 Logische VM-Architektur und Rollenverteilung

Auf der beschriebenen Virtualisierungsebene setzen die einzelnen virtuellen Maschinen auf, die im Folgenden die eigentliche Arbeit übernehmen. Jede dieser VMs übernimmt eine exakt definierte, isolierte Aufgabe, um das FORCAM-System als Ganzes stabil zu betreiben.

Bildliche Analogie für Nicht-ITler: Man kann sich dieses verteilte System wie ein gut organisiertes Unternehmen vorstellen:

- **Die Reporter (VM 102 & 105):** Stehen direkt in der Werkshalle an den Maschinen, übersetzen deren fremde Sprachen in Notizen und funken diese ab.
- **Das Unternehmensarchiv (VM 100):** Ein großes, gesichertes Aktenlager. Hier werden die gefunkten Berichte strukturiert abgelegt und für die Zukunft verwahrt.
- **Die Management-Zentrale (VM 104):** Das Gehirn des Unternehmens. Es wertet die Akten aus dem Archiv aus, berechnet Statistiken, bedient die Bildschirme für die Mitarbeiter und kontrolliert am Gebäude-Eingang, wer überhaupt eintreten darf.

Die folgende Tabelle gibt eine formale Übersicht über diese Aufteilung, bevor die vier zentralen VMs in den nachfolgenden Abschnitten einzeln beschrieben werden:

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

VM ID	Name	Rolle im System	Besonderheiten
VM 100	Datenbankserver	Das Archiv (Datenbank)	Speichert alle erfassten Maschinen- und Betriebsdaten dauerhaft ab
VM 102	Datenerfassung (Produktion)	Der Reporter (FORCAM Edge)	Arbeitet stark am Limit: benötigt bald eine Aufstockung des Arbeitsspeichers
VM 104	Anwendungsserver	Die Zentrale (FORCAM FORCE)	Beherbergt das Web-Interface, die Logik und die externe Passwort-Kontrolle (SSO)
VM 105	Datenerfassung (IWT)	Der Labor-Reporter (Edge)	Zweite, isolierte Einheit nur für Versuchsaufbauten und das IWT-Labor

Tabelle 2 — Server-Rollen & VM-Übersicht

3.5 VM 100 – Datenbankserver und Datenmodellierung (Das Archiv)

Den Ausgangspunkt des dauerhaften Datenflusses bildet der Datenbankserver, in dem sämtliche erfassten Werte langfristig abgelegt werden. Dieser Server ist ausschließlich für die dauerhafte und sichere Speicherung (Persistenz) von Daten zuständig. Hier befindet sich das „Langzeitgedächtnis“ des Systems, oft auch ODS (Operational Data Store) genannt. Alle laufenden Maschinendaten und anfallenden Schicht-Ereignisse werden hier strukturiert aufbewahrt. Für Nutzer gibt es keine eigene Bildschirmoberfläche; andere Server greifen lediglich im Hintergrund auf diese Daten zu.

Herausforderung der Time-Series-Daten: Ein Produktionsumfeld generiert kontinuierliche, hochfrequente Datenströme (Time-Series Data). Das Speichern von Sensordaten im Millisekunden-Takt skaliert in klassischen relationalen Tabellen (wie MSSQL oder PostgreSQL) schlecht, da die Indexstrukturen bei jedem Schreibvorgang aktualisiert werden müssen, was mit wachsender Datenmenge zu Schreibverstärkung (Write Amplification) und Fragmentierung führt. Um dieses High-Throughput-Problem zu lösen, werden Datenbanken für MES-Systeme oft in eine „Hot“- und eine „Cold“-Storage-Ebene aufgeteilt. Aktuelle Echtzeit-Werte der Schicht verbleiben im schnellen Arbeitsspeicher (In-Memory), während ältere Historien-Daten in partitionierten Tabellen auf Festplatten abgelegt werden. Bei der direkten Abfrage

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

von OEE-Daten für das Dashboard muss das Datenbankmanagementsystem (DBMS) große Datenmengen über bestimmte Zeitfenster aggregieren, weshalb optimierte Indizes auf Zeitstempel-Spalten essenziell für kurze Latenzzeiten sind.

Die zentralen Merkmale des Datenbankservers lassen sich wie folgt zusammenfassen:

- **Datenbanktyp**: MSSQL und/oder PostgreSQL
- **Rolle**: Operative Datenbank (ODS – Operational Data Store)
- **Schnittstelle**: Direkte DB-Verbindung von VM 104 (kein direkter Nutzerzugriff)
- **Sicherung**: Backups über Proxmox-Snapshot-Mechanismus empfohlen

3.6 VM 102 – Datenerfassungsserver (Der Reporter)

Am entgegengesetzten Ende des Datenflusses – direkt an den Maschinen – steht der Datenerfassungsserver, der die Rohdaten überhaupt erst in das System einspeist. Dieser Server, auch „FORCAM Edge“ genannt, ist das direkte Bindeglied zur realen Fabrik. Maschinen unterschiedlicher Hersteller sprechen unterschiedliche digitale Sprachen. VM 102 agiert als Protokoll-Konverter: Sie baut persistente TCP/IP-Verbindungen zu diesen Maschinen auf (z.B. über hochfrequentes Polling oder Publish-Subscribe-Muster), liest Binärdaten aus Speicherbausteinen aus, übersetzt diese technischen Rohdaten in einheitliche JSON-Payloads und leitet die Echtzeit-Nachrichten an VM 104 weiter.

Da die fabrikweite Erfassung im Millisekunden-Takt läuft, muss die Maschine große Mengen an I/O-Operationen verarbeiten. Um Packet-Loss bei kurzzeitigen Netzwerkausfällen zu vermeiden, puffert die Edge-Software die Maschinensignale in-memory. Dies erklärt auch den hohen Ressourcenverbrauch: Der zugewiesene Arbeitsspeicher (RAM) ist aktuell fast vollständig ausgelastet (96 %). Ein Memory-Swap (Auslagerung auf die SSD) muss hier zwingend verhindert werden, da die Latenzen sonst deutlich ansteigen würden und das Echtzeit-Timing der OPC-UA-Signale beeinträchtigt wird. Auf diesen kritischen Ressourcen-Engpass wird in Kapitel 5 gesondert eingegangen.

- **Protokolle**: OPC UA (Client-Modus), MQTT (Broker-Verbindungen), Siemens S7 (RFC 1006 / ISO-on-TCP)
- **Betrieb**: Kontinuierlich (24/7) zur lückenlosen OEE-Erfassung
- **Ressourcenlage**: RAM-Auslastung der Maschine kritisch (96 % von 20 GB), Gefahr des Linux-Kernel-OOM-Killers (Out of Memory)

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

- **Administration**: Lokaler Web-Zugriff mit Node Engineer / Super User
- **Lokale API**: Interface für die Verwaltung von Polling-Zyklen und Subscriptions

3.7 VM 104 – Anwendungsserver (Die Zentrale)

Zwischen dem erfassenden Edge-Server und dem archivierenden Datenbankserver bildet der Anwendungsserver das logische Zentrum, in dem die Daten tatsächlich verarbeitet werden. Hier fließen alle zuvor übersetzten Protokolle aus der Werkshalle in einem virtuellen Datenbecken zusammen, dem sogenannten „Data Lake“. Auf diesem Server läuft die eigentliche Logik: Er berechnet, wie effizient die Anlagen laufen (OEE-Wertermittlung), und stellt die Webseiten zur Verfügung, aus denen auch das neue Next.js-Dashboard später seine Daten bezieht. Zudem fungiert VM 104 als digitaler „Türsteher“ für das Gesamtsystem (Keycloak SSO), überprüft also Logins, Passwörter und das Berechtigungskonzept jedes einzelnen Benutzers.

- **Web-Interface**: HTTPS-Zugang auf Port 15443
- **Identity Provider**: Keycloak (Endpunkt auf Port 25443, Pfad: `/ffauth/realms/FORCAM/`)
- **Data Lake**: Zusammenführung aller Maschinendaten (Schnittstelle für Next.js)
- **REST-API**: Vollständige Swagger-UI-Dokumentation verfügbar
- **MES-Funktionen**: Auftragsmanagement, OEE-Berechnung, Maschinendatenerfassung

3.8 VM 105 – Datenerfassung Labor (Der Hochschul-Reporter)

Neben der produktiven Datenerfassung existiert eine zweite, bewusst abgetrennte Edge-Instanz für Forschungs- und Testzwecke. Da dieses Projekt im Kontext der Hochschule stattfindet, ist diese Instanz strikt von der produktiven VM 102 getrennt. Das IWT (Institut für Werkzeug- und Fertigungstechnik) kann diesen Server zum Forschen nutzen, um neue Maschinen anzubinden oder Fehler zu provozieren, ohne die Aufzeichnung der realen Produktionsmaschinen (welche über VM 102 läuft) zu beeinträchtigen.

