Microservices zur modularen Gestaltung von Cloud-Plattformen

Sebastian Hucke, Holger Straede und Thomas Neumann

Im Rahmen des GLASSHOUSE-Projektes wurde ein prototypischer Demonstrator für die Smart Glasses Vuzix M300 erstellt. Dieser benötigt ein Backend mit entsprechenden Diensten, um etwa operative Daten abrufen und an Drittsysteme wie ein Lagerverwaltungssystem weitergeben zu können. Im Projekt wurde daher eine flexible Backend-Infrastruktur konzipiert und implementiert, um diese Dienste effizient betreiben zu können. Hierzu werden zunächst die zentralen Anforderungen an das System erläutert. Anschließend werden die Eckpunkte der konzeptionellen Evolution der Infrastruktur über den Projektverlauf dargestellt – von einem klassischen Setup aus Web- und Datenbankserver hin zu einer modernen container-basierten Lösung unter Einsatz von Kubernetes. Darauf aufbauend folgt die Implementierungsbeschreibung ausgewählter zentraler System-Komponenten.

1 Einleitung

Während Smart Glasses im Fokus des Benutzers stehen wird die eigentliche Arbeit des Systems ungesehen „im Hintergrund“ verrichtet. Die Brille kommuniziert mit Diensten, die ihr benötigte Daten und Funktionen bereitstellen. Hierfür und um sich in die Systemlandschaft des Anwenders zu integrieren, muss die Brille diesen oder weiteren Diensten Daten liefern (bspw. eine Liste verladener Packstücke. Ohne ein solches „Backend“ wäre die Smart Glasses als Technologie zur Prozessunterstützung nutzlos.

Die Daten, um die es sich hierbei handelt, können wiederum aus einer Vielzahl von Quellen stammen. Ebenso können die erzeugten Daten von unterschiedlichsten (Dritt-)Systemen weiterverarbeitet werden. Es stellt sich also die Frage, wie man ein Backend für Smart Glasses in einheitlicher Form bereitstellen und betreiben kann. Im Folgenden werden daher die Konzeption und Implementierung einer solchen Infrastruktur im Rahmen des GLASSHOUSE-Demonstrators beschrieben.

2 Anforderungen des Smart-Glasses-Systems

Wie eingangs erwähnt, liegt das Hauptaugenmerk des GLASSHOUSE-Projektes nicht auf der Backend-Infrastruktur, sondern auf Smart Glasses. Trotzdem kommt eben jenem Backend eine buchstäblich zentrale Bedeutung zu: Es ist der Dreh- und Angelpunkt für all jene Aufgaben, die von den Brillen nicht selbst übernommen werden können oder sollen. Aus einzelnen, isolierten Smart Glasses wird so ein erweitertes Smart-Glasses-System (vgl. Abb. 1). Dieses besteht zunächst grob aus den zwei Subsystemen

- Smart Glasses App und
- GLASSHOUSE Backend.

Dazu kommen externe Systeme, die vom Smart-Glasses-System lediglich genutzt werden bzw. die dieses nutzen. Zu nennen sind hier etwa

- Systeme des Unternehmens (z.B. Lagerverwaltungssysteme, Enterprise-Resource-Planning-Systeme),
- Öffentliche Cloud-Dienste (z.B. zur Spracherkennung oder Videokommunikation),
- Externe Authentifizierungs- und Autorisierungssysteme sowie
- Lokale Dienste und Systeme auf der Smart Glasses an sich

Abb. 1. Architektur eines modularen Smart-Glasses-Systems (Zobel et al. 2018)

Das Smart-Glasses-System stellt sowohl funktionale als auch nicht-funktionale Anforderungen an die technische Realisierung der Systemkomponente „Backend“. Diese Anforderungen wurden in Workshops mit den Projektpartnern und basierend auf den Projektzielen, externer Anforderungen und dem aktuellen Stand der Technik identifiziert. Allgemeine Anforderungen an ein Smart-Glasses-System wurden bereits in Berkemeier et al. (2019) erläutert. Im Fokus dieses Beitrags steht die technische Realisierung des Backends. Daher wird hier nur kurz auf die groben Anforderungen hinsichtlich der technischen Infrastruktur eingegangen:

