Entwicklung von Microservices in der SAP Cloud Platform

Alte Welt, neue Welt

Dieser Artikel stellt die SAP Cloud Platform und darin angebotene Services überblicksartig vor. Anhand einer Beispielanwendung geht er außerdem detaillierter auf die Entwicklung von Microservices-basierten Java-Anwendungen in der SAP Cloud Platform ein.

von Herbert Höbarth

Das Gartner-Modell der bimodalen IT versucht, das Bewährte um Neues und Innovatives zu erweitern. Das Bewährte (Mode 1) ist häufig gleichzusetzen mit einem Legacy-System, während das Innovative (Mode 2) neue Technologien wie Cloud, IoT, Machine Learning oder Blockchain repräsentiert. Durch einen bimodalen IT-Ansatz in der SAP-Welt sollen SAP-Systeme als wichtiges Rückgrat für digitale Geschäftsmodelle und differenzierende Innovationen mit der „neuen Welt" prozessual verbunden werden, um somit für mehr Hybridität und Agilität zu sorgen. Die SAP Cloud Platform kann dabei als Werkzeug dienen, um die Datenströme entlang der Prozessketten und über Unternehmensgrenzen hinweg zu realisieren.

Die SAP Cloud Platform [1] ist eine Enterprise-Platform-as-a-Service-Lösung (PaaS) für die Entwicklung, Erweiterung und Integration von Geschäftsanwendungen in der Cloud und fungiert als die innovative Cloud-Development- und Deployment-Plattform der SAP. Ein sehr guter Startpunkt für einen tieferen Einstieg in die Plattform ist die offizielle Dokumentation [2].

Die Plattform kann sowohl auf SAP-eigenen Rechenzentren als auch auf jenen der IaaS-Partner Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure und Alibaba Cloud betrieben werden.

Die Plattform bietet drei unterschiedliche Umgebungen an, die die Entwicklung und Administration von Geschäftsanwendungen erlauben, nämlich ABAP, Neo und Cloud Foundry.

ABAP

Die ABAP-Umgebung oder Steampunk basiert auf der Cloud-Foundry-Umgebung und ermöglicht die Erweiterung von ABAP-basierten Produkten wie der SAP S/4HANA Cloud bzw. die Transformation bereits existierender ABAP-Programme oder -Erweiterungen in die Cloud. Der Softwarestack der ABAP-Umgebung beinhaltet SAP-Standardtechnologien wie das RESTful-ABAP-Programmiermodell mit SAP Fiori und Core Data Services (CDS).

Neo

Die Neo-Umgebung erlaubt die Entwicklung von Anwendungen mit HTML5 inklusive dem UI-Development-Toolkit SAPUI5, Java und den SAP HANA Extended Application Services (SAP HANA XS). Außerdem ermöglicht die Neo-Umgebung die Verwendung von Virtual Machines, um eigene Applikationen in Szenarien zu betreiben, die nicht durch die Umgebung unterstützt werden. Da die Neo-Umgebung eine proprietäre ist, kann diese nur auf SAP-eigenen Rechenzentren betrieben werden.

Cloud Foundry

Die Cloud-Foundry-Umgebung ist im Gegensatz zur Neo-Umgebung keine proprietäre, sondern basiert auf der Open-Source-Anwendungsplattform der Cloud Foundry Foundation [3] und erlaubt die Verwendung unterschiedlicher Programmiersprachen wie Java, JavaScript und anderer. SAP empfiehlt die Verwendung der Cloud-Foundry-Umgebung vor allem für die Entwicklung von Microservices-basierter und/oder 12-Factor-Anwendungen sowie für IoT- und ML-Szenarien. Für technische Grundlagen zu 12-Factor-Apps und der Cloud-Foundry-Umgebung der SAP Cloud Platform sei auf [4] verwiesen.

Services in der SAP Cloud Platform

Die SAP Cloud Platform bietet eine breite Palette an Services für unterschiedliche Einsatzszenarien an. Das Pricing für Services erfolgt entweder subscription-based oder consumption-based und kann unter [1] einfach und transparent simuliert werden. Nicht alle Services sind auf allen Umgebungen bzw. Datenzentren verfügbar.

Während manche Services wie Git oder die SAP Web IDE ausschließlich in der Neo-Umgebung verfügbar sind, findet man andere wie SAP Leonardo IoT oder Hyperledger Fabric wiederum nur in der Cloud-Foundry-Umgebung. Einen Überblick über alle angebotenen Services kann man sich in der offiziellen Dokumentation [2] verschaffen.

