Urban Data Platform Hamburg: Integration von Echtzeit IoT-Daten mittels SensorThings API

Urban Data Platform Hamburg: Integration of real-time IoT Data using SensorThings API

Michael Fischer | Pierre Gras | Sonja Löwa | Stefan Schuhart*

Zusammenfassung

Ein optimales Anwendungsbeispiel einer Urban Data Platform ist die Vernetzung digitaler Räume in Hamburg. Echtzeitdaten aus dem Internet der Dinge sind ein aktuelles und hoch relevantes Thema für eine Vielzahl dynamischer Anwendungen in der Stadt, erfordern jedoch sowohl wirtschaftlich sinnvolle als auch zuverlässige Wege der Bereitstellung. In diesem Artikel formulieren wir Anforderungen an eine skalierbare Echtzeitdateninfrastruktur und demonstrieren eine mögliche Implementierung (iot.hamburg.de) und Integration in eine Urban Data Platform (www.urbandataplatform.hamburg) mittels der offenen und standardisierten SensorThings API der OGC. Hierbei werden Daten sowohl über eine REST-Schnittstelle als auch eventbasiert über einen MQTT-Endpoint zur Verfügung gestellt. Die hier im Detail beschriebene Umsetzung erfolgt mit Hilfe der Open Source Software FROST-Server in einer skalierbaren Microservice-Architektur auf Basis von Docker, Kubernetes und einem hochverfügbaren PostgreSQL Cluster.

Schlüsselwörter: Urban Data Platform, OGC SensorThings API, FROST-Server, IoT, Echtzeitdaten

Summary

An optimal application example of an Urban Data Platform is the networking of digital spaces in Hamburg with regard to the goals of the UN Agenda 2030. The provision and use of dynamic data from the Internet of Things event-based in real time, with low latency, is becoming increasingly important. In this paper we formulate the requirements and show the implementation of a real-time data infrastructure (iot.hamburg.de) in an Urban Data Platform (www.urbandataplatform.hamburg) using the open and standardised SensorThings API of the OGC. Data is provided both via a REST interface and event-based via an MQTT endpoint. The implementation is carried out with the help of the open source software FROST-Server in a scalable micro service architecture using Docker, Kubernetes and a high availability PostgreSQL cluster and is described in detail.

Keywords: *Urban Data Platform, OGC SensorThings API, FROST-Server, IoT, real-time data*

1 Echtzeitdaten in der Urban Data Platform Hamburg

In allen Bereichen des urbanen Raumes nimmt die Digitalisierung zu und die Datenbestände wachsen stetig. Die Daten werden in den verschiedenen digitalen Räumen (Urbanes Leben, Mobilität & Energie, Wirtschaft & Arbeitswelt, Sicherheit & Rechtswesen, Wissen & Bildung, Kultur & Sport & Freizeit sowie Gesundheit & Soziales) erhoben und genutzt (Hamburg 2020). Eine Vernetzung zwischen diesen Räumen findet bisher kaum statt. Eine *Urban Data Platform* hat das Potenzial, dies grundlegend zu verändern. Ihr liegt ein Konzept zur interdisziplinären Datennutzung und gesamtgesellschaftlicher Mehrwertgenerierung zugrunde (van Oosterhout et al. 2020). Die Integration und Vernetzung von städtischen Daten mittels standardisierter Schnittstellen wird durch eine Urban Data Platform schnell und einfach ermöglicht (Welzel und Eichhorn 2016). Insbesondere wenn diese auf einer etablierten, städtischen Geodateninfrastruktur (GDI) basiert, können IT-Systeme und Dienste aus unterschiedlichen Bereichen miteinander verbunden werden, um Daten automatisiert auszutauschen (Welzel et al. 2019). Die Daten einer Urban Data Platform sind individuell kombinierbar, diskriminierungsfrei und direkt abrufbar. Der einfache, offen standardisierte Zugang zu aktuellen Daten ermöglicht ad-hoc-Analysen und damit eine zügige Bereitstellung von Entscheidungsgrundlagen. Gleichzeitig werden Kosten für doppelte Datenhaltung oder anbieterspezifische Schnittstellen minimiert. Dabei ist es wichtig, dass die öffentliche Verwaltung die Urban Data Platform selbst entwickelt und betreibt und somit den Schlüssel zu einer innovativen und souveränen Stadt selbst in der Hand behält. Damit profitieren Kommunen, Unternehmen, Wissenschaft und Bürger in gleichem Maße. Die Freie und Hansestadt Hamburg verfolgt mit der *Urban Data Platform Hamburg* (*HH_UDP*, www.urbandataplatform.hamburg) und einer umfassenden Digitalstrategie (Hamburg 2020) diesen richtungsweisenden Weg der konsequenten und selbstbestimmten Innovation. Die HH_UDP ist dabei keine Insellösung, sondern folgt, gemeinsam mit vielen anderen europäischen Städten (z. B. Lyon, Helsinki, Nantes, Florenz, Rotterdam) der Urban Data Platform Initiative der European Innovation Partnership on Smart Cities and Communities (EIP SCC 2020, https://eu-smartcities.eu). Die eingesetzte Technologie wird dabei gemeinsam verbessert, um die viel-

* Die Autoren haben alle in gleichem Maße zu diesem Artikel beigetragen.

DOI 10.12902/zfv-0330-2020
© Wißner-Verlag

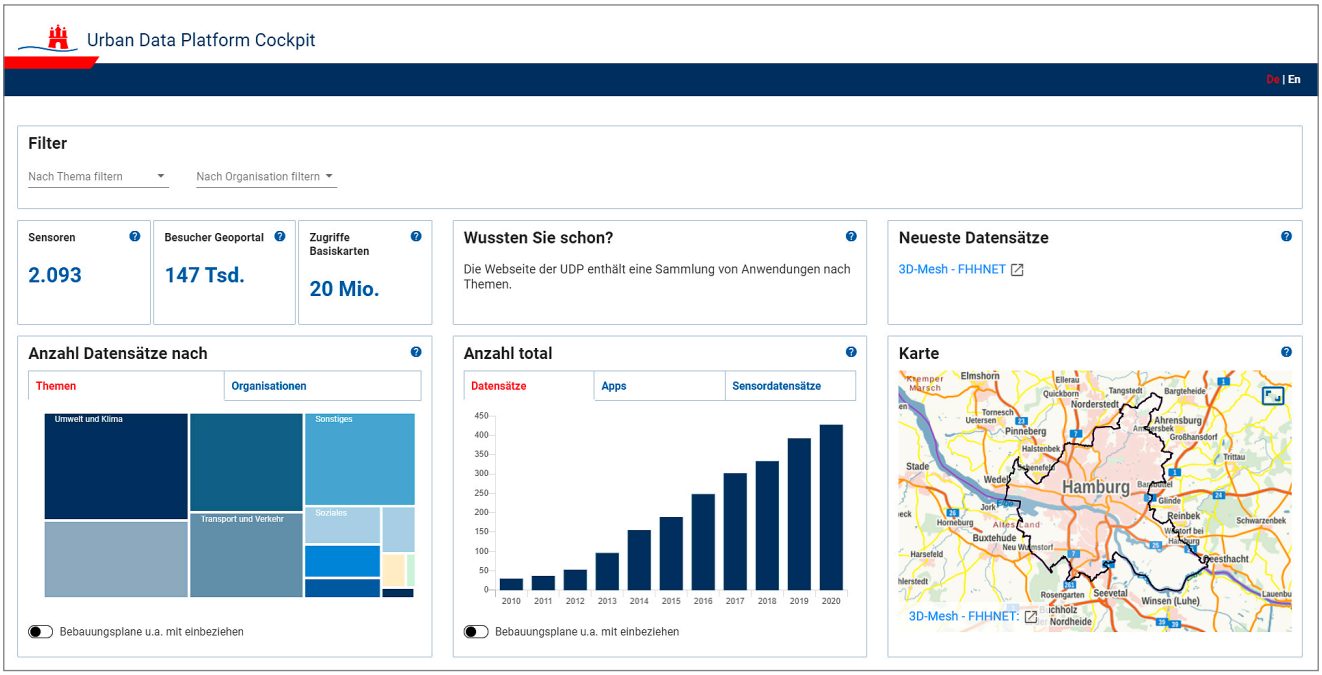

Abb. 1: Urban Data Platform Cockpit (https://geoportal-hamburg.de/udp-cockpit)

fältigen dezentralen Systeme und Datenbanken einer modernen Stadt funktional zu verbinden und so »Datensilos aufzubrechen« – auch für Echtzeitdaten. Diese europäische Initiative wurde in Deutschland mit dem Referenzarchitekturmodell Offene Urbane Plattform (OUP) umgesetzt (EIP SCC 2020 und DIN SPEC 91357:2017-12).