- (REQ1) Die Kommunikation der Smart Glasses mit dem Backend muss über eine standardisierte Schnittstelle erfolgen.
- (REQ2) Das Smart-Glasses-System muss von mehreren Mandanten und/oder Unternehmen genutzt werden können.
- (REQ3) Je Mandant müssen unterschiedliche Drittsysteme für den Datenimport und -export angebunden werden können.
- (REQ4) Der Betrieb der Plattform muss in unterschiedlichen Umgebungen (Public/Private/Hybrid Cloud oder On-Premise) möglich sein.
- (REQ5) Die Benutzer des Systems müssen authentifiziert und autorisiert werden können.
- (REQ6) Es muss ein ausfallsicherer Betrieb des Backends möglich sein.
- (REQ7) Das System muss einfach skalierbar sein.
- (REQ8) Die Kommunikation muss verschlüsselt erfolgen.
- (REQ9) Ein „Vendor-Lock-In“ soll so weit wie möglich vermieden werden.

3 Konzeption der Backend-Infrastruktur

Ziel ist es, eine Systemarchitektur zu konzipieren, deren Implementierung den vorherig aufgezählten Anforderungen gerecht wird. Dabei wurde ein evolutionärer Ansatz verfolgt, um schnell in die Umsetzung einsteigen zu können, letztendlich jedoch ein über das Projektende hinaus tragfähiges Konzept zu entwickeln. In diesem Sinne erstreckte sich der Konzeptionsprozess in iterativer und inkrementeller Weise über einen großen Teil der Projektlaufzeit (Berkemeier et al. 2019). Im Folgenden werden die relevanten Inkremente dargestellt, die schlussendlich in einer cloud- bzw. container-basierten Infrastruktur mündeten.

3.1 Alternative Systemkonzepte

Die zur Verfügung stehenden technischen Alternativen sind in der heutigen Zeit vielfältig. Sie reichen von „klassischen“ Varianten mit dedizierten Web- und Datenbankservern über container-basierte Ansätze bis hin zu „Everything-as-a-Ser-

vice“-Lösungen, bei denen komplette Umgebungen über externe Diensteanbieter realisiert werden und kaum noch dedizierte Server zum Einsatz kommen.

Eine umfassende Betrachtung aller technischen Alternativen ist – insb. aufgrund der umfassenden Kombinationsmöglichkeiten untereinander – im Rahmen dieses Beitrags nicht möglich. Im Folgenden soll daher der Fokus auf der Architektur des tatsächlich realisierten Systems und dessen Evolution im laufenden Forschungsprojekt liegen.

3.1.1 Der Ausgangspunkt: Dedizierte (virtuelle) Server

Zu Beginn des Projekts wurden zunächst zwei virtuelle Server für den Betrieb des Demonstrators eingeplant. Einer davon fungierte als öffentlich erreichbarer Webserver für die API, der zweite als Datenbank-Server in einem separaten Netzwerksegment (vgl. Abb. 2). Die Installation der benötigten Dienste erfolgte manuell.

Abb. 2. Virtuelle Infrastruktur des GLASSHOUSE-Backends zu Beginn des Projekts

Im Rahmen der Anforderungsanalyse und Fachkonzeption wurde jedoch relativ schnell deutlich, dass eine solche, simple Infrastruktur zwar für die Bereitstellung eines prototypischen Demonstrators ausreichend sein mag, jedoch darüber hinaus nicht den Anforderungen eines ganzheitlichen Smart-Glasses-Systems gerecht werden kann. Die Gründe hierfür sind vielfältig, bezogen auf vorgenannte Anforderungen ergeben sich verschiedene Herausforderungen, genannt seien hier insb. folgende Punkte:

- REQ2, Mandantenfähigkeit: Die Realisierung der Mandantenfähigkeit bzw. einer entsprechenden Trennung erstreckt sich nicht auf die Ebene der Infrastruktur. Das heißt sie müsste vollständig softwareseitig in Form von Shared Services implementiert werden.
- REQ4, Betriebsplattformen: Prinzipiell ist ein Betrieb in unterschiedlichen Umgebungen möglich. Letztendlich ist der Cloud-Betrieb dann jedoch auf die Nutzung von Infrastructure-as-a-Service (IaaS) zur Bereitstellung virtueller Maschinen beschränkt. Das GLASSHOUSE-Backend selbst ist dadurch noch kein Cloud-Service, sondern nutzt diese lediglich.

