FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Seminararbeit

Konzept zur Authentifizierung mittels Microservices

Jan Glück

Matrikelnummer: 3566118

1. Prüfer: Prof. Dr. rer. Nat. Sebastian Voss

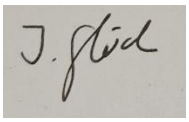
2. Dipl. –Math. Olga Wolf

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel genutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Name: Jan Glück

Aachen, den 16.12.2020



Unterschrift der Studentin / des Studenten

Abstract

In einer zunehmend digitalisierten Welt gewinnen Microservices als Architekturmodell an Bedeutung, insbesondere für die Entwicklung skalierbarer und flexibler Systeme. Diese Arbeit beschäftigt sich mit der Konzeption und Implementierung eines zentralen Microservices zur Verwaltung von Zugriffsrechten, der die bestehenden Tools der Abteilung Produktionsmanagement des Werkzeugmaschinenlabors WZL der RWTH Aachen unterstützt.

Ziel der Arbeit ist es, ein modular aufgebautes System zu entwickeln, das eine zentrale Rechteverwaltung ermöglicht und gleichzeitig eine einfache Integration in bestehende Anwendungen sicherstellt. Dabei werden sowohl technische Herausforderungen wie Skalierbarkeit, Fehlertoleranz und Flexibilität, als auch organisatorische Aspekte berücksichtigt.

Im Verlauf der Arbeit wird ein umfassendes Konzept entwickelt, das auf einer Microservice-Architektur basiert und die Nutzung von Schlüsseltechnologien wie Keycloak und Spring Boot integriert.

Inhaltsverzeichnis

1 Einleitung

In der heutigen zunehmend digitalisierten und vernetzten Arbeitswelt sind Microservices zu einem zentralen Bestandteil moderner Softwarearchitekturen geworden. Sie bieten Unternehmen die Möglichkeit, skalierbare und flexible Anwendungen zu entwickeln, die sich schnell an veränderte Anforderungen anpassen lassen. Ein wichtiger Aspekt dieser Systeme ist das Management von Zugriffsrechten, das sicherstellt, dass Benutzer nur auf die Ressourcen zugreifen können, die für ihre Rolle und Aufgaben notwendig sind.

Die Abteilung Produktionsmanagement des Werkzeugmaschinenlabors WZL der RWTH Aachen ist auf die Optimierung von Prozessen spezialisiert. Dafür sind intern mehrere Tools entwickelt wurden, die unter anderem Prozesse analysieren und optimieren, als auch globale Netzwerke abbilden und dabei unterstützen diese neu zu strukturieren.

Diese Seminararbeit beschäftigt sich mit der Konzeption und Implementierung eines Microservices zur Verwaltung von Zugriffsrechten für Ressourcen von den soeben genannten Tools. Dabei werden sowohl technische als auch organisatorische Herausforderungen beleuchtet und eine Lösung erarbeitet, die später in der Unternehmenspraxis eingesetzt wird.

1.1 Motivation

Da bei den verschiedenen Tools der Abteilung Produktionsmanagement auch der Zugriff für die verschiedenen Ressourcen verwaltet werden muss, soll nun ein System konzeptioniert und umgesetzt werden, welches zentral für alle bestehenden und kommenden Tools die Zugriffsrechte verwaltet. Aktuell wird die Zugriffsberechtigung in jedem Tool einzeln gemanagt und muss somit einzeln entwickelt und gepflegt werden.

Damit man die man diesen Prozess nicht für jedes Tool einzeln wiederholen muss, soll ein zentraler Service konzeptioniert und implementiert werden, welcher die Rechte für

die Ressourcen aller Tools verwaltet. Um dies so modular und skalierbar wie möglich zu machen, soll es mit einer Microservicearchitektur umgesetzt werden.

1.2 Ziel der Arbeit

In einer früheren Bachelorarbeit von Mark Junglas wurde das Problem bereits thematisiert, und es entstand ein erstes Konzept für ein solches System. Allerdings wies dieses Konzept einige Schwächen auf, insbesondere in Bezug auf die Skalierbarkeit und die aufwändige Integration in bestehende Systeme. Ziel dieser Arbeit ist es, ein neues, verbessertes Konzept zu entwickeln, das die zentrale Verwaltung von Zugriffsrechten für Ressourcen innerhalb der bestehenden sowie zukünftigen Tools der Abteilung Produktionsmanagement ermöglicht. Dabei dient das Konzept von Mark Junglas als grundlegende Orientierung, wird jedoch an vielen Stellen überarbeitet und erweitert. Sowohl die Funktionalitäten als auch die Prozessabläufe werden grundlegend angepasst, um die identifizierten Schwachpunkte zu beheben und eine Lösung zu finden.

1.3 Aufbau der Arbeit

In dieser Arbeit werden zu Beginn die Grundlegen Theorien und Techniken zu Microservices in Kapitel 2 erläutert. In Kapitel 3 werden dann die Anforderungen an das System definiert. Daraufhin wird das Konzept für das System in Kapitel 4 entwickelt und die Implementierung und Umsetzung der verschiedenen Microservices wird dann in Kapitel 5 erklärt. Zum Schluss wird es dann in Kapitel 6 noch eine Zusammenfassung und einen Ausblick geben.

2 Grundlagen und Techniken

In dem folgenden Kapitel werden Microservices grundlegend erklärt und welche verschiedenen Vorgänge bzw. Techniken von Microservices angewendet werden.

2.1 Microservices

In den nächsten Abschnitten werden grundlegend Microservices, wichtige Vorgänge und Technologien erklärt.