Statische Daten der städtischen oder nationalen GDI waren bisher Hauptbestandteil einer Urban Data Platform (Schmitz 2012). Aktuell erleben wir eine Erweiterung um dynamische Daten und deren Nutzung in Echtzeit (z. B. Belegungsdaten der Elektroladeinfrastruktur; Abb. 4, Tab. 1). In Europa sollten Daten des öffentlichen Sektors zum Vorteil der europäischen Wirtschaft und Gesellschaft vollständig und echtzeitnah nutzbar sein. Insbesondere wird eine einfache und offene Bereitstellung dynamischer, öffentlicher Daten mittels angemessener technischer Mittel gefordert, um eine Emergenz in der Gesellschaft zu ermöglichen (EU 2019).

Ein ideales Anwendungsbeispiel einer Urban Data Platform ist die Modernisierung des städtischen Verkehrs in Hamburg mit Blick auf die Ziele der Agenda 2030 (BMZ 2017). Um diese Ziele zu erreichen, bedarf es einer diskriminierungsfreien Bereitstellung dynamischer Verkehrsdaten mittels offener, standardisierter, eventbasierter Schnittstellen. Die Lösung der HH_UDP stellt eine ausgereifte Möglichkeit sowohl für Dateneigentümer als auch Datennutzende dar. In Hamburg gibt es bereits diverse Echtzeitdaten von Sensoren, die in Zählschleifen, Lichtsignalanlagen, Umweltmessstationen und Parkhäusern verbaut sind. Hinzu kommen zukünftig tausende neue Sensordatenströme durch Hamburger Projekte zu intelligenten Verkehrssystemen (BVM ITS-Projekte 2020) wie einer automatisierten Verkehrsmengenerfassung (BVM aVME 2020), einem Radverkehrszählnetz (BVM HaRaZäN 2020), einer digitalen Warnbake auf Straßenbaustellen (BVM GeoNetBake 2020) und einem Ampel-Prognosedienst (BVM TLF 2.0 2020). Neben der hohen Anzahl an Sensordatenströmen ist auch die eventbasierte Datenübertragung eine neue Anforderung an die Urban Data Platform. Jedes der erwähnten Sensorsysteme liefert Daten an ein Fachsystem, das für fachspezifische Aufgaben und zweckbezogene Datenhaltung optimiert ist. Der Anschluss dieser Fachsysteme an die HH_UDP ermöglicht umgehend, entsprechend einer Nutzungsfreigabe durch den Dateneigentümer, die stadtweite oder öffentliche Nutzung von Fachdaten, ohne Fachverfahren zu beeinflussen oder Sicherheitslücken zu kreieren. Um dies für Echtzeitdaten zu gewährleisten, müssen Daten eventbasiert mit geringer Latenz bereitgestellt werden. Dies muss kostengünstig und zuverlässig geschehen. Alle gesetzlichen Vorgaben und Sicherheitsstandards sind einzuhalten. Daher haben wir folgende Anforderungen an eine *echtzeitfähige Urban Data Platform* formuliert:

1) **Eventbasierte Datenbereitstellung in Echtzeit:** Daten (z. B. Signalphasen einer Ampel oder Fußgängersignalanforderungen) müssen mit einer anforderungsgerechten Latenz als push-Nachricht bereitgestellt und echtzeitnah verwendet werden können, um eventgetriebene Anwendungen z. B. zur Verkehrsflussoptimierung (z. B. »time-to-green«) zu ermöglichen.
2) **Kostengünstige und zuverlässige Datenbereitstellung:** Die Rechenleistung einer Echtzeitdateninfrastruktur sollte anpassbar (skalierbar) sein, um Lastspitzen verarbeiten zu können und bei geringer Auslastung Kosten zu reduzieren. Daten müssen zuverlässig und stetig ausgeleitet werden, da Echtzeitanwendungen diese verarbeiten und sonst fehlerhafte Ergebnisse produzieren. Die Softwarearchitektur sollte eine horizontale Skalierbarkeit gewährleisten, um die Zuverlässigkeit des Systems zu erhöhen.
3) **Gesetzeskonforme Datenbereitstellung:** Das Geodatenzugangsgesetz (GeoZG 2020) bzw. das gültige

 © Wißner-Verlag

Landesgesetz (Hamburg 2009) setzt die europäische INSPIRE-Richtlinie 2007/2/EG (EU 2007) in nationales Recht um. Neben der Datenbereitstellung selbst wird dort auch die Bereitstellung von Metadaten zu jedem Datensatz verlangt. Weiterhin sind Geodaten gemäß § 8 Abs. 1 GeoZG interoperabel bereitzustellen. In Hamburg muss zusätzlich das Hamburgische Transparenzgesetz (HmbTG) (Hamburg 2012) beachtet werden, das vorschreibt, dass amtliche Geodaten in einem wiederverwendbaren, offenen Format veröffentlicht werden und anonym abrufbar sein müssen.

4) **Sichere Datenbereitstellung:** Die Client-Server-Verbindung zwischen Datenliefersystem und Urban Data Platform muss stets verschlüsselt erfolgen, um eine Datenmanipulation zu verhindern.

2 Lösungsansätze zur Umsetzung einer Echtzeitdateninfrastruktur

2.1 Wie können Daten eventbasiert und in Echtzeit bereitgestellt werden?

Um Daten echtzeitnah und eventbasiert bereitzustellen, sind ausreichend Rechengeschwindigkeit, Bandbreite und schnelle Speicherzugriffe notwendig. Eine eventbasierte Übertragung von Echtzeitdaten ist nicht planbar. Es können Lastspitzen bei der Datenabfrage und der Datenintegration entstehen, die das System abfangen muss. Dies ist mit einer klassischen und nicht automatisiert skalierenden Server-Architektur nur schwer zu realisieren. Das System müsste auf die maximal zu erwartende Last ausgelegt werden und wäre somit nur selten vollständig ausgelastet. Zusätzlich kann die verfügbare Internetbandbreite ein Engpass sein. Eine moderne Lösung liefert eine Kombination aus Cloud Computing und Microservices. Dabei handelt es sich um eine containerbasierte IT-Architektur auf skalierbaren virtuellen Computern *(Virtual Machine, VM)*, die ein Cloud-Provider bereitstellt. Nutzer können Ressourcen entsprechend dem Bedarf manuell oder automatisiert zusammenstellen. Bezahlt wird nur die bestellte Ressourcenleistung, die innerhalb weniger Minuten manuell oder automatisiert angepasst werden kann.

Wenn große Datenmengen hochfrequent übertragen werden, kann die verfügbare Bandbreite überlastet und damit die Latenz verschlechtert werden. Daher sollten benutzte Datenübertragungsprotokolle auf schlanke Datenstrukturen optimiert sein, um die notwendige Bandbreite zu reduzieren. Der de-facto-Standard zur Datenübertragung im Internet ist *HTTP (Hypertext Transfer Protocol)*. Gemäß der Spezifikation dieses Protokolls (W3C 2020) senden Clients Anfragen an Server und bekommen darauf Antworten. Diese Anfrage-Antwort-Kommunikation verursacht bei kurzen Aktualisierungsintervallen einen sehr hohen Protokolloverhead, da bei jedem Request sogenannte HTTP-Header übertragen werden. Auch erlaubt die Anfrage-Antwort-Kommunikation nur eine 1-zu-1-Kommunikation, nicht jedoch eine 1-zu-N-Kommunikation. Daher ist dieses Protokoll zur Übertragung von Echtzeitmassendaten oder Datenströmen mit sehr kurzen Aktualisierungsintervallen nur bedingt geeignet.

Ein stetiger Datenstrom kann durch das Übertragungsprotokoll *Websocket* (IETF 2011) ermöglicht werden. Über dieses Protokoll kann eine permanente, bidirektionale Client-Server-Verbindung aufrechterhalten werden. Wiederholte Client-Anfragen, um neue Daten an einen Server zu übertragen, entfallen und damit auch wiederholte Handshakes. Dabei werden alle Daten des Datenstroms übertragen, eine Filterung auf bestimmte Daten ist nicht möglich.

Eine bessere Alternative zur Übertragung eines Datenstroms ist das MQTT-Protokoll. Es ermöglicht ebenfalls eine permanente Verbindung zwischen Client und Server und stellt darüber hinaus einen Publish/Subscribe-Mechanismus zu Verfügung. Der MQTT-Server stellt Daten über einen sogenannten *Broker* in hierarchisch strukturierten Themen bereit, sodass Clients sich auf ausgewählte Themen abonnieren können und somit eventbasiert die neuesten Daten ausgeliefert bekommen. Dies ermöglicht besonders schlanke Datenströme.