- REQ6, Ausfallsicherheit: Ausfallsicherheit ist mit zwei nicht redundant ausgelegten, virtuellen Maschinen nicht gegeben. Beide sind ein „Single Point of Failure“.
- REQ7, einfache Skalierbarkeit: Die Ressourcen lassen sich lediglich vertikal skalieren (mehr CPU, RAM, Storage etc.). Horizontale Skalierung von Ressourcen ist in der Architektur nicht vorgesehen.

Basierend auf den identifizierten Herausforderungen wurde im nächsten Schritt betrachtet, wie diese mit aktuellen Technologien gelöst werden können.

3.1.2 Das Ziel: Eine container-basierte Service-Plattform

Es wurden verschiedene Optionen evaluiert, um die vorgenannten Probleme zu lösen und die gestellten Anforderungen zu erfüllen. Die Entscheidung fiel letztendlich auf den Einsatz einer Container-Technologie mit einem geeigneten Orchestrierungswerkzeug.

Über die Projektlaufzeit war dieser, relativ gesehen, neue Markt einer gewissen Konsolidierung unterworfen. Stand heute haben sich Docker (2019) als Container-Format und Kubernetes (2019) als Container-Orchestrator als De-facto-Standard etabliert. Dies sind auch die grundsätzlichen Werkzeuge zu deren Gunsten die Entscheidung im Entwicklungsteam gefallen ist.

Eine erste Analyse ergab, dass sich die gestellten Anforderungen mit dem neuen Ansatz voraussichtlich besser erfüllen lassen als mit der ursprünglichen Architektur. Dabei kommt der Entwicklung zugute, dass sich um Docker und Kubernetes bereits ein großes Ökosystem gebildet hat, dem sich mit der Cloud Native Computing Foundation (CNCF 2019) ein Konsortium aus namhaften Größen der Industrie angenommen hat.

Neben Docker als Virtualisierungslösung und Kubernetes als Orchestrierungslösung für Docker-Container fehlte jedoch noch ein Werkzeug, um das GLASS-HOUSE-Backend komfortabel verwalten und betreiben zu können. Die Wahl fiel hierbei auf Rancher (2019), eine Open-Source-Lösung zum Multi-Cluster-Kubernetes-Management. Neben einer Oberfläche zum Management der Backend-Services bringt Rancher den Vorteil mit, dass sich hierüber flexibel unterschiedliche Betriebsszenarien abbilden lassen. So ist etwa auch die Einbindung von „Kubernetes-as-a-Service“-Angeboten möglich. Das erweitert den Gestaltungsspielraum für die zukünftige Evolution des Smart-Glasses-Systems. So ist ein Betrieb „on-premise“ ebenso möglich wie die Einbindung oder ausschließliche Nutzung von Cloud-Providern. Mit der granularen Rechteverwaltung lassen sich zudem Backend-Dienste als Cloud-Services realisieren, die von Anwendern per Katalog konfiguriert und installiert werden können.

In Bezug auf den prototypischen Demonstrator wurden die genannten Herausforderungen auf die folgende Art gelöst:

- REQ2, Mandantenfähigkeit: Mit der Plattform ist grundsätzlich eine strikte Mandantentrennung möglich, bei der jeder Mandant eine nur durch ihn genutzte Umgebung erhält. Zusätzlich kann weiterhin eine softwareseitige Mandantentrennung für weniger kritische bzw. schwach verbundene Bereiche implementiert werden (z.B. zur Trennung von einzelnen Organisationseinheiten oder Präsentations-Szenarien für Messedemos).
- REQ4, Betriebsplattformen: Über Rancher lassen sich flexibel Kubernetes-Cluster bei unterschiedlichen Providern anlegen, sowohl in Public Clouds bekannter Unternehmen als auch in Private Clouds (OpenStack 2019) oder auf physikalischen Maschinen. Ein Betrieb ist damit auf verschiedensten Plattformen sichergestellt.
- REQ6, Ausfallsicherheit: Der Cluster-Betrieb von Kubernetes bringt grundsätzliche Ausfallsicherheit mit sich. Je nach Erfordernissen kann hier konfigurativ der gewünschte Grad justiert werden.
- REQ7, einfache Skalierbarkeit: Eine horizontale Skalierung in Form zusätzlicher Cluster-Nodes ist einfach möglich.

3.2 Systemaufbau der Infrastruktur

Auf Basis der zuvor beschriebenen Entscheidung über die eingesetzten Methoden und Werkzeuge wurde eine entsprechende Systemarchitektur erstellt. Im Folgenden werden die unterschiedlichen Ebenen des Systems „top-down“ erläutert.