- **Basis**: Gleiche technische Grundlage wie VM 102 (FORCAM Edge)
- **Betrieb**: Isolierter Laborbetrieb innerhalb des Hochschulnetzwerks
- **Zweck**: Ermöglicht separate Protokollführung für den IWT-Bereich

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

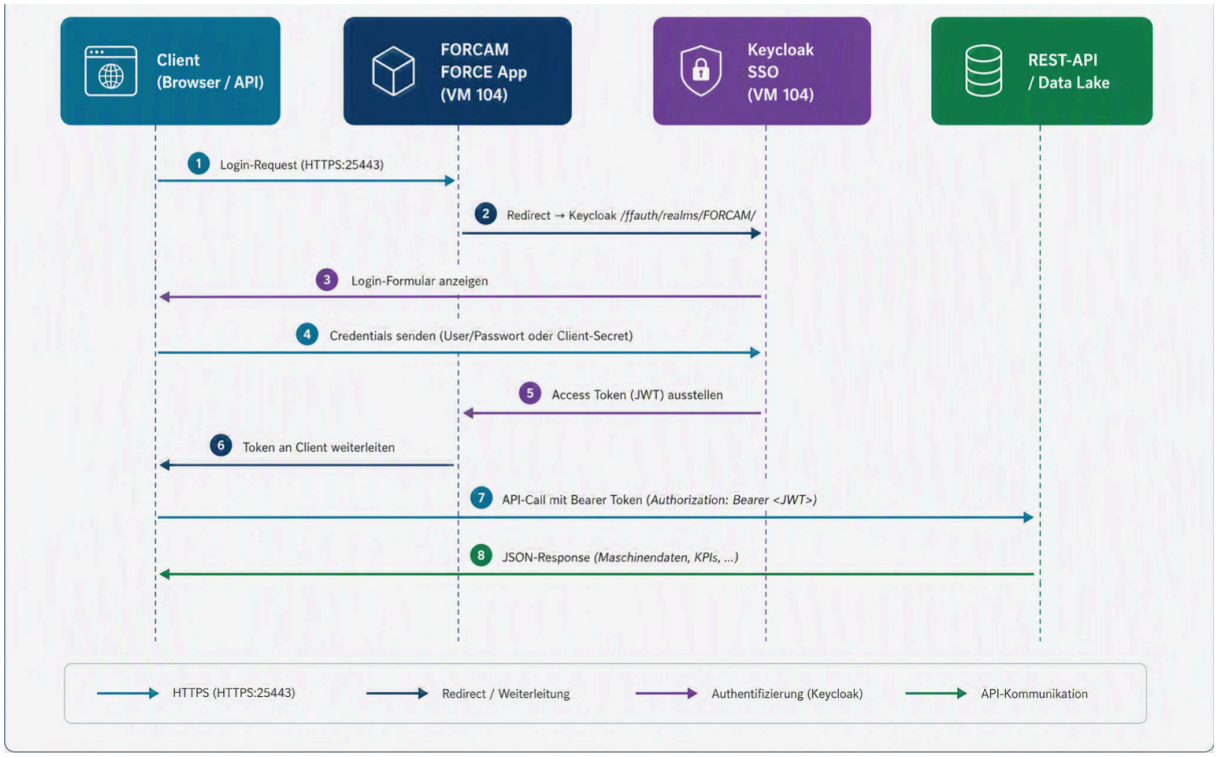


Abbildung 4 — Netzwerk-Topologie – Maschinen ⟶ Edge ⟶ Applikation

3.9 FORCAM Edge – Konnektivitätsschicht (Der Universal-Übersetzer)

Nachdem die einzelnen VMs als physische Bausteine beschrieben wurden, wechselt die Betrachtung nun auf die logische Ebene der Software-Schichten, die auf diesen VMs laufen. Den Einstieg bildet die Konnektivitätsschicht. FORCAM Edge läuft auf den virtuellen Instanzen 102 und 105 und bildet die informationstechnische Brücke der Gesamtarchitektur. Es ist die direkte Schnittstelle zur realen Werkhalle.

Man kann sich das Edge-Gateway bildlich als digitalen Simultanübersetzer vorstellen. Industrielle Maschinen in einer Fabrik sprechen oft unterschiedliche technische Dialekte (sogenannte Industrieprotokolle wie OPC UA oder Siemens S7). Das übergeordnete IT-System (FORCAM FORCE) kann diese Maschinensprachen von Natur aus nicht verstehen. Das FORCAM Edge Gateway schaltet sich exakt dazwischen: Es hört direkt an den Produktionsmaschinen im Millisekundentakt zu, übersetzt alle eingehenden Maschinensignale in ein einheitliches, gut lesbares IT-Format (JSON) und reicht diese bereinigten Datenpakete dann strukturiert an das zentrale Rechenzentrum weiter.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Anbindung an den Data Lake: Die vom Edge normalisierten JSON-Datenpakete verbleiben nicht auf dem Edge-Server, sondern werden kontinuierlich an den zentralen Datenspeicher der Anwendungsschicht – den sogenannten **Data Lake** auf VM 104 – weitergereicht. Ein Data Lake ist ein zentrales, hochperformantes Speicherbecken, in dem die eingehenden Maschinenwerte auf Zeitreihen-Basis unverändert und vollständig historisiert werden. Anders als eine klassische, streng strukturierte Datenbank nimmt der Data Lake die Rohdatenströme zunächst weitgehend in ihrer ursprünglichen Form auf (Prinzip „Schema-on-Read“) und hält sie für spätere Auswertungen vor. Das FORCAM Edge fungiert damit als vorgelagerte Zulieferer- bzw. Ingestion-Schicht: Es sorgt dafür, dass der Data Lake lückenlos und im Millisekundentakt mit sauber übersetzten Signalen gespeist wird, aus denen die Anwendungsschicht anschließend die OEE-Kennzahlen berechnet und über die REST-API bereitstellt. Fällt kurzzeitig die Verbindung zum Rechenzentrum aus, puffert das Edge die Signale lokal zwischen, um die spätere lückenlose Einspeisung in den Data Lake sicherzustellen. Die technische Ausgestaltung dieses Data Lakes wird im folgenden Abschnitt zur Anwendungsschicht näher beschrieben.

Spezialisierte Aufgaben als Gateway:

- **Protokollübersetzung**: Wandelt komplexe, herstellerspezifische Maschinensignale (OPC UA, MQTT) in ein standardisiertes Format um.
- **Lokal integrierter Puffer**: Bricht das Hauptnetzwerk weg, agiert das Edge als lokaler „Notizblock“ und puffert alle Maschinenereignisse offline zwischen.
- **Echtzeiterfassung**: Sichert die lückenlose Messung von Sensorwerten und Stückzahlen ohne signifikanten zeitlichen Verzug ab.

Der administrative Zugriff auf die Edge-Instanzen ist über ein rollenbasiertes Berechtigungskonzept geregelt, das die folgende Tabelle zusammenfasst:

Rolle	**Berechtigungen**
Node Engineer	Konfiguration einzelner Maschinenanschlüsse, Protokollmapping, Debugging
Super User	Vollzugriff auf alle Edge-Konfigurationen, Systemeinstellungen, Updates

Tabelle 3 — Administration & Zugriffsrollen

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

3.10 FORCAM FORCE – Anwendungsschicht

Oberhalb der Konnektivitätsschicht setzt die eigentliche Anwendungsschicht auf, in der die zugelieferten Daten zu verwertbaren Informationen veredelt werden. FORCAM FORCE / MES FLEX (VM 104) ist das zentrale Anwendungssystem und bildet den Kern der MES-Umgebung. „MES FLEX“ bezeichnet dabei die modulare Produktlinie von FORCAM FORCE. Alle Rohdaten aus den Edge-Instanzen werden hier im Data Lake aggregiert, verarbeitet und für Analysen und Berichte bereitgestellt.

- **Web-Interface**: Erreichbar über HTTPS auf Port 15443. Bietet eine vollständige browserbasierte Oberfläche für alle MES-Funktionen: Auftragsmanagement, OEE-Analyse, Betriebsdatenerfassung, Alarmmanagement und Reporting.
- **Identity Provider (Keycloak)**: Zentrales Benutzermanagement via Keycloak (Port 25443). Pfad: `/ffauth/realms/FORCAM/`. Unterstützt OAuth2 und OpenID Connect für SSO. Hier werden alle Benutzer, Rollen und technischen API-Clients verwaltet.
- **Data Lake**: Zentrales Datenaggregations-Repository. Alle Maschinendaten aus den Edge-Instanzen fließen hier zusammen und stehen über die REST-API für externe Analysen bereit.
- **Business Logic**: Verarbeitung von Auftrags- und Maschinendaten, Berechnung von KPIs (OEE, Verfügbarkeit, Leistung, Qualität), Anbindung an ERP-Systeme.