Die diskutierten Protokolle HTTP, Websocket und MQTT sind in proprietären und Open-Source Lösungen zur Datenbereitstellung in Echtzeit unterschiedlich verbreitet. Einige der bekanntesten Implementierungen (Sensor Observation Service, oneM2M, NGSI-LD API (FIWARE), ESRI Stream Services (ArcGIS) und SensorThings API) werden folgend kurz angesprochen.

Der Sensor Observation Service bietet nur Datenhandling mittels SOAP/XML via HTTP unter Berücksichtigung von Ortsinformationen, aber keinen Mechanismus, um Daten eventgetrieben bereitzustellen (OGC 2020a). Auch der von FIWARE benutzte Standard NGSI-LD API sieht in der aktuellen Version nur eine Datenübertragung von JSON-Objekten via REST API unter Berücksichtigung der Ortsinformationen vor (ETSI 2020). Bei der Software ArcGIS von ESRI erfolgt die Übertragung von Echtzeitdaten über proprietäre »Stream Services« (Shrestha et al. 2018). Diese basieren auf der Übertragung von JSON-Objekten via Websocket. Auch Ortsinformationen werden dabei berücksichtigt. OneM2M beherrscht die Datenübertragung via HTTP, MQTT sowie anderer Protokolle (Muhammad et al. 2019). Eine standardisierte Übertragung von Ortsinformationen ist für oneM2M allerdings nicht bekannt. Die SensorThings API stellt Echtzeitdaten als JSON-Objekte sowohl über eine Rest API als auch eventbasiert über MQTT bereit (Hertweck et al. 2019). Dabei werden Ortsinformationen entsprechend der weltweit gültigen und offenen OGC-Standards übertragen. Aus diesem Grund erfolgt der Aufbau der Echtzeitdateninfrastruktur der Freien und Hansestadt Hamburg auf dem Standard der *Sensor Things API*.

Die SensorThings API nutzt JSON-Objekte und folgt den REST-Prinzipien. Neben HTTP wird auch MQTT zur eventbasierten Datenübertragung unterstützt. Die SensorThings API-Spezifikation liegt derzeit in der Version 1.0

vor und wird stetig weiterentwickelt. Die Version 1.1 befand sich Mitte 2020 im Public Comment.

2.2 Wie können Echtzeitdaten wirtschaftlich und zuverlässig bereitgestellt werden?

Zur wirtschaftlichen und zuverlässigen Nutzung des Cloud Computing werden immer häufiger Microservice-Architekturen verwendet. Dabei werden komplexe Anwendungen in einzelne Dienste *(Services)* mit spezifischen Aufgaben aufgeteilt und separat ausgeführt (Red Hat 2020). Die Dienste sind in Containern gekapselt und können in Clustern von VMs *(Nodes)* verteilt ausgeführt werden. Das zurzeit bekannteste Framework für Containertechnologie ist Docker (www.docker.com). Die Verteilung der Container auf die Nodes kann dabei vollautomatisch durch Programme zur Container-Orchestrierung verwaltet werden. Die am weitesten verbreitete Software zur Container-Orchestrierung ist das Open-Source-Programm *Kubernetes* (https://kubernetes.io). Obwohl es Alternativen wie Docker-Swarm oder Apache Mesos gibt, hat sich Kubernetes zu einer frei verfügbaren (Apache Lizenz 2.0) Basistechnologie entwickelt.

Eine Anwendung in einer Microservice-Architektur in einem Kubernetes-Cluster auszuführen, erhöht die Zuverlässigkeit und ermöglicht parallele Datenverarbeitung, da jede Komponente zu jedem Zeitpunkt mehrfach vorliegen kann. Folglich können Aufgaben bei Störungen direkt von anderen Instanzen übernommen werden. Dies ist möglich, da eine einzelne Instanz keine persistenten Informationen speichert, eine Instanz ist also *stateless*. Anwendungen, die in einem Kubernetes-Cluster laufen, können automatisch skalierbar konfiguriert werden, wodurch Lastspitzen sehr gut ausgeglichen und Rechenressourcen optimal genutzt werden können. Kubernetes arbeitet grundsätzlich deklarativ. Das bedeutet, dass Anwendungen mit Textdateien (sog. YAML-Deskriptoren) konfiguriert werden. Eine Versionskontrolle dieser Dateien zum Beispiel mit »Git« (freie Software zur verteilten Versionsverwaltung) vereinfacht die Wartung von Anwendungen und Ressourcen maßgeblich (https://git-scm.com).

2.3 Wie können Echtzeitdaten gesetzeskonform bereitgestellt werden?

Notwendige Bedingung einer gesetzeskonformen Datenbereitstellung ist Interoperabilität des Systems durch die Verwendung offener Schnittstellen. Freie Software verwendet häufiger quelloffene und frei zugängliche Standards als proprietäre Software. Dabei werden grundlegende Richtlinien wie INSPIRE zügig integriert. Die Nutzung von freier, quelloffener Software ermöglicht eine eigenständige Anpassung, um aktualisierten Regularien flexibel und zeitnah zu entsprechen. Auch ist die Weitergabe der Software an andere Behörden uneingeschränkt möglich. Dieses Bestreben wird auf europäischer Ebene von gemeinnützigen Organisationen/Vereinen wie der Free Software Foundation Europe unterstützt (Kirschner 2019).

Damit Daten von Dritten nutzbar sind, müssen diese mit Metadaten beschrieben werden. Insbesondere die SensorThings API vereinfacht die Bereitstellung von Metadaten, da diese im Standard berücksichtigt sind. Über die REST-Schnittstelle ist eine Verlinkung mit Metadatenkatalogen problemlos möglich. Außerdem sind sowohl auf Ebene der GDI-DE als auch auf Ebene von INSPIRE Expertengremien damit befasst, die SensorThings API als Lösung für die Bereitstellung von Echtzeitdaten in diesen Kontexten aufzuzeigen (Kotsev et al. 2018).

2.4 Wie können Echtzeitdaten sicher bereitgestellt werden?

Zur Absicherung der Client-Server-Verbindung zwischen Datenquelle und Urban Data Platform ist eine verschlüsselte Verbindung und eine Zugriffskontrolle erforderlich. Dazu müssen Identität (Authentifizierung) und Zugriffsrechte (Autorisierung) eines Clients geprüft werden. Microservices lassen sich durch OpenID-Connect/OAuth2.0-konforme Identitätsmanagement-Lösungen wie bspw. Keycloak absichern (Keycloak 2020, OpenID 2020, IETF 2012).

3 Integration von Echtzeitdaten in die Urban Data Platform Hamburg

Um Echtzeitdaten in die HH_UDP zu integrieren, sind verschiedene freie und proprietäre Lösungen verfügbar. Im Folgenden wird eine Lösung präsentiert, die allen spezifizierten Anforderungen gerecht wird. Hervorzuheben ist, dass alle Software-Komponenten frei verfügbar sind und offene standardisierte Schnittstellen nutzen, wodurch eine direkte Nutzung von anderen Behörden oder Institutionen problemlos möglich ist. Die Bindung an Technologiepartner wird zudem deutlich abgeschwächt. Die Schematische Darstellung der Echtzeitdateninfrastruktur der Urban Data Platform Hamburg wird in Abb. 2 vorgestellt. Die dortigen Nummern (#) werden im Folgenden referenziert.

3.1 Die OGC SensorThings API als Basis für Echtzeitdaten in der HH_UDP

Die SensorThings API erfüllt alle notwendigen Anforderungen zur Realisierung einer Echtzeitdateninfrastruktur (vgl. #1 und #3). Die Basis der SensorThings API ist ein relationales Datenmodell. Dieses besteht aus acht Entitäten (Thing, Location, HistoricalLocation, Sensor, Observation, FeatureOfInterest, ObservedProperty und Datastream; siehe UML Diagramm der SensorThings API, https://ogc-iot.github.io/ogc-iot-api/datamodel.html). Jede Entität enthält einen Namen, eine Beschreibung und ein *properties*-Feld.

 © Wißner-Verlag

Letzteres wird in der API-Version 1.1 für alle Entitäten außer der Observation integriert. Im properties-Feld können nutzerspezifische Informationen hinterlegt werden. Zusätzlich hat jede Entität spezifische Felder, z. B. ein *location*-Feld für räumliche Informationen im GeoJSON-Format. Detailliertere Informationen sind in der Spezifikation beschrieben (www.ogc.org/standards/sensorthings).