3.2.1 Virtuelle Server

Betrachtet man die Systemkomponenten auf höchster Ebene, so hat sich wenig im Vergleich zum ursprünglichen Ansatz geändert: Es befindet sich eine exponierte Systemkomponente in einer demilitarisierten Zone (DMZ) und eine weitere Systemkomponente im dahinterliegenden LAN-Segment (vgl. Abb. 3). Der Unterschied wird erst bei Betrachtung der Subsysteme deutlich: In der DMZ kommt ein Layer 7 (L7) HA Reverse Proxy Pair im Hot Standby-Betrieb zum Einsatz. Im LAN-Segment stehen drei virtuelle Maschinen als Cluster Nodes bereit. Zu beachten ist: Hierbei handelt es sich um das absolute Minimum für den redundanten Betrieb des Clusters. Für Produktivsysteme wird eine weitere Aufteilung der einzelnen Kubernetes-Rollen (vgl. Abschnitt 3.2.2) empfohlen, sodass sechs bzw. neun virtuelle Maschinen erforderlich werden (d.h. bei diesem erweiterten Setup bilden jeweils drei virtuelle Maschinen disjunkte Cluster für Teilkomponenten).

Als Reverse Proxy können sowohl L7 sowie auch Layer 4 (L4) Load Balancer zum Einsatz kommen. Beide bieten jeweils gewisse Vor- und Nachteile, die hier jedoch nicht genauer betrachtet werden. Der Reverse Proxy, im aktuellen Fall ein L7 Load Balancer, dient auch als SSL-Endpunkt und leitet den eingehenden Datenverkehr an eine der drei virtuellen Maschinen weiter. Diese bilden gemeinsam den Kubernetes-Cluster.

Abb. 3. Virtuelle Server-Infrastruktur des GLASSHOUSE-Backends für den Betrieb mit Kubernetes

3.2.2 Der Kubernetes-Cluster und Rancher

Der Kubernetes-Cluster umspannt die virtuellen Maschinen im LAN-Segment des Systems. Auf diesen laufen verschiedene Kubernetes-Dienste bzw. -Container, die für die grundsätzliche Betriebsfähigkeit benötigt werden. Diese lassen sich in die drei Rollen „Control Plane“, „etcd“ und „Worker“ aufteilen (vgl. hierzu im Folgenden auch [Abb. 4](#)):

- *Control Plane:* Kubernetes liegt eine Master-Slave-Architektur zugrunde. Die Master-Nodes werden als „Control Plane“ bezeichnet. Diese Control Plane umfasst die zur Steuerung benötigten Dienste „API-Server“, „Controller Manager“ und „Scheduler“.
- *API-Server:* Der API-Server stellt eine REST-konforme API bereit, über die Kubernetes (bzw. die API-Objekte wie Services oder Pods) konfiguriert wird.
- *Controller Manager:* Der Controller Manager sorgt für den Betrieb der grundlegenden Kontrollschleifen des Systems. Es handelt sich dabei um Regelkreise, die den gewünschten Zustand des Clusters herstellen und bewahren sollen.
- *Scheduler:* Der Scheduler verteilt die Pods (siehe unten; dies ist eine Gruppe von einem oder mehreren Containern; die kleinste lauffähige Einheit in Kubernetes) gemäß der konfigurierten Regeln auf die Worker Nodes.
- *etcd:* Bei etcd handelt es sich um eine verteilte Key-Value-Datenbank, in der die Cluster-Konfiguration gespeichert wird.
- *Worker:* Die Worker sind diejenigen Nodes, auf denen die Applikationen laufen. Sie umfassen folgende Komponenten:
 - *Kubelet:* Ein Agent, der dafür sorgt, dass die Container – und damit letztendlich die Anwendungen – laufen.
 - *Kube-Proxy:* Diese Komponente dient als Netzwerk-Proxy des internen Kubernetes-Netzwerks.
 - *Container Runtime:* Die Laufzeitumgebung für die Container – im vorliegenden Fall Docker. Andere Runtimes werden aber ebenfalls unterstützt.

Eine weitere wichtige Komponente, die es an dieser Stelle zu nennen gilt, sind die sog. Pods. Es handelt sich dabei jedoch nicht um einen Dienst für den Betrieb von Kubernetes, sondern um ein Entwurfsmuster, mit dem das Ausrollen und die Verwaltung von Anwendungen in Kubernetes vereinfacht werden soll.