3.11 Authentifizierung & Sicherheit mit Keycloak

Da die Anwendungsschicht sensible Produktionsdaten verwaltet, ist ihr ein durchgängiges Sicherheitskonzept vorgelagert, das jeden Zugriff kontrolliert. Wie in den theoretischen Grundlagen (vgl. Kapitel 2) hergeleitet, nutzt das FORCAM-System für das Identity Management ein Single-Sign-On-(SSO)-Verfahren auf Basis von OAuth 2.0 und OpenID Connect. Dieser Abschnitt beleuchtet die konkrete, netzwerkspezifische Konfiguration des Keycloak-Servers in der DHBW-Infrastruktur.

Für fachfremde Betrachter lässt sich das zugrundeliegende Identity Brokering mit der Sicherheits-Passkontrolle an einem großen Firmen-Campus erklären. Anstatt dass jede einzelne Fachabteilung im Gebäude (also jede Software, jedes Dashboard, jede API) ihre eigene Liste mit Passwörtern führt und jeden Nutzer an der Tür separat abfragt, gibt es genau einen zentralen Rezeptionsschalter (die „Keycloak“-VM). Wenn sich ein Mitarbeiter oder ein Computerprogramm anmeldet, kontrolliert diese Rezeption die Identität. Ist alles korrekt, stellt Keycloak ein

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

digitales, zeitlich begrenztes Zugangsticket aus (ein sogenanntes JSON Web Token, JWT). Mit diesem digitalen Besucherausweis kann man sich frei zwischen den zugelassenen Software-Anwendungen bewegen („Single-Sign-On“), da jede Pforte im System nur noch dieses digitale Ticket scannt, anstatt erneut nach Name oder Passwort zu fragen.

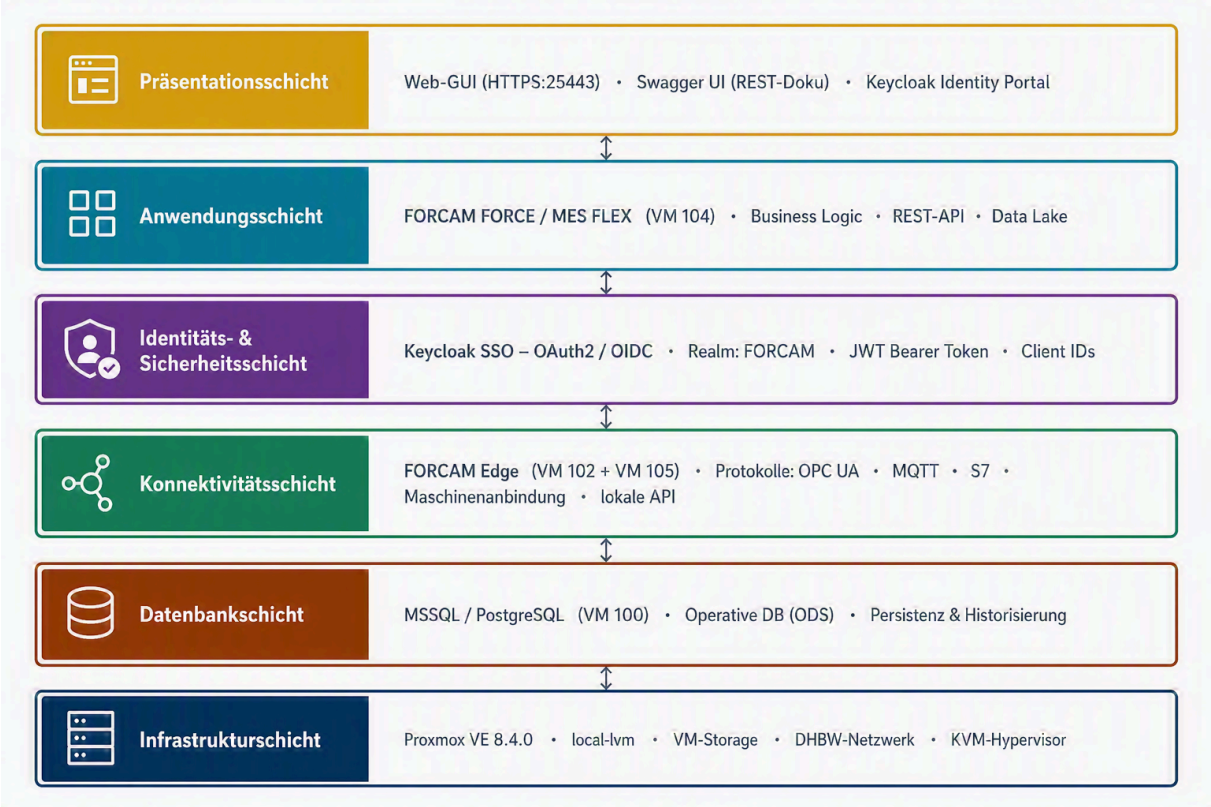


Abbildung 5 — Authentifizierungsfluss – OAuth2 / OpenID Connect (SSO)

Keycloak-Konfiguration

Die konkrete Ausprägung des Keycloak-Servers in der DHBW-Infrastruktur ist in mehreren Parametern hinterlegt, welche die Endpunkte, den zuständigen Sicherheitsbereich (Realm) und das verwendete Token-Format festlegen. Die folgende Tabelle fasst diese zentralen Konfigurationswerte zusammen:

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Parameter	Wert
Identity Provider	Keycloak – Open Source IAM (Identity & Access Management)
Protokolle	OAuth2, OpenID Connect (OIDC); SAML 2.0 verfügbar, im Projekt nicht aktiv genutzt
Realm	FORCAM (Haupt-Sicherheitsbereich)
Endpunkt	https://<anwendungsserver>:25443/ffauth/realms/FORCAM/
Token-Typ	JWT (JSON Web Token) – Bearer Token
Admin-Berechtigungen	Realm-Admin-Rechte für Anlage technischer API-User

Tabelle 4 — Keycloak-Konfiguration

Authentifizierungsmethoden

Auf Basis dieser Konfiguration stellt Keycloak zwei unterschiedliche Verfahren bereit, um sich am System anzumelden – je nachdem, ob ein menschlicher Nutzer oder ein automatisierter Dienst zugreift:

- **Benutzer-Credentials (Username/Password):** Klassische Authentifizierung über Benutzername und Passwort. Geeignet für interaktive Benutzer über das Web-Interface. Keycloak stellt nach erfolgreicher Authentifizierung einen JWT aus.
- **OAuth2 Client Credentials (bevorzugt für API-Zugriffe):** Technische Benutzer (Service Accounts) authentifizieren sich über Client ID und Client Secret. Dies ist die empfohlene Methode für automatisierte API-Abfragen und System-zu-System-Kommunikation. Sie bietet eine feingranulare Berechtigungssteuerung ohne persönliche Credentials.

Sicherheitshinweise

Für den sicheren Betrieb dieses zentralen Zugangssystems sind abschließend einige organisatorische und technische Leitlinien zu beachten:

- Client Secrets und API-Token niemals in Klartext in Konfigurations-Repos speichern (Secret-Management-System verwenden).
- Realm-Admin-Rechte in Keycloak auf eine minimale Anzahl vertrauenswürdiger Accounts beschränken.
- Token-Laufzeiten (Access Token Lifespan) gemäß Sicherheitsrichtlinien konfigurieren.
- HTTPS obligatorisch für alle externen Zugriffe (Ports 15443 und 25443); reinen HTTP-Zugriff deaktivieren.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

- Regelmäßige Überprüfung der aktiven Keycloak-Sessions und Service-Account-Berechtigungen.
- Netzwerk-Firewall: Ports 15443 und 25443 (FORCAM) sowie 8006 (Proxmox) nur aus dem DHBW-Intranet erreichbar.

3.12 Schnittstellen: REST-API und Token-Beschaffung

Damit externe Anwendungen die im Data Lake gesammelten und durch Keycloak geschützten Daten überhaupt nutzen können, stellt FORCAM FORCE eine vollständige REST-API bereit. Sie ermöglicht den Zugriff auf alle Maschinendaten, KPIs, Auftragsdaten und Konfigurationen und bildet damit die technische Grundlage für die spätere Dashboard-Anbindung.