3.2 Der FROST-Server als Kernkomponente für die Echtzeitdateninfrastruktur der HH_UDP

Es existieren verschiedene freie, offene oder proprietäre Implementierungen der SensorThings API. Zum Beispiel der GOST-Server (Alpha-Version: 0.5) von Geodan, eine Implementierung in der Programmiersprache Go (www.gostserver.xyz). Von Mozilla existiert eine Umsetzung in NodeJS (Mozilla 2017), die den ersten Teil der Sensor Things API unvollständig umsetzt. Das kanadische Unternehmen SensorUp Inc., das auch an der Entwicklung der SensorThings API beteiligt ist (OGC 2020b), bietet eine proprietäre Lösung an. Darüber hinaus hat 52°N die SensorThings API in sein System 52°North Sensor Observation Service integriert (52North 2020a und 52North 2020b). In der seit April 2020 verfügbaren Version 2.0.1 ist die SensorThings API nicht vollständig umgesetzt. Beispielsweise fehlt die Unterstützung für MultiDataStreams und Batch-Requests.

Der FROST-Server (Fraunhofer IOSB 2020a) ist eine vom Fraunhofer IOSB entwickelte Open-Source-Implementierung der SensorThings API (https://github.com/FraunhoferIOSB/FROST-Server), geschrieben in Java und veröffentlicht unter der GNU 3.0-Lizenz. Es ist die erste vollständige OpenSource-Implementierung der Sensor Things API Part 1 (FraunhoferIOSB 2020c) und wurde im Jahr 2016 von der OGC als solche zertifiziert. Der FROST-Server ist eine modular aufgebaute Applikation und als web application archive-Datei (WAR) zur Installation in einem Java-Web-Server wie Apache Tomcat oder Wildfly verfügbar. Zusätzlich stehen Docker-Images zur Verfügung.

In einem Proof of Concept (PoC) wurde in Zusammenarbeit von Fraunhofer IOSB, Microsoft und Dataport unter der Leitung des Landesbetrieb Geoinformation und Vermessung (LGV) eine skalierbare Serverarchitektur auf Basis von Docker, Kubernetes und dem FROST-Server entwickelt. Daraus entstand ein Helm Chart (https://helm.sh), bereitgestellt durch das Fraunhofer IOSB, das einen Quick-Start in einem Kubernetes-Cluster erlaubt (Fraunhofer IOSB 2020b). Der FROST-Server ist das zentrale Element der Echtzeitdateninfrastruktur der HH_UDP.

3.3 Aufbau der Echtzeitdateninfrastruktur in der HH_UDP

Die Echtzeitdateninfrastruktur der HH_UDP wurde initial in einem Kubernetes-Cluster in der Microsoft Azure Germany Cloud aufgebaut. Dort stand für Aufbau und Anpassung des Clusters das Kommandozeilentool *AKS (Azure Kubernetes Service)-engine* (Abb. 2 #1) zur Verfügung. In der Microsoft Azure Global Cloud kann dafür der vollständig verwaltete Kubernetes-Dienst AKS genutzt werden. Die AKS-engine benötigt eine YML-Datei, um alle gewünschten Azure-Ressourcen mit Kubernetes bereitzustellen. Die Datei enthält Angaben zu AnzahlSpezifikation und Betriebssystem der VMs sowie die Kubernetes-Version. Die Konfigurationsdatei kann unter Versionskontrolle gestellt und bei Bedarf leicht angepasst werden. In der HH_UDP wird Git (Git 2020) zusammen mit dem Repository-Management-System Bitbucket (Bitbucket 2020) zur Versionskontrolle genutzt.

Der HH_UDP-Kubernetes-Cluster besteht aus drei Mastern und vier Arbeitsknoten. Die Master verwalten das Gesamtsystem, während Anwendungen auf den Arbeitsknoten laufen. Dieses Multi-Master-Setup erhöht die Ausfallsicherheit. Zur Steuerung des Clusters und zur Installation von Anwendungen wird das Kommandozeilenprogramm *kubectl* (Abb. 2 #2) genutzt, welches ebenfalls auf der *Cluster-Administrations-VM* (Abb. 2 #3) installiert ist (https://kubernetes.io/de/docs/tasks/tools/install-kubectl). Administratoren verbinden sich mit dieser Maschine von ihrer lokalen Arbeitsumgebung (Abb. 2 #4), um den Cluster zu administrieren. Mit kubectl können einfache Anwendungen mit wenigen Kubernetes-Objekten problemlos administriert werden. Bei komplexen Microservice-Architekturen ist dies jedoch sehr aufwändig. Daher wird *Helm* (Abb. 2 #5) als Administrationswerkzeug für Konfiguration, Installation und Versionierung genutzt. Das zugehörige Helm Chart enthält alle Komponenten, um die Echtzeitdateninfrastruktur der HH_UDP zu betreiben (https://bitbucket.org/geowerkstatt-hamburg/hh-udp-iot). Es ist frei nutzbar und kann mit Helm heruntergeladen und in einem eigenen Kubernetes-Cluster ausgerollt werden. Damit ist eine einfache Übertragbarkeit der Echtzeitdateninfrastruktur auf andere Urban Data Platformen möglich und in München schon erfolgreich umgesetzt worden.

Die zentrale Komponente der Echtzeitdateninfrastruktur der HH_UDP ist der FROST-Server (Abb. 2 #6). Dieser besteht aus zustandslosen Microservices. Sämtliche Daten werden außerhalb des Kubernetes-Clusters in einem Datenbank-Cluster gespeichert (Abb. 2 #7). Der Datenbank-Cluster besteht aus zwei VMs, die über eine Master-Replica-Beziehung miteinander verbunden sind. Der Master (Abb. 2 #7a) dient als primärer Speicher für neue Daten. Per »streaming replication« werden Daten kontinuierlich auf den Replica übertragen (Abb. 2 #7b). Lesezugriffe sind daher auf beiden Datenbanken möglich, wodurch Ausfallsicherheit und Datendurchsatz erhöht werden.

Damit der FROST-Server über eine verschlüsselte Internetverbindung erreichbar ist, müssen dem Kubernetes-Cluster weitere Anwendungen hinzugefügt werden. Mit einer *Ingress* (Abb. 2 #8) genannten Ressource können HTTP-/HTTPS- und MQTT-Anfragen an einen Microservice geroutet werden. In der HH_UDP wird *Ingress-nginx*