Ein Pod besteht aus einem oder mehreren Containern. Nach außen stellen diese einen einzigen, einheitlichen Service dar, während sie nach innen ihre Ressourcen teilen (IP-Adresse, Ports, persistenten Speicher etc.). So kann etwa ein Pod aus dem Applikations-Container sowie einem Hilfs-Container bestehen, der Logdaten an einen zentralen Logging-Dienst weiterleitet. Wird eine Anwendung auf mehrere Worker Nodes skaliert, so wird immer der gesamte Pod als Einheit skaliert.

Abb. 4. Systemskizze zentraler Kubernetes-Komponenten

Je nach Anforderung können die Nodes eine, mehrere oder alle der vorgenannten drei Rollen annehmen. Für einen hochverfügbaren Cluster wird empfohlen, alle drei Rollen auf jeweils drei separate Maschinen aufzuteilen. Für den Demonstrator ist dieses Ausmaß an Redundanz jedoch nicht erforderlich, weshalb alle drei Maschinen auch alle Rollen im Cluster annehmen.

Rancher selbst läuft innerhalb eines von ihm verwalteten Kubernetes-Clusters. Es wird dabei empfohlen, im produktiven Einsatz einen separaten Kubernetes-Cluster für Rancher zu benutzen und alle weiteren Applikationen in einem oder mehreren separaten Clustern zu betreiben. Für den vorliegenden Beitrag ist dies jedoch, wie auch schon die Aufteilung der Rollen, nicht von Belang.

4 Implementierung der Rancher-Umgebung

Im folgenden Abschnitt wird die Installation der Backend-Infrastruktur erläutert, ausgehend von der Vorbereitung der Server bzw. Kubernetes Nodes über das Deployment des Clusters bis hin zum Setup des Workloads. Exemplarische Anleitungen, die für die hier beschriebene Installation herangezogen wurden, finden sich in der Dokumentation des Rancher-Projektes (Rancher-Doku 2019). Hier wird beispielhaft die Installation erläutert, wie sie zur Bereitstellung des prototypischen Demonstrators verwendet wird.

4.1 Vorbereitung der (virtuellen) Infrastruktur

Bevor mit dem Deployment des Kubernetes-Clusters begonnen werden kann, muss die zugrundeliegende Infrastruktur bereitgestellt werden. Hierzu zählen:

1. Erstellung einer (Sub-)Domain für den Zugriff auf die GLASSHOUSE-API.
2. Anlage von zwei Netzsegmenten: Eine DMZ für das Reverse Proxy Pair mit Zugriff aus dem Internet über Port 443 und ein internes LAN für den Kubernetes-Cluster, wobei der externe Zugriff nur mittelbar über das Reverse Proxy Pair möglich ist (vgl. [Abb. 3](#)).
3. Anlage und Konfiguration des Reverse Proxy Pair: Im vorliegenden Fall wurde eine bereits existierende Installation genutzt. Daher musste lediglich das Routing erweitert werden, so dass Anfragen an die konfigurierte Sub-Domain an einen der Worker Nodes weitergeleitet werden.
4. Installation der virtuellen Maschinen: Des Weiteren wurden drei virtuelle Maschinen mit Ubuntu 18.04 LTS installiert und konfiguriert. Rancher empfiehlt hier für kleine Installation 2 vCPUs und 8 GB RAM je Node. Der administrative Zugriff auf das Betriebssystem erfolgt für alle weiteren Schritte per Secure Shell (SSH) von einem internen Management Client aus.

Nach Abschluss dieser vorbereitenden Arbeiten kann mit dem Deployment des Kubernetes-Clusters fortgefahren werden.

4.2 Deployment eines Kubernetes-Clusters

Die Installation eines Kubernetes-Clusters kann auf unterschiedlichen Wegen erfolgen. Grundsätzlich kann man festhalten, dass dem Vorgang eine relativ hohe Komplexität innewohnt. Rancher bietet hierzu jedoch einen Installer namens Rancher Kubernetes Engine (RKE 2019), der diesen Vorgang stark vereinfacht. Die hierfür notwendigen von einer Linux Workstation aus vorgenommenen Vorgängen werden im Folgenden genauer beschrieben.