2.1.1 Architekturmuster

Ganz grob gesagt sind Microservices nur eine Art von Architekturmuster, bei der mehrere voneinander unabhängige Services ihren gewissen Aufgaben erledigen und miteinander über APIs kommunizieren. [1] Es zielt darauf ab komplexe Anwendungen in kleinere Dienste zu zerlegen. Jeder dieser Dienste kann unabhängig von den anderen entwickelt, deployed und skaliert werden. [2]

Die Microservice-Architektur bildet den Gegenspieler zur monolithischen Architektur, bei der alle Funktionen in eine Anwendung integriert sind. [3] [1] Anhand eines Beispiels lassen die die Architekturen besser verstehen. Nimmt man als Beispiel eine Online-Bibliothek. Solche Web-Anwendungen lassen sich oft in drei Verschiedene Teile aufteilen. Das Frontend, das Backend und die Datenbank. Das Frontend ist die Oberfläche, welche die Daten von Backend visuell darstellt. Das Backend ist für die Logik und das Verarbeiten von Anfragen zuständig und die Datenbank zum Speichern und Anlegen von Daten. Wenn man das Backend mit einer monolithischen Architektur machen würde, hätte man eine komplexe Anwendung mit allen Funktionen. Wenn man nun eine Funktion von der Backend verändern, muss man die gesamte Anwendung neu deployen.

Im Vergleich dazu wäre das Backend bei einer Microservicearchitektur in kleine Anwendungen aufgeteilt und man hat den Vorteil, dass man jede dieser kleinen Anwendungen unabhängig von den anderen weiterentwickeln und deployen kann. Dadurch sinkt der Wartungsaufwand und das Skalieren der Anwendung ist vereinfacht.

2.1.2 Vor- und Nachteile

Wie eben schon erwähnt, sind Microservices einfacher zu skalieren, da jeder Dienst unabhängig skaliert werden kann, wodurch eine effiziente Ressourcennutzung ermöglicht wird [1]. Außerdem hat man eine etwas größere Fehlertoleranz, da bei einem Ausfall eines Microservices nicht der Rest der Anwendung funktionsunfähig

wird. [2] Bei der Entwicklung von Microservice hat man auch den Vorteil, dass kleinere Teams spezifischer an Funktionen arbeiten können, wodurch die Entwicklung beschleunigt werden kann [1]. Die verschiedenen Services können auch nach Belieben mithilfe von verschiedenen Programmiersprachen und Technologien erstellt werden, was den Programmierenden Flexibilität bietet [1].Natürlich bringen Microservices auch gewisse Herausforderungen mit sich. Durch die Anzahl an mehreren Diensten, müssen diese auch miteinander kommunizieren, was zu möglichen Latenzen führen kann. Außerdem muss sich um die Schnittstellen der verschiedenen Services gekümmert werden, das heißt man muss sich um die Übersetzung der Datenkontexte kümmern. [2] [1]

2.1.3 Service Discovery Test

Service Discovery Tests sind ein essenzieller Bestandteil beim Testen von Microservice-Architekturen. Sie gewährleisten, dass Microservices sich gegenseitig finden und erfolgreich miteinander kommunizieren können. Solche Tests decken mehrere Aspekte ab:

- Registrierung und Deregistrierung: Es wird überprüft, ob Microservices sich beim Start korrekt beim Service Discovery Server registrieren und beim Beenden ordnungsgemäß entfernt werden.
- Adressauflösung: Sicherstellen, dass Microservices die Adressen anderer Dienste finden und mit ihnen kommunizieren können.
- Adressaktualisierung: Szenarien, in denen sich die Adresse eines Microservices ändert, werden getestet, um sicherzustellen, dass andere Microservices weiterhin korrekt mit diesem kommunizieren.
- Ausfallszenarien: Tests, bei denen ein Microservice ausfällt, stellen sicher, dass Anfragen an andere Dienste dennoch korrekt verarbeitet werden.

Für die Durchführung solcher Tests stehen verschiedene Technologien zur Verfügung. Eureka oder auch Consul sind zwei bekannte Tools, die solche Tests durchführen können und jeweils weitere Vorteile mit sich bringen.

Zusammenfassend sind Service Discovery Tests essenziell für den reibungslosen Betrieb von Microservice-Umgebungen. Sie tragen dazu bei, eine zuverlässige,

skalierbare und stabile Umgebung zu schaffen, in der Dienste effektiv zusammenarbeiten können.

2.1.4 Kommunikation von Microservices

Es gibt zahlreiche Ansätze, wie Microservices eingesetzt und wie sie miteinander kommunizieren können. In dieser Arbeit konzentrieren wir uns auf eine Kommunikation auf Logik-Ebene, bei der die verschiedenen Services über HTTP-Anfragen miteinander interagieren. Als Architekturansatz verwenden wir dabei den Representational State Transfer (REST). Andere Kommunikationsansätze werden in dieser Arbeit bewusst vernachlässigt.

REST ist ein Architekturstil, der häufig in der Entwicklung von Webservices eingesetzt wird. Dabei werden Ressourcen über eindeutige URLs identifiziert. Ein Beispiel hierfür wäre:

- /user/1 → Diese URL stellt den Benutzer mit der ID 1 dar.

REST verwendet vier zentrale HTTP-Methoden, die den CRUD-Operationen (Create, Read, Update, Delete) entsprechen:

- GET: Zum Abrufen von Ressourcen.
- POST: Zum Erstellen neuer Ressourcen.
- PUT: Zum Aktualisieren bestehender Ressourcen.
- DELETE: Zum Löschen von Ressourcen.

Um die Adressen der Microservices dynamisch zu halten, kann man einen Service Discovery-Mechanismus verwenden. Dieser ermöglicht es, wie in Kapitel 2.3.1 beschrieben, dass die Adressen der Services untereinander bekanntgegeben werden, wodurch eine reibungslose Kommunikation gewährleistet ist.