© Wißner-Verlag

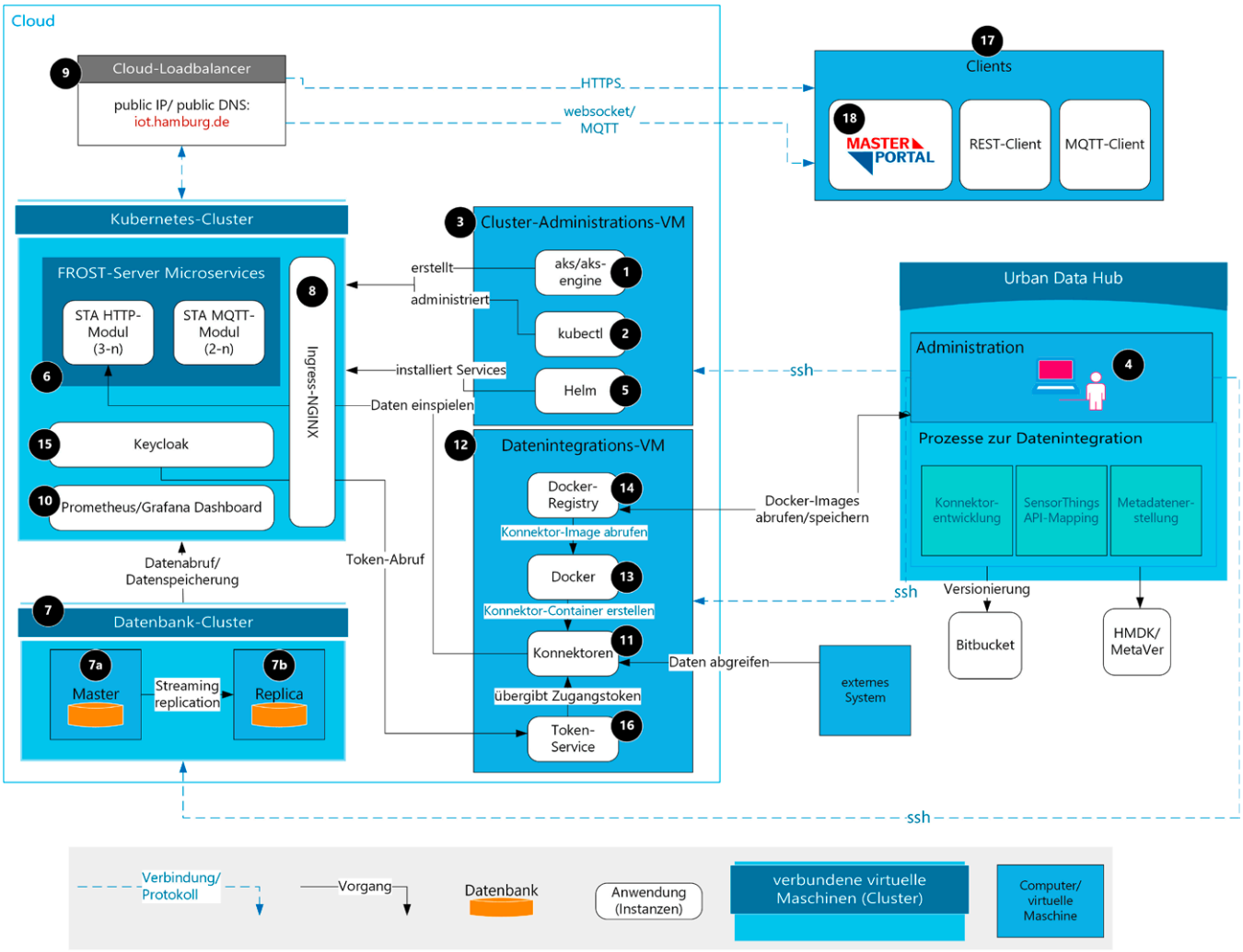

Abb. 2: Schematische Darstellung der Echtzeitdateninfrastruktur der Urban Data Platform Hamburg

(Ingress-nginx 2020) eingesetzt, da er auch das MQTT-Protokoll unterstützt. Bei der Installation eines Ingress im Cluster wird ein Cloud-Loadbalancer (Abb. 2 #9) erstellt, der die im Ingress angegebenen DNS-Einträge und Routen an den jeweiligen Dienst im Cluster weiterleitet. Erst durch den Cloud-Loadbalancer werden die Anwendungen im Cluster über eine externe IP und einen DNS-Eintrag (*iot.hamburg.de*) im Internet erreichbar.

Zum Verschlüsseln der Client-Server-Verbindungen müssen TLS-Zertifikate von einer vertrauenswürdigen Zertifizierungsstelle (z. B. »Let's Encrypt«, https://letsencrypt.org) im Cluster hinterlegt werden. Zur Automatisierung des Zertifizierungsprozesses wird das Programm »cert-manager« verwendet (Jetstack 2020). Es läuft als Kubernetes-Service und ist für den voll automatischen Bezug und die regelmäßige Erneuerung der Zertifikate konfiguriert.

Zur Überwachung des Kubernetes-Clusters wurde ein Prometheus/Grafana-Dashboard (Abb. 2 #10) implementiert. In dem Dashboard können Anzahl, Verteilung und Status der Anwendungsinstanzen auf den VMs überwacht werden, um notwendige Performanceanpassungen vornehmen zu können. Das Dashboard kann über einen eigenen DNS-Eintrag erreicht werden.

Zusätzlich erfolgt eine automatische Überwachung und Skalierung der einzelnen Microservices. Steigt die CPU-Nutzung einer FROST-Server-Komponente, z. B. durch viele Client-Anfragen, werden zusätzliche Instanzen dieser Komponente erzeugt. In Kubernetes werden die kleinsten ausführbaren Einheiten *Pods* genannt, jeder Pod enthält maximal eine FROST-Server-Komponente. Jeder Pod nutzt einen bestimmten Anteil der Rechenleistung eines Arbeitsknotens. Übersteigt die benötigte Leistung eines Pods eine bestimmte Schwelle, werden automatisch zusätzliche Instanzen erzeugt. Dazu muss ein sogenannter »Horizontal Pod Autoscaler« eingerichtet werden. Darin werden die Schwellenwerte für die den Pods zur Verfügung stehenden Rechenressourcen und die minimale und maximale Anzahl der Pods deklariert.

Für die Datenintegration werden zwei ETL-Tools (Extraction, Transformation, Loading) zur Erstellung von Konnektoren (Abb. 2 #11) verwendet: FME-Workbenches, die auf einem FME-Server ausgeführt werden (Safe Software Inc 2020), und containerisierte Pythonskripte, die auf einer separaten VM (Datenintegrations-VM; Abb. 2 #12) laufen, die wiederum mit Docker (Abb. 2 #13) und einer Registry für Docker-Images (Abb. 2 #14) ausgestattet ist. Dabei werden externe Daten über individuelle Schnittstel-

 © Wißner-Verlag

len abgerufen und über die standardisierte REST-Schnittstelle des FROST-Servers in die HH_UDP übertragen. Externe Systeme, deren Schnittstellen auf offenen Standards basieren, sind dabei deutlich einfacher anzubinden als individuell erstellte (meist proprietäre) Schnittstellen.

Um Daten in den FROST-Server zu integrieren, muss jeder Zugriff eines Konnektors autorisiert werden. Dies geschieht via *OpenID Connect* (https://openid.net/connect) mittels *Json Web Token* (https://jwt.io). In der HH_UDP wird die Single Sign-on-Lösung *Keycloak* (Abb. 2 #15) für die Identitäts- und die Zugriffsverwaltung benutzt. Zur automatisierten Json-Web-Token-Bereitstellung für Schreibzugriffe wurde ein Token-Service (Abb. 2 #16) entwickelt, der zeitlich begrenzt gültige Zugangstoken generiert und diese in einem bestimmten Bereich des jeweiligen ETL-Tools abspeichert. In FME und Python werden dazu separate Prozesse betrieben. Beim Übertragen neuer Daten werden diese Zugangstoken im Header der HTTP-POST-Anfrage mitgeschickt und von einem Keycloak-Adapter validiert, der Teil des HTTP-Moduls des FROST-Servers ist. Darüber hinaus können dem Zugangstoken Rechte für weitere Anwendungen im Kubernetes-Cluster oder externe Webressourcen hinzugefügt werden oder zusätzliche Token mit anderen Zugangsrechten ausgestellt werden. Zur Identitätsverwaltung können auch externe Systeme, wie LDAP und Active Directory, genutzt werden.

3.4 Die Komponenten des FROST-Servers

Clients (Abb. 2 #17), die sich mit einem Echtzeitdatendienst der HH_UDP verbinden, erscheint der Cluster als ein System, dass eine REST-Schnittstelle und einen MQTT-Broker entsprechend der SensorThings API-Spezifikation zur Verfügung stellt. Beide sind unter der Adresse iot.hamburg.de erreichbar. Diese beiden Endpunkte werden jeweils durch HTTP- und MQTT-Module des FROST-Servers (Abb. 2 #6) zur Verfügung gestellt.

Der FROST-Server setzt sich aus insgesamt vier Komponenten zusammen:

1) Als Datenspeicher wird eine relationale PostgreSQL-Datenbank mit PostGIS-Erweiterung genutzt.
2) Ein HTTP-Modul exponiert die REST-Schnittstelle; hierüber können die HTTP-Anfragemethoden GET, POST, PUT, PATCH und DELETE ausgeführt werden.
3) Ein MQTT-Modul stellt einen MQTT-Message Broker (Moquette, https://moquette-io.github.io/moquette/) mit Publish/Subscribe-Funktionalität bereit.
4) Ein Message-Bus verteilt Informationen zwischen den Instanzen des HTTP- und des MQTT-Moduls.

Die FROST-Server-Komponenten 2 bis 5 laufen in separaten Pods und spielen wie folgt zusammen: Wenn ein Client (Abb. 3 #1), z. B. ein Konnektor mit einer HTTP-POST-Anfrage, neue Daten in das System überträgt, werden diese vom HTTP-Modul (Abb. 3 #2) angenommen. Das kann z. B. die Anzahl verfügbarer Fahrräder an einer Ausleihstation sein (https://metaver.de/trefferanzeige?docuuid=D18F375E-FA5F-4998-AFF8-557969F44479). Das HTTP-Modul schreibt diese Daten in den Datenbank-Cluster (Abb. 3 #3), kreiert eine SensorThings API-konforme Nachricht mit Navigationslinks auf die nächsten Entitäten, stellt diese auf dem REST-Endpoint bereit und informiert den Message-Bus (Abb. 3 #4) über die Änderung. Der wiederum gibt diese Nachricht an alle MQTT-Module (Abb. 3 #5) weiter, die eine STA-konforme Nachricht an alle auf die entsprechende Entität abonnierten Clients (Abb. 3 #6) ausliefern. Diese Nachricht kann auch über das HTTP-Modul von einem Client abgefragt werden (Abb. 3 #7). Dabei ist es unerheblich, auf welchem Arbeitsknoten ein MQTT-/HTTP-Modul oder der Message-Bus instanziiert ist.