Zunächst muss die Installationsdatei aus dem GitHub-Projekt heruntergeladen werden. Sie sollte an einem Ort abgelegt werden, der in der \$PATH-Variable enthalten ist. So entfällt die explizite Angabe des Pfades.

```
wget
https://github.com/rancher/rke/releases/download/v0.2.4/rke_linux-
amd64
mv rke_linux-amd64 /usr/local/bin/rke
chmod +x /usr/local/bin/rke
```

Im nächsten Schritt muss die Cluster-Konfigurationsdatei `cluster.yml` vorbereitet werden. Für ein Setup aus drei Nodes ist der Aufbau wie folgt:

```
nodes:
- address: 10.10.10.11 # hostname or IP to access nodes
  user: root
  role: [controlplane,etcd,worker]
  ssh_key_path: ~/.ssh/id_rsa_node01
- address: 10.10.10.12
  user: root
  role: [controlplane,etcd,worker]
  ssh_key_path: ~/.ssh/id_rsa_node02
- address: 10.10.10.13
  user: root
  role: [controlplane,etcd,worker]
  ssh_key_path: ~/.ssh/id_rsa_node03
services:
  etcd:
    snapshot: true
    creation: 6h
    retention: 24h
```

Alternativ kann eine interaktive Abfrage aller Konfigurationsparameter über den Befehl `rke config --name cluster.yml` erfolgen.

Folgende Parameter sind in der obigen Datei `cluster.yml` enthalten:

- **nodes:** Dieser Teil enthält die IP-Adressen und Zugangsdaten für einen passwortlosen SSH-Zugriff auf die Nodes. Ebenso werden die Rollen der Nodes festgelegt. Mit diesen Informationen kann sich RKE mit den Nodes verbinden und die Installation bzw. Konfiguration vornehmen.
- **services:** Dieser Teil enthält Angaben zur Erstellung von Sicherungen der Cluster-Konfiguration.

Darüber hinaus existiert noch eine Vielzahl weitere Parameter, die bei Bedarf in der Dokumentation des Projekts zu finden sind. Mit der oben beschriebenen Konfiguration kann der Cluster erzeugt werden. Hierzu genügt es, den Befehl `rke up --config ./rancher-cluster.yml` auszuführen. RKE übernimmt im Hintergrund sämtliche Deployment-Aufgaben und legt im aktuellen Verzeichnis die Datei `kube_config_cluster.yml` ab. Diese enthält die Zugangsdaten, um den Cluster mit den Programmen *kubectl* und *helm* administrieren zu können. Hierzu kann die Datei nach `$HOME/.kube/config` kopiert werden, so dass sie von beiden Programmen automatisch geladen werden kann.

4.3 Installation von Rancher im Cluster

Um einen hochverfügbaren Betrieb der Lösung zu ermöglichen, wird Rancher auf dem zuvor installierten Kubernetes-Cluster mit Hilfe von Helm (2019), einer Paketverwaltung für Kubernetes, eingesetzt. Die Installationspakete werden mittels sog. „Helm Charts“ definiert und können in einem Katalog veröffentlicht werden.

Um Helm für die Installation von Rancher nutzen zu können, muss im Kubernetes-Cluster die Software *tiller* installiert werden. Dies erledigt Helm prinzipiell selbst, jedoch müssen zuvor noch ein Service-Account sowie eine Rollenzuordnung angelegt werden, damit Helm auch über die passenden Berechtigungen verfügt. Die Konfiguration des Clusters wird über das Kommandozeilen-Werkzeug *kubectl* (2019) vorgenommen.

```
kubectl -n kube-system create serviceaccount tiller
kubectl create clusterrolebinding tiller \
  --clusterrole=cluster-admin \
  --serviceaccount=kube-system:tiller
```

kubectl nutzt die im vorherigen Abschnitt bei der Erzeugung des Clusters angelegte Datei `kube_config_cluster.yml`, um eine Verbindung zum Cluster herzustellen. Es wird ein Account namens „tiller“ angelegt und dieser mit der Rolle „cluster-admin“ versehen. Nun kann Helm initialisiert und *tiller* auf dem Cluster installiert werden:

```
helm init --service-account tiller
```

Die Helm Charts von Rancher werden in einem eigenen Katalog (einem sog. Helm Chart Repository) veröffentlicht. Dieses Repository muss vor der Installation hinzugefügt werden:

```
helm repo add rancher-stable
```