Grundsätzlich sind die Services in verschiedene Kategorien wie AI und Machine Learning, Analytics, Blockchain, Mobile oder User Management untergliedert. Für die Entwicklung und die Administration von Anwendungen in der Cloud sind vor allem die Kategorien DevOps und Developer Experience interessant. Hier finden sich Services wie Git, Java Debugging, SAP Web IDE, Java Profiling oder Application Logging. Für die Connectivity zwischen Cloudanwendungen oder die Anbindung von On-Premise-Systemen bietet die Kategorie Integration hilfreiche und notwendige Services wie Connectivity oder Destination. Will man innerhalb einer Anwendung wiederum den Cloud Foundry User Account and Authentication (UAA) Service verwenden, wird man in der Kategorie Data Privacy and Security fündig (Service Authorization and Trust Management).

Architektur einer Cloud-Foundry-Anwendung

Im Rahmen des vorliegenden Artikels soll nun detaillierter auf die Entwicklung von Anwendungen in der Cloud-Foundry-Umgebung der SAP Cloud Platform eingegangen werden. An dieser Stelle sei auf die stets wachsende Tutorialsammlung der SAP unter [5] hingewiesen.

Die Cloud-Foundry-Umgebung ist ein offenes PaaS, das speziell für die Entwicklung und Orchestrierung von Microservices entworfen wurde. Im Allgemeinen unterstützt die Umgebung

- die Entwicklung von polyglotten Anwendungen basierend auf Cloud Foundry Buildpacks.
- die Verwaltung der Lifecycles von Anwendungen durch den Einsatz von Cloud-Foundry-Tools oder des SAP Cloud Platform Cockpits.
- die Optimierung der Anwendungsentwicklung und des Betriebs durch das Binden an ein umfangreiches Set an Ready-to-use Services der SAP Cloud Platform wie Messaging oder Persistenz.
- das laufzeitunabhängige Konsumieren dieser Services.

Für die Entwicklung von Geschäftsanwendungen in der Cloud-Foundry-Umgebung bietet SAP eine Best-Practice-Architektur (**Abb. 1**) an.

Die Hauptkomponenten innerhalb dieser Architektur sind Microservices, Service-Instanzen und Routen. Sobald Microservices in Form von Code und Binaries in die Umgebung gepusht werden, werden mehrere Anwendungsinstanzen erzeugt, wobei jede in einem separaten Container läuft. Services werden in Form von Service-Instanzen an eine Anwendung gebunden. Diese Instanzen beinhalten die Konfiguration und, wenn nötig, die Credentials, um einen Service innerhalb der

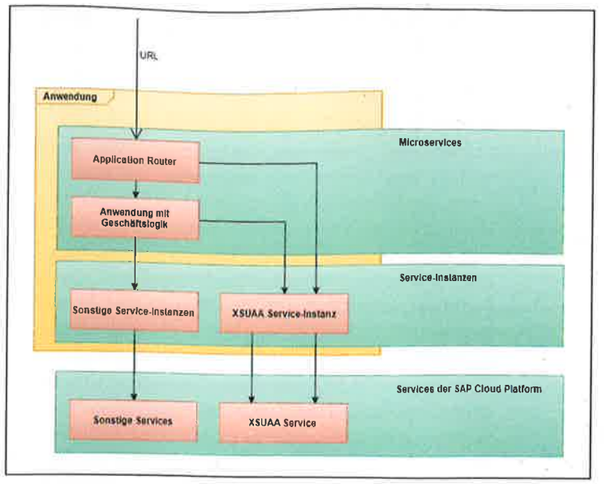

Abb. 1: Architektur für die Entwicklung von Geschäftsanwendungen in der Cloud-Foundry-Umgebung

Anwendung verwenden zu können. Service-Instanzen werden durch einen Service Broker verwaltet. Routen wiederum definieren die Zugangspunkte bzw. URLs zu einer Geschäftsanwendung. Eine Geschäftsanwendung stellt eine Kombination aus Microservices, Service-Instanzen, Service Bindings und Routen dar und kann beispielsweise durch Verwendung des Cloud Foundry CLI durch den Cloud Foundry Controller in die Cloud gepusht werden.

