Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 271

TOSCA4Mashups – Provisionierung und Ausführung von Data Mashups in der Cloud

Daniel Del Gaudio

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Pascal Hirmer
Beginn am:	1. November 2015
Beendet am:	02. Mai 2016
CR-Nummer:	K.1, K.6.4, D.2.12

Kurzfassung

Mit der stetig wachsenden Menge an Daten wird Datenintegration und Datenverarbeitung zunehmend schwieriger. Domänen-Experten ohne IT-Hintergrund sollen aus großen Datenmengen entsprechende Informationen gewinnen. Leicht zu bedienende Werkzeuge sind nötig, um Daten aus heterogenen Datenmengen durch Domänen-Experten zu verarbeiten. Data Mashups ermöglichen die Verarbeitung und Integration verschiedener, heterogener Datenquellen. Dabei nutzen manche Data Mashup-Lösungen dynamische Ausführungsumgebungen. Die Cloud bietet sich für die Provisionierung solcher dynamisch zusammengesetzten Ausführungsumgebungen an, da Rechenressourcen ebenso dynamisch bereitgestellt werden können. Eine Möglichkeit, um Portabilität und Management von Cloud-Anwendungen zu ermöglichen ist OASIS TOSCA. Um Anwendungen mit TOSCA automatisch zu provisionieren werden alle notwendigen Komponenten in einer sogenannten Topologie modelliert und mit allen notwendigen Informationen, um die Anwendung zu betreiben in ein selbst-enthaltenes Dateiformat, sogenannte Cloud Service Archives verpackt. Die TOSCA Laufzeitumgebung kann diese Archivdatei verarbeiten und die Anwendung automatisiert in der Cloud provisionieren. Im Rahmen dieser Bachelorarbeit wird ein Konzept entwickelt, um Data Mashups automatisiert in der Cloud zu Provisionieren und auszuführen. Um das Konzept zu testen wurde ein Prototyp implementiert, der die TOSCA Laufzeitumgebung OpenTOSCA der Universität Stuttgart verwendet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1. Einleitung und Motivation

Mit der immer stärker wachsenden Anzahl an verfügbaren Daten spielt Datenverarbeitung und Datenintegration eine zunehmend wichtige Rolle [MCB+11]. Sogenannte Domänen-Experten, Benutzer ohne IT-Hintergrund, sollen in der Lage sein aus großen heterogenen Datenmengen relevante Informationen zu gewinnen. Probleme, die sich im Zusammenhang mit sehr großen Datenmengen ergeben, werden unter dem Begriff Big Data [MBD+12] zusammengefasst. Data Mashups [DM14] bieten eine Möglichkeit die komplexen Vorgänge der Datenintegration hinter einer für nicht-technische Anwender bedienbaren graphischen Benutzeroberfläche zu verbergen. Je nach Anwendungsfall können bei Mashups verschiedene Aspekte bei der Ausführung priorisiert werden, wie zum Beispiel Effizienz oder Robustheit. Einen derartigen Ansatz unterstützt beispielsweise das an der Universität Stuttgart entwickelte Data Mashup Tool FlexMash [HM16]. Auf Basis nicht-funktionaler Nutzeranforderungen wird dabei die Ausführungsumgebung der Mashups dynamisch, Baukasten-artig zusammengesetzt.

Cloud Computing[MG11] ermöglicht durch das Prinzip der Virtualisierung hochverfügbare *on-demand*-Rechensysteme. An den Anwendungsfall angepasste virtuelle Maschinen können bei Bedarf innerhalb von Sekunden verfügbar gemacht werden. Abrechnung der Kosten bei Cloud-Anbietern geschieht nach dem *pay-as-you-go*-Prinzip [AG10], was bedeutet, dass lediglich die verwendete Rechenleistung bezahlt werden muss. Die Vorteile des Cloud Computing machen dieses besonders geeignet für ad-hoc auszuführende Data Mashups mit variabler Ausführungsumgebung, wie zum Beispiel FlexMesh.

Eine Anwendungstopologie ist die Beschreibung der für eine Anwendung verwendeten Softwarekomponenten und der Kommunikation zwischen diesen Komponenten. Im Zusammenhang mit einer Cloud-Umgebung gehört dazu beispielsweise die virtuelle Maschine, das Betriebssystem und jede weitere installierte Software, von der die auszuführende Software abhängt, wie zum Beispiel eine Datenbank. OASIS[1] TOSCA[2] ist ein Standard der die Möglichkeit bietet solche Anwendungstopologien zu beschreiben und damit portierbar zu machen. Die Anwendungstopologien der verschiedenen Mashup-Ausführungen können unterschiedlich komplex sein, da jede Ausführung aus verschiedenen Anwendungen und Anwendungskomponenten zusammengesetzt sein kann. Dementsprechend benötigt die

[1] https://www.oasis-open.org/
[2] https://www.oasis-open.org/committees/tosca/

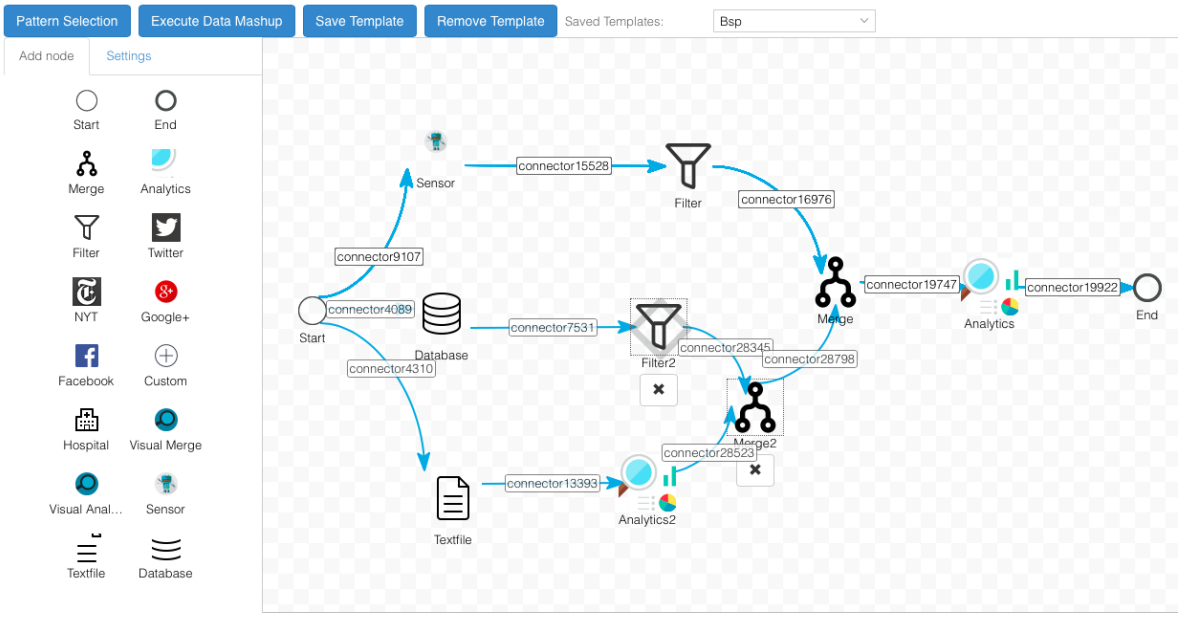

Abbildung 1.1.: Beispiel einer graphischen Oberfläche zur Modellierung von Data Mashups [HM16]

ad-hoc Ausführung von Data Mashups durch Domänen-Experten eine vollautomatisierte Provisionierung, ohne den Anwender mit den Komplikationen dieser zu konfrontieren. In dieser Bachelorarbeit soll ein Konzept hierfür erarbeitet werden.

1.1. Motivationsszenario

In diesem Kapitel wird ein Anwendungsfall beschrieben, der für die Evaluation der Konzepte dieser Arbeit dient.

Ein Produktionsmitarbeiter einer Firma möchte Fehlerfälle einer Fabrik basierend auf verschiedenen Datenquellen automatisiert erkennen. Zu diesen Datenquellen gehören Sensoren, Prozessdatenbanken, Metadatenbanken und textuelle Eingaben von Mitarbeitern. Um dies zu realisieren verwendet er Data Mashups. Um die Zusammenhänge zwischen den einzelnen Datenquellen und die Verarbeitungsschritte zu modellieren entwirft er einen sogenannten Mashup-Plan [HRWM15]. Abbildung 1.1 zeigt eine beispielhafte graphische Benutzeroberfläche, die dazu dient Data Mashups zu modellieren. Links in der Abbildung sind die zur Auswahl stehenden Datenquellen und Datenoperationen, rechts der abstrakte Mashup-Plan. Einen solchen Anwendungsfall, auf den sich diese Arbeit bezieht, beschreiben Kassner und Mitschang [KM15]. Für die Ausführung solcher dynamisch erstellten Data Mashups bietet sich eine Cloud-Umgebung an, da schnell eine angepasste virtuelle Maschine aufgesetzt

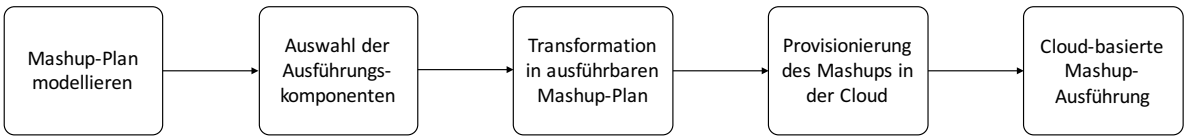

Abbildung 1.2.: Ablauf eines Data Mashups

werden kann. Um Zeit und Kosten zu sparen soll die Provisionierung automatisiert ablaufen, da sonst ein IT-Experte nötig wäre. Da es sich bei dem Benutzer um einen Domänen-Experten, und nicht um einen IT-Experten handelt, kann ihm jedoch auch keine Entscheidung über die Provisionierung überlassen werden. Darum müssen auch alle Entscheidungen, die die Provisionierung betreffen, wie zum Beispiel die Leistung und die Anzahl der virtuellen Maschinen, automatisiert getroffen werden. Nach der Provisionierung soll der Mashup automatisch ausgeführt und das Ergebnis zum Benutzer zurückgeführt werden. Die Verwendung der Cloud-Umgebung soll für den Benutzer vollständig transparent erscheinen. Eine Cloud-Umgebung bietet sich für die Ausführung solcher dynamisch erzeugter Mashups an, da in der Cloud Ressourcen ebenso dynamisch bereitgestellt werden können. Dies birgt jedoch Schwierigkeiten, da bisher keine Möglichkeit besteht, um Data Mashup vollautomatisch in einer Cloud zu provisionieren und auszuführen. Dieses Problem wird in dieser Arbeit gelöst.

Um den Anwendungsfall noch zu verdeutlichen, zeigt Abbildung 1.2 den allgemeinen Ablauf eines dynamisch erstellten Data Mashups. Im ersten Schritt wird der Mashup-Plan vom Anwender in einem nicht-ausführbaren Modell erstellt. Danach wird eine für den Anwendungsfall spezifische Ausführungsumgebung ausgewählt, anhand von Kriterien die der Anwender angibt. Im nächsten Schritt wird der Workflow in einen für die Mashup-Plattform spezifischen ausführbaren Mashup-Plan übersetzt. Nach dem Provisionieren des Mashups in der Cloud wird dieser dort ausgeführt.

1.2. Begriffserklärungen

Häufig in dieser Bachelorarbeit verwendete Begriffe werden in diesem Abschnitt erläutert und ähnliche voneinander abgegrenzt.

1.2.1. Abgrenzung Deployment und Provisionierung

Als Provisionierung werden alle erforderlichen Arbeitsschritte bezeichnet die benötigt werden, um eine Anwendung oder eine Anwendungskomponente in einer Cloud-Umgebung zu installieren und einem oder mehreren Anwendern zur Verfügung zu stellen.

Deployment bezeichnet das Einsetzen von Software in einem System. Bei der Software kann es sich um Dateien und ausführbare Programme handeln. Eine Provisionierung beinhaltet meistens mehrere Deployments.

1.2.2. Abgrenzung Mashup-Plan Workflow und Mashup

Da die Begriffe Workflow, Mashup und Mashup-Plan in dieser Arbeit in einem sehr ähnlichen Kontext verwendet werden, werden diese hier voneinander abgegrenzt.

Ein Mashup-Plan ist eine abstrakte oder technische Beschreibung dessen, was der Benutzer ausgeführen möchte. Dementsprechend können Mashup-Pläne sowohl ausführbar als auch nicht-ausführbar sein. Bei nicht-ausführbaren Mashup-Plänen handelt es sich meist um ein Modell nach dem *Pipes und Filter*-Pattern [Meu95], wie zum Beispiel der BPMN[3]-Notation. Ausführbare Mashup-Pläne sind spezifisch für die jeweilige Ausführungsumgebung. Häufig sind diese im BPEL[4]-Format oder in JSON[5] geschrieben. Im Kontext dieser Arbeit wird aus einem nicht-ausführbaren Mashup-Plan ein ausführbarer Mashup-Plan generiert.

Ausführbare Mashup-Pläne werden ebenfalls als Workflows bezeichnet.

Mashup bezeichnet die gesamte Anwendung zur Ausführung des Mashup-Planes. Im weiteren Sinne bezeichnet Mashup die Menge der Ausführungskomponenten, die für die Ausführung des jeweiligen Workflows benötigt werden. Der Begriff Mashup-Plattform wird mit der selben Bedeutung verwendet.

[3]http://www.bpmn.org/
[4]https://www.oasis-open.org/committees/wsbpel/
[5]http://www.json.org/

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden die nötigen Grundlagen für diese Arbeit vermittelt. Dazu gehören Kenntnisse zu Data Mashups, Cloud Computing und OASIS TOSCA.

Kapitel 3 – Verwandte Arbeiten: Hier werden verschiedene, mit dieser Bachelorarbeit verwandte Arbeiten vorgestellt. Dazu gehören verschiedene Arbeiten, die im Rahmen des Cloud Cycle-Projektes[6] der Universität Stuttgart entstanden sind, auf welchen diese Arbeit aufbaut. Des Weiteren werden verschiedene anderen Cloud-Technologien vorgestellt, von denen sich diese Arbeit abgrenzt.

Kapitel 4 – Konzept: Kapitel 4 stellt den Kern dieser Arbeit dar. In diesem Kapitel wird ein Konzept ausgearbeitet, um Data Mashups mithilfe von OASIS TOSCA automatisiert in einer Cloud-Umgebung zu provisionieren und auszuführen. Dieses Konzept wurde in drei Schritte eingeteilt, nach denen sich die Gliederung des Kapitels richtet. Vorher wird jedoch noch die Architektur des entwickelten Konzepts gezeigt und erläutert.

Kapitel 5 – Implementierung: Um die Funktionalität des Konzeptes zu zeigen wurde ein Prototyp implementiert. Dessen Entwurf und Umsetzung wird in Kapitel 5 beschrieben. Außerdem werden Anwendungsfälle gezeigt, die die Implementierung nicht abdeckt. Zum Schluss wird der Entwurf des Prototypes evaluiert.

Kapitel 6 – Optimierungen: In diesem Kapitel werden verschiedene Möglichkeiten ausgearbeitet, um sowohl das Konzept als auch den Prototyp zu verbessern. Dazu gehört die Wiederverwendung bereits ausgeführter Data Mashups, der Umgang mit Fehlern bei der Provisionierung, die Anpassung der Data Mashups an den jeweiligen Anwendungsfall und die Verwendung anderer Cloud-Provider.

Kapitel 7 – Zusammenfassung und Ausblick Das letzte Kapitel liefert eine Zusammenfassung dieser Arbeit und gibt einen Überblick über mögliche zukünftige Arbeiten, die das Konzept erweitern.

[6] http://www.cloudcycle.org/

2. Grundlagen

In diesem Kapitel werden die für diese Arbeit nötigen Grundlagenkenntnisse vermittelt. Dazu gehören Data Mashups, Cloud Computing und OASIS TOSCA.