<https://releases.rancher.com/server-charts/stable>

Zudem ist vor der eigentlichen Rancher-Installation zu beachten, dass ein Zugriff auf Rancher ausschließlich per HTTPS möglich ist. Da die Rancher-Administrations-Oberfläche im vorliegenden Fall aber nur im internen LAN und nicht per Internet erreichbar sein soll, müssen weitere Vorbereitungen getroffen werden. Zum einen muss ein DNS-Eintrag für die verwendete Intranet-Domain angelegt werden. Zum anderen muss für die automatische Bereitstellung selbst-signierter Zertifikate gesorgt werden. Hierzu wird *cert-manager* aus dem offiziellen Kubernetes Helm Chart Repository installiert.

```
helm install stable/cert-manager \
  --name cert-manager \
  --namespace kube-system \
  --version v0.5.2
```

Im Anschluss kann nun Rancher selbst installiert werden.

```
helm install rancher-stable/rancher \
  --name rancher \
  --namespace cattle-system \
  --set hostname=rancher.intranet.loc
```

Anschließend steht die Rancher-Anwendung unter <https://rancher.intranet.loc/> bereit. Der Aufruf der URL im Browser führt den Benutzer zur Login-Maske (vgl. Abb. 5).

Abb. 5. Login-Dialog Rancher

Nach dem Login kann der Cluster-Status überprüft werden (vgl. Abb. 6). Das System ist bereit zum Deployment der GLASSHOUSE-Dienste.

Abb. 6. Übersicht über den Status des Kubernetes-Clusters

4.4 Deployment der GLASSHOUSE-Services

Basierend auf der Backend-Infrastruktur wird ein GLASSHOUSE-API-Container nebst Datenbank gestartet. Es ist zu beachten, dass verschiedene, für den produktiven Einsatz wichtigen Punkten (so etwa Datenpersistenz, Monitoring, Logging, Deployment Pipelines etc.) im Rahmen dieses Artikels nicht berücksichtigt werden. Ziel ist es, zwei unterschiedliche Optionen zum Starten von Anwendungen im Cluster zu erläutern.

4.4.1 Installation des Datenbank-Managementsystems

Als Datenbank kommt im GLASSHOUSE-Projekt MariaDB (2019) zum Einsatz. Es handelt sich dabei um eine freie, quelloffene Software. Neben klassischen Installationsdateien stehen auch vorgefertigte Docker-Container zum Download bereit (MariaDB-Download 2019). Ebenso gibt es Helm Charts, die speziell für den Betrieb mit Kubernetes angepasst sind.

Im vorherigen Abschnitt wurde Rancher mit Hilfe der Kommandozeile aus einem Helm-Chart-Repository installiert. Rancher bietet nun über seine Weboberfläche eine grafische Möglichkeit, solche Kataloge zu durchsuchen und Installationen vorzunehmen (vgl. Abb. 7).

Abb. 7. Der App-Katalog in Rancher, gefiltert auf die Anwendung MariaDB

Neben zwei vorkonfigurierten Katalogen – das offizielle Helm-Repository (2019) sowie ein Rancher-Repository (2019) – können beliebige weitere, auch eigene Kataloge hinzugefügt werden.

Über die Detailansicht (vgl. Abb. 8) lassen sich erforderliche Konfigurationsparameter setzen (z.B. initiale Account-Informationen oder Replikationsparameter). Mit einem Klick auf „Launch“ wird die Kubernetes-Konfiguration erstellt, woraufhin Kubernetes den gewünschten Cluster-Zustand herstellt (also im Prinzip die Applikation startet).

Abb. 8. Detailansicht des Helm Charts zu MariaDB

Auf der Übersichtseite der installierten Katalog-Anwendungen (vgl. Abb. 9) wird im Anschluss die installierte Anwendung mitsamt ihrem Deployment-Status gelistet.

Abb. 9. Status-Übersicht der installierten Helm Charts

Das Datenbank-Managementsystem steht damit zur Verwendung durch andere Applikationen bereit.

4.4.2 Starten der GLASSHOUSE-API

Eine andere Möglichkeit, einfache Applikationen zu starten, ist das Anlegen sogenannter Workloads. Grundlage für die Erstellung eines Workloads ist ein einzelnes Docker-Image. Zu diesem Image werden über die Rancher-Oberfläche (vgl. Abb. 10) alle relevanten Konfigurationsparameter erfasst, etwa erforderliche Umgebungsvariablen, persistenter Datenspeicher oder Regeln für die Zuweisung auf Nodes.