Um Herausforderungen wie Security und Same Origin Policy in der oben gezeigten Microservices-basierten Architektur zu adressieren, bietet die SAP Cloud Platform eine spezielle Komponente an, nämlich den Application Router. Dieser dient in der Form eines Reverse Proxy als zentraler Einstiegspunkt in eine Geschäftsanwendung und leitet sämtliche Anfragen an dahinterliegende Services weiter. Außerdem beinhaltet er Security-Funktionalitäten wie Login/Logout, Benutzerauthentifizierung und -autorisierung und Schutz vor CSRF. Dazu benötigt der Application Router ein Binding an die SAP-Erweiterung des Cloud Foundry User Account and Authentication Service, nämlich XSUAA. Dieser fungiert innerhalb der SAP Cloud Platform als OAuth2-Provider, der Benutzer gegen die Cloud-Foundry-Umgebung authentifizieren und in weiterer Folge als SSO-Service dienen kann. Der Application Router ist eine Node.js-Applikation, die mittels Verwendung der SAP-eigenen npm Registry *https://npm.sap.com/* installiert werden kann. Details zum Einsatz des Application Routers können in der offiziellen Dokumentation [2] nachgelesen werden. Die Beispielanwendung zu diesem Artikel, die im Folgenden noch erklärt wird, beinhaltet ebenfalls zwei Application Routers.

Java-Entwicklung in der Cloud-Foundry-Umgebung

Für die Entwicklung von Java-Anwendungen in der Cloud-Foundry-Umgebung der SAP Cloud Platform stellt SAP das SAP Java Buildpack zur Verfügung. Die-

ses Buildpack basiert auf der SAP JVM und beinhaltet vier Application Runtimes:

- Tomcat: Apache Tomcat 8
- TomEE: Apache TomEE JAX-RS (Java EE 6)
- TomEE 7: Apache TomEE 7 (Java EE 7 Web Profile)
- Main: Java-Anwendung mit eigener Laufzeitumgebung (z. B. Spring Boot)

Als Entwicklungsumgebung für die Entwicklung von Java-Anwendungen in der Cloud-Foundry-Umgebung bietet sich die SAP Web IDE Full-Stack an, die als Service in der Neo-Umgebung läuft. Die SAP Web IDE unterstützt den Entwickler beim Develop, Build, Run, Test, Deploy und Debug einer Anwendung durch wiederverwendbare Templates und Integration diverser Builder wie Ant oder Maven und anderer Tools wie Git oder SSH. Ein besonderes Feature ist das Hybrid Application Toolkit, das eine einfache, Wizard-getriebene Entwicklung von Apache-Cordova-Apps ermöglicht. Die Web IDE wird in der ersten Hälfte 2020 durch das auf Eclipse Theia basierende SAP Business Application Studio abgelöst werden, das bereits in einer Betaversion in der Cloud-Foundry-Umgebung verfügbar ist.

Alternativ zur SAP Web IDE bzw. dem SAP Business Application Studio können natürlich bereits bekannte Entwicklungsumgebungen verwendet werden, im Falle von Eclipse idealerweise mit den Eclipse Tools for Cloud Foundry [6], die das Arbeiten mit Cloud Foundry sehr gut unterstützen. Anwendungen lassen sich allerdings auch mit dem Cloud Foundry CLI deployen und verwalten und mittels SSH Port Forwarding remote debuggen.

Das SAP Java Buildpack beinhaltet bereits Libraries für Logging und Tracing von Anwendungen. Anwendungen können somit nach Hinzufügen des *com.sap.hcp.cf.logging.servlet.filter.RequestLoggingFilter* in der *web.xml* out of the box Logs im JSON-Format produzieren, die in weiterer Folge von Kibana geparst werden können. Application-Logs können wie gewohnt mittels SLF4J erzeugt und nach dem Binden einer Service-Instanz des Application Logging Service im Cloud Cockpit der SAP Cloud Platform analysiert werden (siehe Beispielanwendung).

Auch für die Verwendung des XSUAA Service und die Validierung der empfangenen JWT-Tokens für die Authentifizierung und Autorisierung von Benutzern bringt das SAP Java Buildpack bereits vorgefertigte Komponenten mit. Die Authentifizierungsmethode XSUAA lässt sich in der *web.xml* konfigurieren (siehe nachfolgendes Beispiel). Die Definition von Security Constraints innerhalb von Servlets kann dann wie gewohnt mittels Annotationen des Servlet API wie *@ServletSecurity* und *@HttpConstraint* erfolgen (siehe Microservice RfcServices in der Beispielanwendung).

```
<login-config>
  <auth-method>XSUAA</auth-method>
</login-config>
```

Auch für Spring-Boot-Anwendungen bietet SAP ein Spring-Security-Modul für die Validierung der vom XSUAA-Service empfangenen JWT-Tokens an. Dazu kann etwa der Spring Boot Starter *com.sap.cloud.security.xsuaa:xsuaa-spring-boot-starter* verwendet werden. Ein Beispiel für eine Security-Konfigurationsklasse findet sich in der Beispielanwendung innerhalb des Spring Boot Microservice DocServices.