Neben einem direkten Aufruf der REST-API im Browser oder einem Abonnement am MQTT-Broker mit einem MQTT-Client ist es möglich, z. B. mit der freien Webportalsoftware »Masterportal« (www.masterportal.org, Abb. 2 #18) Echtzeitdatendienste der HH_UDP als Layer einzubinden und zu visualisieren. Das

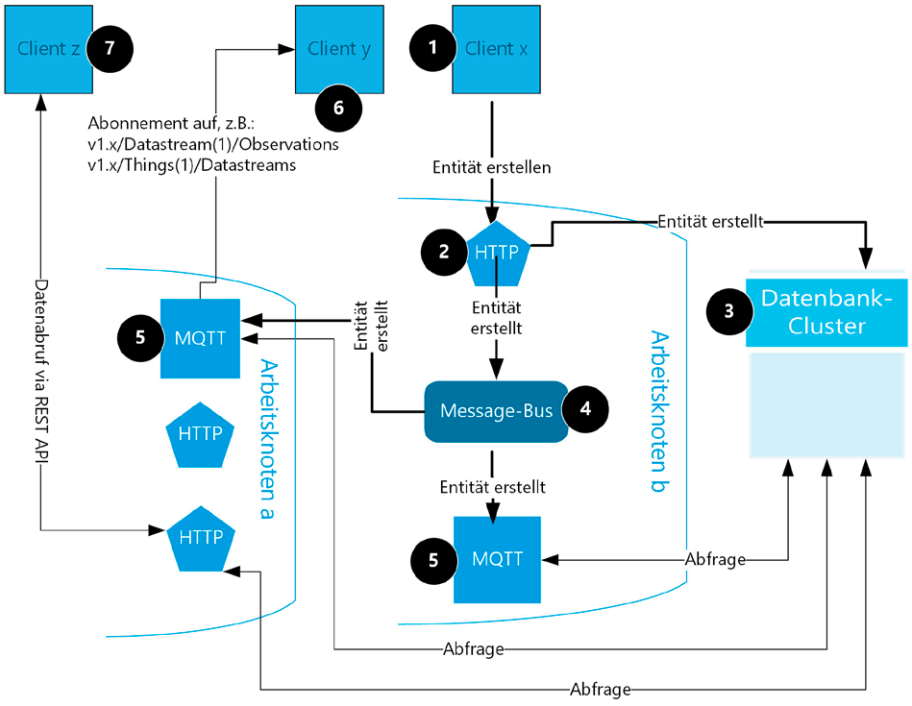

Abb. 3:
Schematische Darstellung der Funktion des FROST-Servers in einem Kubernetes-Cluster

© Wißner-Verlag

Masterportal lädt statische und historische Daten über die REST-API (iot.hamburg.de), während dynamische Daten über MQTT, in Websocket verpackt, bezogen werden. Es werden also die Vorteile der jeweiligen Protokolle in einer Webanwendung verbunden, um die Möglichkeiten der SensorThings API optimal zu nutzen (Landesbetrieb Geoinformation und Vermessung 2020).

3.5 Integration neuer Echtzeitdaten in die HH_UDP

Zur Integration von Echtzeitdaten in die HH_UDP können aktuell alle gängigen echtzeitfähigen ETL-Tools benutzt werden, die HTTP-POST-Anfragen oder MQTT-Publish-Anfragen senden können. Darüber hinaus muss es möglich sein, einen JSON Web Token in einem angepassten HTTP-Header oder bei einer MQTT-Publish-Message als Passwort zu übertragen. Sowohl die Wahl des ETL-Tools als auch die Nutzung bzw. Definition des Mappings auf die Entitäten der SensorThings API sind anforderungs- und entwicklerabhängig und werden deswegen hier nicht weiter ausgeführt.

Um die via SensorThings API bereitgestellten Daten für die Öffentlichkeit technisch und inhaltlich zugänglich zu machen, publiziert der LGV separate Daten- und Dienstbeschreibungen im Hamburger Metadatenkatalog (Hamburger Metadatenkatalog 2020) und im Katalog des Metadatenverbunds METAVER (https://metaver.de). Das bedeutet, dass eine Datensatzbeschreibung mit Informationen zu Datenherausgeber, Nutzungsbedingungen und anderen rechtlichen Rahmensetzungen sowie eine Dienstbeschreibung für jeden Dienst über den Daten veröffentlicht werden, z. B., ob WFS/WMS verfügbar sind. Die Datenabgabe via SensorThings API wird im Folgenden als STA-Dienst bezeichnet. Ein Beispieldatensatz hierfür wären die Informationen zur Belegung der Hamburger Elektro-Ladestandorte (Abb. 4, Tab. 1).

Neben diesen Metadatensatzeinträgen gibt es einen übergeordneten STA-Dienst-Eintrag, in welchem Informationen zur Implementierung der SensorThings API in der HH_UDP hinterlegt sind (https://metaver.de/trefferanzeige?docuuid=19A339AE-FD6E-4551-9AD7-F9624C8A55FF). Zudem gibt es einen Eintrag, der über die Nutzung des MQTT-Brokers erläutert (https://metaver.de/trefferanzeige?cmd=doShowDocument&docuuid=785D987C-AAFF-471D-AE3A-EBCD4C9E23F1).

Alle STA-Dienstbeschreibungen sind standardisiert aufgebaut. In der Beschreibung wird kurz das Mapping der Daten auf die Entitäten der SensorThings API beschrieben

Tab. 1: Metadaten zu Elektro-Ladestandorte Hamburg

Name	Link
Elektro-Ladestandorte Hamburg	https://metaver.de/trefferanzeige?cmd=doShowDocument&docuuid=07302A1A-3250-4F0F-9F6E-96C508288D03
WMS-Elektro-Ladestandorte Hamburg	https://metaver.de/trefferanzeige?cmd=doShowDocument&docuuid=95BABB73-CB02-4DB5-BCFC-8CD593949C28
WFS-Elektro-Ladestandorte Hamburg	https://metaver.de/trefferanzeige?cmd=doShowDocument&docuuid=86611577-BB2D-492B-8809-E6AC8D361F6B
STA-Elektro-Ladestandorte Hamburg	https://metaver.de/trefferanzeige?cmd=doShowDocument&docuuid=0DE61CE3-0DC2-4C5D-8A18-7E388F52AD5E