2.2 Vergleich zu anderen Architekturmodellen

Microservices werden oft mit serviceorientierter Architektur (SOA) verglichen. Beide Ansätze teilen Anwendungen in Services auf. Microservices und SOA haben beide ihre eigenen Vorteile. SOA konzentriert sich auf die Koordination und Verwaltung von

Services, um Flexibilität zu erreichen, während Microservices auf schnelle Deployments und unabhängige Entwicklung setzen. Der Anwendungsbereich von SOA ist außerdem eher unternehmensweit, während Microservices eher die Architektur eines einzelnen Projekts betreffen. SOA verwendet dazu noch monolithisches Deployment mehrerer Services, wobei Microservices einzeln bereitgestellt werden können. Dadurch ist SOA eher für Systeme geeignet, die sich seltener ändern und Microservices für Systeme, die sehr flexibel sind und oft geändert werden sollen.

Monolithische Architektur ist der Gegenspieler zu den Microservices. Microservices haben dafür folgende Vorteile gegenüber der monolithischen Architektur. Durch das einzelne bereitstellen der Services haben sie deutlich besser Skalierbarkeit und auch eine höhere Fehlertoleranz. Bei sehr komplexen Systemen, wobei sehr viele Microservices notwendig wären, ist die monolithische Architektur im Vorteil, da der Wartungsaufwand bei Microservices stark steigt.

Zusammenfassend lässt sich sagen, dass Microservices eine Reihe von Vorteilen gegenüber anderen Architekturmodellen bieten, aber sie bringen auch einige Herausforderungen mit sich. Die Entscheidung, ob Microservices die richtige Architektur für ein bestimmtes Projekt sind, hängt von den spezifischen Anforderungen des Projekts ab. [2] [3] [1]

3 Anforderungen an das System

In den folgenden Abschnitten werden die Anforderung an das System erläutert, welche von der vom der Abteilung Produktionsmanagement am WZL festgelegt wurden.

3.1 Grundlegende Anforderungen an das System

Das System, welches in dieser Arbeit entwickelt wird, soll bei Anfragen von Nutzern ans Backend überprüfen und die Entscheidung treffen, ob der Nutzer für die Aktion, die er ausführen möchte, auch die Rechte hat. Die Rechte der Nutzer werden als

Policies in einer Datenbank gespeichert. In den Policies ist abgespeichert, welche Person oder Gruppe, für welche Ressource, welche Aktionen durchführen darf.

3.2 Skalierbarkeit

Das System soll skalierbar sein und nicht durch eine hohe Anzahl von Nutzern oder Ressourcen in Probleme geraten.

3.3 Flexibilität

Es soll flexibel für alle Tools verwendbar sein und die gewünschte Funktionalität erfüllen.

3.4 Wartbarkeit

Das implementierte System soll möglichst wartbar sein.

3.5 Erweiterbarkeit

In dem Fall, dass man die Policies modularer aufbauen soll bzw. erweitern möchte, soll dies in System möglich sein, ohne alles Bestehende verwerfen zu müssen.

3.6 Fehlertoleranz

Falls es einen Ausfall von einem Teil des Systems gibt oder andere Fehler auftreten, sollen diese vom System abgefangen werden.

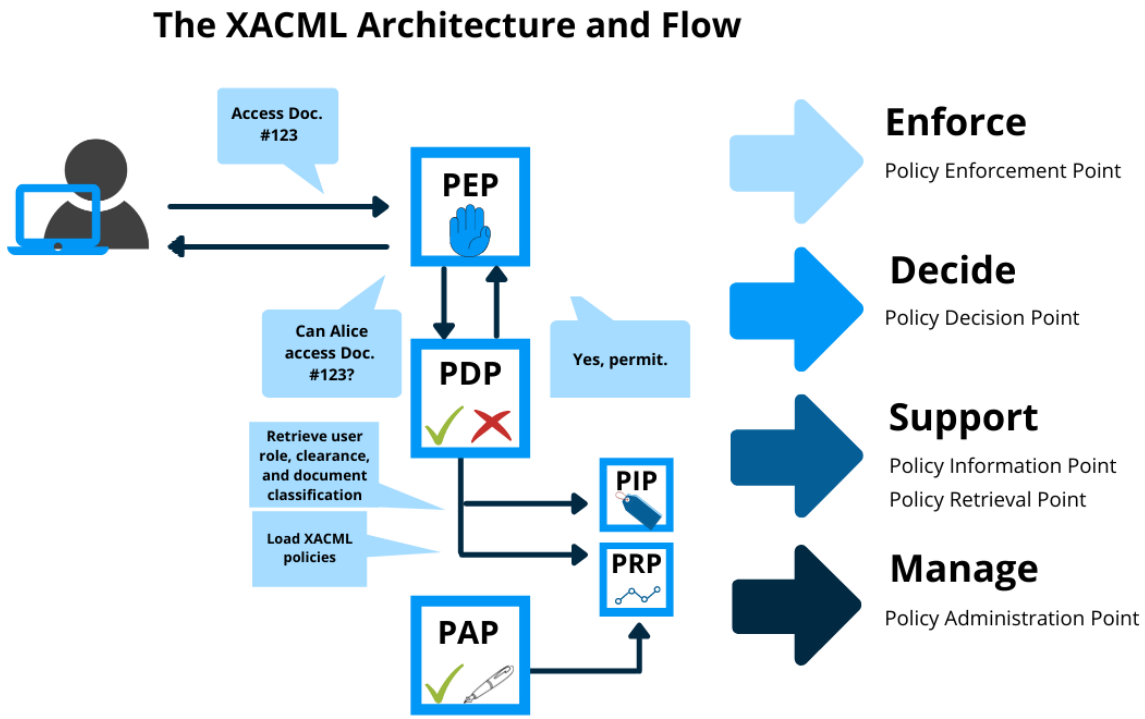
3.7 Integrationsfähigkeit

Das System soll in bestehende Systeme möglichst ohne großen Aufwand integrierbar sein.

4 Konzeption

Für die in dem Kapitel „Anforderungen an das System“ genannten Anforderungen, wurde ein Konzept entwickelt, welches in dem folgenden Kapitel genauer erklärt wird.