Für die effiziente Entwicklung von Geschäftsanwendungen auf der SAP Cloud Platform bietet SAP das SAP Cloud SDK für Java und JavaScript an [7]. Das SAP Cloud SDK für Java beinhaltet sowohl Third Party Frameworks wie Resilience4J und Caffeine als auch ein umfangreiches Set von Komponenten, die eine cloudnative Entwicklung in der SAP Cloud Platform unterstützen. So findet man für die folgenden ausgewählten Funktionen Unterstützung innerhalb des SAP Cloud SDKs:

- Auffinden und Verwendung von Destinations und Services der SAP Cloud Platform (siehe Microservice RfcServices in der Beispielanwendung)
- Auffinden und Verwendung von RFC-Bausteinen auf SAP Backend-Systemen (siehe Microservice RfcServices in der Beispielanwendung)
- Generierung von typsicheren Clientklassen für den Aufruf von OData Services oder BAPIs mit dem Java Virtual Data Model (VDM)
- Einfacher Einsatz von Caches für die Verbesserung von Antwortzeiten (siehe Microservice RfcServices in der Beispielanwendung)
- Unterstützung von Continuous Integration and Delivery
- Einfache Integration von Blockchain Services (MultiChain und Hyperledger Fabric) und SAP Leonardo Machine Learning Services

Für die einzelnen Application Runtimes können Maven Archetypes (z. B. *com.sap.cloud.sdk.archetypes:scp-cf-tomee* oder *com.sap.cloud.sdk.archetypes:scp-cf-spring*) verwendet werden, um sich eine Projektstruktur generieren zu lassen.

Um bereits bekannte SAP-Technologien auch für die Entwicklung von cloudnativen Java- bzw. Node.js-Anwendungen verwenden zu können, entwickelte SAP das SAP Cloud Application Programming Model, das ebenfalls das SAP Cloud SDK verwendet. Dieses Modell dient der End-to-end-Programmierung von Geschäftsanwendungen und baut auf SAP-Technologien wie CDS für die Definition von Datenmodellen und SAPUI5 für die UI-Entwicklung auf.

Zugriff auf das SAP Backend

Das SAP Java Buildpack beinhaltet auch den SAP Java Connector [8], der eine Kommunikation zwischen Java-Anwendungen und On-Premise-SAP-Systemen über das SAP-proprietäre RFC-Protokoll erlaubt. Mit dem Binden der Cloudanwendung an Service-Instanzen der Services Destination und Connectivity werden alle notwendigen Bibliotheken für die Verwendung des SAP JCo geladen. Die Verwendung des SAP JCo ist innerhalb von Spring-Boot-Anwendungen nicht möglich.

Für den einfachen Zugriff auf On-Premise-SAP-Systeme, ohne diese nach außen öffnen zu müssen, empfiehlt sich die Verwendung des SAP Cloud Connectors [9]. Dieser wird im eigenen Netzwerk installiert und verbindet sich über ein Secure VPN mit der SAP Cloud Platform. Mit dem SAP Cloud Connector kann nicht nur auf HTML-Schnittstellen am Backend zugegriffen werden, sondern auch auf RFC-Bausteine. Der Zugriff auf bestimmte URLs und Services bzw. Funktionsbausteine wird dabei einzeln freigegeben. Zusätzlich ermöglicht der SAP Cloud Connector SSO zwischen der SAP Cloud Platform und den angebundenen SAP-Systemen.

Im Tutorial Navigator der SAP [5] sind mehrere Tutorials zur Installation und Konfiguration des SAP Cloud Connectors zu finden, auch zur Verwendung des Application Routers zur sicheren Anbindung von SAP-Backend-Systemen.

Die Beispielanwendung

Die Architektur der Beispielanwendung zu diesem Artikel ist in **Abbildung 2** dargestellt. Der Quellcode sowie ein Readme der einzelnen Schritte für das Set-up in einer eigenen Umgebung sind unter [10] verfügbar.