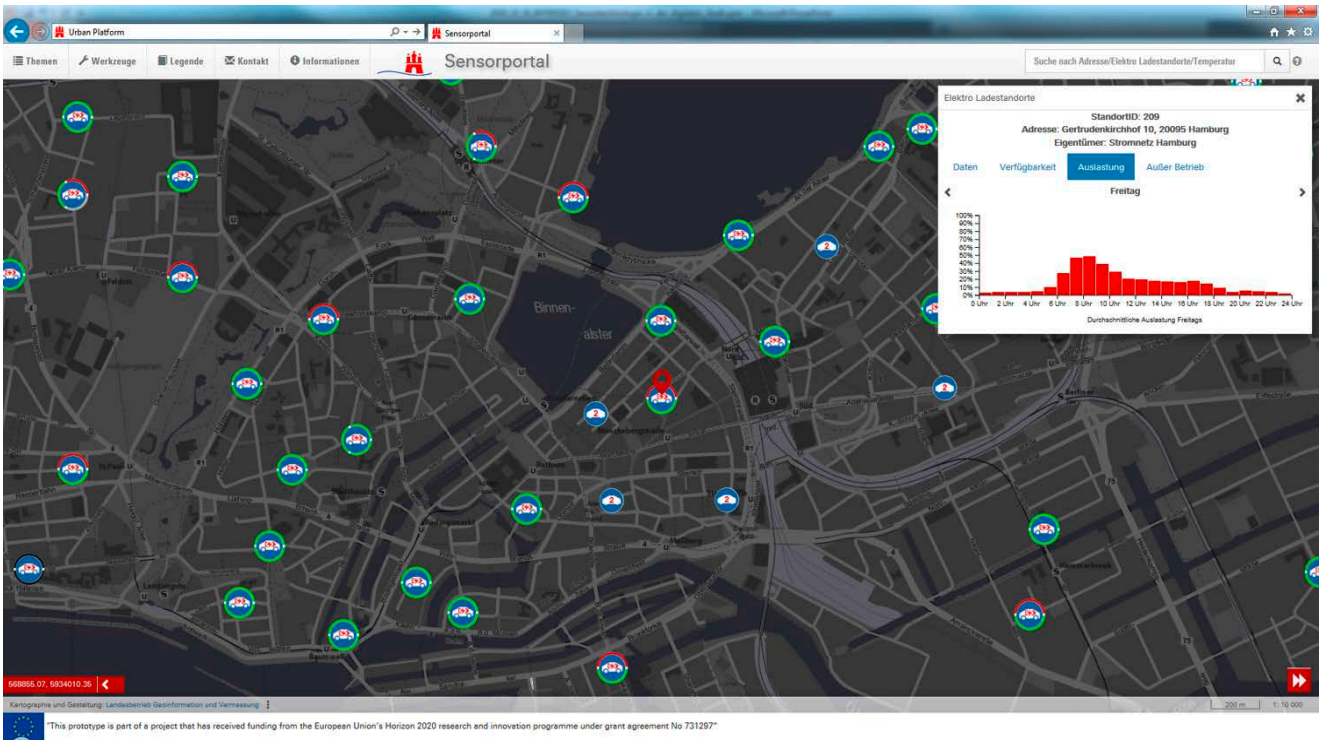

Abb. 4: Sensorportal: STA-Elektro-Ladesäulen

 © Wißner-Verlag

und Informationen zum Umgang mit den JSON-Objekten gegeben. Zur Erleichterung des Zugriffs wurden im properties-Feld des Datastreams die keys »serviceName« und »layerName« eingeführt. Diese können analog zu einem GIS verwendet werden. Entsprechende Key-Value-Paare sind als JSON-Objekte aufgelistet. Als letztes folgen Informationen zum Abonnieren des Datenstroms über den MQTT-Broker.

Unter dem Reiter »Verweise« gibt es folgende Beispiellinks zum Herunterladen der Daten: (1) Eine Gesamtserviceabfrage mit Informationen aller Entitäten inkl. der letzten drei Beobachtungswerte, (2) eine Abfrage für jeden Layer, der Informationen zu den Datastreams sowie die letzten zehn Beobachtungswerte enthält, sowie (3) eine Standortabfrage.

4 Weiterentwicklungen in der Echtzeitdateninfrastruktur der Urban Data Platform Hamburg

Ziel der Weiterentwicklung der Echtzeitdateninfrastruktur der HH_UDP sind einerseits eine Verbesserung der Datenbereitstellung über Konnektoren sowohl innerhalb der verwendeten Programmiersprache Python als auch unter Verwendung weiterer Lösungen wie bspw. Apache Kafka, Apache Flink oder Node-RED. Im Fokus stehen dabei die einfache Erstellung und Wartung sowie die Maximierung der Übertragungsgeschwindigkeit. So werden »LowCode«-Alternativen wie Node-RED (https://nodered.org) geprüft, um die Echtzeitdatenintegration zu vereinfachen und die Abhängigkeit von Softwareentwicklern zu reduzieren (OpenJS Foundation 2020). Node-RED basiert auf NodeJS und bietet ein visuelles Programmierinterface, das im Browser aufgerufen werden kann. Damit können ohne textbasierte Programmierung Anwendungen zur Datenverarbeitung erstellt werden, beispielsweise Konnektoren.

Die Python-Konnektoren werden derzeit auf einer separaten VM ausgeführt. Mit der Containerisierung der Konnektoren ist bereits der Grundstein gelegt worden, diese im Kubernetes-Cluster auszuführen. Dadurch werden alle Vorteile, die Kubernetes bietet, auch für Konnektoren verfügbar – dies würde sowohl Ressourcen- als auch Administrationsaufwand deutlich reduzieren.

5 Langfristige Ziele der Integration von Echtzeitdaten in die Urban Data Platform Hamburg und Fazit

Die HH_UDP liefert sowohl statische als auch dynamische Daten und bildet somit die technische Grundlage eines *Digitalen Urbanen Zwillings (Digital Urban Twin, DUT)* der Stadt Hamburg, mit dem zum Beispiel die Stadtplanung optimiert werden kann. Die Zielsetzung ist dabei folgendem Auszug aus Hamburgs Digitalstrategie zu entnehmen: »*… im DUT können beispielsweise nicht nur Baumaßnahmen optimiert werden. Zeitgleich könnten in einem DUT durch die Verzahnung mit anderen Datenquellen (z. B. DigITALL) potenzielle Wechselwirkungen des städtischen Lebens (Verkehr, Baustellen, Wetterlagen, Events) simuliert und berücksichtigt werden*« (Hamburg 2020, S. 34). Das bedeutet, dass die eventbasierte und hochgradig automatisierte Bereitstellung von Daten in Echtzeit eine zentrale Rolle in allen Bereichen einer modernen und vernetzten Stadt, einer *Smart City*, spielt. In Zukunft ist es denkbar, dass Daten aus einem DUT raum-zeitlich miteinander verknüpft werden. Bauteile eines Gebäudes aus einem dreidimensionalen Stadtmodell könnten selbst Sensoren sein, die einen Ortsbezug haben und selber wiederum Messdaten liefern. Hier gilt es, offene Standards zur Datenbereitstellung und Visualisierung, z. B. im Bereich der virtuellen und erweiterten Realität (VR/AR), zu entwickeln.

Weiterhin sollen innovative Projekte zu intelligenten Transportsystemen den Verkehrsfluss verbessern, um innerstädtische Emissionen sowie Staubildung zu reduzieren. Dabei kann dank der automatisierten Erfassung von Verkehrsmengen in den Projekten aVME (automatisierte Verkehrsmengenerfassung) und HaRaZäN (Hamburger Radzählnetz) der aktuelle Verkehrsfluss in Navigationsanwendungen sowohl für Kraftfahrzeuge als auch für Fahrräder integriert werden. In dem Projekt GeoNetBake werden Warnbaken von Baustellen mit Positionssensoren ausgestattet, damit Absperrungen auf Straßen und Radwegen unmittelbar veröffentlicht werden können. Zudem sollen Echtzeitdaten von Lichtsignalanlagen (Signale, Signalanforderungen z. B. durch Busse oder Fußgänger) bereitgestellt werden und damit berücksichtigt werden können.

Diese und andere dynamische Daten der Stadt Hamburg werden über die SensorThings API und die Domain iot.hamburg.de veröffentlicht. Dadurch soll zu jedem Zeitpunkt ein umfassendes Bild des Geschehens in der Stadt frei oder zugriffsgeschützt online zur Nutzung von Bürgerinnen, Wirtschaft, Wissenschaft und Verwaltung verfügbar sein.

Da Zugriffe auf diese Infrastruktur nicht planbar sind und vielfach anonym erfolgen, braucht es ein flexibles automatisch skalierendes IT-System, das bei geringem manuellen Wartungsaufwand mit der rasant wachsenden Anzahl von Datenquellen und IoT-Sensoren Schritt hält. Eine Blaupause für ein solches Produktivsystem stellt die hier beschriebene Echtzeitdateninfrastruktur der HH_UDP auf Basis von freier Open-Source Software dar. Die Nutzung von freier Software ist dabei unerlässlich, da der Technologieaustausch mit anderen deutschen und europäischen Städten dadurch vereinfacht und somit aktiv gefördert wird. Daher stellt die HH_UDP eine vollständige Dokumentation incl. Helm-Chart zur freien Verfügung (https://bitbucket.org/geowerkstatt-hamburg/hh-udp-iot). Langfristig soll dadurch eine einheitliche Bereitstellung von dynamischen Daten der öffentlichen Verwaltung erreicht und somit die Nutzbarkeit öffentlicher Daten innerhalb der Städte, durch Firmen, interessierte Bürger und in der Wissenschaft massiv vereinfacht werden.

© Wißner-Verlag

Dank

Die Entwicklung der Echtzeitdateninfrastruktur ist Teil eines Projekts, das durch das Forschungs- und Innovationsprogramm Horizont 2020 der Europäischen Union im Rahmen der Zuschussvereinbarung Nr. 731297 finanziert wird. Unser Dank für die sehr gute und konstruktive Zusammenarbeit geht an: Fraunhofer Institut IOSB, Microsoft, Dataport, Daniel Atlan (ENGIE Frankreich), Timo Ruohomäki (Forum Virium, Helsinki), Sascha Tegtmeyer, Thomas Eichhorn und Bianca Lüders.

Literatur

52North (2020a): 52North. https://52north.org/software/software-projects/sos, letzter Zugriff 05/2020.