4.1 Aufbau von dem System



In der Abbildung sieht man nochmal den Ablauf vom System, aber auch die weiteren Services, die dem Policy Decision Point (PDP) helfen Informationen zu sammeln, wie auch den Policy Enforcement Point (PEP), welcher die Anfragen vom Backend empfängt und die Entscheidungen vom PDP weiterleitet. Der Policy Information Point (PIP) ist dafür Informationen zu dem Nutzer zu beschaffen. Dafür erhält dieser die Nutzer-ID vom PDP und fragt dann Keycloak an, um die Rolle und die Gruppen zu welchem der Nutzer gehört zu bekommen und dem PDP diese Informationen zu schicken. Der Policy Retrieval Point hat als Funktion die zur Anfrage passenden Policies rauszusuchen. Dazu bekommt er vom PDP die Ressourcen, Aktion und Nutzerinformationen und sucht in der Datenbank die Policies raus, welche dazu passen. Außerdem gibt es noch den Policy Adminstration Point (PAP), welcher dazu da ist Policies zu erstellen, löschen und zu bearbeiten. Dazu macht dieser dann Anfragen zum PRP, wo diese dann angelegt werden.

4.2 Ablauf von einer Anfrage

Der Ablauf startet mit einem Nutzer, der eine Anfrage ans Backend stellt. Das Backend verarbeitet daraufhin die Anfrage vom Nutzer und sucht in der Datenbank die IDs der Ressourcen, für welche der Zugriff überprüft werden muss, raus. Das Backend schickt dann an den PEP eine Anfrage, welche die Ressourcen, die Aktion und die Nutzer-ID beinhaltet. Der PEP verarbeitet diese Informationen und leitet sie an den PDP weiter. Der PDP fragt weitere Dienste an, um weitere Informationen zu sammeln und die dazu passenden Policies zu erhalten. Anhand von diesen Informationen entscheidet der PDP, ob die Anfrage erlaubt oder abgelehnt wird. Diese Entscheidung wird dann zum PEP geschickt, welche diese dann ans Backend weiterleitet. Das Backend wird dann anhand der Entscheide die Anfrage entweder durchführen oder ablehnen und eine Fehlercode ans Frondend schicken.

4.3 Aufbau von Policies

Die Policies sind die Regeln, die festlegen, ob ein Nutzer die Rechte für eine gewisse Aktion hat.

Zu Beginn muss erwähnt werden, dass wir Keycloak zur Nutzerverwaltung benutzen und dort eine Person auch gewissen Gruppen zugeteilt werden kann. In Kontext würde dann ein Nutzer auch einer Gruppe hinzugefügt, wie zum Beispiel seiner Abteilung oder seiner Organisation. Ein Nutzer kann auch in mehreren Gruppen sein (Organisation -> Abteilung -> Gruppe).

Eine Policy selbst beinhaltet mehrere Informationen. Dazu gehören Tool, Ressource, Aktion, Nutzer und Gruppen. Das Tool und die Ressource sind zwei Strings, welche abspeichern um welches Tool es sich handelt und um welche Ressource innerhalb von diesem Tool. Die Aktion ist ebenfalls ein String und hat drei verschiedenen Optionen („POST“, „DELETE“, „GET“ , „PUT“ und „ADMIN“) und zeigt an für welche Aktion die Policy die Rechte angibt. Die letzten zwei Attribute sind die Nutzer und die Gruppen. Beide Attribute sind Listen aus Strings, da eine Policy direkt für mehrere einzelne Nutzer angelegt werden kann, aber auch für mehrere Gruppen.

4.4 Erstellen/Bearbeiten/Löschen von Policies

Das Erstellen von neuen Policies erfolgt durch eine Anfrage durch das Tool an den PAP. Bei einem Anlegen einer neuen Ressource innerhalb von einem Tool wird automatisch für den Ersteller von der Ressource eine Policy angelegt, welche dem Ersteller die Rechte dazu gibt „ADMIN“-Aktionen in Bezug auf diese Ressource auszuführen. Innerhalb vom Tool hat der Ersteller dann die Möglichkeit anderen Nutzern oder auch ganzen Gruppen auch Rechte auf die Ressource zu geben. Falls er dies tut, wird eine Anfrage an den PAP geschickt, welcher zu Beginn überprüft, ob der Nutzer die Rechte für diese „Admin“-Aktion hat und daraufhin dann eine neue Policy anlegt mit den angegebenen Daten. Natürlich kann der Ersteller auch anderen Nutzern die Rechte wieder wegnehmen oder bearbeiten, woraufhin dementsprechend Policies gelöscht werden oder auch bearbeitet werden.

4.5 Aufgaben und Funktionsweisen der einzelnen Microservices

In den folgenden Kapiteln werden die Aufgaben und Funktionsweisen der einzelnen Microservices, welche im Konzept verwendet, werden erklärt.

4.5.1 Policy Enforcement Point

Der Policy Enforcement Point ist eine wichtige Komponente in dem ganzen Konzept, welches in dieser Arbeit erarbeitet wird. Dieser Dienst hat als Funktion die Anfragen entgegenzunehmen und nach Entscheidung, ob der Nutzer die Rechte für diese Anfrage hat, diese Entscheidung durchzusetzen. Für die Entscheidung selbst sind andere Dienste in dem System zuständig.

Der Policy Enforcement Point ist der erste Schritt in der Kette von Services in dem Konzept. Er empfängt die Anfragen vom Backend und leitet diese weiter an den PDP. Die Anfrage vom Backend hat einen genauen vorgegebenen Aufbau, welcher im nächsten Abschnitt im Policy Decision Point erläutert wird. Um konsistent zu bleiben, werden auch bei einer Anfrage, die nur eine Ressource umfasst, dieses ebenfalls in einer Liste verschickt. Nachdem im PDP die Entscheidung gefällt wurde, schickt dieser die Liste mit den Ressourcen, für die die Aktion durchgeführt werden darf, zurück an

den PEP. Falls die erhaltene Liste leer ist, also der Nutzer nicht die Rechte hat die Anfrage durchzuführen schickt der PEP an das Backend den Statuscode 403. Falls die Liste nicht leer ist, wird die Liste ans Backend weitergeleitet mit dem Statuscode 200.