Die Beispielanwendung besteht grundsätzlich aus zwei Microservices, die jeweils über einen Application Router mit Binding zum Authorization and Trust Management Service erreichbar sind:

- *DocServices:* Dieser Microservice bietet ein API an, um MS-Word-Dokumente mit Content Controls unter Verwendung von docx4j [11] an Custom XML Parts zu binden. Als Ergebnis eines Aufrufs des *merge* Service bekommt man ein Word-Dokument, das aus einer gegebenen Word-Vorlage mit Content Controls und einer XML-Datei zusammengebaut wird. Als Eingabeparameter müssen dem Service demnach zwei Dateien (*.docx* und *.xml*) im Multipart-Format übergeben werden. Der Microservice bringt selbst ein Frontend mit, um zwei Dateien hochzuladen und diese zu mergen. DocServices ist eine Spring-Boot-Anwendung (Version 2.1.5), die für die Authentifizierung und für die Autorisierung den SAP XSUAA Spring Boot Starter verwendet. Die Securitykonfiguration findet in der Klasse *SecurityConfiguration* statt. Außerdem wird Resilienz mit Hystrix beispielhaft im Controller DocController umgesetzt.
- Der Microservice ist an Service-Instanzen der beiden Services Authorization and Trust Management und Application Logging gebunden.

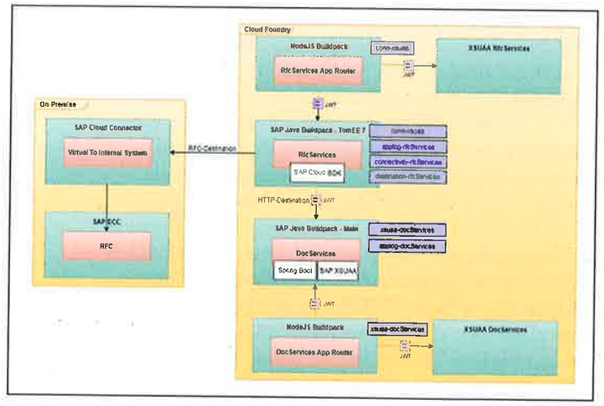

Abb. 2: Architektur der Beispielanwendung

- *RfcServices:* Dieser Microservice bietet ein API an, um einen Auftragsbericht für einen Prozessauftrag aus dem SAP Backend zu erzeugen. Dazu liest der Service für einen gegebenen Prozessauftrag die im SAP Backend gespeicherte Dokumentenvorlage und die Daten zu jenem Auftrag im XML-Format. Die Kommunikation mit dem SAP Backend erfolgt durch Aufrufs eines RFC-Bausteins über den SAP Cloud Connector, der lokal auf dem Entwicklungsrechner installiert wird. Mit diesen beiden Dateien aus dem SAP Backend wird danach der DocServices Microservice aufgerufen, um sie zu mergen. Als Rückgabewert des orderReport-Service des RfcServices Microservice bekommt man letztlich den generierten Auftragsbericht. Auch dieser Microservice bringt ein eigenes Frontend mit, das die Generierung und das Herunterladen eines Auftragsberichts für einen bestimmten Prozessauftrag erlaubt. In diesem Fall wurde das Frontend mit SAPUI5 entwickelt. RfcServices ist eine TomEE-7-Anwendung, die Funktionen aus dem SAP Cloud SDK (Version 3.0.0) wie Handling von Resilienz, Auffinden von Destinationen auf der SAP Cloud Platform sowie Aufruf deren Services und Caching der Ergebnisse verwendet. Der Microservice ist an Service-Instanzen der Services Authorization and Trust Management und Application Logging sowie Destination und Connectivity für den Aufruf des DocServices Microservices und eines Funktionsbausteines des SAP Backends gebunden.

Um das Beispiel lauffähig zu machen, ist ein Account für Cloud Foundry auf der SAP Cloud Platform notwendig. Es empfiehlt sich dafür die Eröffnung eines Trial-Accounts unter [1], der für 90 Tage gültig ist. Nachdem die Microservices wie unter [10] beschrieben gebaut, in der SAP Cloud Platform deployt und an die nötigen Services mittels Service-Instanzen gebunden wurden, sollten vier Anwendungen innerhalb des Cloud Foundry