Swagger UI – das interaktive System-Handbuch

Softwareentwickler, die ihr eigenes Programm (wie beispielsweise das Next.js-Dashboard) mit einer fremden Schnittstelle (API) verknüpfen wollen, brauchen eine präzise Bedienungsanleitung. Ohne zu wissen, an welche digitalen Adressen man die Daten schicken muss und wie die Antworten aufgebaut sind, ist eine Anbindung nicht möglich. Die Swagger-UI löst dieses Problem: Es handelt sich dabei um das einheitliche, interaktive Handbuch der API. Auf dieser Webseite (erreichbar auf VM 104) werden alle verfügbaren API-Abfragen (Endpunkte) übersichtlich aufgelistet. Klickt man auf einen Bereich, zeigt die Seite nicht nur, welche Parameter verlangt werden, sondern sie ist interaktiv: Entwickler können Aufrufe direkt dort ausführen und die tatsächliche Server-Antwort einsehen, ohne dafür bereits eine Zeile Anwendungscode schreiben zu müssen. Die folgende Tabelle fasst die wichtigsten Eckdaten der Swagger-UI zusammen:

Parameter	**Wert**
URL-Schema	https://<anwendungsserver>:25443/swagger-ui
Format	OpenAPI 3.0 / JSON
Interaktiv	Direktes Testen aller Endpunkte über das Browser-Interface
Authentifizierung	Bearer Token direkt im Swagger-Interface konfigurierbar

Tabelle 5 — Swagger UI

REST-API – Technische Spezifikation

Über die Dokumentation hinaus lässt sich die eigentliche Schnittstelle durch einige techni-

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

sche Kerneigenschaften charakterisieren, die für jede Anbindung maßgeblich sind. Die API folgt dem REST-Architekturstil und übermittelt Daten im JSON-Format über verschlüsselte HTTPS-Verbindungen:

Parameter	Wert
Protokoll	REST (Representational State Transfer)
Datenformat	JSON (application/json)
Transport	HTTPS (TLS-verschlüsselt, Port 25443)
Authentifizierung	Bearer Token (JWT), bezogen über Keycloak-Endpunkt

Tabelle 6 — REST-API Technische Spezifikation

Token-Beschaffung (OAuth2 Client Credentials Flow)

Bevor ein Datenabruf erfolgen kann, muss die anfragende Anwendung ein gültiges Zugangs-Token beschaffen. Für automatisierte API-Zugriffe (Machine-to-Machine Communication) wird hierfür der OAuth2 Client Credentials Flow verwendet. Hierbei wird ein technischer Service-Account genutzt, um ein asymmetrisch signiertes JSON Web Token (JWT) zu generieren:

```
POST /ffauth/realms/FORCAM/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=<id>&client_secret=<secret>
```

Das Backend antwortet bei erfolgreicher Verifizierung mit einem temporären Access Token (meist 5 Minuten gültig):

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIg...",
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "token_type": "Bearer",
  "not-before-policy": 0
}
```

Jeder folgende API-Aufruf zur Datenextraktion (z.B. Abruf der Live-OEE-Werte) muss dieses Token zwingend im `Authorization`-Header als Bearer-Token mitsenden:

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD


```
GET /api/v1/workplace/status
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIg...
Accept: application/json
```

3.13 Aktuelle Herausforderungen und Workarounds

Der zuvor beschriebene, ideale Zugriffsweg lässt sich in der Praxis derzeit nicht beschreiten – dieser Abschnitt dokumentiert die konkreten Hürden und die daraus abgeleiteten Lösungsansätze und schließt damit die Bestandsaufnahme des Ist-Zustands ab. Wie bereits bei der theoretischen Betrachtung der Keycloak-Architektur angemerkt, steht das Projekt in der praktischen Umsetzung aktuell vor einer massiven Zugangs-Hürde. Für die Authentifizierung an der Swagger-UI sowie für die Generierung der notwendigen Bearer-Tokens fehlen derzeit gültige administrative Zugangsdaten (Username und Passwort). Dies blockiert den nativen Weg, neue Clients zu registrieren und autorisierte Abfragen über die REST-Schnittstelle an den FORCAM Data Lake zu stellen.

Verschärft wird diese Problematik durch die Tatsache, dass die hier vorliegende Version des MES-Systems durch den Hersteller FORCAM offiziell nicht mehr unterstützt wird (End-of-Support). Ohne aktiven Herstellersupport ist es stark erschwert, die fehlenden Zugangsdaten auf offiziellem Wege zurücksetzen zu lassen oder technische Unterstützung für die API-Anbindung anzufordern.

Um den Fortgang des Dashboards dennoch zu gewährleisten, wird aktuell an technischen Workarounds gearbeitet. Eine primäre Strategie ist die Implementierung einer eigenen, autarken API-Middleware. Anstatt die proprietäre FORCAM-REST-Schnittstelle zu nutzen, könnte diese Eigenentwicklung versuchen, die Maschinendaten durch direkten Zugriff auf die unterliegende PostgreSQL/MSSQL-Datenbank (VM 100) zu extrahieren. Alternativ wird geprüft, ob sich die rohen Maschinensignale (z.B. OPC UA) parallel zu FORCAM abgreifen und in eine eigene Datenstruktur umleiten lassen, um das Dashboard unabhängig von der restriktiven Legacy-API mit Live-Daten zu versorgen.

Konzept der Middleware-Architektur

Sollte die Entscheidung auf eine eigene API-Middleware fallen, muss diese mit hoher Vorsicht in die Bestandsarchitektur integriert werden. Ein direkter Lesezugriff auf das operative

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Produktions-Archiv (VM 100) birgt das Risiko, durch unperformante Datenbankabfragen das gesamte MES-System der Fabrik auszubremsen.

Technologisch würde diese Middleware voraussichtlich mit einem ressourcenschonenden Framework wie **Node.js (Express/NestJS)** oder **Python (FastAPI)** realisiert werden. Die Datenbankanbindung erfolgt zwingend über einen isolierten **Read-Only-Datenbank-User** und implementiert striktes **Connection Pooling**, um die Anzahl der gleichzeitigen SQL-Verbindungen zu limitieren. Die Middleware fungiert hierbei als intelligenter Puffer: Sie liest durch Reverse-Engineering des FORCAM-Datenbankschemas die rohen Tabellen aus, bereinigt und denormalisiert die komplexen Relationen und stellt dem Next.js-Dashboard ausschließlich passgenaue, performante JSON-Endpunkte zur Verfügung.

Domänen-Mapping und Netzwerktopologie (DHBW Ravensburg)
Für die Kommunikation mit dem FORCAM Data Lake und den zugehörigen Identity-Diensten wurden durch die Netzwerkinfrastruktur der Hochschule spezifische Domänen und Ports (Fully Qualified Domain Names – FQDN) freigegeben. Diese strikte Trennung dient der Isolierung von Web-Interfaces und maschinellen API-Zugriffen. Die folgende Tabelle stellt das aktuell vorgesehene Routing dar:

Hostname / FQDN	Port	Verwendungszweck
`anwendungsserver.mes.dhbw-ravensburg.de`	`15443`	Zugang zur grafischen FORCAM-Anmeldemaske und dem nativen Web-Frontend (HTTPS)
`anwendungsserver.mes.dhbw-ravensburg.de`	`25443`	Keycloak-Authentifizierungs-Endpunkt (OAuth2) und REST-API-Zugang für das Dashboard
`mes.dhbw-ravensburg.de` (bzw. `auth...`)	`443`	Vermutete interne Reverse-Proxys oder Load-Balancer der Hochschul-IT

Tabelle 7 — Netzwerk-Routing und Port-Mapping

Kryptografische Hürden: TLS-Zertifikatskonflikte (SAN-Mismatch)
Basierend auf der oben definierten Netzwerkstruktur zeigte sich im Verlauf der API-Integration ein Problem im Bereich der Public-Key-Infrastruktur (PKI). Zwar konnte die initiale Anmeldemaske des FORCAM-Systems über Port `15443` erfolgreich im Browser adressiert

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

werden, jedoch scheiterte der Versuch, ein automatisiertes OAuth-Token am Keycloak-Endpunkt über Port `25443` zu beziehen, mit einem Verbindungsabbruch.

Die nachfolgende Fehleranalyse zeigte, dass der Fehler nicht im OAuth-Protokoll selbst lag, sondern beim vorgeschalteten TLS-Handshake (Transport Layer Security). Das Frontend des Dashboards baut eine TCP-Verbindung auf und überprüft im Anschluss das vom FORCAM-Server ausgelieferte SSL/TLS-Zertifikat. Gemäß den Standards moderner Webbrowser (und Node.js-Runtimes) muss der aufgerufene DNS-Name (`anwendungsserver.mes.dhbw-ravensburg.de`) zwingend als **Subject Alternative Name (SAN)** im Zertifikat des Servers hinterlegt sein.

Die Fehlerausgabe `No subject alternative DNS name matching anwendungsserver.mes.dhbw-ravensburg.de found` bestätigte einen klassischen Zertifikatskonflikt: Das vom Server präsentierte Zertifikat war auf eine abweichende Domäne (beispielsweise auf einen übergeordneten Reverse-Proxy wie `mes.dhbw-ravensburg.de` oder ein internes Test-Zertifikat) ausgestellt. Da der anfragende Client-Hostname somit nicht mit den autorisierten Hostnamen im Zertifikat übereinstimmte, verweigerte das System die Etablierung des sicheren HTTPS-Tunnels, wodurch der Token-Abruf (`[invalid_token_response]`) blockiert wurde.

Eine Behebung dieses Architekturfehlers erfordert die Kooperation mit der IT-Administration der Hochschule. Zu evaluieren sind primär der Austausch des Server-Zertifikats (Neu-Ausstellung mit gepatchtem SAN-Eintrag) oder eine Anpassung des OAuth-Clients auf die exakte, vom aktuellen Zertifikat unterstützte Subdomain. Die in der Softwareentwicklung teils praktizierte Notlösung, die TLS-Validierung im Client temporär zu deaktivieren (mittels Flag-Parametern wie `trust-all-certificates` / `disableSslValidation`), wird für dieses Projekt aufgrund der erheblichen Gefahren durch Man-in-the-Middle-Angriffe (MitM) im industriellen Kontext kategorisch abgelehnt.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

4 Systemintegration: Das Next.js Dashboard

Aufbauend auf dem im vorigen Kapitel beschriebenen Ist-Zustand widmet sich dieses Kapitel der eigentlichen Eigenentwicklung des Projekts. Um die aggregierten Daten aus dem initialisierten FORCAM Data Lake für Endanwender nutzbar zu machen, erweitert die Dashboard-Applikation die zuvor beschriebene Backend-Infrastruktur um eine moderne Präsentationsschicht.

4.1 Architektur und Technologie

Zunächst wird die technische Grundstruktur der Anwendung erläutert, da sie die Rahmenbedingungen für alle weiteren Design-Entscheidungen setzt. Die Visualisierungsoberfläche wird als vollwertige Fullstack-Applikation auf Basis von **Next.js 14** (unter Verwendung des modernen App Routers) programmiert. Diese Architektur teilt sich in zwei essenzielle Rendering-Schichten auf:

1. **React Server Components (RSC):** Die initialen API-Requests an das FORCAM-Backend – inklusive der kryptografischen Authentifizierung via OAuth2 – werden aus Sicherheits- und Performance-Gründen strikt serverseitig in der Node.js-Umgebung von Next.js ausgeführt. Da die `client_secret`-Schlüssel auf dem Server verbleiben, ist die Architektur inhärent vor XSS-Angriffen (Cross-Site Scripting) auf das Token geschützt.
2. **Client Components & State Management:** Die eigentliche interaktive Visualisierung der Maschinen (z.B. dynamische Doughnut-Charts für den OEE-Faktor) wird im Browser des Nutzers gerendert. Für das State-Management und das Live-Polling der Daten wird eine schlanke Lösung wie `Zustand` oder `SWR` (Stale-While-Revalidate) eingesetzt.

Um die Netzwerklast auf den VMs (insbesondere auf der stark ausgelasteten VM 104) gering zu halten, wird das in Next.js integrierte **Data Fetching & Caching** genutzt. Anstatt bei 50 verbundenen Endgeräten 50 individuelle API-Abfragen an FORCAM zu senden, fragt der Next.js-Server die Daten nur einmalig an und verteilt den Cache an alle Clients weiter (Incremental Static Regeneration).