2.1. Data Mashups

Daniel und Matera [DM14] definieren Mashups als zusammengesetzte Anwendungen, welche aus wiederverwendbaren Daten, Anwendungslogik und eventuell einer graphischen Benutzeroberfläche entwickelt werden. Alle Daten, jede Anwendungslogik, sowie jede Benutzeroberfläche wird dabei als Mashup-Komponente bezeichnet. Mashup-Komponenten werden in der Regel, jedoch nicht zwangsweise, aus dem Web bezogen. Die Mashup-Logik legt die Zusammensetzung der einzelnen Komponenten, den Kontrollfluss, den Datenfluss, die Datentransformationen sowie die externen Schnittstellen des Mashups fest. Das bedeutet, dass Mashups als Anwendungen gesehen werden können, welche sich aus verschiedenen anderen Anwendungen zusammensetzen und durch eine simple Sprache, wie zum Beispiel über eine graphische Oberfläche, modelliert und entwickelt werden. Grundlegende Elemente der Data Mashup-Modellierung sind Datenquellen und Datentransformationen, die durch Kommunikationswege verbunden werden können. Diese Kommunikationswege entsprechen meist auch dem Kontrollfluss und Datenfluss der Data Mashup-Anwendung. Diese graphischen Oberflächen sind meist nach dem Pipes und Filter-Pattern [Meu95] aufgebaut, wobei die Pipes den Kommunikationswegen der Daten und die Filter den Datenquellen und Datentransformationen entsprechen. Mashups können auf verschiedene Weise realisiert werden, wie beispielsweise durch service-orientierte Architekturen [Erl05] oder durch Complex Event Processing [Luc02]. Andere Arten von Mashups sind UI-Mashups aus der Web 2.0-Bewegung oder Logic-Mashups [DM14]. Mashup-Arten außer den Data Mashups werden in dieser Arbeit nicht behandelt, weswegen sich der Begriff Mashup hier grundsätzlich auf Data Mashups bezieht.

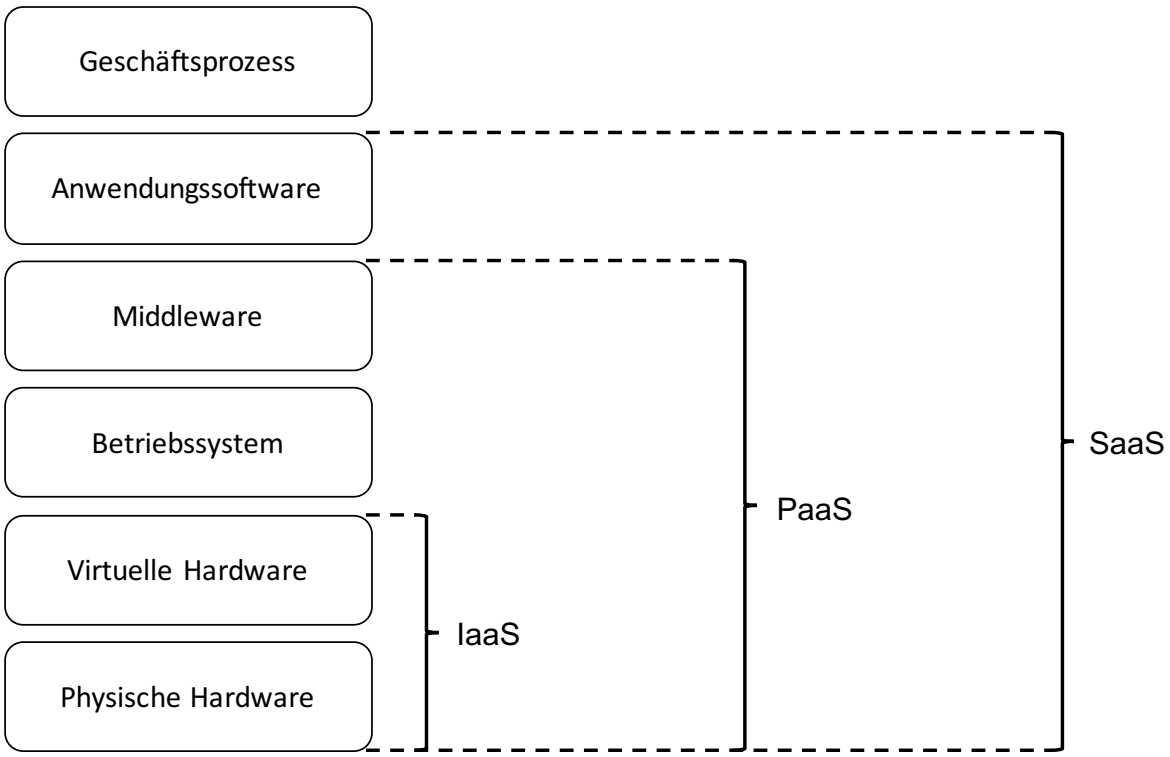

Abbildung 2.1.: Allgemeiner Anwendungsstapel einer Cloud-Anwendung [FLR+14]

2.2. Cloud Computing

Das „National Institute of Standards and Technology“[1] definiert Cloud Computing als „Modell für ubiquitären und komfortablen on-demand Netzwerkzugriff zu geteilten Rechnerressourcen, die schnell und ohne großen Verwaltungsaufwand bereitgestellt und wieder freigegeben werden können“ [MG11]. Dabei muss kaum Kommunikation mit dem Bereitsteller der Ressourcen stattfinden. Das bedeutet, dass Cloud Computing es ermöglicht, angepasste virtuelle Maschinen bei Bedarf innerhalb von kürzester Zeit bereitzustellen. Diese sind dann sofort über eine Netzwerkverbindung verfügbar. Das macht es möglich hochverfügbare und skalierbare Rechensysteme zu schaffen. Da für die gewünschte Rechenleistung keine Hardware direkt bereitgestellt wird, sondern lediglich eine geteilte Nutzung der Hardware durch eine virtuelle Maschine, fallen in der Regel nur Kosten bei der tatsächlichen Nutzung der Ressourcen an. Abbildung 2.1 zeigt auf der linken Seite den allgemeinen Anwendungsstapel einer Cloud-Anwendung [FLR+14]. Als Cloud-Provider wird die Organisation bezeichnet, die je nach Service-Modell entsprechende Software oder Rechenleistung dem Anwender oder Kunden zur Verfügung stellt. Als Cloud-Anwender wird diejenige Person oder Organisation

[1] http://www.nist.gov/

bezeichnet, welche die Software oder Rechenressourcen des Cloud-Providers in Anspruch nimmt. Der Cloud-Provider verwaltet alle Komponenten des Cloud-Anwendungsstapels, der nicht vom Anwender verwaltet wird.

Cloud Computing wird in verschiedene Charakteristika, Service-Modelle und Deployment-Modelle unterteilt, die in den folgenden Abschnitten erläutert werden.

2.2.1. Service-Modelle

Die verschiedenen Service-Modelle unterscheiden sich darin, wie viel Verwaltungsaufwand der Anwender hat, und wie viel Verantwortung dem Cloud-Provider überlassen wird. Auf der rechten Seite von Abbildung 2.1 sind die Verantwortlichkeiten des Cloud-Providers bei den verschiedenen Service-Modellen dargestellt, welche im Folgenden genauer erläutert werden.

Software as a Service (SaaS)

Dem Benutzer wird Software zur Verfügung gestellt, die auf einer Cloud-Infrastruktur läuft [MG11]. Auf diese kann durch Web Browser oder Programmierschnittstellen zugegriffen werden. Der Benutzer kann ohne großen Aufwand verschiedene Softwarepakete, wie zum Beispiel Datenbanken, aufsetzen und verwenden. Für ihn besteht kein Verwaltungsaufwand hinsichtlich dieser. Im Anwendungsstapel von Cloud-Anwendungen muss sich der Benutzer lediglich darum kümmern, dass mit der durch den Cloud-Provider bereitgestellte Anwendungssoftware die eigenen Geschäftsprozesse korrekt durchgeführt werden.

Platform as a Service (PaaS)

Der Benutzer hat die Möglichkeit eigenständig programmierte Anwendungen in der Cloud-Infrastruktur zu deployen und auszuführen [MG11]. Der Vorteil für den Benutzer bei diesem Service-Modell liegt darin, dass er sich lediglich um seine eigene Anwendungssoftware und die durch diese ausgeführten Geschäftsprozesse kümmern muss. Die Teile des Cloud-Anwendungsstapels von der physischen Hardware bis zur Middleware werden durch den Cloud-Provider verwaltet.

Infrastructure as a Service (IaaS)

In diesem Service-Modell werden dem Benutzer fundamentale Rechnerressourcen zur Verfügung gestellt [MG11]. Er erhält Zugang zu virtuellen Maschinen über ein Netzwerk. Der

Verwaltungsaufwand des gesamten Teiles der Cloud-Anwendungsstapels von der virtuellen Hardware bis zum Geschäftsprozess liegt bei dem Benutzer.

2.2.2. Deployment-Modelle

In diesem Abschnitt werden die vier verschiedenen Deployment Modelle des Cloud Computing erläutert. Der Unterschied zwischen den verschiedenen Deployment-Modellen liegt darin, mit welchen anderen Personen und Organisationen die Software und Rechenleistung der Cloud geteilt wird.

Public Cloud

Bei einer Public Cloud-Umgebung werden die Rechenressourcen zur freien Benutzung bereitgestellt. Hierbei wird oft nach dem pay-as-you-go-Prinzip abgerechnet. Das bedeutet, dass der Benutzer nur die wirklich verwendete Rechenleistung bezahlen muss. Die Public Cloud bietet die geringste Sicherheit, da der Cloud-Provider in der Lage ist auf alle Ressourcen zuzugreifen. Die Sicherheit der Daten und Anwendungen ist durch das Vertrauen zum Cloud-Provider beschränkt. Da eine Vielzahl unbekannter Dritter die selbe Cloud verwendet, könnten Sicherheitsfehler schnell verheerend werden. Ein Beispiel für einen möglichen Angriff auf eine virtuelle Maschine in einer Public Cloud ist das sogenannte VM Tunneling [KV10].

Private Cloud

Bei einer Private Cloud stehen die Rechnerressourcen lediglich einer einzigen Organisation zur Verfügung. Sie kann jedoch einer anderen Organisation gehören und auch durch diese verwaltet werden. Eine Private Cloud bietet die sicherste Umgebung unter den Service-Modellen, da niemand auf die virtuellen Maschinen oder auf die physikalische Hardware zugreifen kann.

Hybrid Cloud

Beim Deployment Model Hybric Cloud werden eines oder mehrere der anderen Deploment-Modelle verbunden. Bei einer Hybrid Cloud könnten beispielsweise sicherheitsrelevante Daten auf den Private Cloud-Anteil, und weniger sicherheitsrelevante auf den Public Cloud-Anteil gelegt werden.

Community Cloud

Bei dem Deploment-Modell der Community Cloud stehen alle Rechnerressourcen einer bestimmten Gemeinschaft von Benutzern von Organisationen zur Verfügung, die gemeinsame Anliegen besitzen. Auch hier kann die Cloud durch Dritte betrieben und verwaltet werden. Die Community Cloud bietet eine ähnliche Sicherheit wie eine Public Cloud, jedoch besteht kein Risiko durch unbekannte Dritte.

2.3. OASIS TOSCA

„Topology and Orchestration Specification for Cloud Applications“ von der „Organization for the Advancement of Structured Information Standards“ (OASIS) ist ein Standard zur Beschreibung und Portierung von Cloud-Anwendungen.

Außer einem Sprachstandard besteht die Topology and Orchestration Specification for Cloud Applications noch aus der Definition von *Cloud Service Archive* Dateien und einer beispielhaften Architektur einer TOSCA Laufzeitumgebung.

Mehrere zusammenhängende Anwendungen werden mit TOSCA als Topologien in Form von gerichteten Graphen beschrieben, wobei jeder Knoten eine Instanz einer Anwendung und jede Kante eine Relation zwischen zwei Anwendungen symbolisiert. Abbildung 2.2 zeigt eine Anwendungstopologie mit zwei virtuellen Maschinen, auf denen jeweils ein Linux Betriebssystem provisioniert ist. Auf dem einen ist ein Apache Web Server[2] installiert, auf dem anderen eine MySQL-Datenbank[3]. In der Topologie existieren zwei Arten von Relationen: „hosted on“ und „connect to“. Erstere symbolisiert, dass die Anwendung von der die Kante ausgeht auf der Anwendung zu der die Kante hinführt ausgeführt wird. Die Kante zwischen den Anwendungen „Web Shop“ und „MySQL Database“ symbolisiert den Zugriff der Web-Anwendung auf die Datenbank. Die Knoten in einer Anwendungstopologie werden in TOSCA als *Node Templates*, die Relationen als *Relationship Templates* bezeichnet. Mehrere Anwendungen, von denen jede auf der darunter liegenden Anwendung aufbaut, werden als *Anwendungs-Stack* bezeichnet. Die Topologie in Abbildung 2.2 besteht aus zwei Anwendungs-Stacks. Die vier Anwendungen auf der linken Seite und die vier auf der rechten Seite bilden jeweils einen Stack.

[2]https://httpd.apache.org/
[3]https://www.mysql.de/

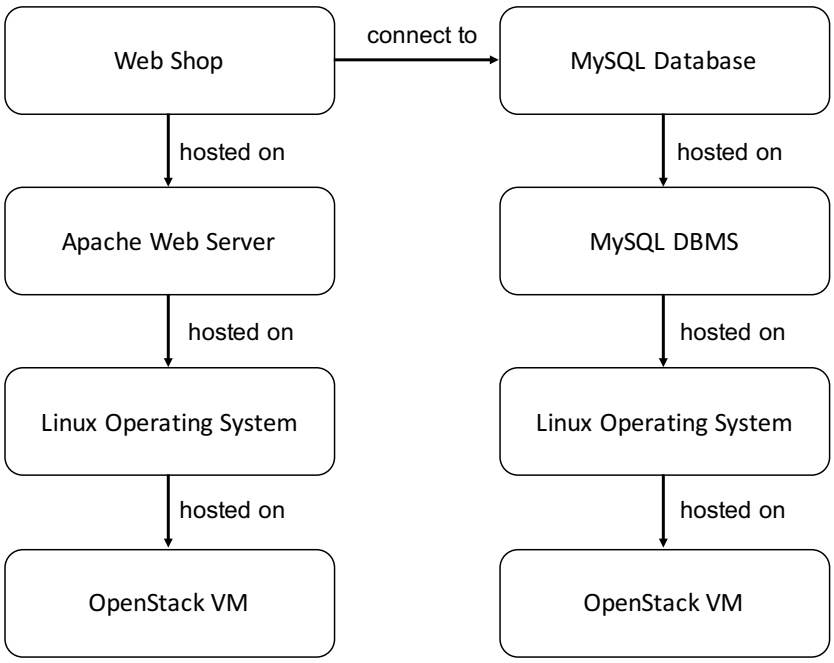

Abbildung 2.2.: Beispiel für eine Anwendungstopologie [BBK+14]

2.3.1. TOSCA Sprachstandard

TOSCA ist ein 2013 von OASIS definierter, auf XML-basierender[4], Sprachstandard [OASb] zur Beschreibung von Cloud-Anwendungen. Aus diesem Grund wird der Begriff „Element“ in dieser Arbeit im Sinne der XML-Definition[5] verwendet. Die für diese Arbeit relevanten TOSCA-Elemente werden in diesem Kapitel erläutert.

Node Types

Ein *Node Type* ist eine wiederverwendbare Einheit, die die Struktur eines oder mehrerer Node Templates vorgibt [OASb]. In den Node Types werden unter anderem *Properties*, *Capabilities* und *Requirements* definiert. Properties sind Eigenschaften der modellierten Anwendung, welche in den aus dem Node Type instanziierten Node Template angegeben werden müssen. Requirements müssen durch Capabilities anderer Node Types erfüllt werden, indem zwischen den Node Templates ein *Relationship Type* instaziiert wird. Listing 2.1 zeigt eine beispielhafte Node Type-Definition, welche eine Node-RED-Umgebung modelliert.