The screenshot shows the 'Deploy Workload' form in the Rancher UI. The form includes the following sections:

- Name:** A text input field containing 'glasshouse'.
- Workload Type:** A dropdown menu showing 'Scalable deployment of 1 pod'.
- Docker Image:** A text input field containing '032884098185.dkr.ecr.eu-central-1.amazonaws.com'.
- Namespace:** A dropdown menu showing 'default'.
- Port Mapping:** A section with a '+ Add Port' button.
- Environment Variables:** A section with a description: 'Set the environment that will be visible to the container, including injecting values from other resources like Secrets.'
- Node Scheduling:** A section with a description: 'Configure what nodes the pods can be deployed to.'

Abb. 10. Anlegen von Workloads über die Rancher-Oberfläche

Nach dem Anlegen des Workloads findet der Datenverkehr jedoch noch nicht automatisch sein Ziel. Auf jedem der drei Kubernetes-Nodes läuft ein sog. Ingress Controller, der eingehenden Datenverkehr an den zuständigen Workload weiterleitet. Hierzu muss dem System noch bekannt gegeben werden, welche Anfragen dies betrifft: Im vorliegenden Fall sollen etwa alle Anfragen an „glasshouse.pco-cloud.de“ an den Workload „glasshouse“ weitergeleitet werden (vgl. Abb. 11).

Ein einfacher Aufruf der API im Browser bestätigt im Anschluss die Funktionalität des erzeugten Workloads.

The screenshot shows the 'Load Balancing' page in the Rancher UI. The page includes a table with the following columns: State, Name, Targets, and Created. The table shows one entry for the 'glasshouse' workload.

State	Name	Targets	Created
Active	glasshouse L7 Ingress	glasshouse.pco-cloud.de > glasshouse	11/09/2018

Abb. 11. Hostname „glasshouse.pco-cloud.de“ mit dem Workload „glasshouse“

5 Fazit und Ausblick

Es wurde eine flexibel einsetzbare Backend-Infrastruktur für den Einsatz im Projekt GLASSHOUSE konzipiert und umgesetzt. Diese lässt sich auch nach Projektende grundsätzlich durch die Projektpartner in verschiedenen Setups für den Weiterbetrieb und die Weiterentwicklung des prototypischen Demonstrators nutzen. Die gewählte Lösung auf Basis von Kubernetes bietet hier vielfältige Optionen, die ein zukunftsfähiges System im Produktivbetrieb benötigt.

6 Literatur

- Berkemeier L, Zobel B, Werning S, Ickerott I, Thomas O (2019) Engineering of Augmented Reality-Based Information Systems. *Business & Information Systems Engineering* 61:67–89
- CNCF (2019) Cloud Native Computing Foundation. <https://www.cncf.io/>, abgerufen am 25.11.2019
- Docker (2019) <https://www.docker.com/>, abgerufen am 25.11.2019
- Helm (2019) <https://helm.sh/>, abgerufen am 25.11.2019
- Helm-Repository (2019) Helm Charts. <https://github.com/helm/charts>, abgerufen am 25.11.2019
- Kubectrl (2019) kubectrl install. <https://kubernetes.io/de/docs/tasks/tools/install-kubectrl/>, abgerufen am 25.11.2019
- Kubernetes (2019) <https://kubernetes.io/>, abgerufen am 25.11.2019
- MariaDB (2019) <https://mariadb.com/>, abgerufen am 25.11.2019
- MariaDB-Download (2019) Dockerhub MariaDB. https://hub.docker.com/_/mariadb, abgerufen am 25.11.2019
- Openstack (2019) <https://www.openstack.org/>, abgerufen am 25.11.2019
- Rancher (2019) <https://rancher.com/>, abgerufen am 25.11.2019
- Rancher-Doku (2019) Rancher Install. <https://rancher.com/docs/rancher/v2.x/en/installation/>, abgerufen am 25.11.2019
- Rancher-Repository (2019) Rancher Charts. <https://github.com/rancher/charts>, abgerufen am 25.11.2019
- RKE (2019) Rancher Kubernetes Engine. <https://github.com/rancher/rke>, abgerufen am 25.11.2019
- Zobel B, Berkemeier L, Werning S, Vogel J, Ickerott I, Thomas O (2018) Towards a Modular Reference Architecture for Smart Glasses-based Systems in the Logistics Domain. In: Fellmann M et al. (Eds) 9th Intern. Workshop on Enterprise Modeling and Inf. Systems Architectures (EMISA). Rostock, 95–99