4.5.2 Policy Decision Point

Der Policy Decision Point ist der nächste Dienst in der Kette. Die Funktion dieses Dienstes ist es die umformulierten Anfragen vom PEP zu empfangen und dann diese Anfrage zu verstehe und sich alle benötigten Informationen dazu zu besorgen und mithilfe von diesen dann zu entscheiden, ob die Anfrage vom Nutzer erlaubt bzw. abgelehnt wird.

Nach dem Empfangen der Anfrage werden dort alle weiteren Informationen gesammelt und entschieden. Die Anfrage liefert im Body mehrere vorgegebene Informationen mit. Zu diesen Informationen gehört zu einem ein Tool, welches als String mitgeschickt wird. Dazu kommen noch die UserID und die Aktion, welche durchgeführt wird. Beides ebenfalls als String. Die Aktion kann aus „READ“, „WRITE“ oder „DELETE“ bestehen. Die letzte Information sind die Ressourcen bzw. die Ressource, welche bei der Anfrage benötigt werden. Diese werden als Liste übergeben. Nachdem die Anfrage erhalten wurde, muss sich der PDP einige Informationen besorgen. Zu einem werden weitere Daten zu dem Nutzer benötigt, wie zum Beispiel seine Rolle oder auch die Gruppen, in denen er Mitglied ist. Um diese Informationen zu erhalten, fragt der PDP den PIP an. Mithilfe von diesen Daten kann der PDP dann den PRP anfragen, um alle Policies zu erhalten die die Ressource, NutzerID oder Gruppen beinhalten. Zu Schluss muss der PDP entscheiden, ob die Anfrage durchgeführt wird. Dazu geht er die Liste mit den Ressourcen durch und überprüft, ob es eine Policy gibt die die gewünschte Aktion für die Ressource erlaubt. Falls dies der Fall ist, wird die Ressource in eine Liste gepackt, für alle Ressourcen, bei denen die Anfrage erlaubt wird. Nachdem er die Liste mit Ressourcen überprüft hat, wird die Liste mit den erlaubten Ressourcen zurück an das PEP geschickt.

4.5.3 Policy Information Point

Der Policy Information Point ist die primäre Quelle für zusätzliche Informationen und Attributen. Dafür sammelt der PIP Informationen durch Anfragen an außenstehende Systeme, wie z.B. Keycloak in unserem Fall für weiter Nutzerinformationen.

In dem Fall dieses Konzepts wird Keycloak das externe System sein. Keycloak ist für die Benutzerverwaltung und die Zuweisung von Nutzern zu Gruppen zuständig. Der Policy Information Point wird mit dem Empfangen einer Anfrage vom PDP die NutzerID mitgeschickt bekommen. Die NutzerID ist die ID, welche von Keycloak für jeden Nutzer angelegt wird, welche bei jeder Anfrage mitgeschickt wird. Mithilfe von der NutzerID ist der PIP in der Lage Keycloak anzufragen, um weitere Informationen von dem Nutzer zu erhalten. In unserem Fall sind die Gruppen, in denen der Nutzer ist und seine Rolle relevant. All diese Informationen werden in einem JSON-Objekt zusammengefasst und dann zum PDP zurückgeschickt.

4.5.4 Policy Retrieval Point

Der Policy Retrieval Point ist die Datenbank für die angelegten Policies zu den Nutzern, Rollen und Gruppen.

Der Dienst hat somit die Aufgabe die Policies zu speichern und rauszusuchen. Dazu werden angelegte Policies in einer Datenbank gespeichert. Policies haben folgende Attribute Tool, Ressource, Aktion, Nutzer und Gruppen, wie in dem Kapitel „Aufbau von Policies“ beschrieben wird. Bei einer Anfrage vom PDP werden dann all diese Attribute in einem JSON-Objekt mitgeschickt. Daraufhin werden in der Datenbank alle Policies rausgesucht die ebenfalls dieselben Werte haben. Falls eine passende Policy gefunden wurde, wird diese zurückgeschickt an den PDP. Es gibt aber noch einen anderen Fall einer Anfrage, nämlich wenn ein Nutzer eine neue Policy erstellen möchte bzw. eine bestehende Policy bearbeiten oder löschen möchte. Diese Anfrage würde vom PAP kommen und die gleichen Attribute beinhalten wie die Anfrage vom PDP. Im Falle vom Bearbeiten oder Löschen wird der PRP wird dann zuerst überprüfen, ob eine Policy vorhanden ist, die dem Nutzer erlaubt ein „ADMIN“-Anfrage durchzuführen. Falls dies der Fall ist, wird die Aktion dementsprechend durchgeführt. Nachdem dies durchgeführt wurde, schickt der PRP ein Statuscode 200 an den PAP zurück.

4.5.5 Policy Adminstration Point

Der Policy Adminstration Point ist die Komponente, welche dafür da ist, Policies zu erstellen, verwalten und zu definieren. Dieser Dienst wird beansprucht, bei der Erstellung, Löschung und Bearbeitung einer Policy

Der Policy Adminstration Point wäre die zweite Schnittstelle mit dem Backend vom Tool. In dem Fall, dass ein Nutzer zu einer Ressource im Tool eine bestehende Policy bearbeiten, löschen oder eine neue erstellen möchte, wird die Anfrage vom Backend direkt an den PAP geschickt. Die Anfrage beinhaltet die Informationen von einer Policy. Dieses JSON-Objekt wird an den PRP geleitet, wo dann dementsprechend gehandelt wird. Nachdem dies vom PRP durchgeführt wurde, erhält der PAP ein Statuscode, welcher er an das Backend weiterleitet.