[4]https://www.w3.org/XML/
[5]https://www.w3.org/TR/REC-xml/

Listing 2.1 Beispiel für eine TOSCA Node Type-Definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tosca:Definitions id="winery-defs-for_ns6-Node-RED"
    targetNamespace="http://types.opentosca.org"
    xmlns:tosca="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:ns2="http://www.eclipse.org/winery/model/selfservice">
   <tosca:NodeType name="NodeRED" abstract="no" final="no"
       targetNamespace="http://types.opentosca.org" winery:bordercolor="#b8f246">
      <tosca:DerivedFrom typeRef="ns1:RootNodeType"
          xmlns:ns1="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"/>
      <tosca:RequirementDefinitions>
         <tosca:RequirementDefinition name="NodeJsContainer"
             requirementType="ns0:NodeJsContainerRequirement" lowerBound="1"
             upperBound="1" xmlns:ns0="http://types.opentosca.org"/>
      </tosca:RequirementDefinitions>
      <tosca:CapabilityDefinitions>
         <tosca:CapabilityDefinition name="HostNodeRedPlan"
             capabilityType="ns0:NodeRedContainerCapability" lowerBound="0"
             upperBound="unbounded" xmlns:ns0="http://types.opentosca.org"/>
      </tosca:CapabilityDefinitions>
      <tosca:Interfaces/>
   </tosca:NodeType>
</tosca:Definitions>
```

Node Templates

Node Templates sind instanziierte Node Types. Jedes Node Template modelliert genau eine Instanz einer zu provisionierenden Anwendung. In Node Templates können die in den Node Types definierten Properties mit Werten belegt werden. Node Templates müssen einen eindeutigen Identifikator besitzen.

Relationship Types

Relationship Types definieren den Typ von einem oder mehrerer Relationship Templates zwischen Node Templates [OASb]. Relationship Types modellieren mögliche Verbindungen zwischen zwei Node Templates bestimmter Node Types. Beispiele dafür sind „hosted on“ und „connect to“ in Abbildung 2.2. In fast jeder Topologie wird eine Relation mit der selben Semantik wie der von „hosted on“ verwendet. Diese symbolisiert, dass die Anwendung von der die Relation ausgeht auf der Anwendung deployt ist, zu der die Relation hinführt.

Relationship Templates

Relationship Templates sind instanziierte Relationship Types. Sie modellieren eine Beziehung zwischen zwei Node Templates. Sie enthalten ein Quell- und ein Zielelement, die jeweils mit dem eindeutigen Identifikator der Quell- und Ziel-Node Templates belegt werden.

Artifact Types

Artifact Types sind wiederverwendbare Einheiten, welche den Typ eines oder mehrerer *Artifact Templates* definieren [OASb]. Artefakte sind Dateien, die sich in der CSAR-Datei befinden. Gängige Artifact Types sind zum Beispiel solche für WAR- oder ZIP-Dateien.

Artifact Templates

Artifact Templates referenzieren eine Datei innerhalb der CSAR-Datei. Dabei wird zwischen Implementierungs- und Deployment-Artefakten unterschieden. Implementierungs-Artefakte sind ausführbare Dateien, welche die Provisionierung einer bestimmten Anwendung durchführen. Deployment-Artefakte sind Dateien, die für die Provisionierung verwendet werden, wie zum Beispiel Archivdateien mit den ausführbaren Dateien einer Anwendung.

Node Type Implementation

Node Type Implementations repräsentieren ein Implementierungs-Artefakt für einen speziellen Node Type. Node Type Implementations bringen alle Node Templates eines bestimmten Node Types mit einem bestimmten Artifact Template in Verbindung. Dadurch weiß die TOSCA Laufzeitumgebung, welche Artefakte für die Provisionierung der Node Templates dieses bestimmten Node Types verwendet werden sollen.

Topology Template

Topology Templates bestehen aus Node Templates und Relationship Templates. Sie bilden das Modell für die dargestellte Topologie. Listing 2.2 zeigt ein TOSCA Topology Template mit einem Node Template, welches aus dem Node Type in Listing 2.1 instanziiert wurde.

Listing 2.2 Beispiel für ein TOSCA Topology Template

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tosca:Definitions id="definitions" name="definitions"
    targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:tosca="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:ns2="http://www.eclipse.org/winery/model/selfservice">
   <tosca:ServiceTemplate id="InstallVMServTemplate"
       targetNamespace="http://types.opentosca.org">
      <tosca:TopologyTemplate>
         <tosca:NodeTemplate name="NodeRED" minInstances="1" maxInstances="1"
             id="NodeRED" type="ns0:NodeRED" winery:x="500" winery:y="250"
             xmlns:ns0="http://types.opentosca.org">
            <tosca:Capabilities>
               <tosca:Capability name="NodeRedContainerCapability" id="ap14z70qp"
                   type="ns0:NodeRedContainerCapability"/>
            </tosca:Capabilities>
         </tosca:NodeTemplate>
      </tosca:TopologyTemplate>
  </tosca:ServiceTemplate>
</tosca:Definitions>
```

Service Template

Im *Service Template* wird das Topology Template, alle Artifact Templates, alle Node Type Implementations und das *Plans*-Element zusammengefügt. Das Plans-Element ist das TOSCA-Element, dass die Schnittstelle zwischen dem Topology Template und dem Provisionierungsplan definiert, welcher in Abschnitt 2.3.2 erläutert wird.

Definitions

Das *Definitions*-Element ist das Wurzelelement jeder TOSCA-Definition. Sowohl Node Type-Definitionen als auch Service Templates müssen sich in einem Definitions-Element befinden. Die Beispiele in den Abbildungen 2.1 und 2.2 veranschaulichen dies.

2.3.2. Provisionierungsplan

Eine TOSCA-Anwendung kann entweder durch eine deklarative oder durch eine imperative Vorgehensweise provisioniert werden. Bei der deklarativen Provisionierung wird lediglich anhand der Beschreibung der Topologie vorgegangen. Bei einer imperativen Provisionierung benötigt es einen sogenannten Provisionierungsplan. Dieser kann in verschiedenen Ausführungssprachen wie zum Beispiel BPEL geschrieben sein. Generell werden für die

Provisionierung die Implementierungs-Artefakte jedes Knoten, beginnend mit dem in der untersten Ebene aufgerufen. Mit der untersten Ebene werden alle Knoten im Topologiegraphen bezeichnet, von denen keine Kanten ausgehen. Im Beispiel in Abbildung 2.2 entspricht das den beiden virtuellen Machinen. Im nächsten Schritt werden alle Node Templates behandelt, zu denen Relationship Templates von den eben behandelten Node Templates ausgehen. Die Implementierungs-Artefakte dieser Node Templates werden aufgerufen, um die damit assoziierten Anwendungen zu provisionieren. Dieser Prozess setzt sich sukzessive fort, bis die behandelten Node Templates keine ausgehenden Relationship Templates mehr besitzen.

In Abschnitt 4.4.2 „Schritt 2.2: Anwendung provisionieren“ wird das Ausführen des Provisionierungsplanes genauer beschrieben.

Die Alternative zu einer sogenannte imperativen Provisionierung mit einem Provisionierungsplan ist die deklarative Provisionierung. Bei dieser wird das Topology Template während der Provisionierung interpretiert und die Implementierungs-Artefakte entsprechend aufgerufen. Breitenbücher et al. beschreiben in „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“ [BBK+14] eine Möglichkeit aus der Beschreibung der Topologie einer Anwendung einen Provisionierungsplan zu generieren, womit die beiden Provisionierungsansätze verbunden werden.

2.3.3. Cloud Service Archive (CSAR)

Alle nötigen Komponenten, um eine TOSCA Cloud-Anwendung aufsetzen zu können werden in einer Cloud Service Archive-Datei, einer ZIP-Datei[6], abgelegt. Dazu gehört das Service Template, sowie alle verwendeten TOSCA Type Definitionen, Artefakte und Pläne. Eine CSAR-Datei muss mindestens die beiden Ordner „TOSCA-Metadata“ und „Definitions“ beinhalten [OASb]. Dem Ersteller der Datei ist es freigestellt beliebige zusätzliche Ordner anzulegen. Der Ordner „TOSCA-Metadata“ muss die Datei „TOSCA.meta“ enthalten, in welcher der restliche Inhalt der CSAR-Datei beschrieben wird. Der „Definitions“-Ordner enthält alle verwendeten TOSCA-Definitionen. Diese können in einer oder mehreren Dateien enthalten sein, welche alle die Endung „.tosca“ haben müssen. Eine dieser Dateien muss ein Definitions-Element mit einem TOSCA Service Template enthalten, welches die Struktur der Cloud-Anwendung beschreibt. Weitere Ordner können zum Beispiel für den Provisionierungsplan oder für Artefakte angelegt werden.

[6] https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT

2.3.4. TOSCA Laufzeitumgebung

In dem Dokument „Tosca Primer“ [OASa] wird auch eine sogenannte *TOSCA Laufzeitumgebung* beschrieben. Die TOSCA Laufzeitumgebung kann CSAR-Dateien und die in ihr enthaltenen TOSCA Definitionen verstehen und die enthaltenen Dateien verarbeiten, um Anwendungen automatisch zu provisionieren.

Die Architektur und die Funktionsweise der TOSCA Laufzeitumgebung wird in Kapitel 4.1.2 „4.1.2“ genauer erläutert.

3. Verwandte Arbeiten

Die für diese Bachelorarbeit relevanten Arbeiten werden in diesem Kapitel beschrieben.

Die verwandte Arbeit, aus der diese Bachelorarbeit hervorgegangen ist, ist „FlexMash - Flexible Data Mashups Based on Pattern-Based Model Transformation“ von Hirmer et. al [HM16]. In der Arbeit wird ein Konzept beschrieben, bei welchem Data Mashup-Modelle auf verschiedene Ausführungskomponenten abgebildet werden. Die Wahl der Ausführungskomponenten hängt dabei von den Bedürfnissen des jeweiligen Benutzers ab. Die Arbeit liefert das Verständnis von dynamischen Data Mashups für diese Bachelorarbeit.

Alle weiteren verwandten Arbeiten werden in die beiden Kapitel 3.1 „OpenTOSCA“ und 3.2 „Alternativen zu TOSCA“ unterteilt.

3.1. OpenTOSCA

OpenTOSCA[1] ist ein Projekt der Universität Stuttgart, in dessen Rahmen verschiedene mit dieser Arbeit verwandte Arbeiten veröffentlicht wurden, die in diesem Abschnitt vorgestellt werden.

Binz et al. [BBH+13] beschreiben in „OpenTOSCA – A Runtime for TOSCA-based Cloud Applications“ eine TOSCA Laufzeitumgebung namens OpenTOSCA. Sie unterstützt eine imperative und planbasierte Provisionierung von Anwendungen. Als Grundlage für diese werden die im Zuge von OASIS TOSCA definierten CSAR-Dateien unterstützt. In der Implementierung dieser Bachelorarbeit wird OpenTOSCA als TOSCA-Laufzeitumgebung verwendet. Zukünftig soll OpenTOSCA noch um eine deklarative Provisionierung erweitert werden. Im Rahmen von OpenTOSCA wurden auch die beiden Tools Winery und Vinothek entwickelt, deren Veröffentlichungen im Folgenden vorgestellt werden.

In „Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA“ [BBK+12] wird eine graphische Notation vorgestellt, die sich für die Modellierung von TOSCA Topology Templates eignet.

[1] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

Kopp et al. [KBBL13] beschreiben ein auf HTML5 basierendes Visualisierungstool namens Winery, um Anwendungstopologien zu modellieren. Um die Topologien intern zu speichern, zu importieren und zu exportieren verwendet es TOSCA. Neben dem Erstellen von TOSCA Topology Templates mit einer einfach zu verwendenden, graphischen Benutzeroberfläche und einer graphischen Notation, bietet es die Möglichkeit TOSCA Typen zu definieren. Die graphische Notation zur Modellierung der Topologien wird in [BBK+12] beschrieben. Verschiedene Funktionen von Winery werden in der Implementierung zu dieser Bachelorarbeit verwendet. Dazu gehört die Funktion Topologien zu vervollständigen, die von Hirmer et al. [HBBL14] beschrieben wird, und ein Plangenerator, der von Breitenbücher et al. in „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“ [BBK+14] ausgearbeitet wurde. Winery wird inzwischen als Open Source Projekt der Eclipse Software Foundation[2] geführt.

Der Artikel „Vinothek - A Self-Service Portal for TOSCA“ [BBKL14] handelt von einem Selbstbedienungsportal für TOSCA-basierte Cloud Anwendungen, welches auf Java Server Pages und HTML5 basiert. Ziel dabei ist es, Benutzern eine bedienungsfreundliche graphische Benutzeroberfläche zu bieten, um Anwendungen in der Cloud zu provisionieren. Im Gegensatz zur Vinothek wird bei dieser Bachelorarbeit ein Konzept erarbeitet, bei dem der gesamte Provisionierungsvorgang sowie die Ausführung der Anwendung vor dem Benutzer versteckt wird.

Alle Arbeiten zu OpenTOSCA liefern eine Grundlage für die Problemstellung dieser Arbeit, deren Konzepte sind jedoch nicht suffizient für eine vollautomatisierte Provisionierung von Data Mashups. Um eine neue Anwendungstopologie zu provisionieren setzen sie die Modellierung dieser mit TOSCA oder mithilfe des Tools Winery voraus. Dafür benötigt es jedoch Kenntnisse über die genauen Ausführungskomponenten, welche von einem Domänen-Experten nicht erwartet werden können. Diese Arbeit löst das Problem durch einen vollautomatisierten Ansatz zur Provisionierung.

3.2. Alternativen zu TOSCA

Außer TOSCA existieren noch andere Standards, um Anwendungen automatisiert in einer Cloud-Umgebung zu provisionieren.

Speziell für die Cloud-Umgebung Amazon Web Services[3] (AWS) existiert CloudFormation[4]. Mit CloudFormation können Vorlagen zur Beschreibung von AWS-Ressourcen erstellt werden, um diese dann automatisch zu provisionieren und bereitzustellen.

[2]https://www.eclipse.org/proposals/soa.winery/
[3]https://aws.amazon.com/de/
[4]https://aws.amazon.com/de/cloudformation/

Das Äquivalent zu CloudFormation für die Cloud-Umgebung OpenStack[5] ist Heat[6]. Mit Heat können in Form von sogennanten Heat Orchestration Templates Topologien von Cloud-Anwendungen definiert werden.

Bei CloudFormation und Heat handelt es sich um deklarative Systeme. Das heißt, dass nur solche Anwendungen provisioniert werden können, die von der jeweiligen Laufzeit- oder Cloud-Umgebung erkannt werden. Aus diesem Grund konzentriert sich diese Arbeit auf OASIS TOSCA anstelle von Heat oder CloudFormation. Bei TOSCA können nicht nur vorgefertigte Templates verwendet, sondern Node Types für beliebige Anwendungen erstellt werden. Zusammen mit einer imperativen Provisionierung anhand eines Provisionierungsplanes macht dies TOSCA vollständig generisch.

Mit Cloudify[7] wurde ebenfalls eine TOSCA Laufzeitumgebung implementiert. Für diese Arbeit wurde jedoch OpenTOSCA aufgrund dessen Toolsupports verwendet. Dazu gehören zum Beispiel Winery und der Plan Generator [BBK+14]. Zudem wurde OpenTOSCA an der Universität Stuttgart entwickelt, was eine große Unterstützung für diese Arbeit gewährleistet.

Cloud Foundry[8] ist ein Open Source PaaS-System, welches Container für Software in verschiedenen Sprachen wie zum Beispiel Java, Ruby und Python bietet. Eine weitere PaaS-Software ist OpenShift[9]. In dieser Bachelorarbeit wurde TOSCA anstelle von PaaS-Systemen verwendet, da mit TOSCA generisch jede Art von Software provisioniert werden kann, und nicht nur vorgefertigte Container verwendet werden können.

Eine weitere Möglichkeit Anwendungen in einer Cloud automatisiert zu verwalten bieten die Konfigurationswerkzeuge Puppet[10] und Chef [11]. Bei Puppet muss eine Maschine, der sogenannte Puppet-Master, über sogenannte Puppet-Manifeste deklarativ konfiguriert werden. In diesen Manifesten kann der gewollte Zustand eines Systems beschrieben werden, wie zum Beispiel welche Anwendungen darauf installiert sein sollen. Alle mit dem Puppet-Master verbundenen und entsprechend eingerichtete Maschinen passen sich dann diesem an. Da es sich dabei um ein deklaratives System handelt, können nur Anwendungen provisioniert werden, die dem Ausführungssystem von Puppet bekannt sind. Bei Chef müssen sogenannte Rezepte geschrieben werden anhand welcher, ähnlich wie bei Puppet, ausgewählte Rechner konfiguriert werden. Chef besitzt die selben Nachteile wie Puppet und dient deswegen auch nicht für die Lösung der Problemstellung dieser Arbeit.