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

4.2 Shopfloor UI/UX und Echtzeit-Streaming

Neben der technischen Architektur stellt das industrielle Einsatzumfeld besondere Anforderungen an Bedienung und Datenaktualität, die sich deutlich vom klassischen Büro-Kontext unterscheiden. Ein interaktives Dashboard für das industrielle Produktionsumfeld (Shopfloor) unterliegt daher anderen Usability-Anforderungen als herkömmliche Web-Applikationen im Office-Bereich:

- **Dark Mode und Kontrast:** Industrie-Displays oder Tablets sind oft in rauen Umgebungen mit greller Hallenbeleuchtung montiert. Das Dashboard implementiert daher primär hochkontrastige, dunkle Farbpaletten (Dark Mode). Dies reduziert zum einen die Blendwirkung und stellt zum anderen sicher, dass Warnsignale (z.B. rote Maschinenstillstände) auch aus mehreren Metern Entfernung sofort erfasst werden (Glanceability).
- **Touch-Optimierung:** Da die Endgeräte an den Maschinen meist per Touchscreen und teils mit Arbeitshandschuhen bedient werden, müssen interaktive Elemente (Buttons, Tabs) eine deutlich vergrößerte Hitbox (Touch-Target-Size) aufweisen.
- **Live-Daten-Streaming:** Da die OEE-Leistungskennzahlen in Echtzeit auf dem Dashboard aktualisiert werden müssen, erzeugt traditionelles HTTP-Polling (z.B. Frontend-Abfrage alle 2 Sekunden) unnötigen Netzwerk-Overhead. Technologisch wird evaluiert, auf **Server-Sent Events (SSE)** oder **WebSockets** zu migrieren. Bei diesen Push-Technologien wird ein einzelner, persistenter Kanal vom Next.js-Server zum Browser-Client offengehalten, über den das Backend neue OEE-Berechnungen sofort und ressourcenschonend direkt auf die Tablets in der Werkhalle überträgt.

4.3 Aktueller Status: Prototyp und nächste Implementierungsschritte

Abschließend wird der zum jetzigen Zeitpunkt der Praxisphase erreichte Entwicklungsstand eingeordnet und den noch offenen Arbeitspaketen gegenübergestellt. Bislang wurde ein visuell funktionaler UI-Prototyp fertiggestellt, der das grundsätzliche Layout mithilfe responsiver CSS-Grids abbildet und Ansichten für die OEE-Kennzahlen simuliert.

- **Bereits umgesetzt:** Setup der Frontend-Architektur (App Router), hierarchisches Layout-System, globale CSS-Variablen für das Theming und das Design erster interaktiver Metrik- und Chart-Komponenten (basierend auf Bibliotheken wie Recharts).
- **Fehlend (nächste Schritte):** Der Prototyp arbeitet aktuell isoliert auf Mock-Daten. Im nächsten Meilenstein wird die serverseitige Anbindungslogik (die `fetch`-Routen)

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

implementiert. Hierbei muss ein automatisierter Token-Refresh-Mechanismus entwickelt werden, um sicherzustellen, dass das Dashboard nach Ablauf der 5-minütigen JWT-Lebensdauer nicht die Verbindung zum MES-Backend verliert. Anschließend werden die statischen Platzhalter durch authentifizierte Live-Abfragen aus dem Data Lake abgelöst.

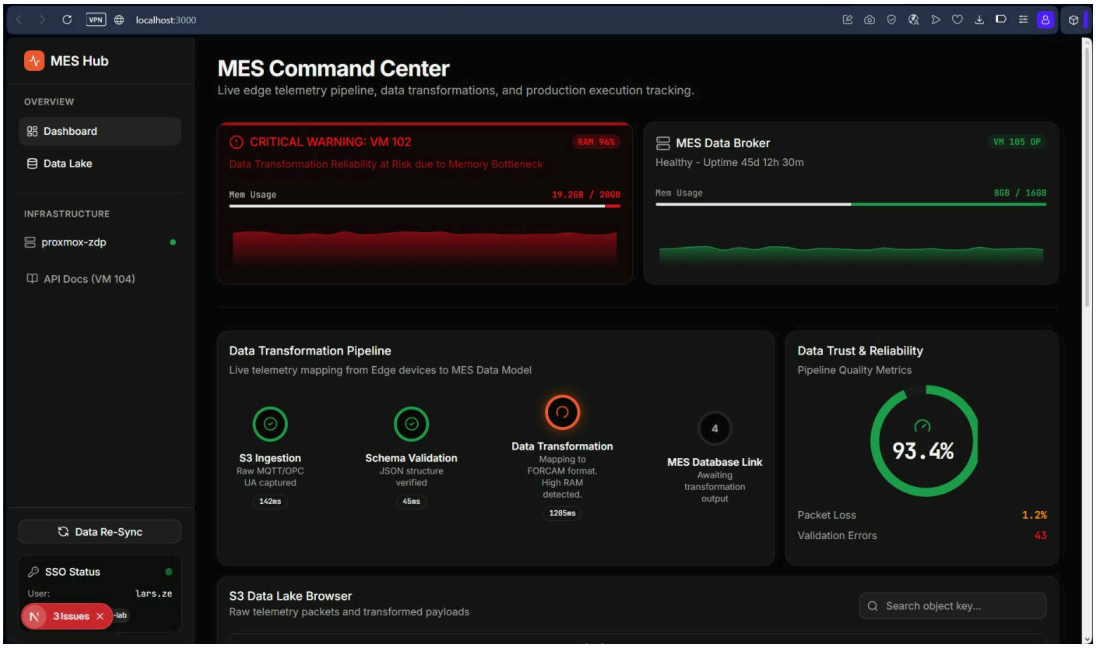


Abbildung 6 — Aktueller Entwicklungsstand des Next.js-Dashboard-Prototyps

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

5 Monitoring & Wartungshinweise

Ein produktiv betriebenes MES-System erfordert über die reine Entwicklung hinaus einen planvollen laufenden Betrieb. Die regelmäßige Überwachung der Systemressourcen und die planmäßige Wartung sind entscheidend für den stabilen Betrieb der FORCAM-Umgebung. Dieses Kapitel bündelt daher die kritischen Beobachtungen aus der Bestandsaufnahme sowie konkrete Handlungsempfehlungen. Folgende Aspekte bedürfen dabei besonderer Aufmerksamkeit:

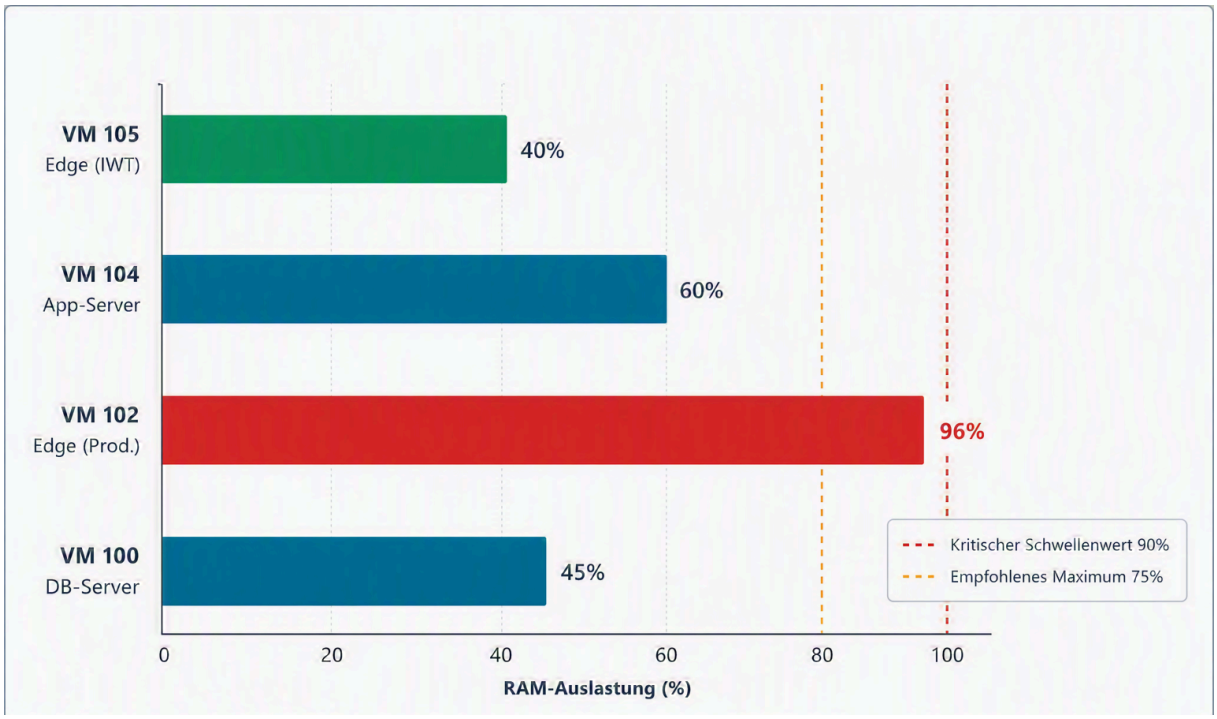


Abbildung 7 — Arbeitsspeicher-Auslastung der VMs (Schätzwerte)

5.1 Kritischer Ressourcen-Engpass – VM 102

Die dringlichste Beobachtung aus dem Monitoring betrifft den Arbeitsspeicher des produktiven Edge-Servers, weshalb dieser Punkt vorangestellt wird. VM 102 (Datenerfassungsserver – Produktion) arbeitet dauerhaft am Limit ihres zugewiesenen Arbeitsspeichers: ca. 19 GB von 20 GB sind belegt (96 %). Dies führt zu erhöhter Swap-Nutzung und kann bei intensiven API-Abfragen oder Peak-Loads zu Stabilitätsproblemen und OOM-Kills führen.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Die folgende Tabelle ordnet die empfohlenen Gegenmaßnahmen nach ihrem jeweiligen Zeithorizont:

Zeithorizont	Maßnahme
Kurzfristig	RAM-Aufstockung über Proxmox-GUI: VM 102 → Hardware → Memory von 20 GB auf mindestens 24 GB erhöhen (Neustart erforderlich)
Mittelfristig	Aufstockung auf 32 GB empfohlen, um ausreichend Puffer für zukünftige Erweiterungen (mehr Maschinenanbindungen) zu haben
Monitoring	Einrichtung von Proxmox-Alerts bei RAM-Auslastung > 85 % zur proaktiven Benachrichtigung des Administrators
Analyse	Überprüfung von FORCAM-Edge-Prozessen auf Memory Leaks und unnötige Datenpufferung

Tabelle 8 — Empfohlene Maßnahmen

5.2 Wartungsroutinen

Über die akute Behebung des Engpasses hinaus sichert ein fest getakteter Wartungsrhythmus den langfristig stabilen Betrieb. Die folgenden Routinen sind nach ihrer empfohlenen Frequenz gegliedert:

Wöchentlich:

- Proxmox-Dashboard: Ressourcenauslastung aller VMs prüfen
- FORCAM-Systemlog auf Fehler und Warnungen durchsuchen
- Edge-Verbindungen: alle Maschinen-Konnektivitäten verifizieren

Monatlich:

- System-Updates: `apt-get update && apt-get upgrade` auf allen VMs
- Package Database aktualisieren (FORCAM-Update-Funktion nutzen)
- Keycloak: inaktive Benutzer und abgelaufene Service Accounts entfernen
- Proxmox: Snapshots verwalten und veraltete Backups löschen

Quartalsweise:

- Vollständiges Backup aller VMs inklusive Konfigurationen erstellen
- Disaster-Recovery-Test: VM-Wiederherstellung aus Backup verifizieren
- Sicherheitsaudit: SSL-Zertifikate, Keycloak-Konfiguration, Firewall-Regeln

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

- Kapazitätsplanung: Wachstum der Datenbankgröße und des RAM-Bedarfs bewerten

5.3 Update-Strategie

Da Aktualisierungen an einem produktiven MES stets ein Ausfallrisiko bergen, bedarf es einer klar definierten Vorgehensweise. Das System verfügt über eine integrierte Update-Funktion für die Package Database auf Basis von `apt-get`; für deren Einsatz wird folgende Strategie empfohlen:

- Updates zunächst auf einem Testsystem oder VM 105 (IWT) validieren, bevor sie auf das Produktionssystem (VM 102) angewendet werden
- Vor jedem Update: Proxmox-Snapshot der betreffenden VM erstellen
- FORCAM-spezifische Updates ausschließlich in Wartungsfenstern einspielen
- Update-Log dokumentieren: Datum, Version, durchführende Person, Ergebnis
- Nach Updates: Funktionstest aller Maschinenverbindungen und der Web-GUI

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

6 Alternative MES-Architektur (Eigenentwicklung)

Die in den vorangegangenen Kapiteln geschilderten Hürden – ein durch den Hersteller nicht mehr unterstütztes System (End-of-Support), fehlende administrative Zugangsdaten sowie eine restriktive, schwer zugängliche Legacy-API – werfen zwangsläufig die grundsätzliche Frage auf, ob die reine Anbindung an das proprietäre FORCAM-System langfristig die tragfähigste Lösung darstellt. Als strategische Alternative zu diesem „Buy"-Ansatz wird in diesem Kapitel konzeptionell entworfen, wie ein von Grund auf eigenständig entwickeltes Manufacturing Execution System („Build") auf Basis moderner, quelloffener Technologien aufgebaut sein könnte. Ziel dieses Kapitels ist nicht die vollständige Implementierung, sondern die architektonische Evaluation eines Referenzentwurfs, der die zuvor erarbeiteten theoretischen Grundlagen (vgl. Kapitel 2) zu einem konsistenten Gesamtsystem zusammenführt.

6.1 Motivation: Build-vs-Buy-Entscheidung

Am Anfang der Überlegung steht die grundsätzliche wirtschaftliche und technische Abwägung, die den Rahmen für den gesamten Entwurf absteckt. In der Wirtschaftsinformatik beschreibt die „Build-vs-Buy"-Entscheidung den strategischen Abwägungsprozess zwischen dem Zukauf einer fertigen Standardsoftware und der Eigenentwicklung einer maßgeschneiderten Lösung. Kommerzielle MES-Suiten wie FORCAM FORCE bieten den Vorteil eines sofort einsatzfähigen, erprobten Funktionsumfangs und eines (im Regelfall) verfügbaren Hersteller-Supports. Dem stehen jedoch erhebliche Nachteile gegenüber, die sich im vorliegenden Projekt exemplarisch manifestieren.