52North (2020b): 52North/sensorweb-server-sta. https://github.com/52North/sensorweb-server-sta, letzter Zugriff 15.05.2020.

Bitbucket (2020): Bitbucket. https://bitbucket.org, letzter Zugriff 05/2020.

BMZ (2017): Zukunftsvertrag für die Welt – Agenda 2030 für nachhaltige Entwicklung. Bundesministerium für wirtschaftliche Zusammenarbeit und Entwicklung (BMZ), Referat Öffentlichkeitsarbeit, digitale Kommunikation, Besucherdienst Bundesministerium für wirtschaftliche Zusammenarbeit und Entwicklung (BMZ). www.bmz.de/de/themen/2030_agenda, letzter Zugriff 09/2020.

BVM aVME (2020): Automatisierte Verkehrsmengenerfassung – aVME. Behörde für Verkehr und Mobilitätswende. www.hamburg.de/bvm/weltkongress-2021/12758118/news-verkehrszaehlprojekt, letzter Zugriff 09/2020.

BVM GeoNetBake (2020): Einsatz sensorgestützter Baken auf Straßenbaustellen – GeoNetBake. https://lsbg.hamburg.de/its-projekte/#anker_7, letzter Zugriff 09/2020.

BVM HaRaZäN (2020): Hamburger Radverkehrszählnetz – HaRaZäN. Behörde für Verkehr und Mobilitätswende. www.hamburg.de/bvm/projekte-its/12323778/radverkehrszaehlnetz, letzter Zugriff 09/2020.

BVM ITS-Projekte (2020): Intelligente Verkehrs- und Transportsysteme – ITS. www.hamburg.de/bvm/projekte-its, letzter Zugriff 09/2020.

BVM TLF 2.0 (2020): Traffic Light Forecast 2.0 – TLF 2.0. https://lsbg.hamburg.de/its-projekte/#anker_4, letzter Zugriff 09/2020.

DIN SPEC 91357:2017-12: Referenzarchitekturmodell Offene Urbane Plattform (OUP).

EIP SCC (2020): https://eu-smartcities.eu, letzter Zugriff 15.05.2020.

ETSI (2020): www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.02.02_60/gs_CIM009v010202p.pdf, letzter Zugriff 15.05.2020.

EU (2007): Richtlinie 2007/2/EG des Europäischen Parlaments und des Rates vom 14. März 2007 zur Schaffung einer Geodateninfrastruktur in der Europäischen Gemeinschaft (INSPIRE). Amtsblatt der Europäischen Union, 14.03.2007.

EU (2019): RL (EU) 2019/1024.

Fraunhofer (2020a): Fraunhofer IOSB. www.iosb.fraunhofer.de/servlet/is/80113, letzter Zugriff 15.05.2020.

Fraunhofer (2020b): FraunhoferIOSB/FROST-Server. https://github.com/FraunhoferIOSB/FROST-Server/tree/master/helm/frost-server, letzter Zugriff 15.05.2020.

Fraunhofer (2020c): FraunhioferIOSB/FROST-Server/releases. https://github.com/FraunhoferIOSB/FROST-Server/releases, letzter Zugriff 15.05.2020.

GeoZG (2020): Gesetz über den Zugang zu digitalen Geodaten (Geodatenzugangsgesetz – GeoZG). www.gesetze-im-internet.de/geozg/BJNR027800009.html, letzter Zugriff 09/2020.

Git (2020): Git. https://git-scm.com, letzter Zugriff 15.05.2020.

Hamburg (2009): Hamburgisches Geodateninfrastrukturgesetz (Hmb GDIG). Hamburgisches Gesetz- und Verordnungsblatt, 2009: 528.

Hamburg (2012): Hamburgisches Transparenzgesetz (HmbTG). Hamburgisches Gesetz- und Verordnungsblatt, 2012: 271.

Hamburg (2020): Digitalstrategie Hamburg. Strategiepapier, Hamburg: Freie und Hansestadt Hamburg Senatskanzlei – Amt für IT und Digitalisierung, Drucksache Nr. 21/19800 (auszugsweise, 2. überarbeitete Aufl.).

Hamburger Metadatenkatalog (2020): http://hmdk.fhhnet.stadt.hamburg.de, letzter Zugriff 15.05.2020.

Hertweck, P., Hellmund, T., van der Schaaf, H., Moßgraber, J, Blume, J.-W. (2019): Management of Sensor Data with Open Standards. 16th International Conference on Information Systems for Crisis Response and Management, ISCRAM 2019. Conference proceedings. Valencia (Spain). 1126–1139.

IETF (2020): The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749, letzter Zugriff 15.05.2020.

IETF (2011): The Websocket Protocol. https://tools.ietf.org/html/rfc6455, letzter Zugriff 15.05.2020.

Ingress-nginx (2020): NGINX Ingress Controller. https://kubernetes.github.io/ingress-nginx, letzter Zugriff 15.05.2020.

Jetstack (2020): cert-manager. www.jetstack.io/cert-manager, letzter Zugriff 15.05.2020.

Keycloak (2020): Keycloak. www.keycloak.org, letzter Zugriff 15.05.2020.

Kirschner, M. (2019): Public Money Public Code – Modernising Public Infrastructure with Free Software. Free Software Foundation Europe (FSFE). Berlin (Germany).

Kotsev, A., Schleidt, K., Liang, S., van der Schaaf, H., Khalafbeigi, T., Grellet, S., Lutz, M., Jirka, S., Beaufils, M. (2018): Extending INSPIRE to the Internet of Things through SensorThings API. Geosciences. 221.

Landesbetrieb Geoinformation und Vermessung (2020): Masterportal. www.masterportal.org, letzter Zugriff 15.05.2020.

Mozilla (2017): mozilla-sensorweb/sensorthings. https://github.com/mozilla-sensorweb/sensorthings, letzter Zugriff 15.05.2020.

Muhammad, A., Afzal, B., Imran, B., Tanwir, A., Akbar, A. H., Shah, G. (2019): OneM2M Architecture Based Secure MQTT Binding in Mbed OS. IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Stockholm (Schweden). 48–56.

OGC (2020a): Sensor Observation Service OGC. www.ogc.org/standards/sos, letzter Zugriff 04/2020.

OGC (2020b): Open Geospatial Consortium. http://docs.opengeospatial.org/is/15-078r6/15-078r6.html, letzter Zugriff 15.05.2020.

OpenID (2020): OpenID. https://openid.net/developers/specs, letzter Zugriff 15.05.2020.

OpenJS Foundation (2020): Node-RED. https://nodered.org, letzter Zugriff 15.05.2020.

Red Hat (2020): Microservices. www.redhat.com/de/topics/microservices/what-are-microservices, letzter Zugriff 15.05.2020.

Safe Software Inc. (2020): www.safe.com, letzter Zugriff 15.05.2020.

Schmitz, S. (2012): Geoportal.DE – Ein Blick in die Geodateninfrastruktur Deutschland. In: zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement, Heft 2/2012, 137. Jg., 69–74.

Shrestha, S. R., Tisdale, M., Kopp, S., Rose, B. (2018): Earth Observation and Geospatial Implementation: Fueling Innovation in a Changing World. In: Earth Observation Open Science and Innovation, Vol. 15, by Mathieu, P. P., Aubrecht, C., edited by ISSI Scientific Report Series. Springer.

van Oosterhout, M., Holst, J. A., Sheombar, H., van Heck, E. (2020): Research Study on Urban Data Platforms in Europe. Forschungsbericht. Rotterdam: Erasmus Centre for Data Analytics.

W3C (2020): www.w3.org/Protocols, letzter Zugriff 15.05.2020.

Welzel, R.-W., Eichhorn, T. (2016): Stadtentwicklung in der Digitalen Stadt Hamburg – modern, innovativ, zukunftssicher. In: zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement, Heft 5/2016, 141. Jg., 322–329. DOI: 10.12902/zfv-0135-2016.

Welzel, R.-W., Hopfstock, A., Fischer, M., Friehl, M., Neumetzger, T. (2019). GDI-DE – Aktuelle Herausforderungen und Perspektiven. In: fub, 5/2019: 1–6.

Kontakt

Dr. Michael Fischer | Dr. Pierre Gras | Sonja Löwa | Stefan Schuhart
Urban Data Hub, Landesbetrieb Geoinformation und Vermessung
Neuenfelder Straße 19, 21109 Hamburg
michael.fischer1@gv.hamburg.de | pierre.gras@gv.hamburg.de
sonja.loewa@gv.hamburg.de | stefan.schuhart@gv.hamburg.de

 © Wißner-Verlag