[5] https://www.openstack.org/
[6] https://wiki.openstack.org/wiki/Heat
[7] http://getcloudify.org/
[8] https://www.cloudfoundry.org/
[9] https://www.openshift.com/
[10] https://puppetlabs.com/
[11] https://www.chef.io/chef/

4. Konzept

Um beliebige Mashups automatisiert auf einer Cloud-Umgebung zu provisionieren und anschließend auszuführen wurde ein Konzept erarbeitet. Dieses stellt den Kern dieser Bachelarbeit dar und wird in diesem Kapitel erläutert. Dieses Kapitel unterteilt sich in die Abschnitte 4.1 Architektur, 4.2 Methode/Übersicht, 4.3 CSAR erstellen, 4.4 CSAR deployen und Anwendung provisionieren und 4.5 Mashup ausführen. 4.1 gibt eine Übersicht über die Architektur des Systems und stellt die verwendeten Komponenten vor. 4.2 zeigt die drei Schritte, in welche die Methode unterteilt wurden. 4.3, 4.4 und 4.5 entsprechen jeweils einem dieser drei Schritte.

4.1. Architektur

Die Architektur des gesamten Systems besteht aus sechs Komponenten. In diesem Kapitel wird in Abschnitt 4.1.1 zunächst eine Übersicht über die Architektur vorgestellt. In Abschnitt 4.1.2 werden dann die einzelnen Komponenten der Architektur genauer erläutert.

4.1.1. Übersicht

Abbildung 4.1 zeigt eine schematische Übersicht über alle involvierten Komponenten. Bei den beiden Komponenten *Node Type Repository* und *Artifact Repository* handelt es sich um zwei Datenbanken, auf welche die beiden Komponenten *Topology Completer* und *TOSCA4Mashups* lesend zugreifen. Topology Completer ist eine Komponente zur Vervollständigung von TOSCA-Topologien. Die TOSCA4Mashups-Komponente generiert die CSAR-Datei mithilfe der anderen Komponenten und übergibt diese an die *TOSCA Runtime*, bei der es sich um eine TOSCA Laufzeitumgebung handelt. Die Kommunikation zwischen TOSCA4Mashups und den beiden Komponenten Topology Completer und *Plan Generator* läuft nach dem Anfrage-Antwort-Prinzip, wobei TOSCA4Mashups als der Anfragende fungiert. Bei *Plan Generator* handelt es sich um einen Plangenerator im Sinne von Breitenbücher et al. [BBK+14]. TOSCA4Mashups kommuniziert mit der TOSCA Laufzeitumgebung auf nur einem Weg, da lediglich die fertige CSAR-Datei zur Provisionierung übergeben wird.

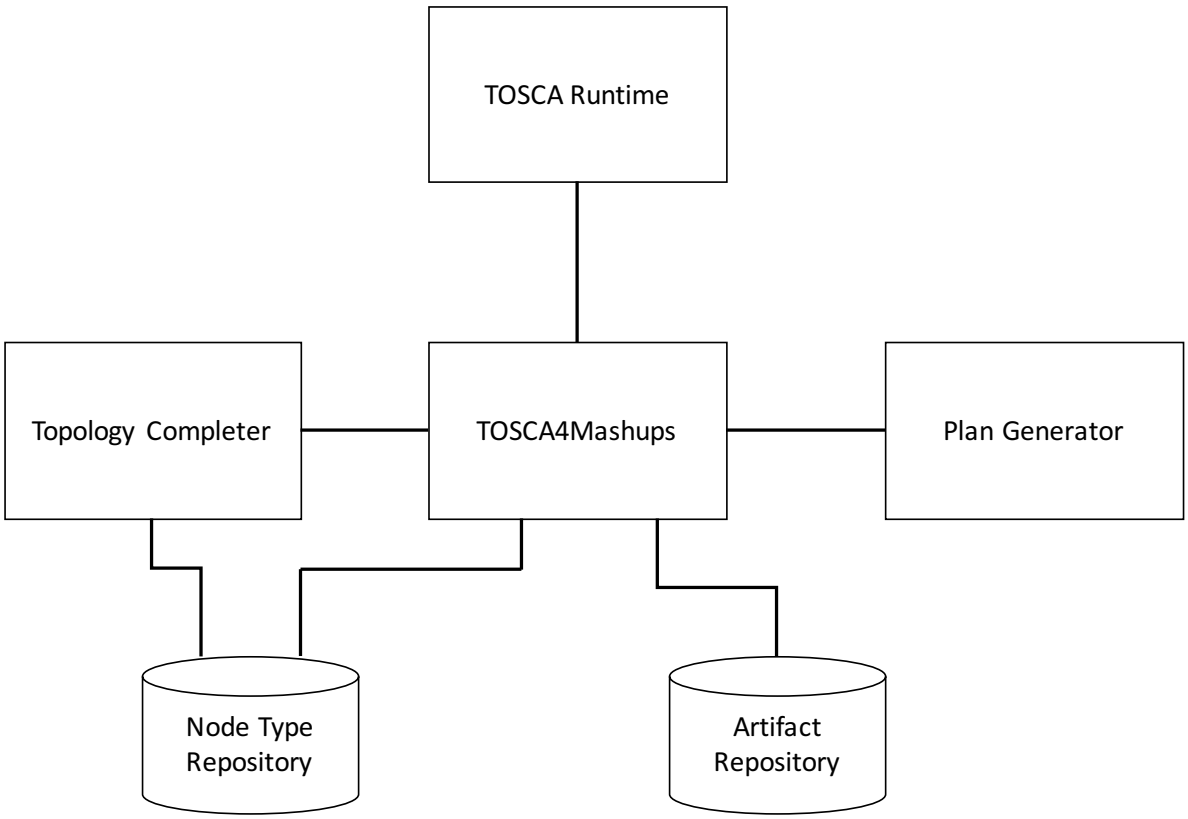

Abbildung 4.1.: Architektur des gesamten Systems

4.1.2. Komponenten

Im Folgenden werden die einzelnen Komponenten der Architektur genauer erläutert.

TOSCA4Mashups

Diese Komponente stellt die Schnittstelle zwischen allen anderen Komponenten dar. Außer dem eigentlichen Zusammenstellen der CSAR-Datei ruft sie den Topology Completer auf, um die Topologie vervollständigen zu lassen, und den Plan Genererator, um den Plan erstellen zu lassen. Zuletzt übergibt TOSCA4Mashups die fertige CSAR-Datei der TOSCA Laufzeitumgebung, um die Anwendung automatisiert provisionieren zu lassen. Abbildung 4.2 zeigt die innere Struktur der TOSCA4Mashups Komponente. Der TOSCA Runtime Adapter übernimmt die Kommunikation mit der jeweiligen TOSCA Laufzeitumgebung, der Plan Generator Adapter die mit dem Plan Generator, der Topology Completer Adapter die mit dem Topology Completer. Um auf die beiden Repositories für die Node Types und die Artefakte zuzugreifen besteht der Repository Adapter. Der Sevice Generator ist in der Lage mit Hilfe des Repository Adapters und der Topology Completer Adapters ein vollständiges TOSCA

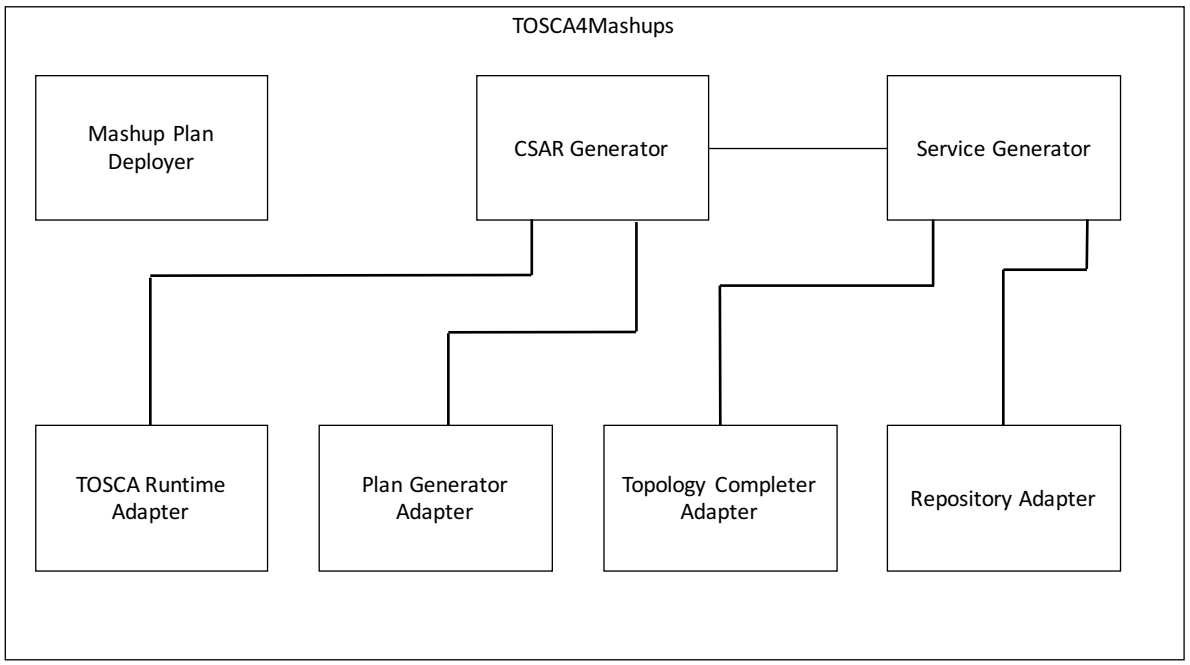

Abbildung 4.2.: Architektur der TOSCA4Mashups-Komponente

Service Template zu erstellen. Der CSAR-Generator übernimmt das Zusammenfügen aller Artefakte, der TOSCA Definitionen und des Provisionierungsplanes zu einer vollständigen, selbst-enthaltenen CSAR-Datei. Der *Mashup Plan Deployer* ist eine optionale Komponente, was in den Kapiteln 4.3.1 und 4.5.1 genauer erläutert wird. Dieser ist in der Lage einen ausführbaren Mashup-Plan in eine passende Ausführungsumgebung zu deployen.

TOSCA Laufzeitumgebung

Die TOSCA Laufzeitumgebung führt die eigentliche Provisionierung der Anwendung aus. Abbildung 4.3 zeigt ein reduziertes Modell einer TOSCA Laufzeitumgebung, das jedoch alle für diese Anwendung nötigen Komponenten enthält. Dabei wurde auf die Möglichkeit einer deklarativen Provisionierung verzichtet und lediglich die Komponenten für eine imperative Provisionierung miteinbezogen. Dies ist ausreichend, da der Plangenerator generisch für jede Topologie einen Provisionierungsplan generieren kann. Der sogenannte *CSAR Processor* führt das Entpacken der CSAR-Datei durch. Der *Definition Manager* verwaltet die aus der CSAR entnommenen TOSCA Definitionen, der *Artifact Manager* die entnommenen Artefakte. Die *Process Engine* ist in der Lage Provisionierungspläne durchzuführen und wird vom *Instance Manager* angestoßen. Davor deployt der *Deploy Manager* den Provisionierungsplan in der Process Engine und alle Implementierungs-Artefakte in der Umgebung an entsprechender Stelle.

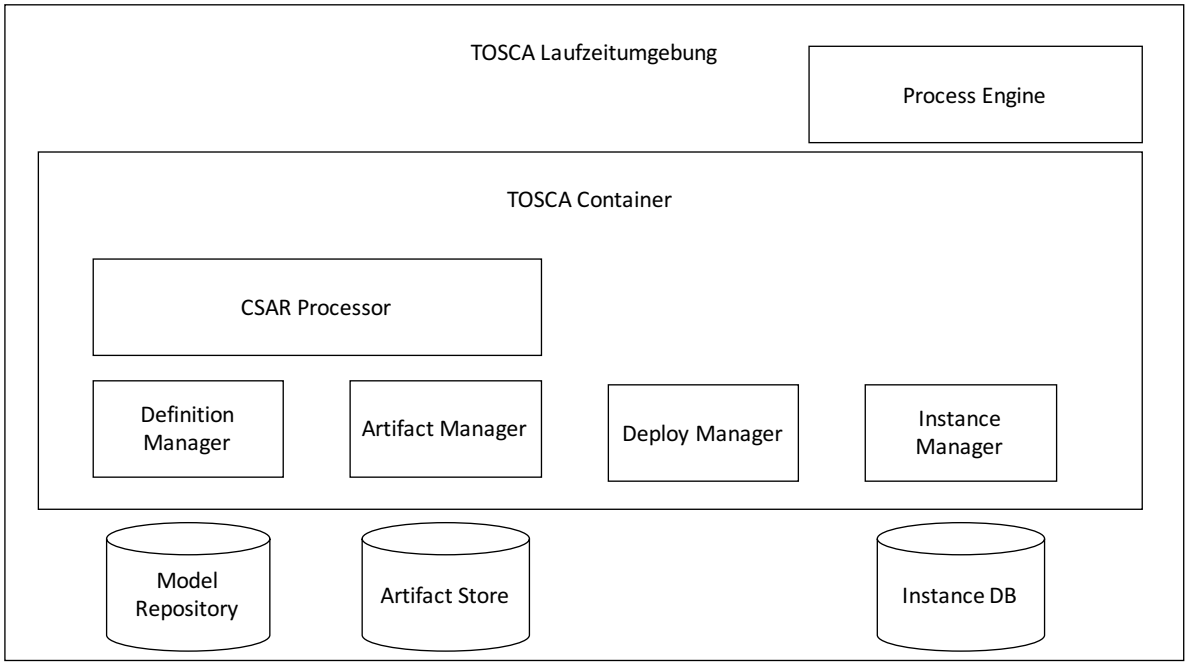

Abbildung 4.3.: Architektur einer TOSCA Laufzeitumgebung [OASa]

Topology Completer

Der Topology Completer generiert aus einem Topology Template, das nur anwendungsspezifische Node Templates enthält, ein vollständige Topologie. Dafür verwendet er die Node Types aus dem Node Type Repository. Grundlage für den Topology Completer bietet die Arbeit von Hirmer et al. [HM16]. Die Methode, die angewendet wird, um eine Topologie zu vervollständigen, wird in Abschnitt 4.3.1 „Topologie vervollständigen“ genauer beschrieben.

Plan Generator

Der Plan Generator ist in der Lage aus einem TOSCA Topology Template einen Provisionierungsplan zu generieren. Der Plan Generator ermöglicht eine imperative Provisionierung auf Grundlage der deklarativen Beschreibung einer Topologie. In dem Artikel „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“ [BBK+14] wird ein Konzept für einen Plangenerator vorgestellt, auf welchem die Plan Generator-Komponente basiert. Dieses wird in Abschnitt 4.3.3 „Schritt 1.3: Plan generieren“ erläutert.

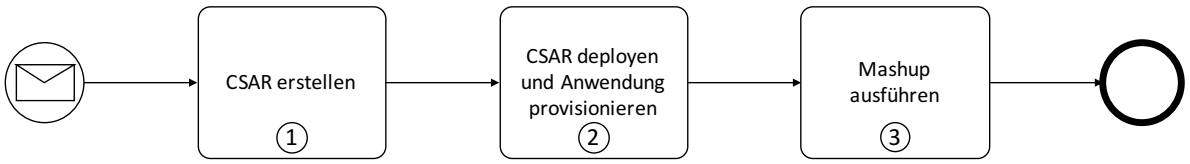

Abbildung 4.4.: Überblick der Methode

Node Type Repository

Das Node Type Repository muss alle Node Types beinhalten, welche für die Topologie benötigt werden. Die TOSCA4Mashups-Komponente sucht hier nach einem geeigneten Node Type für den Mashup-Plan und der Topology Completer nach allen weiteren Node Types, welche für die Topologie benötigt werden. Außerdem müssen alle anderen benötigten TOSCA-Komponenten enthalten sein, wie die Relationship Types, Artifact Types und Node Type Implementations.

Artifact Repository

Im Artifact Repository sind die Deployment- und Implementierungs-Artefakte für alle Node Types im Node Type Repository abgelegt. Die TOSCA4Mashups-Komponente greift darauf zu, um sie zu der CSAR-Datei hinzuzufügen.

4.2. Methode zur automatischen Provisionierung von Data Mashups

Die Methode um eine Data Mashup-Anwendung in einer Cloud-Umgebung zu provisionieren und auszuführen ist in drei Schritte aufgeteilt. Im ersten Schritt wird eine CSAR Datei erstellt, im zweiten Schritt wird diese in der Cloud provisioniert und im dritten Schritt wird das Mashup ausgeführt. Abbildung 4.4 zeigt die drei grundlegenden Schritte der Methode.