Der gravierendste Nachteil ist der **Vendor-Lock-in**: Die Abhängigkeit von einem einzigen Hersteller führt dazu, dass mit dem Auslaufen des Supports (End-of-Life) das gesamte System zu einem nur schwer wartbaren Altlast-Faktor (Technical Debt) wird. Hinzu kommen wiederkehrende Lizenzkosten, eine begrenzte Anpassbarkeit der starren Benutzeroberflächen sowie proprietäre Datenformate, die einen Datenexport erschweren. Eine Eigenentwicklung verspricht demgegenüber volle Datenhoheit, vollständige Anpassbarkeit an die spezifischen Anforderungen des Shopfloors und die Vermeidung von Lizenzgebühren – erkauft wird

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

dies allerdings durch einen erheblichen initialen Entwicklungsaufwand und die dauerhafte Verantwortung für Wartung und Betrieb (vgl. Tabelle 10).

6.2 Architekturparadigma: Vom Monolithen zur Microservice-Architektur

Aus der Entscheidung für eine Eigenentwicklung ergibt sich unmittelbar die Frage nach dem passenden Architekturstil. Klassische MES-Lösungen sind häufig als monolithische Applikationen konzipiert, bei denen Datenerfassung, Geschäftslogik und Präsentationsschicht in einer einzigen, eng gekoppelten Codebasis vereint sind. Ein solcher Monolith skaliert schlecht und ist wartungsintensiv, da bereits kleine Änderungen ein vollständiges Neu-Deployment des Gesamtsystems erfordern.

Der hier vorgeschlagene Referenzentwurf bricht bewusst mit diesem Muster und folgt einer **ereignisgetriebenen Microservice-Architektur** (Event-Driven Architecture). Das System wird dabei in mehrere fachlich klar abgegrenzte, unabhängig voneinander deploybare Dienste zerlegt (z.B. einen Dienst für die OEE-Berechnung, einen für das Auftragsmanagement und einen für die Alarmierung). Diese Dienste kommunizieren nicht direkt und synchron miteinander, sondern lose gekoppelt über einen zentralen Nachrichtenstrom (Message Broker). Wie in Tabelle 9 ersichtlich, lassen sich sämtliche Bausteine einer solchen Architektur durch etablierte Open-Source-Komponenten abbilden. Technologisch würden diese Dienste – analog zur Überlegung einer autarken Middleware (vgl. Kapitel 3) – in ressourcenschonenden Containern (Docker) betrieben und über eine Orchestrierungsplattform verwaltet.

6.3 Schichtenmodell des Eigenentwurfs

Konkret gliedert sich der Referenzentwurf in fünf logische Schichten, die den Datenfluss von der physischen Maschine bis zum Endanwender abbilden und im Folgenden einzeln vorgestellt werden. Jede Schicht erfüllt eine dedizierte Aufgabe und ist über standardisierte Schnittstellen von der jeweils benachbarten Schicht entkoppelt.

Konnektivitätsschicht (Data Ingestion)

Die unterste Schicht übernimmt – analog zum FORCAM Edge – die Rolle des Protokoll-Übersetzers zwischen der heterogenen Maschinenwelt und dem digitalen System. Anstelle eines proprietären Gateways kommen hier quelloffene Bibliotheken und Werkzeuge zum Einsatz. Für die Anbindung von OPC-UA-fähigen Anlagen eignet sich beispielsweise die Bibliothek

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

`node-opcua`, für ältere Siemens-Steuerungen (S7 über RFC 1006) das Framework `Apache PLC4X`, und für die Erfassung leichtgewichtiger Telemetrie das ereignisgesteuerte MQTT-Protokoll in Verbindung mit einem Broker wie `Eclipse Mosquitto`. Jeder Maschinentyp wird durch einen kleinen, dedizierten Konnektor-Dienst angebunden, der die rohen Binär- oder Protokolldaten ausliest, in ein einheitliches JSON-Format normalisiert und mit einem Zeitstempel versieht.

Ein zentrales Architekturkonzept moderner IIoT-Systeme (Industrial Internet of Things) ist hierbei der sogenannte **Unified Namespace (UNS)**: Sämtliche Maschinendaten werden – strukturiert nach der ISA-95-Hierarchie (Werk / Bereich / Linie / Maschine) – in einer einzigen, semantisch eindeutigen Namensstruktur auf dem MQTT-Broker publiziert. Der UNS fungiert damit als „Single Source of Truth“ für den gesamten aktuellen Zustand der Produktion, aus dem sich beliebige nachgelagerte Dienste bedienen können, ohne die einzelnen Maschinen erneut direkt abfragen zu müssen.

Streaming- und Nachrichtenschicht

Das Herzstück der Entkopplung bildet ein persistenter Nachrichtenbus. Während für reine Zustandsdaten der MQTT-Broker genügt, empfiehlt sich für die zuverlässige, ausfallsichere Verarbeitung hochfrequenter Datenströme der Einsatz einer Streaming-Plattform wie `Apache Kafka`. Kafka puffert die eingehenden Ereignisse dauerhaft in einem verteilten Log und erlaubt es, denselben Datenstrom von mehreren Diensten unabhängig konsumieren zu lassen. Ein entscheidender Vorteil gegenüber der rein flüchtigen In-Memory-Pufferung des FORCAM Edge (welche den kritischen RAM-Engpass auf VM 102 mitverursacht, vgl. Kapitel 5) ist die **Replay-Fähigkeit**: Fällt ein nachgelagerter Verarbeitungsdienst aus, kann er nach dem Neustart die verpassten Nachrichten lückenlos aus dem Kafka-Log nachverarbeiten, ohne dass Maschinendaten verloren gehen.

Persistenzschicht (Time-Series und Stammdaten)

Wie bereits bei der Betrachtung von VM 100 erläutert, skalieren klassische relationale Tabellen bei hochfrequenten Zeitreihendaten schlecht. Der Eigenentwurf trennt die Persistenz daher konsequent nach Datencharakteristik:

- **Zeitreihen-Datenbank:** Für die massiven, kontinuierlichen Sensordatenströme kommt eine spezialisierte Time-Series Database (TSDB) wie `TimescaleDB` (eine PostgreSQL-Erwei-

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

terung) oder `InfluxDB` zum Einsatz. Diese sind auf hohe Schreibraten (High-Throughput Ingestion) und die effiziente Aggregation über Zeitfenster optimiert und lösen damit das im Data Lake beschriebene Hot-/Cold-Storage-Problem nativ.

- **Relationale Datenbank:** Strukturierte Stammdaten mit klaren Relationen – etwa Fertigungsaufträge, Artikelstämme, Schichtmodelle und Benutzerdaten – werden weiterhin in einer klassischen relationalen Datenbank wie `PostgreSQL` gehalten, da hier die referentielle Integrität (ACID-Eigenschaften) im Vordergrund steht.

Verarbeitungsschicht (OEE-Engine und Geschäftslogik)

In dieser Schicht wird aus den rohen Maschinensignalen verwertbares Wissen. Ein dedizierter **OEE-Microservice** konsumiert den Datenstrom aus Kafka und berechnet in Echtzeit die in Kapitel 2 hergeleitete Gesamtanlageneffektivität aus den Faktoren Verfügbarkeit, Leistung und Qualität:

$$\text{OEE} = \text{Verfügbarkeitsfaktor} \times \text{Leistungsfaktor} \times \text{Qualitätsfaktor} \tag{2}$$

Für die kontinuierliche Auswertung der Datenströme (Stream Processing) eignen sich Frameworks wie `Kafka Streams` oder `Apache Flink`, welche Zustandsübergänge (etwa den Wechsel einer Maschine von „Produktion“ zu „Störung“) ereignisbasiert erkennen und daraus fortlaufend aggregierte Kennzahlen ableiten. Die eigentlichen fachlichen Dienste (Auftrags-, Alarm- und Reporting-Service) werden als ressourcenschonende Backend-Applikationen realisiert, beispielsweise auf Basis von `Node.js` (mit den Frameworks Express oder NestJS) oder `Python` (mit FastAPI).

API- und Präsentationsschicht

Nach außen exponiert das System seine Daten über eine wohldefinierte API. Wie in Kapitel 2 gegenübergestellt, offenbart der klassische REST-Ansatz [4] im Dashboard-Kontext die Schwächen des Over- und Underfetching. Ein Eigenentwurf kann daher gezielt ein `GraphQL`-Gateway (z.B. auf Basis von Apollo Server) als zentralen Einstiegspunkt vorsehen, über das das Frontend exakt die für eine Ansicht benötigten Felder in einer einzigen Abfrage anfordert – was die Netzwerklast im Shopfloor reduziert. Ergänzend kann für die maschinelle System-zu-System-Integration weiterhin eine dokumentierte REST-Schnittstelle (OpenAPI/Swagger) bereitgestellt werden.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Die Präsentationsschicht selbst bliebe technologisch identisch zum bereits konzipierten Prototyp (vgl. Kapitel 4): ein in `Next.js` realisiertes Fullstack-Dashboard, dessen serverseitige React Server Components die Authentifizierung übernehmen und die Live-Daten performant an die Thin-Clients in der Werkhalle ausliefern.