4.6 Keycloak

Für die schon erstellten Tools in der Abteilung Produktionsmanagement des Werkzeugmaschinenlabors WZL der RWTH Aachen wird Keycloak als Software benutzt, um die User und deren Zugriff für die Tools zu managen.

Keycloak selbst ist eine Open-Source Software und basiert auf OAuth und OpenID Connect und verwendet für die Authentifizierung und Autorisierung JSON Web Tokens. Die Software läuft auf einem eigenständigen Server getrennt von der Anwendung, welche es schützt. Als Hauptfunktion bietet Keycloak Single Sign-On an, welches ermöglicht Benutzer sich mit einem einzigen Login bei mehreren Anwendungen anzumelden. Dazu hat man eine zentrale Benutzerverwaltung und eine sichere Zugriffsverwaltung. In der Benutzerverwaltung hat man die Möglichkeiten Benutzerkonten zu verwalten und die Authentifizierung und Zugriffskontrolle zu bearbeiten. Außerdem kann man mehrere Benutzer auch in Gruppen zusammenfassen und man hat die Möglichkeit einer Gruppen Child-Groups zu geben, wodurch man ein gewisse Gruppenhierarchie aufbauen kann.

In dem Konzept wird man per Anfrage an Keycloak die Informationen vom Nutzer und somit die Gruppen erhalten, in denen er ist, woraufhin man die dazu passenden Policies anfragen kann.

Damit man die Policies nicht für jede einzelne Person anlegen muss, nutzt man die Funktion in Keycloak mehrere Nutzer in Gruppen zusammenzufassen. Dadurch kann man Policies erstellen, die für alle Gruppenmitglieder einer Gruppe gelten. Im Kontext von dem Konzept und der Abteilung Produktionsmanagement wäre es dann so, dass Nutzer X beim Erstellen in Keycloak direkt in mehrere Gruppen hinzugefügt wird. Darunter würde dann die Abteilung „Produktionsmanagement“ und seine Arbeitsgruppe „Produktmanagment“ zählen. Da seine Arbeitsgruppe eine Child-Group von der Abteilung ist, würde man den Nutzer nur dieser hinzufügen und er würde automatisch auch Teil der Parent-Groups werden, woraufhin der Nutzer alle Rechte dieser zwei Gruppen erhalten würde.

4.7 Einbau vom System in bestehende Tools

Dadurch, dass das System nur im Backend als Hilfstool eingesetzt wird, müssen Anfragen vom Frontend ans Backend in bestehenden Tools nicht verändert werden. Aktuell werden in bestehenden Tools der Abteilung Produktionsmanagement die Rechte für die jeweiligen Ressourcen in der Datenbank mit abgespeichert und bei Anfragen vom Frontend ans Backend muss dort innerhalb der Funktionen überprüft werden, ob der jeweilige Nutzer auch Rechte hat. Diese Funktion zum Überprüfen soll dann vom System übernommen werden. Dafür muss man dann dem PEP eine Anfrage schicken mit der NutzerID, die Aktion, die der Nutzer ausführen will, einer Liste mit den Ressourcen, die die Anfrage beinhaltet und das Tool. Als Antwort bekommt das Backend die Liste mit Ressourcen, für welcher der Nutzer die Anfrage durchführen darf. Es gibt noch den Fall in dem ein Nutzer auch Policies erstellen bzw. löschen oder bearbeiten möchte. Diese Anfrage würde direkt an den PAP gehen und dort durchgeführt, welcher dann mit einem Statuscode antwortet.

5 Umsetzung und Implementierung

In dem folgenden Kapitel werden die Umsetzung und die Implementierung von dem bereits erklärten Konzept genau erläutert.

5.1 Verwendete Technologien

In den Tools von der Abteilung Produktionsmanagement wird im Backend immer Spring-Boot als Framework verwendet. Da sich Spring-boot für Microservices eignet und um die Struktur einheitlich zu behalten, wird somit Spring-Boot ebenfalls in diesem Konzept für die verschiedenen Services verwendet.

5.2 Spring-Boot

Spring-Boot ist ein Java-Framework, welches die Entwicklung von Anwendungen vereinfacht, durch das Übernehmen von verschiedenen Aufgaben und voreingestellten Konfigurationen. Wie schon erwähnt wird bei der Erstellung von einem Projekt schon einige Voreinstellungen vorgenommen, wodurch der Einstieg in ein neues Project erleichtert wird und die Notwendigkeit von manuellen Konfigurationen minimiert wird. Außerdem bietet Spring einen eingebetteten Webserver, wie Tomcat, Jetty oder Undertow, wodurch man keinen externen Server bereitstellen muss. All diese Vorteile spricht auch für Spring bei der Entwicklung von Microservices, da man somit schnell und einfach verschiedene kleine startbereite Dienste programmieren kann.

5.3 Der Aufbau der Microservices

Die Services folgen einer einheitlichen Struktur, die aus Controllern, Services, Klassen und gegebenenfalls Repositories besteht.

- **Controller**: Der Controller dient dazu, eingehende HTTP-Anfragen zu empfangen, die mitgelieferten Daten auszulesen und die Anfragen zu verarbeiten. Dabei delegiert er die eigentliche Logik an den entsprechenden Service.
- **Service**: Der Service kapselt die Funktionalität des Controllers und übernimmt die Verarbeitung der Anfrage. Der Controller ruft lediglich die passende Funktion im Service auf, um die Anfrage zu bearbeiten.

- **Klassen**: Klassen werden verwendet, um Daten einheitlich zu strukturieren und zu formatieren. Sie sorgen für Konsistenz und erleichtern die Datenverarbeitung.
- **Repositories:** Zusätzlich kommen Repositories zum Einsatz, die für die Interaktion mit der Datenbank verantwortlich sind. Sie ermöglichen das Erstellen, Bearbeiten und Löschen von Daten in der Datenbank.

Diese Architektur stellt sicher, dass die Verantwortlichkeiten klar getrennt sind und der Code wartungsfreundlich bleibt.

5.4 Implementation PEP

Die Aufgabe vom PEP ist simpel gehalten. Der Controller vom PEP hat eine Methode, um die http-Anfrage vom Backend zu empfangen. In dieser Anfrage werden per RequestBody die Daten zu der Erlaubnisanfrage per JSON-Objekt, der Klasse RequestData mitgeschickt.

Die Klasse RequestData hat folgende Attribute:

1. **Tool (String)**: Spezifiziert das verwendete Tool.
2. **userId (String)**: Die ID des Benutzers, der die Anfrage stellt.
3. **ressource (List<String>)**: Eine Liste von Strings, die die Ressourcen beschreibt, auf die zugegriffen wird.
4. **action (String)**: Definiert die Art der Aktion, die mit der Anfrage ausgeführt werden soll.

Im Service werden die mitgeschickten Daten geprüft. Bei der Prüfung wird kontrolliert, ob jedes Attribut gesetzt ist und ob die „action“ auch entweder „GET“, „POST“, „PUT“ oder „DELETE“ als Wert hat. Nach den Überprüfungen schickt der PEP eine http-Anfrage an den PDP, welche als Antwort eine List mit den erlaubten Ressourcen für die Anfrage zurückschickt.

5.5 Implementation PDP

Der PDP ist der zentrale Service in diesem Konzept und ist für die Kommunikation zwischen den verschiedenen Diensten verantwortlich. Dafür muss der Microservice

nach dem Empfangen der http-Anfrage des PEPS, weitere http-Anfragen and den PIP und PRP senden. Der PDP selbst empfängt im Controller nur die Get-Anfrage vom PEP, in welchem im Body mehrere Informationen mitschickt. Der Request-Body enthält ein JSON-Objekt, der Klasse RequestData.

Der PDP schickt danach eine http-Anfrage an den PIP. Für diese Anfrage liest der PDP die UserId aus der RequestData und schickt dieser in Header mit an der PIP, welcher dann ein JSON-Objekt, der Klasse User zurückschickt.

Diese Klasse hat folgende Attribute:

1. **userId (String)**: Eine Zeichenkette, die die eindeutige Identifikation eines Benutzers darstellt.
2. **Role (String)**: Eine Zeichenkette, die die Rolle des Benutzers im System beschreibt (z. B. Administrator oder Nutzer).
3. **groups**: Eine Liste von Zeichenketten, die die Gruppen angibt, denen der Benutzer angehört.

Falls diese Anfrage keinen Nutzer zurückschickt, wird die Entscheidung vom PDP direkt abgelehnt. Zu Beginn wird nun überprüft, ob der Nutzer die Rolle „admin“ hat. In diesem Fall werden alle angefragten Aktionen erlaubt. Im Fall der Rolle „user“ wird für jede Ressourse in der RequestData, eine Anfrage an den PRP geschickt, welcher überprüft, ob eine passende Policy in der Datenbank vorhanden ist und in diesem Fall diese dem PDP zurückschickt. Nach der Überprüfung von allen Ressourcen wird an den PEP eine Liste mit den erlaubten Ressourcen zurückgeschickt.

5.6 Implementation PIP

Der PIP ist der erste von zwei Microservices, der dem PDP auf Anfrage die benötigten Daten bereitstellt. Dabei ruft der PIP die Informationen zum Nutzer über eine Anfrage an Keycloak ab. Die Verbindung zu Keycloak wird mithilfe eines KeycloakBuilders hergestellt, der durch Angabe von „KeycloakUrl“, „KeycloakRealm“, „KeycloakClient“ und „KeycloakClientSecret“ konfiguriert wird. Diese Parameter müssen manuell gesetzt werden. Der KeycloakBuilder wird durch das Hinzufügen der Dependency „keycloak“ in das Projekt eingebunden.

Im Controller des PIP empfängt der Service die Anfrage vom PDP, welche die UserId als PathVariable übermittelt. Mithilfe der UserId und der bestehenden Keycloak-Verbindung können zusätzliche Informationen zum Nutzer abgerufen werden, wie beispielsweise Rollen und Gruppen. Diese Daten werden als JSON-Objekt, basierend auf der Klasse **User** (wie in Kapitel 6.4 beschrieben), an den PDP zurückgegeben.

5.7 Implementation PAP

Der PAP ist die zweite Schnittstelle mit dem Backend vom jeweiligen Tool. Im Controller sind drei verschiedene Methoden, für die alle möglichen Anfragen vom Backend, jeweils eine Methode für die HTTP-Methoden **PUT**, **POST** und **DELETE**.

- **POST-Methode**: Diese Methode dient dazu, eine neue Policy zu erstellen, bei einer Erstellung einer neuen Ressource
- **DELETE-Methode**: Diese Methode verwendet, falls eine Ressource gelöscht wurde und somit auch die Policies gelöscht werden können.
- **PUT-Methode**: Die Methode wird verwendet, um eine bestehende Policy zu bearbeiten.

Alle drei Methoden erwarten im RequestBody ein Policy. Im Service wird daraufhin eine weitere http-Anfrage an den PRP gemacht, welcher dann die Policy dementsprechend in der Datenbank bearbeitet.