4.3. Schritt 1: CSAR erstellen

Der erste notwendige Schritt besteht darin eine vollständige CSAR-Datei zu erstellen. Diese muss alle notwendigen Dateien enthalten, die für die Provisionierung notwendig sind. Die Eingabe von TOSCA4Mashups besteht aus zwei Teilen:

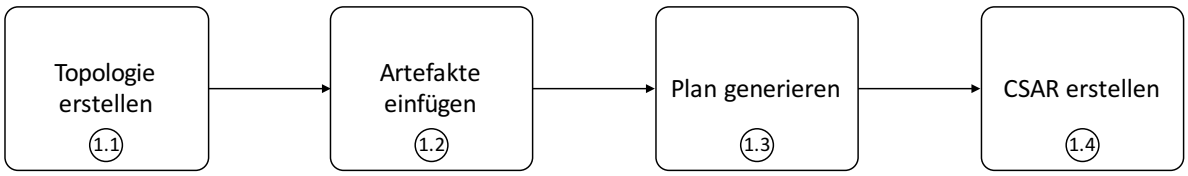

Abbildung 4.5.: Methode, um das Cloud Service Archive zu erstellen

1. Eine Liste mit den Bezeichnern der für die Ausführung des Mashups benötigten Ausführungskomponenten.
2. Ein ausführbarer Mashup-Plan.

Auf Basis der Bezeichner der Ausführungskomponenten muss eine vollständige CSAR-Datei generiert werden, die außer allen benötigten TOSCA-Beschreibungen und Artefakten auch den Mashup-Plan enthält. Um eine CSAR-Datei auf dieser Grundlage zu generieren wurden vier Teilschritte ausgearbeitet. Abbildung 4.5 zeigt eine schematische Übersicht über diese vier Schritte. Die folgenden Abschnitte erläutern jeweils einen der vier Schritte.

4.3.1. Schritt 1.1: Topologie erstellen

Anhand der Ausführungskomponenten muss eine vollständige TOSCA-Topologie erstellt werden. Als Grundlage dafür dienen die Node Types aus dem Node Type Repository. Die Methode, um eine Topologie automatisiert zu generieren wurde in drei Schritte aufgeteilt:

1. Passende Node Types suchen und instanziieren
2. Topologie vervollständigen
3. Mashup-Plan Deployment innerhalb der Topologie

Diese werden im Folgenden ausführlich beschrieben. Auf den letzten Schritt kann auch verzichtet werden, indem der Mashup-Plan durch die TOSCA4Mashups-Anwendung nach der Provisionierung deployt wird, anstatt während dem Provisionierungsvorgang.

Passende Node Types suchen und instanziieren

Für die zu provisionierenden Anwendungskomponenten muss nun jeweils ein zugehöriger Node Type aus dem Node Type Repository ausgesucht werden. Hierfür müssen die Node Types entweder den selben Namen besitzen wie die möglichen Anwendungskomponenten auf oberster Ebene, oder es muss eine injektive Abbildung zwischen beiden Namensräumen bestehen. Eine weitere Möglichkeit wäre es semantisch äquivalente Node Types zu suchen. Die gefundenen Node Types müssen als Node Templates instanziiert und in ein Topology Template eingefügt werden. Für die Vervollständigung der Topologie ist es notwendig, dass

die Requirements aus den zu instanziierenden Node Types entnommen und in die Node Templates eingefügt werden.

Topologie vervollständigen

Die Vervollständigung der Topologie wird durch die Komponente *Topology Completer* durchgeführt. Dafür übergibt TOSCA4Mashups dem Topology Completer ein TOSCA Definitions-Element, welches ein Topology Template mit den anwendungsspezifischen Node Templates enthält. Um eine Topologie zu vervollständigen wird für jedes Requirement in jedem Topology Template nach einem Node Type mit einer Capability gesucht, die das jeweilige Requirement erfüllt. Der Node Type wird dann zu einem Node Template instanziiert und in die Topologie eingefügt, wobei hier wieder die Requirements aus den Node Types übernommen werden. Die beiden Node Templates werden durch ein Relationship Template verbunden und das erfüllte Requirement wird gelöscht. Dadurch kann die Topologie sukzessive, durch das rekursive Verfahren, vervollständigt werden, bis keine Requirements mehr vorhanden sind. Hirmer et al. [HBBL14] zeigen eine Methode, um aus unvollständigen Topology Templates vollständige zu generieren. Das von Hirmer et al. vorgestellte Konzept wird durch diese Arbeit insofern erweitert, als dass aus der vollständigen Topologie im Verlauf des hier vorgestellten Konzeptes ein gesamtes Service Template mit entsprechenden Artefakten generiert wird. Des Weiteren wird aus diesem Service Template eine vollständige CSAR-Datei generiert.

Mashup-Plan Deployment innerhalb der Topologie

Zusätzlich den benötigten Anwendungskomponenten muss der ausführbare Mashup-Plan deployt werden. Dies kann nur geschehen nachdem alle Anwendungskomponenten provisioniert wurden. Eine Möglichkeit besteht darin, den Mashup-Plan ebenfalls durch ein Deployment Artifact zu modellieren. Ist dies der Fall, wird in Schritt 4.3.1 „Passende Node Types suchen und instanziieren“ nicht nach passenden Node Types für die Ausführungskomponenten, sondern nach dem Node Type für den entsprechenden Mashup-Plan gesucht. Der Plan selbst kann als Deployment Artifact in einer Datei in der CSAR liegen und wird im Service Template durch ein Artifact Template repräsentiert. Eine andere Möglichkeit wäre es, den Mashup-Plan in einer externen Datenbank abzulegen. Für jeden Mashup Plan Node Type besteht zusätzlich ein Implementation Artifact. Dieses muss für jede Art von Ausführungskomponente angepasst werden. Es deployt den in dem Deployment Artifact abgelegten Mashup Plan in der jeweiligen Ausführungskomponente.

Eine alternative Möglichkeit den Mashup-Plan zu deployen wird in Abschnitt 4.5.1 beschrieben. Diese besteht darin, den Mashup-Plan nach der Provisionierung der Anwendung in der Cloud durch TOSCA4Mashups zu deployen.

4.3.2. Schritt 1.2: Artefakte einfügen

Nachdem die Topologie vervollständigt wurde können die entsprechenden Implementierungs- und Deployment-Artefakte angefügt werden. Diese können aus dem Artifact Repository entnommen und in die CSAR an entsprechender Stelle eingefügt werden. In den Artifact Templates im Service Template wird durch das TOSCA-Element „ArtifactReference“ der Pfad für das jeweilige Artefakt angegeben.

4.3.3. Schritt 1.3: Plan generieren

Die vollständige Topologie kann nun an den Plan Generator übergeben werden, welcher in der Lage ist anhand eines Service Templates einen ausführbaren Provisionierungsplan zu erstellen. Breitenbücher et al. [BBK+14] beschreiben einen Algorithmus, um aus einer TOSCA-Topologie einen ausführbaren Provisionierungsplan zu generieren. Dabei wird aus dem Topology Template ein gerichteter Provisionierungsgraph, und aus diesem ein Provisionierungsplanskelett generiert. Anhand des Skelettes und der Informationen des Service Teplates kann nun ein Provisionierungsplan in einer beliebigen Ausführungssprache automatisiert generiert werden.

4.3.4. Schritt 1.4: CSAR erstellen

Alle von der TOSCA Ausführungsumgebung für die Provisionierung benötigten Komponenten müssen in einer CSAR-Datei verpackt werden. Dazu gehört das Service Template, der Provisionierungsplan, alle Typdefinitionen, sowie alle benötigten Artefakte aus dem Artifact Repository. Diese werden alle in ein Dateisystem gelegt, das dem in Kapitel 2.3.3 „Cloud Service Archive (CSAR)“ entspricht. Das Dateisystem wird dann zu einer ZIP-Datei verpackt, die mit der Endung „.csar“ versehen wird.

4.4. Schritt 2: CSAR deployen und Anwendung provisionieren

in Abschnitt 4.3 wurde gezeigt wie eine CSAR-Datei für jede Art von Mashup automatisiert erstellt werden kann. Die TOSCA Laufzeitumgebung führt nun die eigentliche Provisionierung des Data Mashups durch. Dieses Vorgehen kann in zwei Schritte eingeteilt werden, welche im Folgenden erläutert werden. Abbildung 4.6 zeigt die beiden Teilschritte, um die Anwendung in der Cloud zu provisionieren.

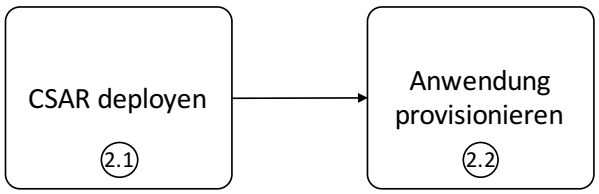

Abbildung 4.6.: Methode, um die Anwendung zu provisionieren

4.4.1. Schritt 2.1: CSAR deployen

Die vollständige CSAR-Datei wird an die TOSCA Laufzeitumgebung übergeben. Hier wird sie entpackt und der Provisionierungsplan sowie die Artefakte werden an entsprechenden Stellen abgelegt. Der CSAR Processor entnimmt alle TOSCA Definitionen aus der CSAR-Datei und übergibt diese dem Definition Manager. Der Definition Manager speichert alle Definitionen im Model Repository, um spätere Zugriffe darauf zu gewährleisten. Des Weiteren entnimmt der CSAR Processor alle Implementierungs- und Deployment-Artefakte aus der CSAR-Datei und übergibt sie dem Artifact Manager. Dieser speichert alle Artefakte im Artifact Store. Daraufhin deployt der Deploy Manager alle Implementierungs-Artefakte an der richtigen Stelle in der Umgebung, sowie den Provisionierungsplan in der Process Engine. Der Instance Manager führt die Erstellung von Instanzen der Cloud-Anwendung durch, indem er den Process Manager aufruft, um den Provisionierungsplan auszuführen.

4.4.2. Schritt 2.2: Anwendung provisionieren

Um die virtuelle Maschine aufzusetzen und alle Anwendungskomponenten zu installieren wird der Provisionierungsplan ausgeführt. Das führt die Process Engine der TOSCA Laufzeitumgebung durch, indem sie die Implementierungs-Artefakte in entsprechender Reihenfolge und mit entsprechenden Eingaben ausführt. Der Deploy Manager der TOSCA Laufzeitumgebung übergibt den Provisionierungsplan der Process Engine und deployt alle Implementierungs-Artefakte an entsprechender Stelle in der Umgebung. Der Instance Manager stoßt die Process Engine an, damit diese den Provisionierungsplan ausführt. Der Plan muss dabei mit den Node Templates beginnen, von denen keine Relationship Templates ausgehen, da diese in der Topologie auf unterster Ebene liegen. Dies ist notwendig, da jede Anwendung nur provisioniert werden kann, wenn die in der Topologie darunter liegende Anwendung bereits provisioniert wurde. Mehrere Stacks einer Topologie können parallel provisioniert werden, solange sie nicht voneinander abhängen. Abbildung 4.7 zeigt auf der linken Seite eine schematische Darstellung des Provisionierungsplanes in BPMN-Notation zu der Topologie auf der rechten Seite. Bei diesem Beispiel können die beiden Anwendungs-Stacks parallel provisioniert werden, was sich im Provisionierungsplan widerspiegelt. Der linke Strang des Planes provisioniert dabei den linken Anwendungs-Stack der Topologie, der rechte den rechten Anwendungs-Stack. Es werden zuerst die beiden virtuellen Maschinen

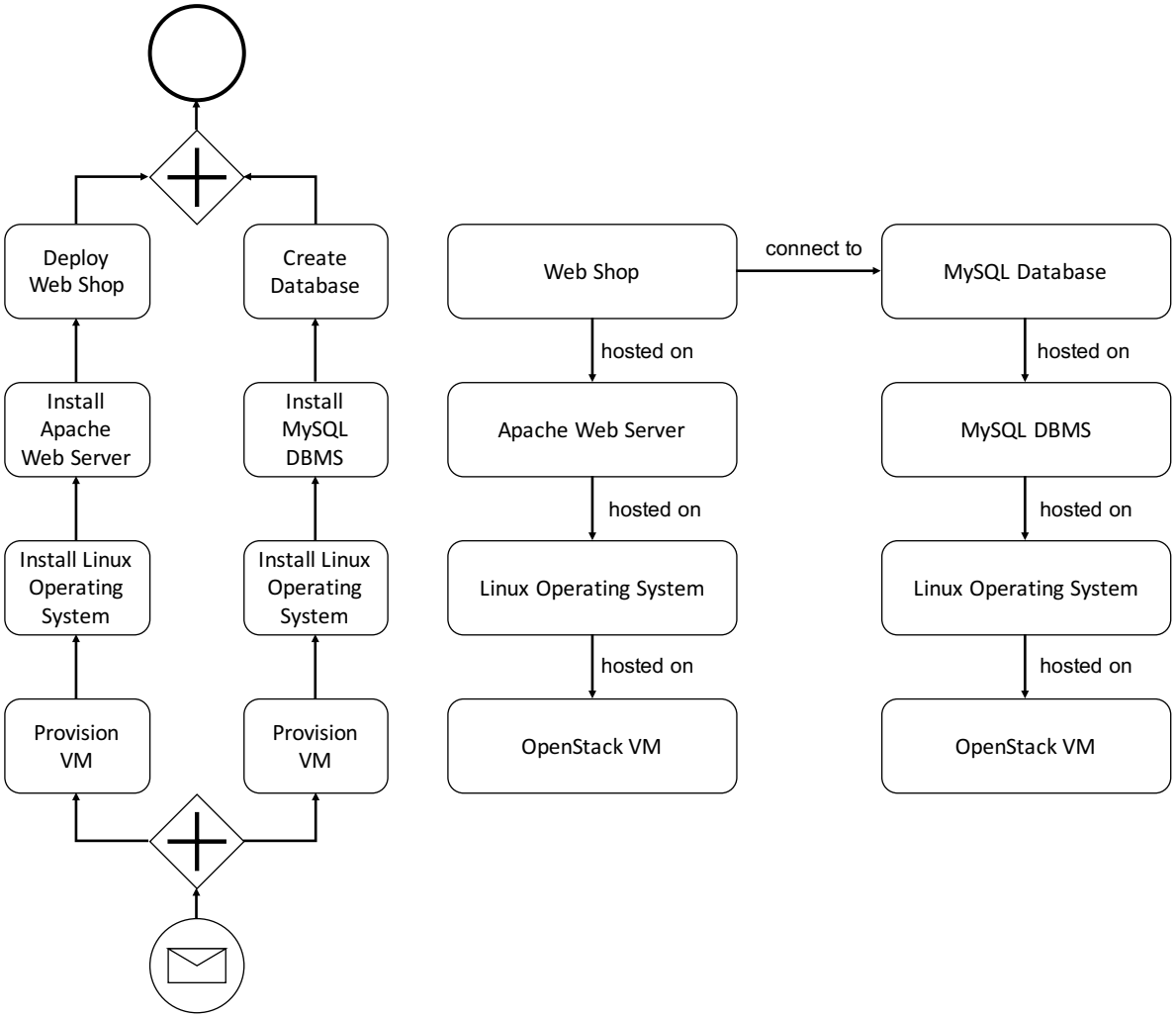

Abbildung 4.7.: Anwendung provisionieren. Links der Provisionierungsplan, rechts die Topologie.

in der Cloud aufgesetzt. Im Anschluss wird auf jeder Maschine ein Linux Betriebssystem installiert. Diese beiden Schritte können je nach Cloud-Umgebung in einem Schritt im Plan geschehen. Nun wird auf dem einen Betriebssystem ein Apache Web Server und auf dem anderen eine MySQL-Datenbank installiert. Auf dem Webserver wird ein Web Shop aufgesetzt, in dem Datenbanksystem eine Datenbank für den Webserver angelegt. Die Verbindung mit der Beschriftung „connect to“ zwischen dem Web Shop und der MySQL Database spiegelt sich im Plan nicht wider, da sie durch die Web Shop-Anwendung durchgeführt wird. Der Vorgang der Provisionierung wird von einer typischen TOSCA Laufzeitumgebung, entsprechend dem TOSCA Primer [OASa], nicht automatisch ausgeführt. In diesem Fall muss TOSCA4Mashups, nachdem die CSAR-Datei in der TOSCA Laufzeitumgebung deployt wurde, den Provisionierungsprozess anstoßen.