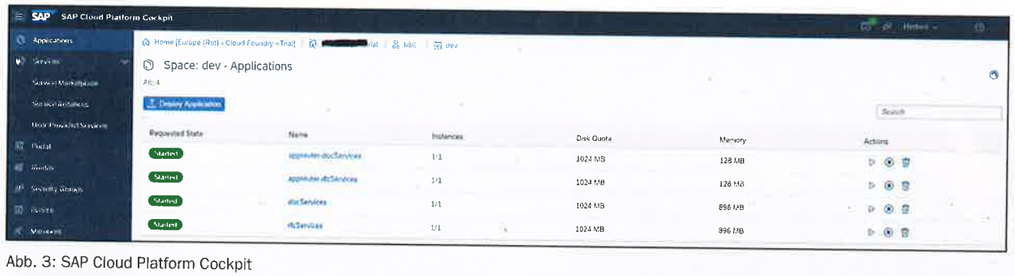

Abb. 3: SAP Cloud Platform Cockpit

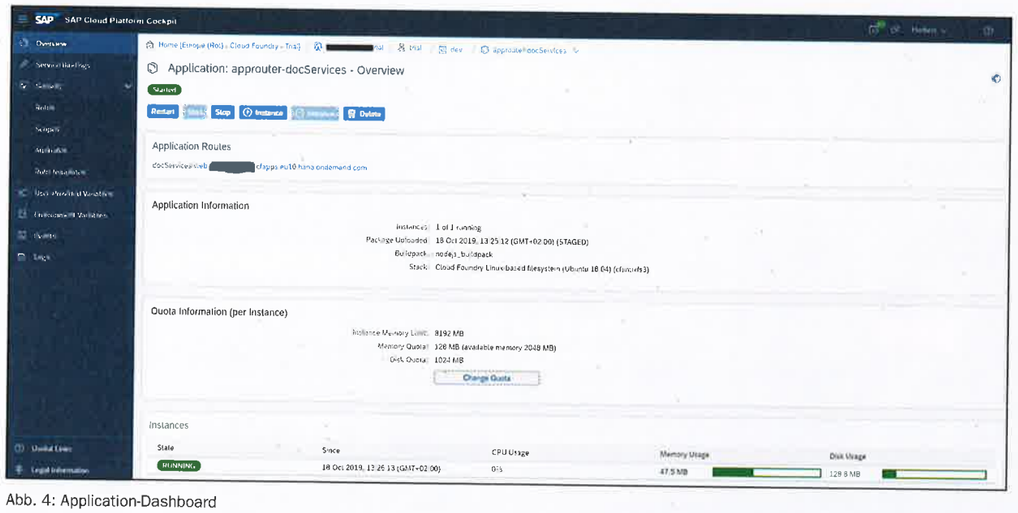

Abb. 4: Application-Dashboard

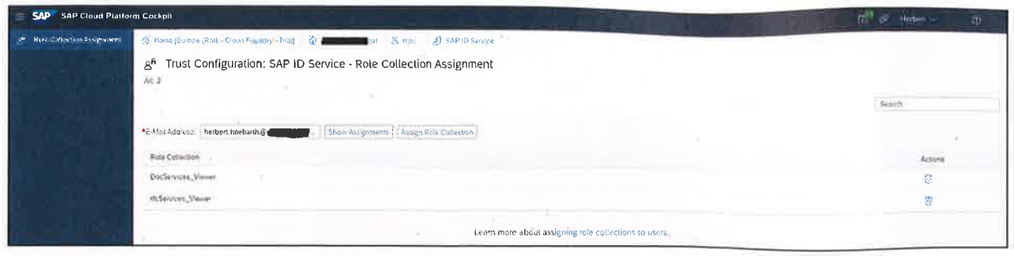

Abb. 5: Zuordnung von Role Collections in der Trust Configuration

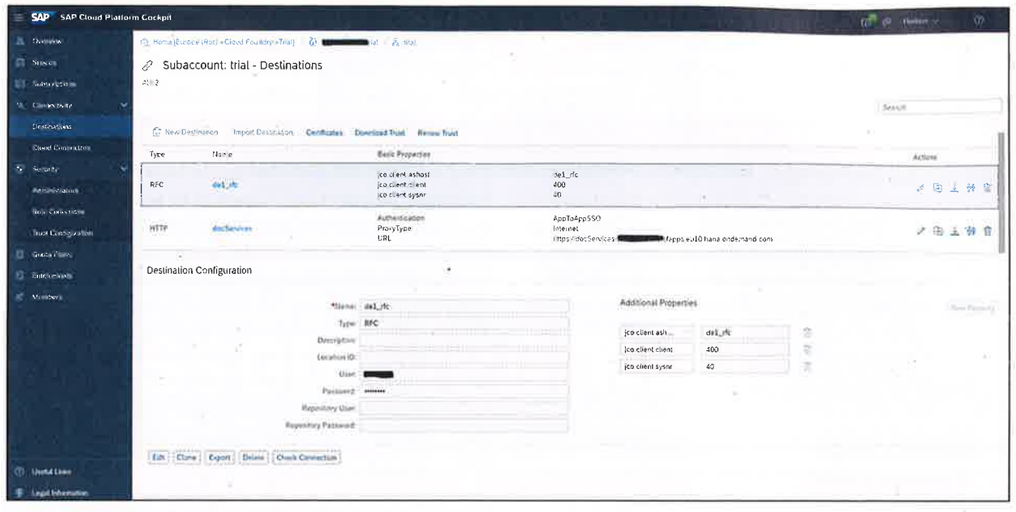

Abb. 6: Destinationen in der SAP Cloud Platform

Spaces im SAP Cloud Platform Cockpit sichtbar sein (**Abb. 3**). Nun können die Frontends der Microservices über die Application Routes im Dashboard ihrer Application-Router-Anwendungen aufgerufen werden (**Abb. 4**). Der Application Router sorgt dafür, dass sich der Benutzer nur einmal an der SAP Cloud Platform anmelden muss. Für die Aufrufe anderer Services in der SAP Cloud Platform generiert er ein JWT-Token und leitet dieses entsprechend weiter.

Da in beiden Microservices auf bestimmte Berechtigungen geprüft wird, müssen dem eigenen Benutzer die Role Collections *DocServices_Viewer* und *rfcServices_Viewer* in der Trust Configuration des Subaccounts zugeordnet werden (**Abb. 5**).

Um den Aufruf des DocServices Microservice aus dem RfcServices Microservice zu ermöglichen, ist eine HTTP Destination innerhalb des Subaccounts anzulegen (**Abb. 6**). Der Name der Destination wird im RcfServices Microservice verwendet, um die Destination in der Klasse *MergeFilesCommand* zu lokalisieren (Listing 1).

Ist ein SAP-System vorhanden, so kann man dieses an die Cloudanwendung binden. Dabei ist der SAP Cloud Connector zu installieren und zu konfigurieren. Im SAP Backend ist ein Funktionsbaustein *Z_PP_PRPP_GET_FILES* wie unter [10] beschrieben anzulegen und dieser im SAP Cloud Connector freizugeben (**Abb. 7**).

Danach ist eine RFC-Destination innerhalb des Subaccounts anzulegen (**Abb. 5**). Dabei ist darauf zu achten, dass die Properties *jco.client.ashost* und *jco.client.sysnr* dem virtuellen Host bzw. virtuellen Port des Systemmappings im SAP Cloud Connector entsprechen. Der Name der Destination wird wiederum im RfcServices Microservice verwendet, um sie in der Klasse *GetOrder-*

Listing 1: Lokalisieren der Destination

```
private byte[] run() {
  try {
    final Destination httpDestination = DestinationAccessor.
                                        getDestination("docServices");
    if(httpDestination != null)
    {
      logger.info("Got the HTTP destination");
      documentService.setHttpDestionation(httpDestination);
    }
  }
  catch (DestinationNotFoundException | DestinationAccessException e) {
    logger.error(e.getMessage());
    throw new ResilienceRuntimeException(e);
  }
  return documentService.merge(docx, xml);
}
```

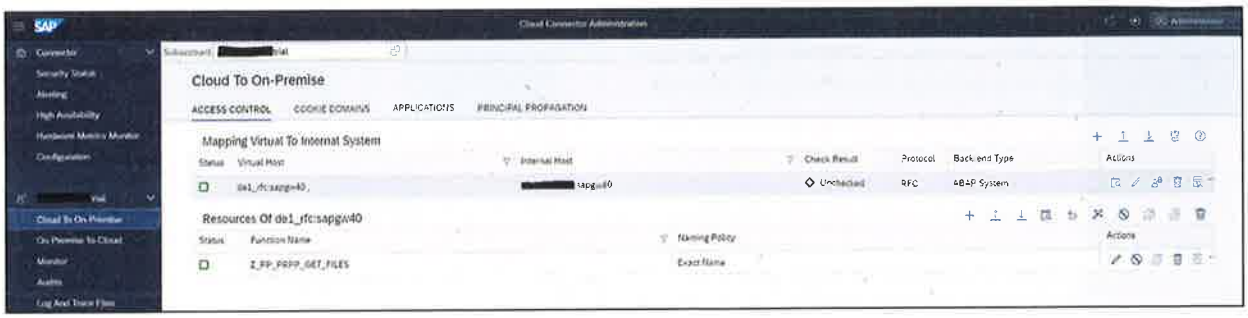

Abb. 7: Cloud Connector Administration

FilesCommand zu lokalisieren und damit einen RFC-Baustein im SAP Backend aufzurufen (Listing 2).

Falls kein SAP Backend vorhanden und konfiguriert ist, greift der RfcServices Microservice als Fallback-Szenario auf zwei Beispieldokumente im WEB-INF-Ordner zurück. Die Beispielanwendung funktioniert also auch ohne SAP-Backend-System.

Fazit und Ausblick

Mit der SAP Cloud Platform bietet SAP seinen Kunden die Möglichkeit, eine bimodale IT-Architektur für ihre SAP-Systemlandschaft aufzubauen, um agiler auf neue Geschäftsanforderungen reagieren zu können. SAP setzt dabei unter anderem auf die offene Cloud-Foundry-Plattform, um Multicloudanwendungen zu ermöglichen und Vendor Lock-ins zu reduzieren. Zusätzlich bietet SAP ein umfangreiches Set von vorgefertigten Services in der Cloud, die einfach in die eigene Anwendung integriert werden können. Mit einem eigenen Java Buildpack und dem SAP Cloud SDK wird die cloudnative Anwendungsentwicklung und der Zugriff auf SAP-Backend-Systeme sehr gut unterstützt. Durch den Fokus auf den Cloud-Foundry-Standard ist die SAP Cloud Platform allerdings auch für Nichtbestandskunden eine interessante Alternative zu anderen PaaS-Anbietern.

Als Board Member der Cloud Foundry Foundation und der Cloud Native Computing Foundation ist SAP auch aktiver Contributor der Projekte Eirini und Quarks, die die bessere Integration von Cloud Foundry und Kubernetes zum Ziel haben.

Listing 2: Aufruf eines RFC-Bausteins

```
@Override
public Map<String, byte[]> getOrderFiles(String order) {
  RfmRequestResult rfmResult;
  try {
    rfmResult = __getRfmRequest(order).execute(rfcDestination);
  } catch (RequestSerializationException | DestinationNotFoundException |
DestinationAccessException
      | RequestExecutionException e) {
    logger.error(e.getMessage());
    throw new ResilienceRuntimeException(e);
  }
  logger.info("RFC executed");
  logger.info("docx length: " + rfmResult.get("E_DOCX").asString().length());

  Map<String, byte[]> result = new HashMap<>();
  result.put("docx", rfmResult.get("E_DOCX").asString().getBytes());
  result.put("xml", rfmResult.get("E_XML").asString().getBytes());
  return result;
}

private RfmRequest __getRfmRequest(String order) {
  RfmRequest rfmRequest = new RfmRequest("Z_PP_PRPP_GET_FILES", false);

  rfmRequest.withExporting("I_AUFNR", "AUFNR", order)
  .withExporting("I_SPRAS", "SPRAS", "DE")
  .withImporting("E_XML", "STRING", xml)
  .withImporting("E_DOCX", "STRING", docx);

  logger.info("rfmRequest created");

  return rfmRequest;
}
```

Herbert Höbarth arbeitet als SAP Application Administrator/Development in einem produzierenden pharmazeutischen Unternehmen in Wien. Dort ist er für kundenspezifische Weiterentwicklungen der eingesetzten SAP-Standardsoftware und die Einhaltung der Entwicklerrichtlinien und Codequalitätsanforderungen zuständig. Dabei setzt er sich auch mit innovativen Technologien innerhalb und außerhalb des SAP-Ökosystems auseinander.

herbert.hoebarth@gmx.at

Links & Literatur

[1] https://www.sap.com/products/cloud-platform.html

[2] https://help.sap.com/viewer/65de2977205c403bbc107264b8eccf4b/Cloud/en-US/73beb06e127f4e47b849aa95344aabe1.html

[3] https://www.cloudfoundry.org/

[4] https://entwickler.de/online/cloud/cloud-foundry-2-579862494.html

[5] https://developers.sap.com/tutorial-navigator.html?tag=products:technology-platform/sap-cloud-platform

[6] https://projects.eclipse.org/proposals/eclipse-tools-cloud-foundry

[7] https://developers.sap.com/topics/cloud-sdk.html

[8] https://support.sap.com/en/product/connectors/jco.html

[9] https://help.sap.com/viewer/cca91383641e40ffbe03bdc78f00f681/Cloud/en-US/e6c7616abb5710148cfcf3e75d96d596.html

[10] https://github.com/hhoebarth/SCPDemo

[11] https://www.docx4java.org/trac/docx4j