5.8 Implementation PRP

Der PRP ist der einzige Microservice, welcher auch ein Zugang zu seiner Datenbank für die verschiedenen Policies benötigt. Dazu wurde die „postgressql“ dependency installiert, wodurch man direkt eine Verbindung zu seiner postgres-Datenbank aufbauen kann. Die Datenbank selbst benötigt nur drei zusammenhängende Tabellen. Dazu muss man drei Entity-Klassen schreiben. Die Klassen, welcher mit der „@Entity“-Anotation gekennzeichnet werden auch in der Datenbank als Tabelle dargestellt und jedes Attribut dieser Klasse entspricht einer Spalte in einem relationalen

Datenbanksystem.
Die Entity-Klasse Policy repräsentiert eine Tabelle in der Datenbank, die Benutzerinformationen speichert. Sie enthält folgende Attribute:

1. **id** (Long): Eindeutige Identifikationsnummer der Policy (Primärschlüssel).
2. **tool** (String): Ein Tool, auf welches sich die Policy bezieht.
3. **ressource** (String): Die Ressource, auf welche sich die Policy in dem Tool bezieht.
4. **action** (String): Die Action, beschreibt welche Aktion mit der Policy erlaubt wird (besteht aus „GET“, „POST“, „DELETE“ & „ADMIN“).
5. **user** (List<User>): Die User, ist eine Liste aus Strings mit den UserId´s, für welche die Policy etwas erlaubt.
6. **group** (List<Group>): Die Group, ist eine Liste aus Strings mit den Gruppennamen, für welche die Policy etwas erlaubt.

Die Attribute „user“ und „group“ sind Listen aus 2 weiteren Entity Klassen. Die Klasse User hat nur das eine Attribut:

1. **id** (Long): Die UserID (Primärschlüssel).

Die Klasse Group hat folgendes Attribut:

1. **id** (Long): Die ID der Gruppe (Primärschlüssel).

Somit bildet sich die Datenbank zusammen aus diesen drei Tabellen. Dadurch das die Attribute „user“ und „group“ von der Policy Klasse jeweils eine „@JoinColumn“-Annotation haben wird in den zwei Tabellen „user“ und „group“ auch eine Spalte mit der Policy-Id hinzugefügt, wodurch man sie zu einer Policy zuordnen kann. Um den Microservice anzusprechen muss noch ein Controller geschrieben werden, welcher dann noch den passenden Service und Repository anspricht.

Der Policy-Controller umfasst drei verschiedene Methoden, die eingehende HTTP-Anfragen verarbeiten: jeweils eine Methode für die HTTP-Methoden **GET**, **POST** und **DELETE**.

- **GET-Methode**: Diese Methode erwartet im Request-Body eine Policy ohne ID. Sie wird aufgerufen, wenn der Policy Decision Point (PDP) alle relevanten

Informationen zu einer Anfrage besitzt und prüfen möchte, ob eine passende Policy vorhanden ist. Falls eine Policy vorhanden ist, wird diese zurückgeschickt

- **POST-Methode**: Diese Methode dient dazu, eine neue Policy zu erstellen. Sie wird vom Policy Administration Point (PAP) genutzt, wenn eine neue Policy hinzugefügt.
- **DELETE-Methode**: Diese Methode wird verwendet, um eine Policy zu löschen.
- **PUT-Methode**: Diese Methode wird verwendet, um eine bestehende Policy zu bearbeiten.

Durch diese vier Methoden stellt der Controller sicher, dass die notwendigen Funktionen zur Verwaltung von Policies effizient umgesetzt werden. Bei der DELETE- und PUT-Methode muss zuerst überprüft werden, ob der Nutzer auch eine bestehende Policy hat, welche ihm „ADMIN“-Rechte gibt für diese Policy. Die Umsetzung der Funktionalität der Methoden wird ausgelagert in den Service. Dort gibt es dann für jede Methode die erwünschten Funktionalitäten umgesetzt mithilfe von dem Repository, welches in der Datenbank mithilfe von automatisch erstellten SQL-Anfragen die Einträge raussucht.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein umfassendes Konzept für die zentrale Verwaltung von Zugriffsrechten auf Basis einer Microservice-Architektur entwickelt und implementiert. Das vorgestellte System bietet eine flexible und wartungsfreundliche Grundlage für die zentrale Verwaltung von Policies und deren Integration in bestehende Tools der Abteilung Produktionsmanagement.

Durch die verwendeten Techniken und die gewählte Architektur wurden zentrale Anforderungen wie Skalierbarkeit und Fehlertoleranz erfüllt. Die modulare Struktur des Systems ermöglicht darüber hinaus eine einfache Erweiterbarkeit, wodurch das System langfristig anpassbar bleibt.

In der Zukunft sind mehrere Erweiterungsmöglichkeiten denkbar. Dazu gehört die Entwicklung einer Benutzeroberfläche für eine effiziente Übersicht über alle erstellten Policies sowie deren Verwaltung. Ebenso sollte eine Performance-Analyse mit einer

großen Anzahl von Nutzern und komplexen Anwendungsfällen durchgeführt werden, um die Belastbarkeit des Systems zu prüfen. Zusätzlich wäre es sinnvoll, einen Service-Discovery-Test zu integrieren, um sicherzustellen, dass die dynamische Erkennung und Kommunikation zwischen den Microservices unter realistischen Bedingungen zuverlässig funktioniert.

Zusammenfassend kann gesagt werden, dass die Ergebnisse dieser Arbeit eine stabile Grundlage für den praktischen Einsatz des Systems in der Abteilung bieten. Gleichzeitig zeigen sie Wege und Potenziale auf, wie das Konzept zukünftig weiterentwickelt werden kann, um den steigenden Anforderungen an moderne IT-Systeme gerecht zu werden.

7 Literaturverzeichnis

[1] E. Wolff, Microservices: Ein Überblick, dpunkt, 2018.

[2] A. Goldbeck, *Ist neu immer besser? Vergleich der Qualität von Microservice Architektur und SOA,* Doktorarbeit: Campus 02 Fachhochschule der Wirtschaft, 2017.

[3] O. Al-Debagy and P. Martinek, A comparative review of microservices and monolithic architectures, International Symposium on Computational Intelligence and Informatics (CINTI), 2018.

[4] Nuha, Alshuqayran, A. Nour and R. Evans, *A systematic mapping study in microservice architecture,* IEEE 9th international conference on service-oriented computing and applications, 2016.