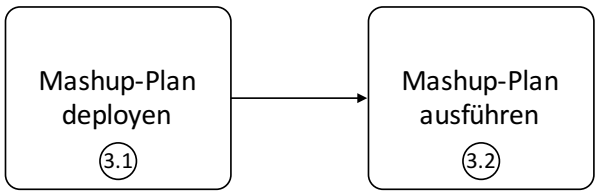

Abbildung 4.8.: Mashup ausführen

4.5. Schritt 3: Mashup ausführen

Bis zu diesem Zeitpunkt wurden alle Anwendungskomponenten in eine Cloud-Umgebung provisioniert, die nötig sind, um den Mashup auszuführen. Eventuell wurde auch schon der Mashup-Plan entsprechend deployt. Falls nicht, muss dies noch geschehen, außerdem muss dieser noch ausgeführt werden. Dafür wurden zwei methodische Schritte ausgearbeitet, die in diesem Abschnitt erläutert werden. Abbildung 4.8 zeigt zwei Schritte die nötig sind, um den Mashup auszuführen. Auf den Schritt „Mashup-Plan deployen“ kann verzichtet werden wenn der Schritt „Mashup-Plan Deployment innerhalb der Topologie“, der in Kapitel 4.3.1 erläutert wird, angewendet wurde.

4.5.1. Schritt 3.1: Mashup-Plan Deployment außerhalb der Topologie

Eine alternative Möglichkeit dazu den Mashup-Plan in die Topologie zu integrieren ist es diesen nach Provisionierung der Anwendungskomponenten ohne Verwendung der TOSCA-Laufzeitumgebung zu deployen. Dafür existiert in der Architektur von TOSCA4Mashups in Abbildung 4.2 der Mashup Plan Deployer. Dieser muss jedoch für jede Art von zu provisionierenden Anwendungen um eine API-Anbindung erweitert werden. Des Weiteren müssen die jeweiligen Mashup-Komponenten über eine API oder eine Möglichkeit verfügen Mashups in Form von Dateien über das Dateisystem zu deployen.

Da das Erweitern von TOSCA4Mashups einen größeren Aufwand darstellt, als die für das Deployen des Mashup-Planes nötigen Node Types mit zugehörigem Implementation Artifact zu implementieren, wird die Methode in Abschnitt 4.3.1 empfohlen.

4.5.2. Schritt 3.2: Mashup-Plan ausführen

Der TOSCA-Standard enthält keine Möglichkeit die Ausführung von provisionierten Anwendungen zu modellieren. Die Ausführung des Mashups kann durch zwei Möglichkeiten durchgeführt werden. Entweder wird sie durch ein weiteres Node Template im Topology Template modelliert und durch ein entsprechendes Implementierungs-Artefakt durchgeführt, oder sie wird nach der Provisionierung durch die TOSCA4Mashups-Komponente

durchgeführt. Hierfür müsste jedoch wieder TOSCA4Mashups für jede Art von Mashup erweitert werden. Anstatt ein eigenes Node Template und Implementierungs-Artefakt für das Anstoßen der Ausführung zu implementieren, könnte dies auch das Implementation Artifact des Mashup-Planes übernehmen. Dieses könnte einfach nach dem Deployen des Mashup-Planes dessen Ausführung starten.

Da dies den geringsten Aufwand hinsichtlich Erweiterbarkeit darstellt, wird diese Möglichkeit der Mashup-Ausführung empfohlen.

Der letzte Schritt des Mashup-Plans sollte es sein dem Benutzer das Ergebnis zurückzuführen. Dies kann einfach dadurch geschehen, dass das Ergebnis in eine Datenbank geschrieben wird oder dem Benutzer über HTTP visualisiert wird, indem es einem Webserver übergeben wird.

5. Implementierung

Um die Funktionalität des Konzeptes in Kapitel 4 zu zeigen wurde ein Prototyp implementiert. Als Grundlage für die Implementierung dient der Anwendungsfall aus Kapitel 1.1. Abbildung 5.1 zeigt die Topologie, die für einen Node-RED Workflow provisioniert werden muss. Der Knoten „NodeREDPlan“ modelliert dabei das Topology Template für einen ausführbaren Mashup-Plan für Node-RED. Dieser wird in JavaScript Object Notation (JSON) an die TOSCA4Mashups-Komponente übergeben, womit der Ablauf der Implementierung beginnt. Das resultierende TOSCA Service Template findet sich in Anhang A in den Listings A.1, A.2, A.3, A.4, A.5 und A.6.

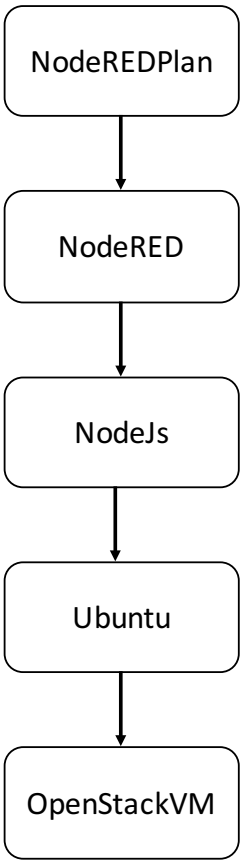

Abbildung 5.1.: Beispieltopologie für Node-RED

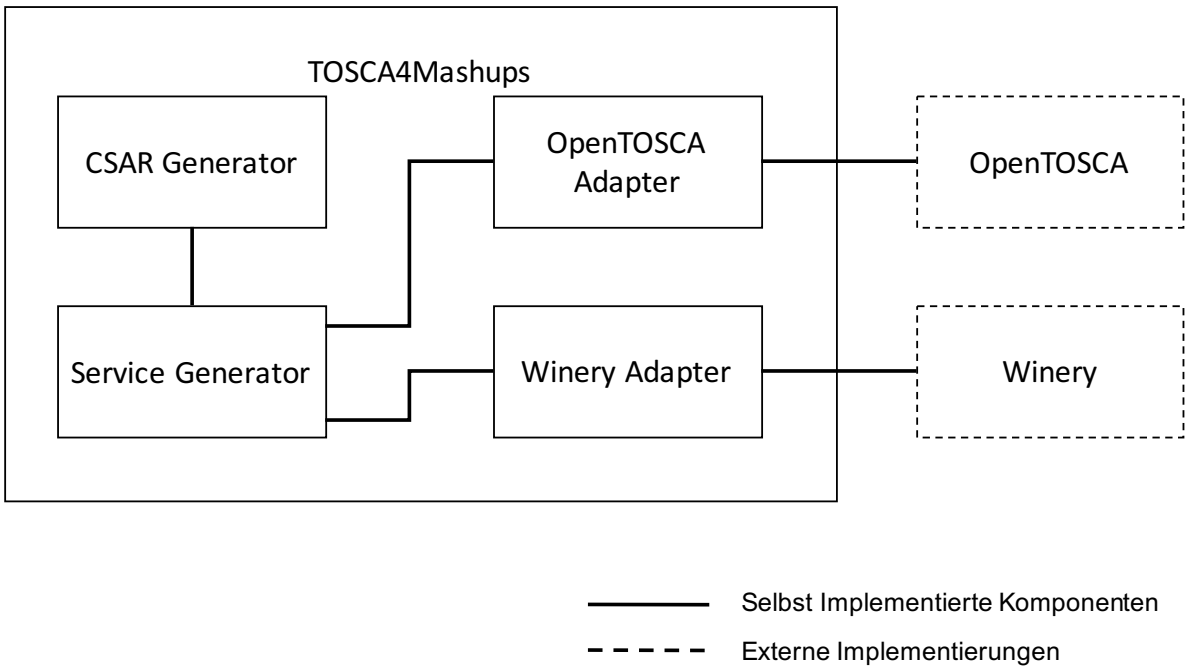

Abbildung 5.2.: Entwurf TOSCA4Mashups

Um die Implementierung zu erläutern wird zunächst in Abschnitt 5.1 ein auf den Anwendungsfall und die ausgewählten Komponenten zugeschnittener Entwurf gezeigt. In Abschnitt 5.2 wird die Umsetzung dieses Entwurfes erklärt. Abschnitt 5.3 beschäftigt sich mit Fällen, welche nicht durch diese Implementierung abgedeckt werden, in Abschnitt 5.4 wird die gesamte Implementierung bewertet.

5.1. Entwurf

Abbildung 5.2 zeigt einen Überblick über die Architektur der implementierten Anwendung. TOSCA4Mashups und die darin enthaltenen Komponenten wurden im Zuge dieser Bachelorarbeit entworfen und implementiert. Im Folgenden werden die einzelnen Komponenten genauer erläutert.

5.1.1. OpenTOSCA

Als TOSCA Laufzeitumgebung dient OpenTOSCA, eine Implementierung des Institutes für Architektur von Anwendungssystemen der Universität Stuttgart. OpenTOSCA enthält unter anderem einen Container für CSAR-Dateien und eine BPEL Engine für die Ausführung des

Provisionierungsplanes. Eine genaue Beschreibung der Architektur von OpenTOSCA findet sich in [BBH+13].

5.1.2. Winery

Winery ist ein TOSCA Topology Modellierungs-Tool[1]. Außer einer graphischen Oberfläche zur Modellierung der Topologien enthält es ein Node Type Repository, einen Topology Completer und einen Plan Generator. Alle Komponenten lassen sich auch über eine REST API ansprechen. Der Plan Generator generiert aus einer Topologie einen Provisionierungsplan in BPEL.

5.1.3. TOSCA4Mashups

Die Komponente TOSCA4Mashups wurde im Zuge dieser Arbeit implementiert. Dafür wurden die Programmiersprache Java und verschiedene Bibliotheken für HTTP und JSON verwendet. TOSCA4Mashups ist die Implementierung für die gleichnamige Komponente im Konzept in Kapitel 4.

OpenTOSCA Adapter

Der OpenTOSCA Adapter stellt eine Verbindung mit der OpenTOSCA API her, um die generierte CSAR-Datei hochzuladen und die Provisionierung anzustoßen. Dabei handelt es sich um eine REST-API die sich über HTTP ansprechen lässt.

Winery Adapter

Der Winery Adapter übernimmt die Kommunikation mit der Winery, die den größten Anteil an Kommunikation zwischen den Komponenten einnimmt. Winery bietet ebenfalls die Möglichkeit eine REST-API über HTTP zu bedienen. Dabei werden Metainformationen zu den in der Winery gespeicherten Definitionen im JSON-Format übertragen.

Sowohl der Winery Adapter als auch der OpenTOSCA Adapter wurde eingeführt, um die Verbindung zwischen TOSCA4Mashups, OpenTOSCA und Winery zu abstrahieren. Dies hat den Vorteil, dass bei einem Austausch der TOSCA Laufzeitumgebung lediglich diese beiden Komponenten neu implementiert werden müssen.

[1] https://projects.eclipse.org/projects/soa.winery

Service Generator

Das Instanziieren der Node Types und das Zusammenfügen der Node Templates zu einem Service Template übernimmt der Service Generator.

CSAR Generator

Nach der Generierung des Service Templates erstellt der CSAR Generator eine vollständige CSAR-Datei.

5.2. Umsetzung

Wie die TOSCA4Mashups-Komponente des Entwurfs umgesetzt und implementiert wurde wird in diesem Abschnitt erläutert. Zunächst werden die für die Implementierung verwendeten Technologien erläutert, im Anschluss wird der Ablauf einer automatischen Provisionierung eines Node-RED-basierten Data Mashups dargestellt.

5.2.1. Technologien

Als Programmiersprache für die Implementierung von TOSCA4Mashups wurde Java gewählt. Die Kommunikation zwischen TOSCA4Mashups, OpenTOSCA und der Winery funktioniert mit HTTP, wodurch diese auf unterschiedlichen Maschinen ausgeführt werden können. So kann zum Beispiel OpenTOSCA in der Cloud-Umgebung selbst, und Winery mit dem Node Type Repository auf einer lokalen Machine ausgeführt werden. OpenTOSCA basiert auf einem Tomcat-Server, einem Jetty-Server für den Container und einer WSO2BPS[2] Workflow Engine für die Ausführung der Provisionierungspläne. Als Implementierungs-Artefakte dienen zwei WAR-Dateien. Die eine führt die Provisionierung einer virtuellen Maschine auf einer OpenStack-Umgebung aus, die andere ist in der Lage beliebige Skripte auf einem Linux Betriebssystem auszuführen. Auch diese wurden im Rahmen von OpenTOSCA von der Universität Stuttgart entwickelt. Zur Verarbeitung von TOSCA wurden die beiden Bibliotheken JAXB[3] und W3C DOM[4] verwendet. Als Cloud-Umgebung wird für die Implementierung OpenStack verwendet.

[2]http://wso2.com/products/business-process-server/
[3]https://jaxb.java.net/
[4]https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html

5.2.2. Ablauf

Im Folgenden wird der Ablauf einer Provisionierung beschrieben:

1. Node Types von Winery abfragen:

   Für jede benötigte Anwendungskomponente wird der entsprechende Node Type von der Winery benötigt. Dafür müssen erst die Namen aller Node Types mit ihren zugehörigen Namespaces abgefragt werden. Wird für jede benötigte Anwendungskomponente ein gleichnamiger Node Type gefunden, können deren TOSCA-Definitionen von der Winery abgefragt werden.

2. Node Templates bauen und in Topology Template einfügen:

   Im nächsten Schritt werden alle Node Types zu Node Templates instanziiert. Dafür wird für jeden benötigten Node Type ein Node Template-Element erstellt und die Attribute „id“ und „name“ entsprechend dem Namen des Node Types gesetzt. Außerdem müssen alle Requirements und Capabilities aus dem Node Type entnommen und in das Node Template eingebaut werden. Die fertigen Node Templates werden in ein Topology Template eingefügt.

3. Topologie an Winery schicken und vervollständigen lassen:

   Bis hier besteht das Topology Template nur aus den anwendungsspezifischen Node Templates. Um die Topologie zu vervollständigen wird die Unvollständige an den Topology Completer der Winery geschickt. Dieser verwendet die im Node Type Repository der Winery liegenden Node Types, um die Topologie zu vervollständigen und speichert sie danach innerhalb eines Service Templates ab. Über die REST-API kann dieses Service Template wieder abgerufen werden.

4. Node Type Implementations und Artifacts einfügen:

   Nachdem die Topologie vervollständigt wurde, müssen nun alle nötigen Artifact Templates und Node Type Implementations in das Service Template eingefügt werden. Dafür werden wieder alle Namen und Namespaces der in der Winery abgelegten Artifact Types abgefragt. Aus diesen werden die ausgesucht, die den benötigten Node Templates entsprechen. Jetzt können deren Definitionen der Winery entnommen werden. Aus diesen Artifact Types können dann die Artifact Templates und Node Type Implementations erstellt und in das Service Template eingefügt werden.

5. Plan einfügen:

   Ein weiteres wichtiges Element in einem Service Template ist das TOSCA Plan-Element. Dieses enthält eine Referenz auf den in der CSAR-Datei liegenden Provisionierungsplan, damit die TOSCA-Laufzeitumgebung darauf zugreifen kann.

6. Properties einfügen:

   Die Node Templates müssen noch mit entsprechenden Properties bestückt werden. In dem Anwendungsfall handelt es sich dabei zum Beispiel um die OpenStack-Zugangsdaten im OpenStackVM Node Template. Außerdem wurde das gesamte Skript, um Node-RED zu installieren und den Mashup-Plan zu deployen als Property in das Node Template, das das Betriebssystem repräsentiert eingefügt, was lediglich der Einfachheit dient. Dieses Skript entnimmt OpenTOSCA dann bei der Provisionierung, übergibt es der WSO2BPS, welche dieses dann dem Implementierungs-Artefakt übergibt, das es auf dem Betriebssystem der virtuellen Maschine ausführt.

7. Service Template in Winery ablegen:

   Nachdem alle weiteren nötigen TOSCA-Elemente eingefügt wurden kann das Definitions-Element wieder in der Winery gespeichert werden. Dies geschieht wieder über die REST-API der Winery.

8. Provisionierungsplan generieren:

   Der Plan sollte durch den Plan Generator der Winery erstellt werden. Da dieser jedoch noch nicht funktioniert, muss er der Provisionierungsplan vorher von Hand erstellt und in die Winery geladen werden. Dann wird der Plan mit allen nötigen Dateien an der entsprechenden Stelle in der CSAR-Datei abgelegt.