Identity & Access Management

Bemerkenswert ist, dass die Sicherheitsschicht des Eigenentwurfs vollständig auf die bereits im Bestandssystem eingesetzte Open-Source-Software `Keycloak` zurückgreifen kann. Da Keycloak herstellerunabhängig ist und die Standardprotokolle OAuth 2.0 [5] und OpenID Connect implementiert, entfällt hier jegliche Neuentwicklung. Im Gegensatz zur aktuellen Problematik (fehlende Realm-Admin-Rechte auf der proprietären FORCAM-Instanz) läge die vollständige administrative Kontrolle über den `FORCAM`-Realm, die Registrierung neuer Clients und die Anlage von Service-Accounts hier durchgehend beim Entwicklungsteam selbst.

6.4 Deployment und Betrieb

Ergänzend zur fachlichen Architektur ist zu klären, wie ein solcher Stack betrieben und ausfallsicher gehalten würde. Der gesamte Technologie-Stack würde konsequent containerisiert betrieben. Jeder Microservice, jede Datenbank und der Message Broker liefen als isolierter `Docker`-Container. Für den produktiven Betrieb mit automatischer Skalierung, Ausfallsicherheit (Self-Healing) und rollierenden Updates bietet sich eine Orchestrierungsplattform wie `Kubernetes` an; für kleinere Installationen oder das hochschulinterne Laborumfeld genügt bereits `Docker Compose` in Verbindung mit einer schlanken Verwaltungsoberfläche wie `Portainer`. Ergänzt wird der Betrieb durch einen dedizierten Monitoring-Stack aus `Prometheus` (Metrik-Erfassung) und `Grafana` (Visualisierung), der die im Bestandssystem nur rudimentär vorhandene Ressourcenüberwachung deutlich professionalisiert.

6.5 Technologie-Stack im Überblick

Um die bisher einzeln vorgestellten Schichten abschließend zu bündeln, stellt die folgende Tabelle die proprietären Komponenten der bestehenden FORCAM-Umgebung den quelloffenen Bausteinen des skizzierten Eigenentwurfs gegenüber und verdeutlicht, dass für jede Funktionsebene ausgereifte Open-Source-Alternativen existieren.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Schicht / Funktion	FORCAM (Bestand)	Eigenentwurf (Open Source)
Konnektivität / Edge	FORCAM Edge (proprietär)	node-opcua, Apache PLC4X, Eclipse Mosquitto
Nachrichtenbus / Streaming	intern / proprietär	Apache Kafka, MQTT (Unified Namespace)
Zeitreihen-Speicher	Data Lake (proprietär)	TimescaleDB / InfluxDB
Stammdaten-Datenbank	MSSQL / PostgreSQL	PostgreSQL
OEE- / Geschäftslogik	FORCAM Business Logic	NestJS / FastAPI + Kafka Streams / Flink
API-Schicht	REST (Swagger)	GraphQL (Apollo) + REST (OpenAPI)
Identity & Access	Keycloak	Keycloak (identisch)
Frontend / Dashboard	FORCAM Web-UI (Port 15443)	Next.js
Virtualisierung / Betrieb	Proxmox VE (VMs)	Docker + Kubernetes / Portainer
Monitoring	Proxmox-Bordmittel	Prometheus + Grafana

Tabelle 9 — Gegenüberstellung der Technologie-Stacks

6.6 Kritische Würdigung und Trade-offs

So überzeugend der technische Entwurf zunächst wirkt, ist er einer ehrlichen Abwägung seiner Nachteile zu unterziehen. Bei aller technischen Eleganz eines maßgeschneiderten Systems darf die Eigenentwicklung nicht als pauschal überlegene Lösung missverstanden werden. Die vollständige Nachbildung eines über Jahre gereiften MES-Funktionsumfangs (inklusive Feinplanung, Qualitätsmanagement, Chargenrückverfolgung und normkonformer Dokumentation) stellt einen erheblichen personellen und zeitlichen Aufwand dar, der die Kapazitäten eines Hochschulprojekts oder einer kleinen Entwicklungsabteilung schnell übersteigt. Zudem verlagert sich die volle Verantwortung für Wartung, Sicherheitsaktualisierungen und Ausfallsicherheit vom Hersteller auf das eigene Team – ein Aspekt, der bei der Betrachtung der Gesamtbetriebskosten (Total Cost of Ownership, TCO) über die reine Betrachtung der Lizenzkosten hinausgeht.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Die nachfolgende Tabelle fasst die zentralen Abwägungskriterien gegenüberstellend zusammen:

Kriterium	**Standardsoftware (FORCAM)**	**Eigenentwicklung**
Initialer Aufwand	Gering (sofort einsatzfähig)	Sehr hoch (vollständige Entwicklung)
Laufende Lizenzkosten	Hoch (herstellerabhängig)	Keine (nur Betrieb/Personal)
Anpassbarkeit	Gering (starre Oberflächen)	Vollständig frei
Time-to-Market	Kurz	Lang
Vendor-Lock-in	Hoch (EoL-Risiko)	Keiner (volle Datenhoheit)
Wartungsverantwortung	Beim Hersteller	Vollständig beim eigenen Team
Benötigtes Know-how	Anwenderwissen	Tiefes Entwickler- & Betriebs-Know-how
Funktionsreife	Hoch (erprobt)	Anfangs gering (Eigenrisiko)

Tabelle 10 — Abwägung: Standardsoftware (Buy) vs. Eigenentwicklung (Build)

Zusammenfassend lässt sich festhalten, dass ein eigenentwickeltes MES auf Basis des skizzierten, quelloffenen Microservice-Stacks technisch durchaus realisierbar ist und die konkreten Schwachpunkte des Bestandssystems – insbesondere den Vendor-Lock-in, die fehlende Datenhoheit und die restriktive API – gezielt adressiert. Für das vorliegende Praxisprojekt bietet sich dabei ein pragmatischer **hybrider Zwischenschritt** an: Anstatt das gesamte MES sofort zu ersetzen, kann zunächst nur die problematische Datenzugriffsschicht durch die bereits in Kapitel 3 diskutierte, schlanke Eigen-Middleware ersetzt werden, während das bewährte FORCAM-System für unkritische Funktionen zunächst weiterläuft. Ein solcher evolutionärer Ansatz (Strangler-Fig-Muster) erlaubt es, die Vorteile der Eigenentwicklung schrittweise und risikoarm zu erschließen, ohne den laufenden Produktionsbetrieb zu gefährden.

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

7 Anhang: Wichtige URLs & Referenzen

Zur schnellen Orientierung im laufenden Betrieb bündelt die folgende Tabelle die im Projekt relevanten Zugangspunkte und Endpunkte:

Bezeichnung	**URL**	**Beschreibung**
Proxmox Web-GUI	https://141.68.41.5:8006	Hypervisor-Verwaltung, VM-Konsolen, Ressourcenmonitoring
FORCAM Portal (Web-GUI)	https://<anwendungsserver>:15443/	Hauptzugang zum MES-System, grafisches Web-Frontend
Keycloak Admin	https://<anwendungsserver>:25443/ffauth/	Identity-Management-Portal
Keycloak Realm	https://<anwendungsserver>:25443/ffauth/realms/FORCAM/	FORCAM-Sicherheitsbereich
Token-Endpunkt	.../protocol/openid-connect/token	OAuth2-Token-Beschaffung (Client Credentials)
Swagger UI	https://<anwendungsserver>:25443/swagger-ui	Interaktive REST-API-Dokumentation

Tabelle 11 — Wichtige URLs & Referenzen

7.1 Verwendete Technologien & Versionen

Abschließend listet die folgende Tabelle die eingesetzten Kerntechnologien mit ihren Versionsständen und Einsatzbereichen auf:

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Technologie	Version	Einsatzbereich
Proxmox VE	8.4.0	Hypervisor-Plattform
FORCAM Edge	Aktuell	Konnektivitätsschicht / Maschinenanbindung
FORCAM FORCE MES FLEX	Aktuell	Anwendungsschicht / MES-System
Keycloak	Integriert	Identity & Access Management (IAM)
MSSQL	Aktuell	Relationale Datenbank (Primär)
PostgreSQL	Aktuell	Relationale Datenbank (Alternativ)
OAuth2/OIDC	RFC 6749	Authentifizierungsprotokoll

Tabelle 12 — Verwendete Technologien & Versionen

Docusign Envelope ID: A1B1B171-4ADE-869A-835F-03D9116B2BDD

Literatur

[1] H. Kagermann, W. Wahlster, und J. Helbig, „Securing the Future of German Manufacturing Industry: Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0“, Frankfurt am Main, 2013.

[2] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standards in Manufacturing*. ISA – International Society of Automation, 2007.

[3] S. Nakajima, *Introduction to TPM: Total Productive Maintenance*. Cambridge, MA: Productivity Press, 1988.

[4] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures“, 2000. [Online]. Verfügbar unter: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

[5] D. Hardt, „The OAuth 2.0 Authorization Framework“. [Online]. Verfügbar unter: https://www.rfc-editor.org/info/rfc6749

[6] Meta Platforms, Inc., „React Documentation: Preserving and Resetting State“. Zugegriffen: 10. April 2024. [Online]. Verfügbar unter: https://react.dev/learn/preserving-and-resetting-state

[7] Vercel Inc., „Next.js Documentation: Routing and Rendering“. Zugegriffen: 10. April 2024. [Online]. Verfügbar unter: https://nextjs.org/docs

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, und A. Liguori, „kvm: the Linux Virtual Machine Monitor“, in *Proceedings of the 2007 Linux Symposium*, 2007, S. 225–230.