9. CSAR an OpenTOSCA schicken:

   OpenTOSCA bietet die Möglichkeit eine CSAR-Datei direkt aus der Winery zu laden. Dafür muss ein entsprechender Aufruf an die API von OpenTOSCA durch den OpenTOSCA Adapter durchgeführt werden.

10. Anwendung provisionieren:

    Zum Schluss muss OpenTOSCA noch dazu aufgefordert werden die Anwendung auf Grundlage der CSAR-Datei zu provisionieren.

5.3. Sonderfälle

Bisher können mit der Implementierung nur Anwendungen provisioniert werden, die sich durch einen einzigen Shell Shell-Skript-Befehl installieren lassen. Anwendungen die zusätzliche Implementierungs-Artefakte benötigen werden nicht unterstützt. Da auch nur der Shell-Skript-Befehl zur Installation von Node-RED implementiert wurde, lassen sich nur solche Mashups provisionieren, die auf Node-RED basieren.

Da der Plan Generator der Winery nur beschränkt funktioniert, muss der Provisionierungsplan bisher im Voraus erstellt und in Winery importiert werden. Deswegen können auch

lediglich Anwendungen mit der selben Topologie wie in Abbildung 5.1 provisioniert werden.

5.4. Evaluation

Aufgrund der starken Einschränkung auf Node-RED-Mashups ist die Implementierung nicht sehr brauchbar. Allerdings beweist sie die Funktionalität des Konzeptes, da der Entwurf auf alle Arten von Mashups erweiterbar ist.

OpenTOSCA hat sich als gute Wahl für eine TOSCA Laufzeitumgebung herausgestellt, da es sehr gut mit der Winery harmoniert. Diese ist wiederum für diese Implementierung dank des Topology Completers und des Plan Generators von großem Vorteil.

Da die meiste Zeit beim gesamten Provisionierungsvorgang die Kommunikation zwischen den Komponenten und die Ausführung des Provisionierungsplanes in Anspruch nimmt, wurden keine Zeitmessungen der reinen CSAR-Generierung gemacht.

6. Optimierungen

Sowohl für das Konzept als auch für die Implementierung, die in dieser Arbeit vorgestellt wurden, ergeben sich viele Möglichkeiten diese zu verbessern. In diesem Kapitel werden Möglichkeiten erörtert das Konzept oder die Implementierung zu verbessern. Dazu gehört die Wiederverwendung bereits generierter Services, der Umgang mit auftretenden Fehlern, Anpassungen der Topologie an den jeweiligen Anwendungsfall sowie Anpassung an verschiedene Cloud-Provider.

6.1. Wiederverwendung bereits generierter Services

Je nach Umfeld des Anwendungsfalles kann es sein, dass oft verschiedene Mashup-Pläne auf den gleichen Ausführungskomponenten ausgeführt werden sollen. In diesem Falle macht es wenig Sinn bei jeder neuen Provisionierung alle Schritte zur Generierung von CSAR-Dateien durchzuführen. Bereits fertige Data Mashup-CSARs könnten für Weitere mit den selben Komponenten wiederverwendet werden. Wenn nur der Workflow aber nicht die Ausführungskomponenten geändert werden, muss nur der Workflow deployt werden. In diesem Fall bietet sich die Möglichkeit aus Abschnitt 4.5.1 des Mashup-Plan Deployens an, bei der dieser nach Provisionierung der Anwendung deployt wird. Auf diese Weise kann die gesamte CSAR-Datei wiederverwendet werden, und lediglich ein anderer Mashup-Plan nach der Provisionierung deployt werden.

6.2. Fehlerbehandlung

Eine weitere Möglichkeit die Anwendung zu optimieren ist es, einen robusten Umgang in Bezug auf mögliche Fehlerquellen einzubauen. Dies ist insbesondere in der Hinsicht relevant, da es sich bei den Benutzern nicht um IT-Experten handelt. Einige Möglichkeiten werden im Folgenden behandelt.

6.2.1. Keine passenden Node Types

Der erste Fehler, welcher bei der Ausführung auftreten kann ist, dass für den zu provisionierenden Mashup kein passender Node Type im Node Type Repository gefunden wird. Um dies zu verhindern ist es sinnvoll das Mapping auf die Anwendungskomponenten den im Node Type Repository vorhandenen Node Types anzupassen. Wenn der Benutzer die Wahl zwischen verschiedenen Ausführungsmustern bekommt, sollten nur solche zur Verfügung stehen, die auch im Node Type Repository durch Node Types hinterlegt sind.

6.2.2. Fehler beim Installieren der Anwendungskomponenten

Die schwerwiegendste Fehlerquelle liegt in der Installation der Ausführungskomponenten. Hier auftretende Fehler sind für den Benutzer kaum erkennbar, da sie sich eventuell nur in den Log-Dateien der TOSCA-Laufzeitumgebung zeigen. Solche Fehler wären durch einen Domain-Experten aber auch nicht behebbar. Aus diesem Grunde wäre es eine wichtige Optimierung solche Fehler zu erkennen, und im Zweifelsfall die Provisionierung mit einer Fehlermeldung abzubrechen.

6.3. Anpassung der Topologie

Beim Aufsetzen einer virtuellen Maschine in einer Cloud-Umgebung bestehen meistens verschiedene Wahlmöglichkeiten bezüglich der Leistung und der Speicherkapazität. Bei Mashups die aus mehreren Anwendungskomponenten, wie zum Beispiel einem Webserver und einer Datenbank, bestehen, gibt es außerdem die Möglichkeit diese auf einer oder mehrerer virtuellen Maschinen zu provisionieren. Diese Entscheidungen sind kosten- und leistungsrelevant, und sollten deshalb dem Betreiber des Mashups überlassen werden. Da es sich bei dem Benutzer jedoch um einen Domain-Experten handelt, und nicht um einen IT-Spezialisten, können diesem solche Entscheidungen nicht überlassen werden. Eine Möglichkeit wäre es den Benutzer bei der Modellierung des Mashups zwischen einer kosten- oder leistungseffizienten Ausführung wählen zu lassen. Abbildung 6.1 zeigt eine alternative Topologie zu der in Abbildung 2.2, bei welcher der Webserver und die MySQL-Datenbank auf einer gemeinsamen virtuellen Maschine angesiedelt sind. Diese Art der Provisionierung kann je nach Cloud-Provider günstiger ausfallen, jedoch auch weniger leistungseffizient und weniger skalierbar sein.

Außer der Topologie selbst kann auch die Wahl der virtuellen Maschine an den Anwendungsfall angepasst werden. Dabei könnten ebenso verschiedene Kriterien wie zum Beispiel der Grad an benötigter Effizienz und der Durchsatz der anfallenden Daten auf eine virtuelle Machine mit einer bestimmten Rechenleistung abgebildet werden.

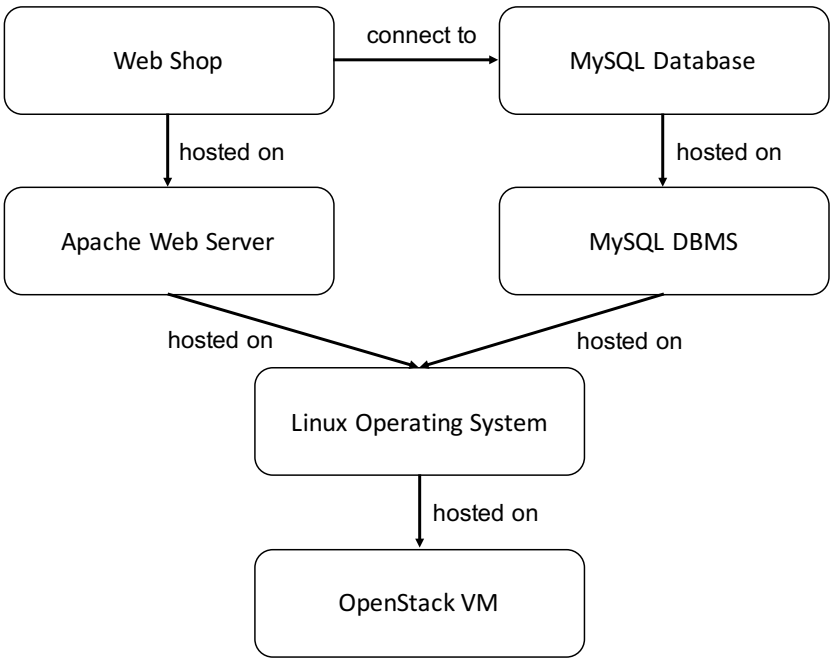

Abbildung 6.1.: Alternative Topologie zu Abbildung 2.2

6.4. Verwendung anderer Cloud-Anbieter

Die Optimierungsmöglichkeit in Abschnitt 6.4.1 in diesem Kapitel bezieht sich speziell auf die Implementierung, da die Anwendung konzeptionell auf allen möglichen Cloud-Architekturen mit entsprechenden Anbindungsmöglichkeiten funktioniert.

6.4.1. Ausführung auf anderen Cloud-Architekturen

Außer der in der Implementierung verwendeten Cloud-Architektur OpenStack ist es noch möglich andere wie zum Beispiel Amazon EC2[1] oder Google Cloud Platform[2] zu verwenden. Dafür müsste lediglich OpenTOSCA um eine Anbindung an die jeweils andere Cloud Architektur erweitert werden, wobei speziell Amazon EC2 schon von OpenTOSCA unterstützt wird. Eventuell müssen auch nur die Implementierungs-Artefakte an die andere Architektur angepasst werden, die die Provisionierung der virtuellen Maschine und des Betriebssystems ausführen.

[1] https://aws.amazon.com/de/ec2/
[2] https://cloud.google.com/

6.4.2. Anpassung an den Cloud-Provider

Cloud-Anbieter wie Amazon Web Services und IBM Bluemix[3] bieten neben IaaS-Angeboten auch verschiedene PaaS- und SaaS-Angebote, wie zum Beispiel Webserver und Datenbanken. Unter Verwendung solcher Cloud-Anbieter wäre es vorteilhaft bestehende Angebote zu verwenden, anstatt diese selbst aufzusetzen, da diese durch den Anbieter verwaltet werden und somit eine höhere Robustheit besitzen. Um solche Anwendungen verwenden zu können, wäre es möglich spezielle TOSCA Node Types mit entsprechenden Implementierungs-Artefakten zu entwerfen. Die Implementierungs-Artefakte müssten dann lediglich auf die API des jeweiligen Cloud-Providers zugreifen, um entsprechende Anwendungen zu provisionieren.

In den nächsten beiden Abschnitten wird untersucht, wie sich TOSCA4Mashups an die beiden Cloud-Provider IBM Bluemix und Amazon Web Services anpassen lässt, um innerhalb dieser Cloud-Umgebungen Mashups effizienter und robuster zu provisionieren.

Deklarative OpenStack-Implementierung

Neben der imperativen OpenStack-Implementierung, die in der Implementierung dieser Arbeit verwendet wird, existiert noch die deklarative OpenStack Implementierung. Um Anwendungen in dieser automatisiert zu provisionieren kann OpenStack Heat verwendet werden. Die Anpassung des in dieser Bachelorarbeit vorgestellten Konzeptes an die deklarative OpenStack-Implementierung wäre zum Beispiel möglich, indem man das TOSCA Service Template in Heat übersetzt. Für diesen Zweck existiert ein OpenStack-Projekt[4], welches unter der Apache 2-Lizenz[5] steht. Dieses Tool ist in der Lage TOSCA Templates in Heat Orchestration Templates umzuwandeln. Daraus ergeben sich jedoch wieder die Nachteile einer deklarativen Beschreibung; es sind nur solche Anwendungen verwendbar, die von Heat erkannt werden. Wenn die von TOSCA4Mashups generierten Topology Templates nur zum Teil in ein Heat Orchestration Template übersetzt werden, ist es möglich von den Vorteilen beider Möglichkeiten der Provisionierung zu profitieren.

Amazon Web Services

Ein weiteres Beispiel für einen Cloud-Provider ist Amazon Web Services. Um Anwendungen automatisiert in der AWS Cloud-Umgebung zu provisionieren existiert CloudFormation. Beim Schreiben dieser Bachelorarbeit konnte keine Arbeit zu einer Übersetzung von TOSCA in eine CloudFormation-Beschreibung gefunden werden. Eine andere Möglichkeit,

[3] https://www.ibm.com/cloud-computing/bluemix/
[4] https://pypi.python.org/pypi/heat-translator
[5] https://www.apache.org/licenses/LICENSE-2.0

die Provisionierung von Anwendungen an einen speziellen Cloud-Provider, wie zum Beispiel AWS, anzupassen, ist es spezielle Implementierungs-Artefakte für diesen zu schreiben. Die Implementierungs-Artefakte können über die API des Providers die benötigten SaaS-Angebote provisionieren lassen.

IBM Bluemix

In der Cloud-Umgebung Bluemix von IBM wird Cloud Foundry als PaaS-Plattform verwendet. Für die automatisierte Provisionierung mit Cloud Foundry können sogenannte *Application Manifest*-Dateien[6] erstellt werden. Über eine Transformierung zwischen TOSCA Templates und Application Manifest-Dateien wurde ebenfalls nichts gefunden. Diese wäre dennoch denkbar.

[6]https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept vorgestellt, das es ermöglicht Data Mashups automatisiert in einer Cloud-Umgebung zu provisionieren und auszuführen. Dieses Konzept wurde in drei Schritte unterteilt, welche selbst in weitere Einzelschritte aufgeteilt wurden. Im Anschluss wurde eine Implementierung des Konzeptes vorgestellt, das die von der Universität Stuttgart entwickelte TOSCA-Laufzeitumgebung OpenTOSCA verwendet. OpenTOSCA wurde unter anderem wegen des großen Toolsupportes gewählt. Dazu gehört auch Winery, eine graphische Oberfläche zur Modellierung von TOSCA-Topologien, welche außer einem Node Type Repository auch einen Plan Generator und einen Topology Completer enthält. Die Funktionalität des Konzeptes wurde durch die Implementierung eines Prototypen gezeigt. Desweiteren wurden mehrere Möglichkeiten erörtert um sowohl das Konzept, als auch die Implementierung zu verbessern. Dazu gehört unter anderem die Anpassung der Topologie an den jeweiligen Anwendungsfall und die Anpassung des Konzeptes an verschiedene Cloud-Provider.

Ausblick

Durch eine Verbindung zwischen TOSCA und bestehenden SaaS- und PaaS-Angebote könnten Data Mashups in Cloud-Umgebungen, deren Provider solche Dienste anbieten schneller und einfach provisioniert werden. Es müssen keine Implementierungs-Artefakte für die Installation von bestimmten Anwendungen wie zum Beispiel Datenbanken oder Web Containern erstellt werden. Stattdessen könnten generische Implementierungs-Artefakte für die jeweiligen Cloud-Provider erstellt werden, welche die gewünschten Anwendungen aus dem Repertoire der Cloud-Provider instanziieren. Zukünftig wäre es möglich das Konzept auf alle Arten von verteilten Anwendungen zu erweitern, und nicht nur auf Data Mashups zu beschränken. Diese Problemstellung bietet sich für eine Masterarbeit an.

A. Anhang

Listing A.1 Resultierendes Service Template der Implementierung 1

```
<tosca:Definitions xmlns:tosca="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:ns2="http://www.eclipse.org/winery/model/selfservice" id="definitions"
    name="definitions" targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12">
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/relationshiptypes/http%253A%252F%252Ftypes.
opentosca.org/NodeRedPlanHostedOnNodeRed/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/relationshiptypes/http%253A%252F%252Ftypes.
opentosca.org/OperatingSystemHostedOnOpenStack/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/nodetypes/http%253A%252F%252Ftypes.
opentosca.org/InstallOpenStackVM/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/nodetypes/http%253A%252F%252Ftypes.
opentosca.org/NodeREDPlan/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/capabilitytypes/http%253A%252F%252Ftypes.
opentosca.org/OperatingSystemContainerCapability/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/nodetypes/http%253A%252F%252Ftypes.
opentosca.org/NodeRED/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/nodetypes/http%253A%252F%252Ftypes.
opentosca.org/NodeJs/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/capabilitytypes/http%253A%252F%252Ftypes.
opentosca.org/NodeRedContainerCapability/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
```

Listing A.2 Resultierendes Service Template der Implementierung 2

```
<tosca:Import namespace="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"
    location="http://localhost:8080/winery/nodetypes/http%253A%252F%252Fdocs.
oasis-open.org%252Ftosca%252Fns%252F2011%252F12%252FToscaBaseTypes/OperatingSystem/
?definitions" importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/relationshiptypes/http%253A%252F%252Ftypes.
opentosca.org/NodeRedHostedOnNodeJs/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.
opentosca.org"
    location="http://localhost:8080/winery/capabilitytypes/http%253A%252F%252Ftypes.
opentosca.org/VirtualMachineContainerCapability/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/capabilitytypes/http%253A%252F%252Ftypes.
opentosca.org/NodeJsContainerCapability/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:Import namespace="http://types.opentosca.org"
    location="http://localhost:8080/winery/relationshiptypes/http%253A%252F%252Ftypes.
opentosca.org/NodeJsHostedOnOperatingSystem/?definitions"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>
<tosca:ServiceTemplate id="InstallVMServTemplate"
    targetNamespace="http://types.opentosca.org">
<tosca:TopologyTemplate>
<tosca:NodeTemplate xmlns:ns0="http://types.opentosca.org" name="NodeREDPlan"
    minInstances="1" maxInstances="1" id="NodeREDPlan" type="ns0:NodeREDPlan"
    winery:x="500" winery:y="100">
<tosca:Properties>
<Properties:Properties
    xmlns:Properties="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:ns6="http://types.opentosca.org"/>
</tosca:Properties>
</tosca:NodeTemplate>
<tosca:NodeTemplate xmlns:ns0="http://types.opentosca.org" name="NodeRED"
    minInstances="1" maxInstances="1" id="NodeRED" type="ns0:NodeRED" winery:x="500"
    winery:y="250">
<tosca:Properties>
<Properties:Properties
    xmlns:Properties="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns:ns2="http://types.opentosca.org"
    xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"/>
</tosca:Properties>
<tosca:Capabilities>
<tosca:Capability name="NodeRedContainerCapability" id="ap14z70qp"
    type="ns0:NodeRedContainerCapability"/>
</tosca:Capabilities>
</tosca:NodeTemplate>
```

Listing A.3 Resultierendes Service Template der Implementierung 3

```
<tosca:RelationshipTemplate xmlns:ns0="http://types.opentosca.org"
    name="NodeRedPlanHostedOnNodeRed" id="NodeRedPlanHostedOnNodeRed"
    type="ns0:NodeRedPlanHostedOnNodeRed">
<tosca:SourceElement ref="NodeREDPlan"/>
<tosca:TargetElement ref="NodeRED"/>
</tosca:RelationshipTemplate>
<tosca:NodeTemplate xmlns:ns0="http://types.opentosca.org" name="NodeJs"
    minInstances="1" maxInstances="1" id="NodeJs" type="ns0:NodeJs" winery:x="500"
    winery:y="400">
<tosca:Properties>
<Properties:Properties
    xmlns:Properties="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns:ns2="http://types.opentosca.org"
    xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"/>
</tosca:Properties>
<tosca:Capabilities>
<tosca:Capability name="NodeJsContainerCapability" id="avlf03cab"
    type="ns0:NodeJsContainerCapability"/>
</tosca:Capabilities>
</tosca:NodeTemplate>
<tosca:RelationshipTemplate xmlns:ns0="http://types.opentosca.org"
    name="NodeRedHostedOnNodeJs" id="NodeRedHostedOnNodeJs"
    type="ns0:NodeRedHostedOnNodeJs">
<tosca:SourceElement ref="NodeRED"/>
<tosca:TargetElement ref="NodeJs"/>
</tosca:RelationshipTemplate>
<tosca:NodeTemplate
    xmlns:ns1="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"
    name="OperatingSystem" minInstances="1" maxInstances="1" id="LinuxOperatingSystem"
    type="ns1:OperatingSystem" winery:x="500" winery:y="550">
<tosca:Properties>
<Properties:Properties
    xmlns:Properties="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns:ns2="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"
    xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/2013/02/12">
<hostname/>
<sshUser>ubuntu</sshUser>
<sshKey/>
```

Listing A.4 Resultierendes Service Template der Implementierung 4

```
<script>
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash - && sudo apt-get install
    -y nodejs && sudo npm install -g --unsafe-perm node-red && sudo npm install -g pm2
    && pm2 start node-red > /dev/null && sleep 10 && touch
    ~/.node-red/flows_‘hostname‘.json && echo
    "[{\"id\":\"58ffae9d.a7005\",\"type\":\"debug\",\"name\":\"\",\"active\":true,
\"complete\":false,\"x\":640,\"y\":200,\"wires\":[]},{\"id\":\"17626462.e89d9c\",
\"type\":\"inject\",\"name\":\"\",\"topic\":\"\",\"payload\":\"\",\"repeat\":\"\",
\"once\":false,\"x\":240,\"y\":200,\"wires\":[[\"2921667d.d6de9a\"]]},
{\"id\":\"2921667d.d6de9a\",\"type\":\"function\",\"name\":\"Format timestamp\",
\"func\":\" var date = new Date(msg.payload);msg.payload = date.toString();return
    msg;\",\"outputs\":1,\"x\":440,\"y\":200,\"wires\":[[\"58ffae9d.a7005\"]]}]" >
    ~/.node-red/flows_‘hostname‘.json && sleep 5 && pm2 restart all > /dev/null
</script>
</Properties:Properties>
</tosca:Properties>
<tosca:Capabilities>
<tosca:Capability xmlns:ns0="http://types.opentosca.org"
    name="OperatingSystemContainerCapability" id="at1q10tzr"
    type="ns0:OperatingSystemContainerCapability"/>
</tosca:Capabilities>
</tosca:NodeTemplate>
<tosca:RelationshipTemplate xmlns:ns0="http://types.opentosca.org"
    name="NodeJsHostedOnOperatingSystem" id="NodeJsHostedOnOperatingSystem"
    type="ns0:NodeJsHostedOnOperatingSystem">
<tosca:SourceElement ref="NodeJs"/>
<tosca:TargetElement ref="LinuxOperatingSystem"/>
</tosca:RelationshipTemplate>
<tosca:NodeTemplate xmlns:ns0="http://types.opentosca.org" name="InstallOpenStackVM"
    minInstances="1" maxInstances="1" id="InstallOpenStackVMTemplate"
    type="ns0:InstallOpenStackVM" winery:x="500" winery:y="700">
<tosca:Properties>
<Properties:Properties
    xmlns:Properties="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns="http://types.opentosca.org/propertiesdefinition/winery"
    xmlns:ns2="http://types.opentosca.org"
    xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/2013/02/12">
<credentials>
{"auth":{"tenantId":"81313f66325f427d9907b1e2674a3834","passwordCredentials":
{"username":"daniel.del.gaudio","password":"hfduhfdfg"}}}
</credentials>
<endpointsAPI>
{"os-identity-api":"http:\/\/129.69.209.127:5000\/v2.0",
"os-tenantId":"81313f66325f427d9907b1e2674a3834"}
</endpointsAPI>
<flavorId>3</flavorId>
<keypair/>
<imageId/>
<imageName>ubuntu-12.04-server-cloudimg-amd64</imageName>
```

Listing A.5 Resultierendes Service Template der Implementierung 5

```
<minDisk/>
<minRAM/>
<floatingIp/>
<serverId/>
<privKey/>
</Properties:Properties>
</tosca:Properties>
<tosca:Capabilities>
<tosca:Capability name="VirtualMachineContainerCapability" id="a8l5vlr7w"
    type="ns0:VirtualMachineContainerCapability"/>
</tosca:Capabilities>
</tosca:NodeTemplate>
<tosca:RelationshipTemplate xmlns:ns0="http://types.opentosca.org"
    name="OperatingSystemHostedOnOpenStack" id="OperatingSystemHostedOnOpenStack"
    type="ns0:OperatingSystemHostedOnOpenStack">
<tosca:SourceElement ref="LinuxOperatingSystem"/>
<tosca:TargetElement ref="InstallOpenStackVMTemplate"/>
</tosca:RelationshipTemplate>
</tosca:TopologyTemplate>
<tosca:Plans>
<tosca:Plan id="InstallVMServTemplateBuildPlan" name="InstallVMServTemplateBuildPlan"
    planType="http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
    planLanguage="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
<tosca:PlanModelReference
    reference="../servicetemplates/http%253A%252F%252Ftypes.opentosca.org/
InstallVMServTemplate/plans/InstallVMServTemplateBuildPlan/
InstallVMServTemplateBuildPlan.zip"/>
</tosca:Plan>
</tosca:Plans>
</tosca:ServiceTemplate>
<tosca:ArtifactTemplate xmlns:ns6="http://www.example.com/ToscaTypes"
    id="InstallOpenStackVM_IA" type="ns6:WAR">
<tosca:Properties>
<ns6:WSProperties xmlns:ns6="http://www.uni-stuttgart.de/opentosca"
    xmlns="http://www.uni-stuttgart.de/opentosca"
    xmlns:ns0="http://www.eclipse.org/winery/model/selfservice"
    xmlns:ns2="http://www.example.com/ToscaTypes"
    xmlns:ns31="http://www.eclipse.org/winery/model/selfservice"
    xmlns:ns4="http://www.example.com/ToscaTypes">
<ServiceEndpoint>/services/InstallOpenStackVM_Custom_InstallVMPort</ServiceEndpoint>
<PortType>
{http://types.opentosca.org}InstallOpenStackVM_Custom_InstallVM
</PortType>
<InvocationType>SOAP/HTTP</InvocationType>
</ns6:WSProperties>
</tosca:Properties>
```

Listing A.6 Resultierendes Service Template der Implementierung 6

```
<tosca:ArtifactReferences>
<tosca:ArtifactReference
    reference="artifacttemplates/http%253A%252F%252Ftypes.opentosca.org/
InstallOpenStackVM_IA/files/InstallOpenStackVM_Custom_InstallVM.war"/>
</tosca:ArtifactReferences>
</tosca:ArtifactTemplate>
<tosca:NodeTypeImplementation xmlns:ns0="http://types.opentosca.org"
    name="InstallOpenStackVM_impl" targetNamespace="http://types.opentosca.org"
    nodeType="ns0:InstallOpenStackVM" abstract="no" final="no">
<tosca:ImplementationArtifacts>
<tosca:ImplementationArtifact xmlns:ns6="http://www.example.com/ToscaTypes"
    name="InstallOpenStackVM_IA" interfaceName="Custom_InstallVM"
    artifactType="ns6:WAR" artifactRef="ns0:InstallOpenStackVM_IA"/>
</tosca:ImplementationArtifacts>
</tosca:NodeTypeImplementation>
</tosca:Definitions>
```

Literaturverzeichnis

[AG10] N. Antonopoulos und L. Gillam. *Cloud Computing: Principles, Systems and Applications*. Computer Communications and Networks. Springer London, 2010. ISBN: 9781849962414. URL: https://books.google.de/books?id=SbSbdkqibwIC (Zitiert auf S. 9).

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak und S. Wagner. „OpenTOSCA - A Runtime for TOSCA-based Cloud Applications“. English. In: *Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13)*. Bd. 8274. LNCS. Springer Berlin Heidelberg, Dez. 2013, S. 692–695. DOI: 10.1007/978-3-642-45005-1_62. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-45&engl=1 (Zitiert auf S. 27, 45).

[BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und D. Schumm. „Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA“. Englisch. In: *Proceedings of the 20th International Conference on Cooperative Information Systems (CoopIS 2012)*. Lecture Notes in Computer Science. Springer-Verlag, Sep. 2012. DOI: 10.1007/978-3-642-33606-5_25. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2012-33&engl=0 (Zitiert auf S. 27, 28).

[BBK+14] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann und J. Wettinger. „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“. Englisch. In: *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*. IEEE Computer Society, März 2014, S. 87–96. DOI: 10.1109/IC2E.2014.56. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-21&engl=0 (Zitiert auf S. 20, 24, 28, 29, 31, 34, 38).

[BBKL14] U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Vinothek - A Self-Service Portal for TOSCA“. Englisch. In: *Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014)*. Hrsg. von N. Herzberg und M. Kunze. Bd. 1140. CEUR Workshop Proceedings. CEUR-WS.org, März 2014, S. 69–72. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-25&engl=0 (Zitiert auf S. 28).

[DM14] F. Daniel und M. Matera. *Mashups - Concepts, Models and Architectures*. Data-Centric Systems und Applications. Springer., 2014 (Zitiert auf S. 9, 15).

[Erl05] T. as Erl. „Service-Oriented Architecture (SOA) Concepts, Technology and Design“. In: (2005) (Zitiert auf S. 15).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck und P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Publishing Company, Incorporated, 2014. ISBN: 3709115671, 9783709115671 (Zitiert auf S. 16).

[HBBL14] P. Hirmer, U. Breitenbücher, T. Binz und F. Leymann. „Automatic Topology Completion of TOSCA-based Cloud Applications“. English. In: *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. Bd. 232. LNI. Bonn: Gesellschaft für Informatik e.V. (GI), Sep. 2014, S. 247–258. ISBN: 978-3-88579-626-8. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-66&engl=1 (Zitiert auf S. 28, 37).

[HM16] P. Hirmer und B. Mitschang. „FlexMash - Flexible Data Mashups Based on Pattern-Based Model Transformation“. English. In: *Rapid Mashup Development Tools*. Hrsg. von F. Daniel und C. Pautasso. Bd. 591. Communications in Computer and Information Science. Springer International Publishing, 2016, S. 12–30. ISBN: 978-3-319-28726-3. DOI: 10.1007/978-3-319-28727-0_2. URL: http://dx.doi.org/10.1007/978-3-319-28727-0_2 (Zitiert auf S. 9, 10, 27, 34).

[HRWM15] P. Hirmer, P. Reimann, M. Wieland und B. Mitschang. „Extended Techniques for Flexible Modeling and Execution of Data Mashups“. In: *DATA 2015 - Proceedings of 4th International Conference on Data Management Technologies and Applications, Colmar, Alsace, France, 20-22 July, 2015.* 2015, S. 111–122. DOI: 10.5220/0005558201110122. URL: http://dx.doi.org/10.5220/0005558201110122 (Zitiert auf S. 10).

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher und F. Leymann. „Winery - A Modeling Tool for TOSCA-based Cloud Applications“. English. In: *Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13)*. Bd. 8274. LNCS. Springer Berlin Heidelberg, Dez. 2013, S. 700–704. DOI: 10.1007/978-3-642-45005-1_64. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-46&engl=1 (Zitiert auf S. 28).

[KM15] L. B. Kassner und B. Mitschang. „MaXCept–Decision Support in Exception Handling through Unstructured Data Integration in the Production Context. An Integral Part of the Smart Factory.“ In: *In Proceedings of the 48th Hawaii International Conference on System Sciences* (2015) (Zitiert auf S. 10).

[KV10] R. L. Krutz und R. D. Vines. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley Publishing, 2010. ISBN: 0470589876, 9780470589878 (Zitiert auf S. 18).

[Luc02] D. Luckham. *The power of events*. Bd. 204. Addison-Wesley Reading, 2002 (Zitiert auf S. 15).

[MBD+12] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil und D. Barton. „Big data“. In: *The management revolution. Harvard Bus Rev* 90.10 (2012), S. 61–67 (Zitiert auf S. 9).

[MCB+11] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh und A. H. Byers. *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, Mai 2011. URL: http://www.mckinsey.com/Insights/MGI/Research/Technology%5C_and%5C_Innovation/Big%5C_data%5C_The%5C_next%5C_frontier%5C_for%5C_innovation (Zitiert auf S. 9).

[Meu95] R. Meunier. „Pattern Languages of Program Design“. In: Hrsg. von J. O. Coplien und D. C. Schmidt. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995. Kap. The Pipes and Filters Architecture, S. 427–440. ISBN: 0-201-60734-4. URL: http://dl.acm.org/citation.cfm?id=218662.218694 (Zitiert auf S. 12, 15).

[MG11] P. M. Mell und T. Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Techn. Ber. Gaithersburg, MD, United States, 2011 (Zitiert auf S. 9, 16, 17).

[OASa] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. 31 January 2013. OASIS Committee Note Draft 01. http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.html.* (Zitiert auf S. 25, 34, 40).

[OASb] OASIS. *Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. OASIS Standard. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.* (Zitiert auf S. 20–22, 24).

Alle URLs wurden zuletzt am 11. 04. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift