Received March 31, 2022, accepted May 11, 2022, date of publication June 6, 2022, date of current version June 13, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3180365

Toward Reference Architectures: A Cloud-Agnostic Data Analytics Platform Empowering Autonomous Systems

ATTILA CSABA MAROSI[1], MÁRK EMŐDI[1], ATTILA FARKAS[1], RÓBERT LOVAS[1], RICHÁRD BEREGI[2], GIANFRANCO PEDONE[2], BALÁZS NÉMETH[3], (Member, IEEE), AND PÉTER GÁSPÁR[3]

[1]Laboratory of Parallel and Distributed Systems, Institute for Computer Science and Control, Eötvös Lóránd Research Network, 1518 Budapest, Hungary
[2]Centre of Excellence in Production Informatics and Control, Institute for Computer Science and Control, 1111 Budapest, Hungary
[3]Systems and Control Laboratory, Institute for Computer Science and Control, Eötvös Lóránd Research Network, 1518 Budapest, Hungary

Corresponding author: Attila Csaba Marosi (atisu@sztaki.hu)

This work was supported in part by the National Research, Development and Innovation (NRDI) Office—NKFIH, Hungary, under Grant ED_18-2-2018-0006 and Grant SNN 129178; and in part by the Ministry of Innovation and Technology NRDI Office within the framework of the Autonomous Systems National Laboratory Program. The work of Róbert Lovas was supported by the Janos Bolyai Research Scholarship of the Hungarian Academy of Sciences.

ABSTRACT This work introduces a scalable, cloud-agnostic and fault-tolerant data analytics platform for state-of-the-art autonomous systems that is built from open-source, reusable building blocks. As the baseline for further new reference architectures, it represents an architecture blueprint for processing, enriching and analyzing various feeds of structured and non-structured input data from advanced Internet-of-Things (IoT) based use cases. The platform builds on industry best practices, leverages on solid open-source components in a reusable fashion, and is based on our experience gathered from numerous IoT and Big Data research projects. The platform is currently used in the framework of the National Laboratory for Autonomous Systems in Hungary (abbreviated as ARNL). The platform is demonstrated through selected use cases from ARNL including the areas of smart/autonomous production systems (collaborative robotic assembly) and autonomous vehicles (mobile robots with smart vehicle control). Finally, we validate the platform through the evaluation of its streaming ingestion capabilities.

INDEX TERMS Reference architecture, blueprint, data analytics, autonomous systems, IoT, IIoT, big data, mobile robots, collaborative robots, smart control.

I. INTRODUCTION

Data is everywhere and the capability of fast processing and data-based decision making is more and more distinguishing top organizations from the rest. In the world of Internet of Things (IoT), the capability of ingesting, analyzing and fast response was always a basic requirement, however building solutions that enable this is still not a straight-forward process. Architecture blueprints allow the reuse of existing knowledge and best practices when creating new solutions. There are different definitions proposed, e.g., [1]–[4] with the main ideas of (i) promoting re-usability, (ii) incorporating best practices, (iii) using high or low abstraction levels, and (iv) serving certain use cases. However, most existing architecture blueprints either are too abstract; not end-to-end; not vendor agnostic or not open-source; or some combination of the above. For example Microsoft Azure currently offers 77 architecture blueprints for data analytics [5], and AWS offers 55 reference architectures for big data and data analytics [6]. These are all aimed at commercializing their cloud-based services, respectively. On the contrary, the "Industrial Internet Reference Architecture" [7] and "Reference Architectural Model Industry 4.0" [8] represent the joint effort of several organizations and companies, however, they are high-level concepts. In this work we augment these results by presenting our cloud-agnostic and open-source data analytics platform. It represents a reusable architecture blueprint for processing, enriching and analyzing different feeds of structured and non-structured input data. The platform is defined both at a high-level, and also we

The associate editor coordinating the review of this manuscript and approving it for publication was Md Arafatur Rahman.

present a current implementation used for IoT and Industrial IoT (IIoT) use cases.

The main contributions of this work are as follows. First, we propose a scalable, cloud-agnostic and fault-tolerant data analytics platform for advanced IoT use cases. The platform is built from different reusable building blocks and itself is a reusable architecture blueprint. It is designed for processing, enriching and analyzing different feeds of structured and non-structured input data. It is built on our previous results [9]–[11], and it is continuously adapting to the requirements of existing and new use cases. For each major block (represented by stages in our architecture) we rely on stable open-source components that have also multiple commercial platform-based alternatives when needed.

Second, we demonstrate the value of the platform through two use cases. The first use case being the guaranteeing safe motion of mobile robots for logistic and transport process using cloud-aided learning. The second presented use case addresses IIoT data collection and enrichment from a robotic assembly scenario over a service-oriented manufacturing execution system, in which the collaborative robotic arm near real time-data provision provides the foundation necessary for a better understanding of the while process.

The paper is organized as follows. Section II introduces related works in the fields of similar data analytics platforms developed either for the use cases within an organization or as a blueprint or reference architecture available from a public cloud provider. Section III details the design decisions and the architecture of the platform. This includes the iterations of services used for the different part of the data pipeline within the platform, the deployment methods. Section IV introduces the two use cases from the ARNL project that utilize the platform for data collection and as a cloud-based extension of their own infrastructure. This includes the custom components developed and the interfaces used. Finally, section V concludes the paper and discusses future work.

II. RELATED WORKS

We evaluate related works for data analytics platforms in three categories. We discuss reference architectures using either (i) high; or (ii) low abstraction levels. Next, we discuss (iii) implementations and relate them to reference architectures. We note here, that we relate the presented related works concepts to our solution in section III.

High abstraction level architectures typically contain approaches and system design principles, but lack concrete implementation references. Contrary, low-level architectures focus on the implementation details, typically using platform (PAAS) and software (SAAS) based services of a particular cloud provider, and/or rely on open-source services.

An example of a high-level reference architecture is the Lambda Architecture [12]. It introduced two parallel pipelines on the same data sources. One pipeline for streaming ("speed layer") and one for batch processing ("batch layer"). An extra layer ("serving layer") stores the output and responds to ad-hoc queries. The batch layer utilizes a distributed processing system to process all available data. As processing takes time, the batch layer can push fewer updates. Additionally, to fix errors the system needs to recompute the complete dataset. The speed layer processes the most recent data and provides a (near) real-time view of the data. In its original form, it did not aim for completeness as it tried to fill the gap caused by the batch layer's lag. However, real-time data is a requirement for today's data platforms. Finally, the serving layer stores output from the batch and speed layers. Additionally, it also responds to ad-hoc queries.

The Kappa Architecture [13] aims to be a simplification and evolution of the Lambda Architecture by removing the batch processing component. In the Kappa Architecture the canonical data store is an append-only immutable log. It is used instead of a relational database management system (RDBMS) or noSQL store like Apache Cassandra [14]. From this log, data is streamed through the processing pipeline and fed into supplementary stores for serving. All data is moved through the stream processing system quickly eliminating the need for the batch layer.

The Open Group Architecture Framework (TOGAF) [4] is a widely used framework for general enterprise architecture. It includes an approach for designing, planning, delivering, and governing an enterprise information technology architecture. It distinguishes two building block types: Architecture Building Blocks (ABBs) and Solution Building Blocks (SBBs). ABBs are higher level architecture models. They encapsulate a set of technology, application, business and data requirements and steer the development of SBBs. Additionally, they incorporate the relationship and interoperability with other building blocks, and should be reusable. SBBs on the other hand, are lower level components and are directly guided by ABBs.

As discussed in section I, public cloud providers publish low-level reference architectures for different use cases to help commercialize their services. One such provider based architecture is the Azure IoT Reference Architecture [15]. It defines three data paths for data analytics. The hot path represents near real-time insights using stream analytics; the warm path uses fine-grain methods, but incurs a higher latency; finally the cold path represents batch processing performed at greater intervals. Microsoft also provides more generic data processing architectures, for example in [16] Apache Kafka [17] is used as a streaming source to ingest data. In these architectures as a best practice for ingestion, data is always fed into a streaming service. For example, in the discussed Azure IoT Reference Architecture, data ingested into Azure EventHubs, which is the Azure platform equivalent of Apache Kafka.

FIWARE [18] provides reference architectures, consisting of several building block like components that facilitate architecture building. FIWARE is a framework of open-source components for data management, for developing interoperable smart solutions. As part of the framework FIWARE provides different components such as the "Orion context broker" for data management, IoT agents for connecting smart devices, components for storing historical data such as "Comet", and also components for data analytics among many others. Additionally to a standards-based platform, FIWARE provides a complete ecosystem as well. It was originally driven by the European Union to develop a platform for the Future Internet.

Isah and Zulkernine [19] present a fault-tolerant and scalable data management framework for many feeds of structured and unstructured data. The presented framework consists of reusable building blocks. Additionally, the authors demonstrate the framework using a real-world data streaming case study. They rely heavily on Apache NiFi [20] to provide a dataflow that integrates Apache Kafka [17] and HDFS [21]. Their architecture is stream processing only, and is an implementation of the Kappa architecture, although not referenced by the authors.

Airbnb built the Minerva [22]–[24] metric platform to standardize data analytics at scale within the company. Minerva is an internal plaform at Airbnb, and is used across the company as the single source of truth. More than 12000 metrics, 4000 dimensions are in Minerva. More than 200 data sources originating from different teams like core product, payment and different organisations like product management, finance and engineering are integrated. Minerva is built using open-source products: Apache Airflow [25] for workflow orchestration, Apache Spark [26] as computation engine, and Apache Druid [27] and Presto [28] for data consumption. The platform provides analytics, reporting and experimentation capabilities for Airbnb. Data is defined at a single place. Anyone can look up definitions without confusion. Minerva is focused on metrics and dimensions as opposed to tables and columns. When a metric is defined its authors are required to provide self-defining metadata. At heart of the Minerva's configuration system are event and dimension sources. These correspond to fact and dimension tables in a star schema [29] design. Users can select the metric they would like to explore and then are transferred to an internal component called Metric Explorer. This allows basic investigation of the data. For deeper insights Apache Superset [30] is available within the platform to explore the data. Additionally, Minerva provides a metric-serving API layer between upstream data models and downstream consumer applications.

III. DATA ANALYTICS PLATFORM

Our goal was to build a reusable, open-source, vendor agnostic and end-to-end platform for IoT data analytics using industry best practices. We designed the platform to primarily support the ingestion of high-frequency time series data. All ingested data is stored in raw format. This allows to execute new and improved versions of our data processing tasks on the complete dataset. The platform provides data exploratory tools and also capabilities to move developed data processing services into production. Data access is provided through standardized interfaces such as Amazon S3, Apache Kafka and ODBC for data warehouse access.

The platform is container based. We use Docker for convenience purposes when a full virtual machine (VM) would be dedicated to a service component. In this case a single container is running on a dedicated VM managed via Terraform [31] and Ansible [32]. We use Kubernetes in all other cases. However, we allow exceptions for custom components of use cases (see figure 1) that may require e.g., Windows based VMs.

The platform is built using orchestration tools and utilizing reusable building blocks. For the deployment of the platform we are relying on Terraform and Ansible descriptors. Terraform is responsible for the cloud communication part. It manages and maintains all the cloud objects like security groups, firewall rules, volumes and virtual machines. Furthermore, it can prepare the Ansible host configuration in the correct form as Terraform knows all the necessary endpoints and properties. We use Ansible to manage the basic configurations. It implements two different roles. First, it distributes and manages access credentials on every host in the platform. Second, it manages monitoring components on each host.

The platform consist of two main parts. The first part contains the custom components for the different supported use cases, while the second part hosts the core components. figure 1 depicts the architecture of the current version of the platform. Custom components of each use case have access to a dedicated set of resources assigned to that particular use case. These components include for example VPN endpoints connecting smart vehicles via cellular network; or Robot Operating System (ROS) [33] based components for cloud-based processing. The second part of the platform contains the core components of the platform. There a single shared instance of the core components serves all use cases.

The platform is stream processing based for delivering ingested data to the end user. It is similar to the presented Kappa Architecture (see section II). However, we retain the capability of executing batch jobs to support existing workloads. Our platform implements the hot-warm-cold path for data presented in the Azure IoT Reference Architecture (see section II). The hot path provides a low-latency pipeline relying mainly on MQTT and Apache Kafka for ingestion; with Logstash, Elasticsearch and Kibana (the "Elastic stack" [34]) for the real-time analytics stage. The warm path is used for more complex processing and delivery to the data warehouse for further analytics. The warm path also relies on Apache Kafka, however, supports non-structured data uploads using an Amazon S3 compatible interface as well. Apache NiFi [19], [20] and ksqlDB [35] is used for data transformation and enrichment, while TimescaleDB [36] acts as the data warehouse. We use dbt [37] in the data warehouse for batch workload on a as-needed basis, thus, implementing the cold path.

A typical analytics platform should consists of three main stages at least [38]. These include (a) ingestion,

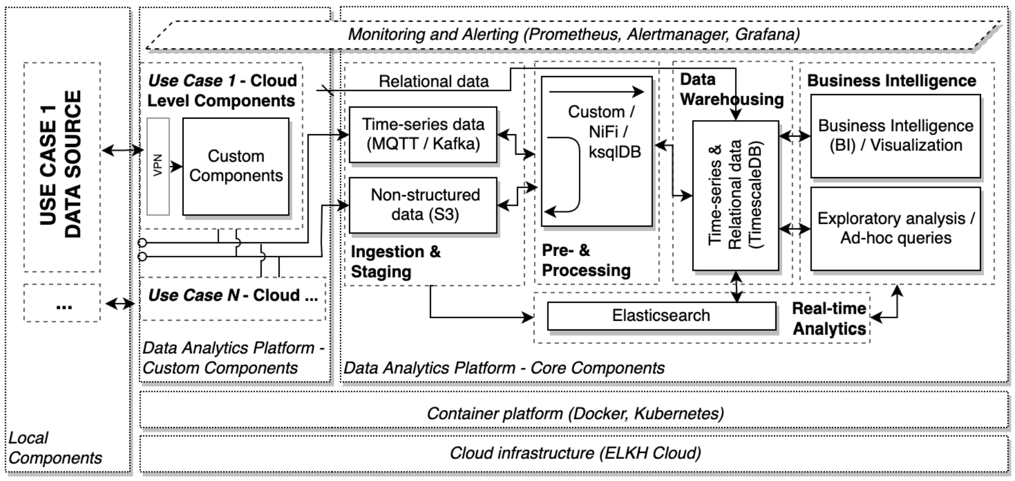

Monitoring and Alerting (Prometheus, Alertmanager, Grafana)
USE CASE 1 DATA SOURCE
...
Use Case 1 - Cloud Level Components
VPN
Custom Components
Use Case N - Cloud ...
Data Analytics Platform - Custom Components
Relational data
Time-series data (MQTT / Kafka)
Non-structured data (S3)
Ingestion & Staging
Custom / NiFi / ksqlDB
Pre- & Processing
Data Warehousing
Time-series & Relational data (TimescaleDB)
Business Intelligence
Business Intelligence (BI) / Visualization
Exploratory analysis / Ad-hoc queries
Elasticsearch
Real-time Analytics
Data Analytics Platform - Core Components
Container platform (Docker, Kubernetes)
Cloud infrastructure (ELKH Cloud)
Local Components

FIGURE 1. Logical architecture of the data analytics platform running on ELKH Cloud. Arrows represent the data flow between core and custom components, including the different stages of the platform.

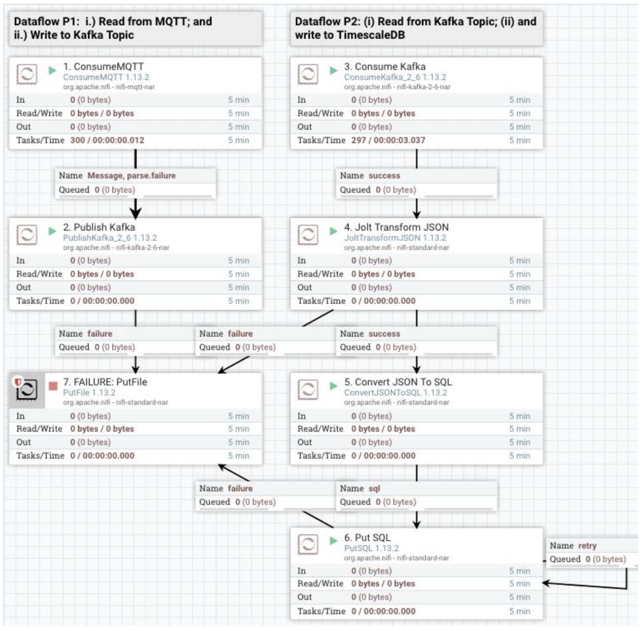
(b) processing, and (c) storage. Contrary, the core part of our platform consists of five stages. These stages are (i) ingestion and staging; (ii) (pre-)processing; (iii) data warehousing; (iv) real-time analytics; and (v) business intelligence (including visualisation). See figure 1 for more details. Our platform implements a hot, warm and cold path (see Azure IoT Reference Architecture in section II and in [15]). The warm path utilizes stages $i \rightarrow ii \rightarrow iii \rightarrow v$, while the hot path is implemented by stages $i \rightarrow iv \rightarrow v$; see more details in figure 1. The cold path involves the same stages as the warm path ($i \rightarrow ii \rightarrow iii \rightarrow v$), and executes batch workloads using data in the data warehouse. Additionally, we provide a dedicated business intelligence stage (v) that is responsible both for visualization and data insights. Next, we are going to discuss the stages in detail.

The ingestion and staging stage is responsible for accepting and storing incoming data. The platform supports both time-series and blob (unstructured) data. Currently, MQTT and native Kafka for streaming data; Amazon S3 for non-structured or blob data; and Open Database Connectivity (ODBC) for relational data are supported. In our experience MQTT is common ground for IoT devices and similarly, S3 is for blob data. For MQTT there is no restriction on the format and structuring of data, JSON payload and/ or Sparkplug-B [39] structuring is preferred. We currently use Eclipse Mosquitto [40] as MQTT broker for ingesting data. The throughput of Mosquitto is limited by single-thread CPU performance [41]. Currently this does not cause a bottleneck for the use cases of the platform. However, we are investigating alternative, distributed solutions such as EMQX [42]. Additionally, the first stage of the platform acts as a staging area for all incoming data. Data here is kept in its raw source format, thus this area is effectively the data lake of the platform.

The processing stage is responsible for enriching, transforming and transferring data to persistent storage and real-time analytics. For this we rely mainly on a set of services. First, we use Apache NiFi for orchestrating the dataflow between the building blocks of the architecture. Additionally, ksqlDB and custom Kafka micro-services are responsible for additional data transformation and enrichment within Apache Kafka.

Apache NiFi is responsible for ingesting data from the MQTT broker acting as the MQTT interface of the platform. NiFi is also a Kafka producer and pushes the ingested data to the desired Kafka topic. Additionally, it is a Kafka consumer, it reads data from the topic, performs simple conversions (e.g., flattening JSON structures and type conversions) and pushes the results to the data warehouse, TimescaleDB [36] in our case. An example data-flow is shown in figure 2. Apache NiFi relies on the "Zero-Leader Clustering paradigm", this means that each node in the cluster has the same data flow and performs the same tasks on the data. The cluster automatically distributes the data between the nodes. In case of running multiple dataflows (e.g., different use cases) on a single cluster, all nodes instantiate all components, meaning the CPU and memory footprint is increased on all nodes. Thus, all nodes in the cluster must be scaled up vertically, however, this is not always feasible. The best-practice is to separate each dataflow to its dedicated cluster [43]. Based on a sizing guide by Cloudera [44] the throughput of NiFi scales nearly linearly when introducing additional nodes. However, the guide only reports results up to 9 nodes. We are currently utilizing up to 3 nodes in our cluster, and this cluster is shared between all use cases. We are currently not impacted by the vertical scaling limit, but are in the process of moving each use case (dataflow) to its dedicated NiFi cluster.

Additionally, NiFi is used both for data enrichment and pre-processing purposes. Enrichment is used during transfer from the MQTT broker into the streaming layer (Apache Kafka), and from Apache Kafka to the Data Warehouse (TimescaleDB). When moving data into Kafka we always



```
Dataflow P1: i.) Read from MQTT; and
ii.) Write to Kafka Topic

1. ConsumeMQTT
ConsumeMQTT 1.13.2
org.apache.nifi - nifi-mqtt-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  300 / 00:00:00.012       5 min

Name  Message, parse.failure
Queued  0 (0 bytes)

2. Publish Kafka
PublishKafka_2_6 1.13.2
org.apache.nifi - nifi-kafka-2-6-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min

Name  failure
Queued  0 (0 bytes)

Name  failure
Queued  0 (0 bytes)

7. FAILURE: PutFile
PutFile 1.13.2
org.apache.nifi - nifi-standard-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min

Name  failure
Queued  0 (0 bytes)

Dataflow P2: (i) Read from Kafka Topic; (ii) and
write to TimescaleDB

3. Consume Kafka
ConsumeKafka_2_6 1.13.2
org.apache.nifi - nifi-kafka-2-6-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  297 / 00:00:03.037       5 min

Name  success
Queued  0 (0 bytes)

4. Jolt Transform JSON
JoltTransformJSON 1.13.2
org.apache.nifi - nifi-standard-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min

Name  success
Queued  0 (0 bytes)

5. Convert JSON To SQL
ConvertJSONToSQL 1.13.2
org.apache.nifi - nifi-standard-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min

Name  sql
Queued  0 (0 bytes)

6. Put SQL
PutSQL 1.13.2
org.apache.nifi - nifi-standard-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min

Name  retry
Queued  0 (0 bytes)
```

FIGURE 2. **The NiFi dataflow used in the collaborative robotic assembly use case (see section III-A for more details) for transformation and streaming into the data warehouse.**

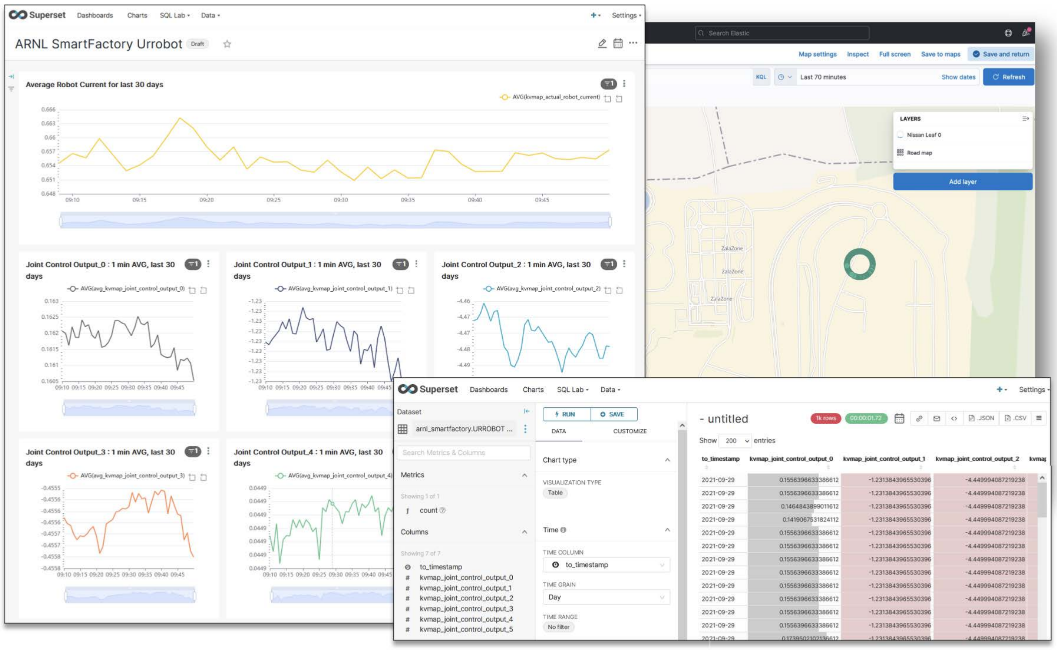
augment it with metadata from the source system. In case of MQTT this means the broker and topic name, Quality of Service (QoS) level, and whether the message is retained and/ or a duplicate. Next, when moving between from Apache Kafka, metadata from Kafka is added such as the topic name, partition and offset numbers, and the Kafka ingestion timestamp. These enrichment cascade into the final destination and enable data lineage, meaning understanding and recording the flow of data. Additionally, it enables to replay and re-process the data using the ingested raw version. Pre-processing with Apache NiFi involves light transformations. With our use cases (see sections III-A and III-B) we ingest data in JSON format and use the JOLT transformer of NiFi to e.g., convert between data types, flatten nested structures, and include additional metadata.

ksqlDB allows working with Kafka based streams using an SQL-like language with the scalability of Kafka Streams. In turn Kafka Streams is a client-side Java library that abstracts the low-level Kafka APIs (Producer and Consumer). It utilizes a streaming approach: it ingests from sources, performs transformations on the ingested data, and finally forwards data to sinks. Additionally, it allows scaling by automatically handling the assignment and balancing of the workload based on the number of partitions and application instances running (via Kafka consumer groups). However, as it is a library it does not contain a resource manager or scheduler, as the application developer should bring its own. This means that Kafka Streams applications are suited for containerization, especially deploying on Kubernetes. We perform additional stream-based transformations using ksqlDB running on a Kubernetes cluster within the platform, and support Kafka Streams-based custom micro-services as well.

The Data Warehouse stage is responsible for storing and serving data. We rely on TimescaleDB [36] for storing time-series data. It is an extension on top of PostgreSQL, which is in turn used for storing relational data and data warehousing functionality overall. This allows storing both relational and time-series data in a single data store. The data warehouse is based on a relational database (PostgreSQL - TimescaleDB), thus, incoming data must be transformed into this schema. We rely on the ConvertJSONtoSQL processor of NiFi for JSON type messages, see "5. Convert JSON To SQL" in figure 2 for more details. We would like to note here, this is pure transformation as enrichment is done in a previous step using the JoltTransformationJSON processor, see "4. Jolt Transform JSON" in figure 2 for more details. The resulting data forms the base core dataset for each use case. Using this dataset we perform further near real time processing in TimescaleDB, e.g., aggregate using different time periods; and also run offline batch workloads as needed.

The Business Intelligence (BI) stage is responsible for providing dashboard and visualization capabilities. Currently, Apache Superset is extensively used on top of the data warehouse, with Grafana as second choice, and Kibana in the hot path, see figure 3 for more details. Additionally, both ad-hoc data queries and self-service data exploration are supported either via Superset or via Jupyter notebooks. In general, any visualization or BI tool (such as Microsoft Power BI) can be connected that supports the ODBC interface. We choose Grafana as it can be extended easily with custom charts, and allows speeding up visualization by switching to different aggregations when using PostgreSQL (TimescaleDB) [45] as shown in figure 6 for the Collaborative Robotic Assembly (CRA) use case (see section III-A for more details about the use case). Kibana has extensive charting capabilities, however, as we rely on the Basic license of Elasticsearch, we do not have access to all features such as visualization layers for creating tracks from geospatial data. Additionally, our finding is that the OpenStreetMap based maps used by Kibana are less accurate than the commercial offerings such as Google Maps. This can be seen in the top right image of Figure 3 where circle is drawn using GPS vehicle data in what is seemingly an empty area. However, in reality there is a roundabout there with connected roads.

For data lineage and life-cycle management we rely on metadata and a data catalogue. We allow to differentiate experiments within each use case by assigning an unique identifier to the incoming data. Additionally, data at each stage is enriched with additional metadata such as the time of ingestion at the stage and various stage specifics (e.g., partition and offset in Kafka, topic in MQTT). We use a data catalogue to hold additional metadata that provides further description for the experiment for later reference. Currently, raw data is kept in the staging environment indefinitely by default, and removed on a per experiment or use case basis. In the data warehouse we use the


Superset
Dashboards
Charts
SQL Lab
Data
Settings
ARNL SmartFactory Urrobot
Draft
Average Robot Current for last 30 days
AVG(kvmap_actual_robot_current)
Joint Control Output_0 : 1 min AVG, last 30 days
Joint Control Output_1 : 1 min AVG, last 30 days
Joint Control Output_2 : 1 min AVG, last 30 days
Joint Control Output_3 : 1 min AVG, last 30 days
Joint Control Output_4 : 1 min AVG, last 30 days
Search Elastic
Map settings
Inspect
Full screen
Save to maps
Save and return
KQL
Last 70 minutes
Show dates
Refresh
LAYERS
Nissan Leaf 0
Road map
Add layer
Dataset
arnl_smartfactory.URROBOT
RUN
SAVE
DATA
CUSTOMIZE
Search Metrics & Columns
Metrics
count
Columns
to_timestamp
kvmap_joint_control_output_0
kvmap_joint_control_output_1
kvmap_joint_control_output_2
kvmap_joint_control_output_3
kvmap_joint_control_output_4
kvmap_joint_control_output_5
Chart type
VISUALIZATION TYPE
Table
Time
TIME COLUMN
to_timestamp
TIME GRAIN
Day
TIME RANGE
No filter
- untitled
1k rows
00:00:01.72
.JSON
.CSV
Show 200 entries

to_timestamp	kvmap_joint_control_output_0	kvmap_joint_control_output_1	kvmap_joint_control_output_2
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1464843899011612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1419067531824112	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1556396633386612	-1.2313843965530396	-4.449994087219238
2021-09-29	0.1739502102136612	-1.2313843965530396	-4.449994087219238

FIGURE 3. **Data exploration and visualization using Apache Superset for the collaborative robotic assembly use case (left and bottom); and exploring GPS vehicle data using Kibana (Elasticsearch) for another use case (top-right).**

aggregation feature of TimescaleDB to down-sample the data as allowed by the experiment and use case. Higher-frequency data can be removed later as it can be reproduced when needed from the raw data stored in the data lake (ingestion stage). The removal depends on the use case and experiment.

The platform uses Prometheus [46] for monitoring its services and infrastructure. Prometheus relies on so-called ''exporters'' for monitoring components and services. These exporters are created for a specific purpose, such as monitoring the resource usage of containers [47], hosts [48], or services such as Java-based applications [49] among others. Prometheus provides a rule based engine for defining alerting rules. These alerts, when triggered, are forwarded to the Alertmanager component of Prometheus. Alertmanager is able to de-duplicate and group together alerts, and then use different media (e.g., email, slack or webhooks) for alert notifications. Finally, we use Grafana for charting the different monitored metrics.

The platform is deployed on the federated, community cloud of the Eotvos Lorand Research Network, called ELKH Cloud. The goal of ELKH Cloud is supporting Hungarian scientists with elastic, virtualized computing resources. This science cloud provides a combined 5904 vCPUs, with 28TB RAM, 1248TB HDD and 338TB SSD storage. It also offers a total of 68 GPUs with 2400GB RAM, and 584TFlops double precision, 1174TFlops single precision and 13768TFlops FP16 Tensor performance. The presented platform is deployed in the site of the Institute of Computer Science and Control (abbreviated as SZTAKI) of the federated cloud. This site contains (among others) 10 compute nodes, each with 2x Intel Xeon Gold 6230R 26-Core CPUs and 768GB of RAM. Additionally, there are 6 HDD and 6 SSD storage nodes, each with 192TB and 92TB of raw storage, respectively. The cloud is based on OpenStack Wallaby, and the storage backend is built using Ceph Pacific. All virtual machines have 10Gbps networking by default, with optional configuration settings for 30-35Gbps. The ELKH Cloud is connected to the Hungarian academic backbone with 100Gbs network.

We currently allocate 106 vCPU cores, 264 GB of RAM and 4 TB of SSD storage for the platform. The benchmark platform which is described in section IV is deployed in this allocated resource pool as well. The first version of the platform is deployed on the cloud of the Hungarian Academy of Sciences, called MTA Cloud. This cloud is the predecessor of the ELKH Cloud with the same purpose but with less resources. This initial version of the platform is using 13 virtual machines with 48 vCPUS, 96GB of RAM and 480GB HDD storage.

A. USE CASE: COLLABORATIVE ROBOTIC ASSEMBLY

Collaborative Robotic Assembly (CRA) refers to the joint assembly process of workpieces performed by human operators and collaborative robotic systems, the core elements of a common Cyber-physical System (CPS). The advantage of the application of operators with high situation recognition capability and flexibility supported by the accuracy and speed of the robot arms can be enormous, if the safety is guaranteed. This synergistic relationship can only be achieved if such system can provide the necessary amount of information both in real time for monitoring, and in the long run to improve efficiency by analysis and evaluation.

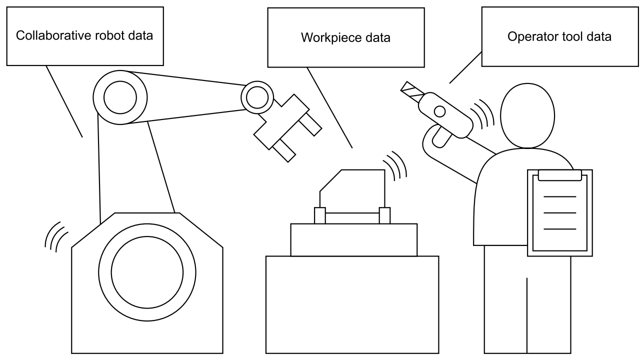


FIGURE 4. **Schematic representation of typical collaborative robotic assembly scenario.**

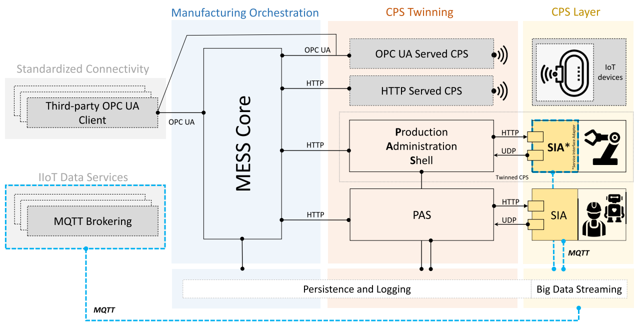


FIGURE 5. **MESS overall architecture with highlights on use-case involved components and communication (blue bold dashed lines).**

Figure 4 schematically enumerates the actors and possible data sources in a CRA scenario: i.) the human operators and their tools with network connection; ii.) the collaborative robotic arm; and iii.) the smart workpiece and/or fixture. For this specific use-case, the focus is on the robotic arm, because the sheer amount of data it can provide is sufficient for testing the platform. The robotic arm is one of the Universal Robots' UR-series, which can provide 84 different parameters (from which eighteen are six-element float arrays) at 125Hz via the Real-Time Data Exchange (RTDE) interface.

These parameters include—without claiming to be exhaustive—mechanical properties (e.g. position, speed and acceleration), thermal metrics, energy consumption indicators (e.g. currant and voltage), status information and I/O states. We can even enrich the data with difference formation of targeted and realized values to gain the scale of errors. This is a sufficient amount of data to make meaningful analysis from which relevant information can be indirectly obtained about the performance of the operators, as well.

The platform's visualisation features of the CRA presents the time distribution of data proportional to energy consumption of the robotic arms and so to a much wider concept of sustainability for the assembly phases considered (visible in figure 6). This is one of many physical aspects of a CRA that can be represented and analyzed on the basis of the provided data and the usage of the platform.

The CRA is connected to the data analytics platform by means of an MQTT-capable Service Interface Adapter (SIA) developed on top of the CRA CPS controller. It can be referred to as "dew" (as defined in research work from [50]): this is the very first entry-point in "traditional" manufacturing systems where legacy devices are converted into network-able, embedded components for the distributed production scenarios of Industry 4.0; the lowest-level cloud deployable component of a much wider manufacturing execution system architecture, as briefly explained in the following subsection.

1) MANUFACTURING EXECUTION SYSTEM AS A SERVICE

The Manufacturing Execution System as a Service (MESS in the following) is a new attempt to model, integrate, and orchestrate essential CPS services which are at the basis of modern, digitized manufacturing facility. The architecture of the MESS facilitates a novel, generic, and simple way for the collaboration of distributed, autonomous manufacturing entities. As suggested by the acronym, the MESS has been designed as a set of cloud-deployable services where the specific layer's requirements permit it, taking into account aspects such as service availability, latency time for communication and data exchange rates, primarily.

Main components of the MESS are depicted on figure 5 whereas their major characteristics are briefly introduced hereafter (for a comprehensive presentation of the MESS please refer to [51]).

The MESS enables service-oriented orchestration and production improvement by supporting and improving the communication among the CPSs inside the facility, as well as between the production and the other activities in the enterprise (such as process planning, process simulation, resource planning and sharing, etc.). This is fundamental in order to provide updated communication and information analysis to the management, offer a clear and simple interface to the end-users, and to monitor and operate the system capability. Briefly, the MESS is expected to provide an interoperable, reconfigurable and reliable information system to the overall production and resource sharing activity.

The MESS Core is essentially a time-invariant, event-based sequence control of processes, based on the monitored statuses of resources (e.g., management of production processes). It controls the synchronization with the CPSs and acknowledges the other system components about the occurring events.

Fundamental element of the MESS architecture is the Cyber-physical System (CPS) which is in charge of performing the actual production activities. A CPS can be abstracted to almost anything: a robot, a human worker with network connected device, a camera, a PLC controlled manufacturing unit, a pool of tools, etc. To the MESS, a CPS is like a black box that makes the physical layer seamless, providing a set of production related capabilities and physical dimension values. The production capabilities of the CPS can be reached by the MESS Core directly or through a digital counterpart

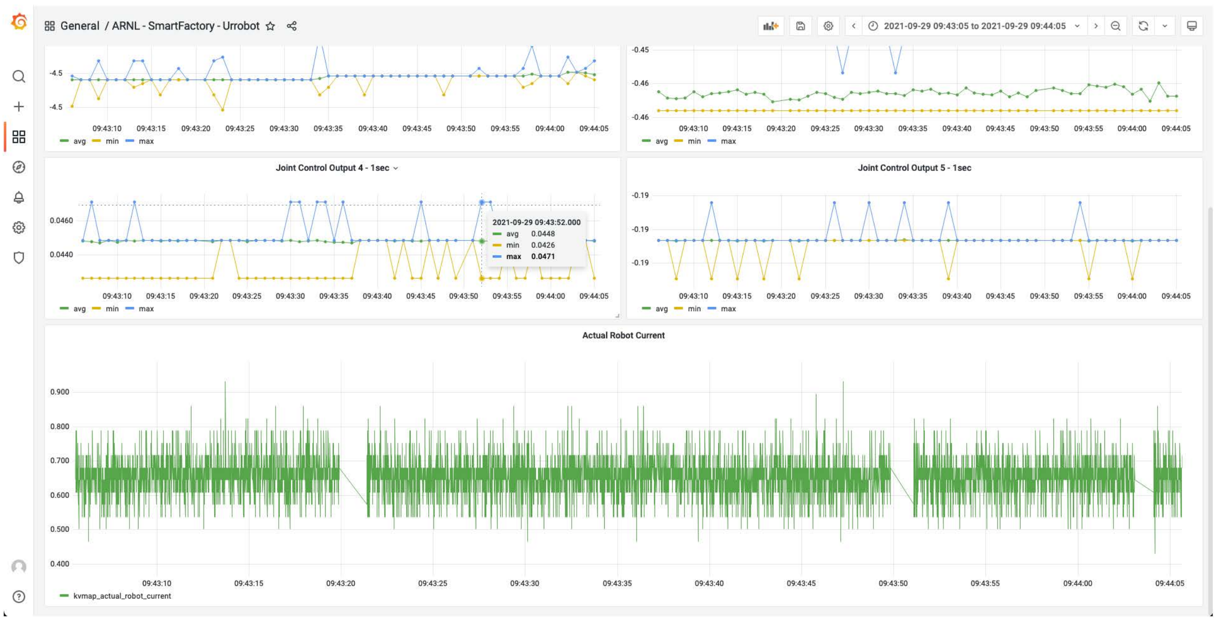

General / ARNL - SmartFactory - Urrobot
2021-09-29 09:43:05 to 2021-09-29 09:44:05
Joint Control Output 4 - 1sec
Joint Control Output 5 - 1sec
Actual Robot Current
avg
min
max
kvmap_actual_robot_current

FIGURE 6. Custom Grafana-based dashboard for visualizing CRA use case data at different resolutions.

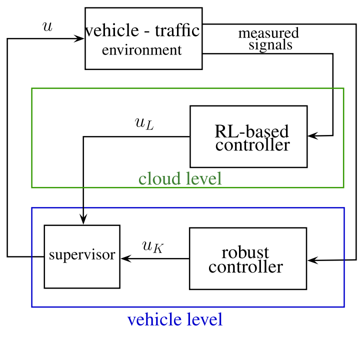
(Digital Twin) with standardized interface. From the point of view of an external component (i.e., user interface, scheduler, and so forth – out of scope in this paper), only these twinned CPSs are discoverable and actionable by the MESS, while physical devices are kept hidden for security and competency reasons.

Devices represent the endpoints of the MESS and are generally controlled by their own specific software, with its own information and service model. In order to connect to the MESS, on the other hand, a CPS needs to adapt its arbitrary service model to the one prescribed by the MESS, which simplifies and standardizes this mechanism. This crucial aspect of a device integration at the lowest level (entry-point) of the architecture is guaranteed by the Service Integration Adapter (SIA). SIA is (also) the point where CPS data can be captured and provided at a rate closest to the physical capability of a device (in our use-case the UR robots can provide data of their physical operations at 125Hz) and so generate big-data streaming for successive analytical IIoT data services (figure 5). The goal of the SIA is basically to enable the device controllers to communicate with the MESS framework through the MESS Production Administration Shell (PAS), which is in charge of the CPS digital twinning with the MESS Core. The SIA is a. Net library written in C# language. Besides HTTP and UDP connection capabilities, it is also equipped with MQTT technology, a standardized back-bone for IIoT data provision for additional integration and analytics services (as part of this use-case). The great advantage of this CPS twinning solution is that, regardless of the specific method chosen by the designer for the CPS integration, once the SIA is connected to the PAS (and to the MESS Core), the overall system will seamlessly provide a standardized OPC UA [52] equivalent transformation of the CPS service and information model to the external world. MESS Core (cloud), PAS (edge/cloud) and SIA (dew/edge) are all components of a ''fluid'' manufacturing architecture [50].

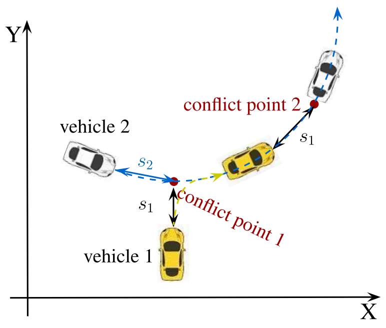
B. USE CASE: HARDWARE-IN-LOOP SMART VEHICLE CONTROL

Cloud computing for automated vehicles has high relevance in control applications, which require high computational effort, e.g., learning and coordinated control features. Typical example on Vehicle-to-Cloud connection is route planning, where comfort-based [53] and safety-based [54] route planning approaches on the cloud with access to real-time information on the environment are implemented. Another example is real-time computation of a stochastic model predictive control intervention for a suspension system [55]. In case of high number of vehicles connected to the cloud, the problem of best quality service against stochastic communication delays and task deadline in automotive context is a hot topic, see e.g., [56]. Another important application of Vehicle-to-Cloud connection is the coordination of automated vehicles in non-signalized intersections, one- and double lane roundabouts [57]. Due to the requirements of real-time information on the traffic scenario and of enhanced performance level on vehicle dynamic requires the application of learning-based methods for the motion of the automated vehicle, see e.g., [58]. Nevertheless, it is necessary to develop control structures, which can provide guarantees on safety performances, e.g., collision-free motion in roundabouts.

This demonstration provides a hierarchical control for automated vehicles, with which their safe and efficient motion in roundabouts can be guaranteed. The control hierarchy contains vehicle level and cloud level. The aim of the cloud-level is to achieve enhanced control performances

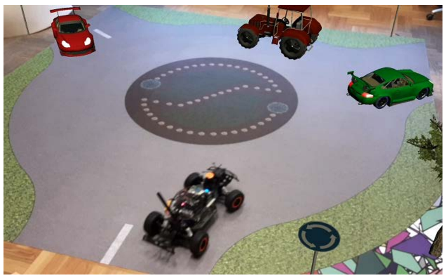


(a) Illustration of the control architecture

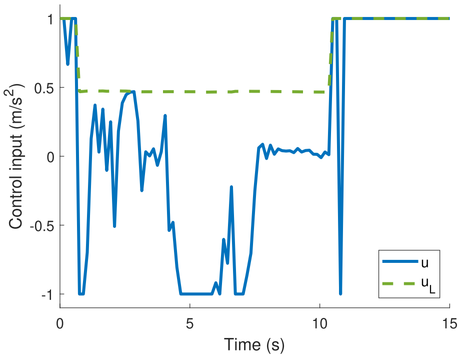


(b) Vehicle interactions in roundabout

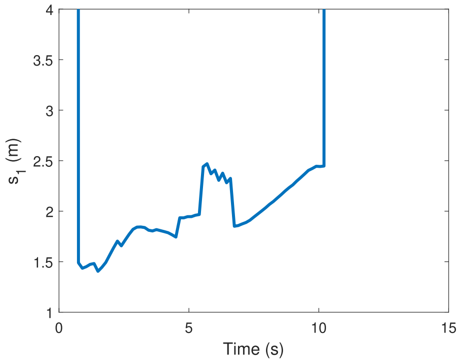
FIGURE 7. Illustrations on the methodology of the vehicle control.



(a) Illustration of the vehicle motion



(b) Control input



(c) Distance from the conflict point

FIGURE 8. Simulation example.

using the high computation capacity of the cloud. Thus, reinforcement learning on the cloud level for achieving maximum speed of the vehicles is implemented. Moreover, on the vehicle level the safety requirement, i.e., collision avoidance, is guaranteed. The advantage of the solution is that safe performance specifications even at the degradation of the communication in the network can be guaranteed. Significant novel content of this work is the implementation of the method using indoor test vehicle environment with cloud connection.

The architecture of the hierarchical control with each levels is illustrated in figure 7(a). The goal of the control is to provide single motion input $u(k)$ for a given individual vehicle, i.e., longitudinal acceleration command $a_1(k)$, with which the vehicle moves along its route. $u(k)$ is computed by the supervisor, such as $u(k) = u_K(k) + \Delta(k)$, where $u_K(k)$ is the output of the robust controller on the vehicle level. $\Delta(k) \in \hat{\Delta}$ is an additional term of the control input and $\hat{\Delta}$ is the finite domain of $\Delta(k)$. In the control architecture, $u_L(k)$ is a candidate control input, which is suggested by the reinforcement (RL)-learning-based controller. The value of $\Delta(k)$ is a result of an optimization process in the supervisor, which minimizes the difference between $u(k)$ and $u_L(k)$ and guarantees collision avoidance between the automated and the other vehicles [59].

The constraint between the vehicles through a method of conflict points is formulated. Roundabout can be handled as a complex scenario with intersection and vehicle following tasks, i.e., safe motion requires the modification of the conflict point during the motion of the vehicle, see figure 7(b). In case of entering into the roundabout the conflict point can be defined as the crossing of the vehicle routes. After entering into roundabout, automated vehicle must follow preceding vehicle. In this case, the actual position of the preceding vehicle is the continuously varying conflict point. The aim of the constraint is to keep safety distance s_{safe} between automated and other vehicles, e.g., further automated vehicles or human-driven vehicles: $s_1^2(k+1)+s_2^2(k+1) \leq s_{safe}^2$, where $k+1$ represents the next time step.

The effectiveness of the proposed method through a Hardware-in-the-Loop (HiL) simulation with three 1/10 sized wheeled RC vehicles is demonstrated, see figure 8. The candidate control input $u_L(k)$ from ELKH Cloud and the position of the automated vehicle through Wi-Fi on the Robot Operating System (ROS) network are transmitted.

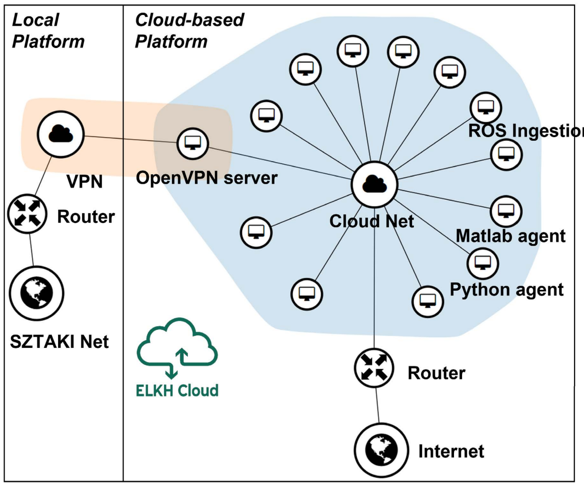


FIGURE 9. High-level architecture presenting the local to cloud connection and components.

In the presented example the vehicle enters into roundabout, takes a round and exits at the same direction. figure 8(a) shows the moment when the vehicle decides to enter into the roundabout. During its route three simulated virtual vehicles also move in the roundabout. Thus, the automated vehicle has interactions with all vehicles. figure 8(b) presents the recommended RL-based control input u_L, which is transmitted by the cloud. It shows that u_L facilitates the entrance of the vehicle into the roundabout, i.e., traveling time is reduced, which is the goal of the RL-based agent. figure 8(c) illustrated the distance of the vehicle from its actual conflict point, i.e., from the surrounding vehicles. It can be seen that $s_{safe} = 1m$ during the entire simulation can be guaranteed.

ELKH Cloud hosts the cloud level control of the HiL environment as presented in figure 9. The connection between the local environment and the cloud-based one is provided via a secure OpenVPN site-to-site VPN connection (OpenVPN server). The Reinforcement Learning Toolbox of Matlab is running on a dedicated Windows-based virtual machine (Matlab agent) due to performance considerations. The Python-based RL agent is being evaluated as an alternative to Matlab and deployed on a separated Ubuntu-based virtual machine (Python agent). These machines can directly communicate with the Hil environment through the VPN connection with ROS protocol. Lastly, the ingestion component (ROS ingestion) is running on another Linux based virtual machine. This component acts as a ROS-MQTT bridge for ingesting data from the HiL environment with ROS protocol and forwarding it to the Kafka streaming component via MQTT protocol for further analysis.

IV. EVALUATION OF THE PLATFORM

We wanted to evaluate the performance of the platform for additional use cases. During the next period of the Autonomous National Lab project additional use cases will be introduced. Data will be ingested from outdoor sources that include data originating from multiple autonomous cars.

We wanted to evaluate the platform as a whole, and also the ingestion stage separately. The evaluation of the platform as a whole provides a clear view of the limitations and bottlenecks, if any. Additionally, the most critical part of the platform is the ingestion & staging part. Any bottleneck here will result in a data loss, while the later stages typically only introduce latency for the system as a whole.

A. METHODOLOGY

For the ingestion stage, we evaluated the Kafka ingestion endpoint. We measured different number of data sources using different rate of transmission and message sizes. This includes the number of concurrent data sources, the sustainable message transfer frequency and message size for these. We also evaluated the scalability of the endpoint in terms of expected performance uplift by increasing the cluster size. We did not increase the hardware resources allocated to the cluster nodes as there is a limitation stemming from the cloud infrastructure, and we rather go wide than tall.

We decided to use a loader cluster to generate load for the Kafka cluster in a controlled way. The whole ingestion part, including the Kafka cluster and the Kafka load generator cluster, is monitored by Prometheus and its Node exporter. The Node exporter of Prometheus collects the most relevant metrics data from the virtual machines and makes it available at an HTTP endpoint. Prometheus scrapes the metrics from the HTTP endpoint and stores it for visualisation and later use. The Kafka cluster uses the m2.xlarge VM flavor, which contains 8 vCPUs and 16 GB of RAM. The loader cluster utilises the m2.large VM flavor, which includes 4 vCPUs and 8 GB of RAM. Both clusters are based on Ubuntu 20.04 operating system and uses Kafka version 3. The deployment of the temporary load cluster was done via Terraform. Terraform makes the benchmark process more straightforward, easy to replicate and customizable as needed. The benchmark started with a Python script that uses Ansible as a core component to distribute the benchmark configuration and start the benchmark process on the load cluster.

We created two different scenarios covering the two main aspects, the throughput and latency of data ingestion systems during the benchmark measurement. In addition, we investigated and measured the ingestion scalability with cluster sizes of three and five nodes. These measurements help to analyze the system's scalability and identify bottlenecks in the system. Our preliminary tests show that one thread cannot use all of the available CPU power on the loader hosts. Therefore, we created two active benchmark processes on the loader cluster during the throughput optimized scenario. We created the configuration based on the vendor guideline [60] to create those scenarios and fine-tune the Kafka cluster to achieve better results. The benchmark aimed to measure the system throughput and latency. Therefore the throughput is not limited on the Kafka loader components.

TABLE 1. The different configuration parameters during the two scenarios.

	Latency optimized	Throughput optimized
Batch.size	32,768 (32 kB)	100,000 (100 kB)
Linger.ms	0 ms	10 ms
Compression.type	none	lz4
Partitions	3 / 5 depends on the cluster size	
Buffer.memory	335,544,32 (32 MB)	
ACKS	1	
Throughput	-1	
Replication.factor	3	

In the configuration, "-1" denotes that the throughput is not limited. *Bootstrap.server* points to one of the Kafka endpoints, and Kafka distributes the traffic between the nodes. *Buffer.memory* can adjust how much memory is allocated during the producer benchmark. We do not expect to use many partitions during the use-cases, so we are using 32 MB buffer memory regarding the guideline. One of the best ways to optimize throughput is to increase the producer batch size via the *Batch.size* option. Larger batch sizes result in fewer processing requests on the broker side and fewer requests to generate on the loader side, i.e., the system can achieve higher throughput with lower CPU utilization and spare the network bandwidth. We used 100 kB batch size during the throughput optimized strategy, and in the latency optimized scenario, we used 32 kB. According to the guideline, the second-best parameter to increase the throughput is to increase *Linger.ms* value. *Linger.ms* parameter sets the waiting time for the batch to fill up with messages. In the throughput benchmark, we increased the default value to 10 ms. Compressing data batches improves throughput and reduces the load on storage but might not be suitable for low-latency applications where compression or decompression costs are inevitable. In our throughput optimized experiment, we used the lz4 compression method. *Replication.factor* ensures that messages are copied and stored on multiple brokers. We used a replication factor of three during the measurements. Kafka's topics are divided into several partitions. While the topic is a logical concept in Kafka, a *partition* is the smallest storage unit that holds a subset of records owned by a topic. *ACKS* defines the number of acknowledgements the producer requires the leader broker to have received before considering a request complete. In a low latency application, it is not reasonable to get an acknowledgement from every node. However, one acknowledgement can assure that the data is processed, and Kafka will synchronize the data to achieve the desired replication factor. Table 1 summarizes the different configurations used for the measurements.

B. RESULTS

Figure 10 represents a typical benchmark load on one of the Kafka cluster node. CPU load graphs 10 (**a, b**) represent the CPU load on one of the Kafka nodes. The first peek shows the active data handling and synchronization process across the cluster. As soon as the peak load decreases, the loader cluster has finished the data sending, and the rest of the load, which is around 15-25% corresponds to the synchronization process between the Kafka cluster. We can observe the same behaviour on the network 10 (**c**) and disk performance 10 (**e, f**) graphs. The active data handling and processing stages are the more resource-demanding tasks compared to the data synchronization part. Memory 10 (**d**) and disk-related 10 (**e, f**) graphs show that the data is written directly to the disk. Thus, we can achieve approximately constant throughput speeds. Disk graphs point out the system bottleneck, which is the cloud underlying storage performance. The peak node performance values were the following: CPU utilization 50.9%, 3.74 Gb/s input and 1.49 Gb/s output network traffic, 468.6 MB/s write, and 91.96 MB/s read performance for the storage.

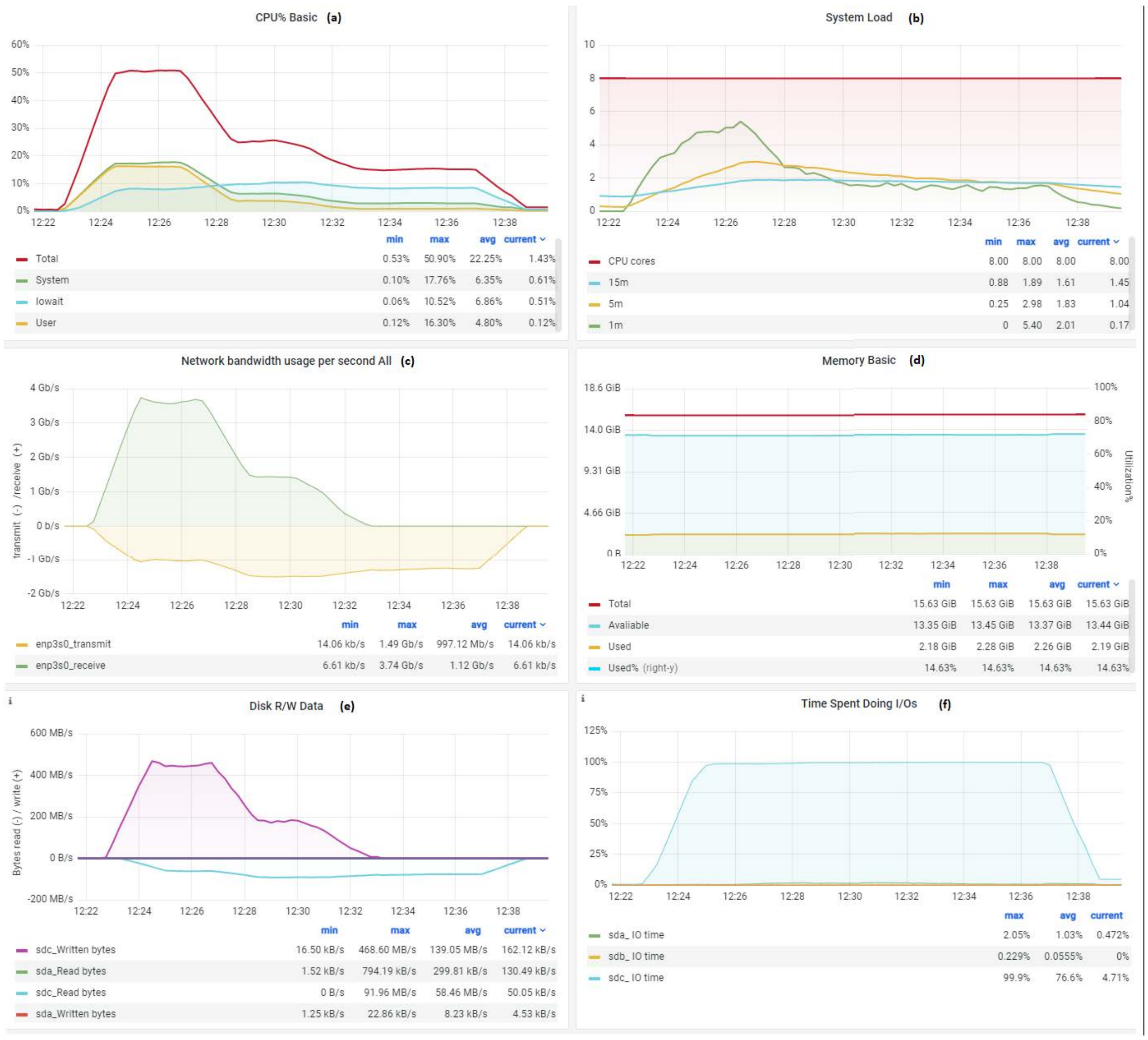
For each measurement, tables were presented based on the median and P_{90} values obtained by repeating the experiments five times. Table 2 represents the benchmark on three Kafka nodes in latency optimized configuration. The rows represent the different packet sizes, and the columns are the median and the 90th percentile of the measured values. Table 3 represents the measurements for three Kafka nodes in throughput optimized configuration and table 4, 5 represents five node Kafka cluster with the configuration fine-tuning explained earlier.

TABLE 2. Ingestion stage evaluation results using a three node Kafka cluster for ingestion (latency optimized).

Record Size (Byte)	**Throughput (Record/sec)**		**Throughput (MB/sec)**		**Latency (ms)**	
	Median (Q2)	P_{90}	**Median (Q2)**	P_{90}	**Median (Q2)**	P_{90}
64	2,810,456.30	2,912,822.08	171.55	177.79	2.00	2.66
256	1,033,260.80	1,062,845.08	252.26	259.49	5.64	14.81
512	499,156.00	499,156.00	243.74	243.74	3.68	3.68
1,024	263,493.20	263,493.20	257.31	257.31	3.35	3.35
2,048	136,952.00	136,952.00	267.49	267.49	6.63	6.63
4,096	69,099.80	69,099.80	269.92	269.92	3.03	3.03
8,192	37,186.80	37,186.80	290.53	290.53	4.10	4.10
16,384	18,660.20	18,660.20	291.57	291.57	2.05	2.05
32,768	9,614.40	9,614.40	300.44	300.44	1.90	1.90

TABLE 3. Ingestion stage evaluation results using a three node Kafka cluster for ingestion (throughput optimized).

Record Size (Byte)	**Throughput (Record/sec)**		**Throughput (MB/sec)**		**Latency (ms)**	
	Median (Q2)	P_{90}	**Median (Q2)**	P_{90}	**Median (Q2)**	P_{90}
64	5,868,120.40	6,017,009.60	358.17	367.26	4.86	4.49
256	2,023,865.15	2,072,222.63	494.10	505.92	3.76	3.44
512	1,113,407.95	1,131,852.04	543.66	552.67	3.48	3.36
1,024	587,245.80	595,734.09	573.48	581.77	3.41	3.31
2,048	305,104.70	310,562.78	595.92	606.58	3.49	3.24
4,096	153,871.80	156,013.32	601.06	609.43	3.47	3.20
8,192	77,257.90	78,739.30	603.58	615.16	3.50	3.30
16,384	38,731.00	39,555.89	605.17	618.06	3.36	3.11
32,768	18,818.00	19,218.00	588.07	600.58	3.17	3.04


CPU% Basic (a)
System Load (b)
Network bandwidth usage per second All (c)
transmit (-) /receive (+)
Memory Basic (d)
Utilization%
Disk R/W Data (e)
Bytes read (-) / write (+)
Time Spent Doing I/Os (f)

	min	max	avg	current
Total	0.53%	50.90%	22.25%	1.43%
System	0.10%	17.76%	6.35%	0.61%
Iowait	0.06%	10.52%	6.86%	0.51%
User	0.12%	16.30%	4.80%	0.12%

	min	max	avg	current
CPU cores	8.00	8.00	8.00	8.00
15m	0.88	1.89	1.61	1.45
5m	0.25	2.98	1.83	1.04
1m	0	5.40	2.01	0.17

	min	max	avg	current
enp3s0_transmit	14.06 kb/s	1.49 Gb/s	997.12 Mb/s	14.06 kb/s
enp3s0_receive	6.61 kb/s	3.74 Gb/s	1.12 Gb/s	6.61 kb/s

	min	max	avg	current
Total	15.63 GiB	15.63 GiB	15.63 GiB	15.63 GiB
Avaliable	13.35 GiB	13.45 GiB	13.37 GiB	13.44 GiB
Used	2.18 GiB	2.28 GiB	2.26 GiB	2.19 GiB
Used% (right-y)	14.63%	14.63%	14.63%	14.63%

	min	max	avg	current
sdc_Written bytes	16.50 kB/s	468.60 MB/s	139.05 MB/s	162.12 kB/s
sda_Read bytes	1.52 kB/s	794.19 kB/s	299.81 kB/s	130.49 kB/s
sdc_Read bytes	0 B/s	91.96 MB/s	58.46 MB/s	50.05 kB/s
sda_Written bytes	1.25 kB/s	22.86 kB/s	8.23 kB/s	4.53 kB/s

	max	avg	current
sda_ IO time	2.05%	1.03%	0.472%
sdb_ IO time	0.229%	0.0555%	0%
sdc_ IO time	99.9%	76.6%	4.71%

FIGURE 10. **A typical resource usage pattern during the benchmark process in one of the Kafka node. (a) Total CPU usage of the system (percentage); (b) System load over a period of time (1m, 5m, 15m); (c) Received and transmitted network bandwidth usage per second; (d) Total memory usage of the system (percentage); (e) Disk read and write speed; and (f) The time spent on I/O in the natural time of each second.**

TABLE 4. **Ingestion stage evaluation results using a five node Kafka cluster for ingestion (latency optimized).**

Record Size (Byte)	Throughput (Record/sec)		Throughput (MB/sec)		Latency (ms)	
	Median (Q2)	P_{90}	Median (Q2)	P_{90}	Median (Q2)	P_{90}
64	3,735,536.30	3,939,183.85	228.00	240.43	2.97	4.42
256	1,421,410.60	1,465,821.06	347.03	357.88	15.58	38.40
512	771,748.50	797,557.54	376.83	389.43	17.67	42.92
1,024	417,547.50	431,058.05	407.77	420.96	23.88	60.74
2,048	229,341.80	235,790.56	447.93	460.53	30.09	72.88
4,096	118,957.80	122,364.20	464.68	477.99	11.75	29.79
8,192	61,528.20	62,904.92	480.70	491.45	3.28	8.31
16,384	31,782.20	32,288.40	496.60	504.51	1.28	3.57
32,768	16,151.70	16,387.98	504.76	512.12	1.43	3.60

TABLE 5. **Ingestion stage evaluation results using a five node Kafka cluster for ingestion (throughput optimized).**

Record Size (Byte)	Throughput (Record/sec)		Throughput (MB/sec)		Latency (ms)	
	Median (Q2)	P_{90}	Median (Q2)	P_{90}	Median (Q2)	P_{90}
64	8,675,443.50	8,895,890.04	529.50	542.97	4.60	5.95
256	3,006,083.30	3,082,123.83	733.91	752.47	3.25	4.33
512	1,622,703.75	1,648,402.52	792.32	804.87	3.05	3.95
1,024	857,376.60	867,659.89	837.27	847.32	2.90	4.02
2,048	448,829.60	459,277.92	876.62	897.03	2.86	3.95
4,096	225,413.40	227,977.70	880.52	890.53	2.86	3.87
8,192	113,094.05	115,949.29	883.56	905.85	2.89	3.55
16,384	56,418.65	57,212.70	881.54	893.95	2.77	3.52
32,768	27,610.65	28,216.83	862.84	881.79	2.73	3.55

Figure 11 presents the relation between throughput and records size. For Y-axis logarithmic scale is used in this chart. Figure 12 shows the relation between bandwidth throughput performance and records size. Based on our measurements, the peak throughput speed in the cluster was 883.56 MB/s with 8,192 byte records. Between the record

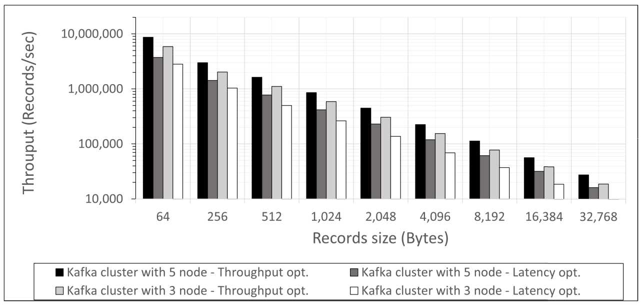


FIGURE 11. Effect of batch size on throughput in records/s in Kafka — higher is better; Note: The scale on the y-axis is logarithmic.

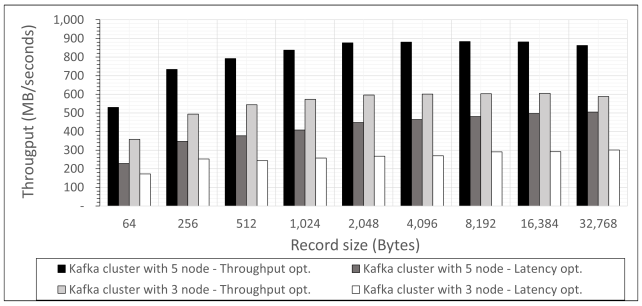


FIGURE 12. Effect of batch size on throughput in bandwidth in Kafka — higher is better.

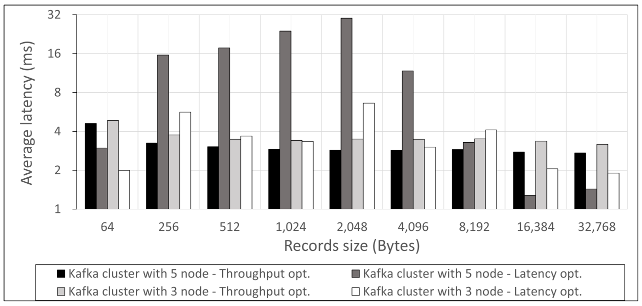


FIGURE 13. Effect of batch size on throughput in records/s in Kafka — lower is better; Note: The scale on the y-axis is logarithmic.

sizes of 2,048 and 32,768 in the same scenario, the reduction in the throughput speed is negligible. Figure 13 represents the values of the latency during the benchmark using logarithmic scale for the Y-axis. Interestingly, during the experiment, we got higher latency values between the record sizes of 256 and 8,192 in the latency optimized configuration.

Table 6 and 7 show the scalability measurements of the Kafka cluster. The tables show the difference in the throughput and latency using a three and five node sized Kafka cluster for the different scenarios. The green backgrounds in the table cell mark the system property improvement, and red backgrounds signify the values' deterioration. Table 6 data shows that latency configuration fine-tuning is not scaling very well in terms of latency in case of small record sizes. As a matter of throughput, scale-up provide ≈40% higher throughput. Table 7 data indicates better scalability where throughput increased ≈67% and latency decreased ≈14, 55% during the measurements.

TABLE 6. Evaluation of the speedup of the latency optimized Kafka clusters using three vs. five nodes.

latency optimized

Record Size (Byte)	Throughput (record/s)		Latency (ms)	
	Difference	Gain	Difference	Gain
64	925,080.00	75.24%	-0.97	-48.33%
256	388,149.80	27.31%	-9.95	-176.42%
512	272,592.50	35.32%	-13.99	-380.73%
1,024	154,054.30	36.90%	-20.53	-612.69%
2,048	92,389.80	40.28%	-23.47	-354.21%
4,096	49,858.00	41.91%	-8.73	-288.43%
8,192	24,341.40	39.56%	0.82	19.92%
16,384	13,122.00	41.29%	0.78	37.80%
32,768	6,537.30	40.47%	0.47	24.56%

TABLE 7. Evaluation of the speedup of the throughput optimized Kafka clusters using three vs. five nodes.

throughput optimized

Record Size (Byte)	Throughput (record/s)		Latency (ms)	
	Difference	Gain	Difference	Gain
64	2,807,323.10	67.60%	0.26	5.30%
256	982,218.15	65.63%	0.51	13.50%
512	509,295.80	69.24%	0.43	12.48%
1,024	270,130.80	69.27%	0.50	14.76%
2,048	143,724.90	69.49%	0.63	18.04%
4,096	71,541.60	69.35%	0.62	17.71%
8,192	35,836.15	67.12%	0.61	17.35%
16,384	17,687.65	66.93%	0.60	17.73%
32,768	8,792.65	65.18%	0.45	14.07%

As a summary, using the throughput optimized configuration our system is able to achieve overall better throughput with acceptable latency, and with overall better scalability. Based on the performance measurements, we found that processor and memory utilization do not pose a bottleneck in the system. The most resource-demanding state of the system utilizes ≈50.9% CPU. This suggests that it may be worthwhile to choose another type of flavor for better cloud resource utilization. Further, we investigated the resource-demanding states in the system and endeavour to gain better utilization on the cloud side.

Based on the outcome of the benchmarks, we can determine the system's performance conditions and fine-tune the values based on the needs of the experiments. As a result, we were able to better understand the performance characteristics of our system, and what it is possible when scaling our current architecture.

V. CONCLUSION

In this paper we introduced a scalable, cloud-agnostic and fault-tolerant data analytics platform for advanced IoT use cases. Additionally, our work relies on stable open-source components to build a reusable architecture blueprint also known as reference architecture. The applied components have commercial and/or cloud-based alternatives that can be swapped in as needed. Additionally, we demonstrated the value of the platforms through two uses cases. The first use case being the guaranteeing safe motion of mobile robots for logistic and transport process using cloud-aided learning. The second presented use case addresses Industrial IoT (IIoT) data collection and enrichment from a robotic assembly scenario over a service-oriented manufacturing execution system, in which the collaborative robotic arm near real time-data provision provides the foundation necessary for a better understanding of the while process.

Current limitations include the reliance on single core performance for MQTT-based ingestion due to the limitations of Eclipse Mosquitto. However, for the ingestion stage our main limitation is the Kafka ingestion cluster. Based on our evaluations, we understand what further system adjustments, including allocation of more system resources, and cluster sizing is required there. Based on the outcome of the benchmarks, we understand performance conditions of the system and can fine-tune the values based on the needs of the experiments. As a result, we were able to better understand the performance characteristics of our system, including the scalability of the current architecture.

We are implementing our findings in anticipation of new use cases and increasing traffic volume from existing ones. Further, we plan to introduce dedicated Apache NiFi clusters for the use cases, allowing better separation of workloads. For the data warehouse we are not expecting issues, however, TimescaleDB supports clustering via the standard PostgreSQL replication and also supports distributed hypertables [61]. We are investigating the latter for a future use case involving live data collection from autonomous vehicles over 5G cellular network.

Finally, we are working on releasing the complete platform and infrastructure descriptors as an open-source reference architecture using Occopus [62] and MiCADO-Edge [63] cloud-agnostic orchestrators.

ACKNOWLEDGMENT

The authors would like to thank for the usage of ELKH Cloud (https://science-cloud.hu/) that significantly helped them achieving the results published in this paper (On behalf of project "ARNL: GPU-Enabled Cloud-Based Big Data/AI Research Platform").

REFERENCES

[1] *What is a Reference Architecture?—Enterprise it Definitions*. Accessed: Feb. 20, 2022. [Online]. Available: https://www.hpe.com/us/en/what-is/reference-architecture.html

[2] P. Pääkkönen and D. Pakkala, "Reference architecture and classification of technologies, products and services for big data systems," *Big Data Res.*, vol. 2, no. 4, pp. 166–186, Dec. 2015.

[3] *Microsoft Azure Documentation Reference Architectures*. Accessed: Feb. 20, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/browse/

[4] *The TOGAF Standard, Version 9.2 Overview*. Accessed: Feb. 12, 2022. [Online]. Available: https://www.opengroup.org/togaf

[5] *Microsoft Azure Documentation Reference Architectures for Data Analytics*. Accessed: Mar. 26, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/browse/?azure_categ%ories=analytics

[6] *AWS Architecture Center Architecture Best Practices for Analytics & Big Data*. Accessed: Mar. 26, 2022. [Online]. Available: https://aws.amazon.com/architecture/analytics-big-data/?cards-all.sort-%by=item.additionalFields.sortDate&cards-all.sort-order=desc&awsf.content-type=%content-type%23reference-arch-diagram&awsf.methodology=*all

[7] S.-W. Lin, B. Murphy, E. Clauer, U. Loewen, R. Neubert, G. Bachmann, M. Pai, and M. Hankel, "Architecture alignment and interoperability: An industrial internet consortium and platform industrie 4.0 joint whitepaper," Ind. Internet Consortium, Boston, MA, USA, White Paper, 2017. [Online]. Available: https://www.iiconsortium.org/pdf/JTG2_Whitepaper_final_20171205.pdf

[8] K. Schweichhart. (2016). *Reference Architectural Model Industrie 4.0 (Rami 4.0)*. An Introduction. [Online]. Available: https://www.plattform-i40.deI

[9] A. C. Marosi, R. Lovas, A. Kisari, and E. Simonyi, "A novel IoT platform for the era of connected cars," in *Proc. IEEE Int. Conf. Future IoT Technol. (Future IoT)*, Jan. 2018, pp. 1–11.

[10] A. C. Marosi, A. Farkas, and R. Lovas, "An adaptive cloud-based IoT back-end architecture and its applications," in *Proc. 26th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2018, pp. 513–520.

[11] R. Lovas, A. C. Marosi, M. Emodi, A. Kisari, E. Simonyi, and P. Gaspar, "PaaS-oriented IoT platform with connected cars use cases," in *Proc. Int. Conf. Sensor Netw. Signal Process. (SNSP)*, Oct. 2018, pp. 409–420.

[12] N. Marz. (2011). *How to Beat the Cap Theorem*. Accessed: Mar. 2, 2022. [Online]. Available: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

[13] J. Kreps. (2014). *Questioning the Lambda Architecture*. Accessed: Feb. 3, 2022. [Online]. Available: https://www.oreilly.com/radar/questioning-the-lambda-architecture/

[14] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[15] *Azure IoT Reference Architecture*. Accessed: Mar. 1, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/reference-architect%ures/iot

[16] *Microsoft Azure Real Time Analytics*. Accessed: Mar. 1, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/arti%cles/real-time-analytics

[17] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. USENIX Int. Workshop Netw. Meets Databases*, vol. 11, 2011, pp. 1–7.

[18] F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng, and E. Kovacs, "A standard-based open source IoT platform: FIWARE," *IEEE Internet Things Mag.*, vol. 2, no. 3, pp. 12–18, Sep. 2019.

[19] H. Isah and F. Zulkernine, "A scalable and robust framework for data stream ingestion," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2900–2905.

[20] *Apache Software Foundation*. Apache NIFI. Accessed: Feb. 1, 2022. [Online]. Available: https://nifi.apache.org

[21] *Apache Hadoop: HDFS Architecture*. Accessed: Mar. 7, 2022. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/H%dfsDesign.html

[22] *How Airbnb Achieved Metric Consistency at Scale*. Accessed: Mar. 7, 2022. [Online]. Available: https://medium.com/airbnb-engineering/how-airbnb-achieved-metric-consis%tency-at-scale-f23cc53dea70

[23] *How Airbnb Standardized Metric Computation at Scale*. Accessed: Mar. 7, 2022. [Online]. Available: https://medium.com/airbnb-engineering/airbnb-metric-computation-with-mi%nerva-part-2-9afe6695b486

[24] *How Airbnb Enables Consistent Data Consumption at Scale*. Accessed: Mar. 7, 2022. [Online]. Available: https://medium.com/airbnb-engineering/how-airbnb-enables-consistent-dat%a-consumption-at-scale-1c0b6a8b9206

[25] *Apache Airflow: Platform Created by the Community to Programmatically Author, Schedule and Monitor Workflows*. Accessed: Mar. 7, 2022. [Online]. Available: https://airflow.apache.org/

[26] M. Zaharia, "Apache Spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Nov. 2016.

[27] *Apache Druid: A High Performance Real-Time Analytics Database*. Accessed: Mar. 7, 2022. [Online]. Available: https://druid.apache.org/

[28] *Presto: Distributed Sql Query Engine for Big Data*. Accessed: Mar. 7, 2022. [Online]. Available: https://prestodb.io/

[29] D. L. Moody, "From enterprise models to dimensional models: A methodology for data warehouse and data mart design," in *Proc. DMDW*, Jun. 2000, p. 5.

[30] *Apache Superset is a Data Visualization and Data Exploration Platform*. Accessed: Mar. 7, 2022. [Online]. Available: https://superset.apache.org/

[31] *Terraform by Hashicorp: Terraform is an Open-Source Infrastructure as Code Software Tool That Enables You to Safely and Predictably Create, Change, and Improve Infrastructure*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.terraform.io

[32] *Ansible is Simple it Automation*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.ansible.com

[33] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3. Kobe, Japan, 2009, p. 5.

[34] *Elastic Stack: Elasticsearch, Kibana, Beats & Logstash*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.elastic.co/elastic-stack/

[35] *KSQLDB: The Database Purpose-Built for Stream Processing Applications*. Accessed: Feb. 1, 2022. [Online]. Available: https://ksqldb.io

[36] *Timescaledb: Time-Series Data Simplified*. Accessed: Feb. 1, 2022. [Online]. Available: https://www.timescale.com

[37] *DBT Transform Data in Your Warehouse*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.getdbt.com

[38] O.-C. Marcu, A. Costan, G. Antoniu, M. S. Perez-Hernandez, R. Tudoran, S. Bortoli, and B. Nicolae, "Towards a unified storage and ingestion architecture for stream processing," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 2402–2407.

[39] Eclipse Foundation. *Sparkplug Topic Namespace and State ManagementV2.2-With Appendix B Format*. Accessed: Feb. 23, 2022. [Online]. Available: https://www.eclipse.org/tahu/spec/Sparkplug%20Topic%20Namespace%20and%2%0State%20ManagementV2.2-with%20appendix%20B%20format%20-%20Eclipse.pdf

[40] R. A. Light, "Mosquitto: Server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, p. 265, May 2017.

[41] B. Mishra, B. Mishra, and A. Kertesz, "Stress-testing MQTT brokers: A comparative analysis of performance measurements," *Energies*, vol. 14, no. 18, p. 5817, Sep. 2021.

[42] *EMQX: Open-Source, Cloud-Native MQTT Broker for IoT*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.emqx.io/

[43] *Cloudera Dataflow (CDF): Deployment Options*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.cloudera.com/products/cdf.html

[44] Cloudera. (2019). *NIFI Sizing Guide and Deployment Best Practices*. Accessed: Feb. 1, 2022. https://community.cloudera.com/t5/Community-Articles/NiFi-Sizing-Guide-%Deployment-Best-Practices/ta-p/246781

[45] TimescaleDB. *Speed up Grafana by Auto-Switching Between Different Aggregations, Using Postgresql*. Accessed: Mar. 25, 2022. [Online]. Available: https://www.timescale.com/blog/speed-up-grafana-autoswitching-postgresq%l/

[46] *Prometheus: Monitoring System & Time-Series Database*. Accessed: Feb. 1, 2022. [Online]. Available: https://prometheus.io

[47] *Cadvisor: Analyzes Resource Usage and Performance Characteristics of Running Containers*. Accessed: Feb. 1, 2022. [Online]. Available: https://github.com/google/cadvisor

[48] *Prometheus Node_Exporter: Exporter for Machine Metrics*. Accessed: Feb. 1, 2022. [Online]. Available: https://github.com/prometheus/node_exporter

[49] *Prometheus JMX_Exporter: A Process for Exposing JMX Beans*. Accessed: Feb. 1, 2022. [Online]. Available: https://github.com/prometheus/jmx_exporter

[50] R. Beregi, G. Pedone, and I. Mezgár, "A novel fluid architecture for cyber-physical production systems," *Int. J. Comput. Integr. Manuf.*, vol. 32, nos. 4–5, pp. 340–351, May 2019.

[51] R. Beregi, G. Pedone, B. Háy, and J. Váncza, "Manufacturing execution system integration through the standardization of a common service model for cyber-physical production systems," *Appl. Sci.*, vol. 11, no. 16, p. 7581, Aug. 2021.

[52] OPC Foundation. (2019). *OPC Unified Architecture. Interoperability for Industrie 4.0 and the Internet of Things*. Accessed: Feb. 3, 2022. [Online]. Available: https://www.festo.com/rep/en-gb_gb/assets/pdf/GB-OPC-UA-Interoperabilit%y-For-Industrie4-and-IoT-EN-v5-June-2017.pdf

[53] Z. Li, I. V. Kolmanovsky, E. M. Atkins, J. Lu, D. P. Filev, and Y. Bai, "Road disturbance estimation and cloud-aided comfort-based route planning," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3879–3891, Nov. 2017.

[54] Z. Li, I. Kolmanovsky, E. Atkins, J. Lu, D. Filev, and J. Michelini, "Cloud aided safety-based route planning," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2014, pp. 2495–2500.

[55] J. Guanetti and F. Borrelli, "Stochastic MPC for cloud-aided suspension control," in *Proc. IEEE 56th Annu. Conf. Decis. Control (CDC)*, Dec. 2017, pp. 238–243.

[56] Z. Li, T. Chu, I. V. Kolmanovsky, X. Yin, and X. Yin, "Cloud resource allocation for cloud-based automotive applications," *Mechatronics*, vol. 50, pp. 356–365, Apr./May 2018.

[57] M. Rodrigues, A. McGordon, G. Gest, and J. Marco, "Autonomous navigation in interaction-based environments—A case of non-signalized roundabouts," *IEEE Trans. Intell. Vehicles*, vol. 3, no. 4, pp. 425–438, Dec. 2018.

[58] M. Z. Abnili and N. L. Azad, "On-line situational awareness for autonomous driving at roundabouts using artificial intelligence," *J. Mach. Intell. Data Sci.*, vol. 2, pp. 17–24, Jan. 2021.

[59] B. Németh and P. Gáspár, "The design of performance guaranteed autonomous vehicle control for optimal motion in unsignalized intersections," *Appl. Sci.*, vol. 11, no. 8, p. 3464, Apr. 2021.

[60] Y. Byzek, "White paper: Optimizing your apache Kafka deployment," Confluent, Inc., Mountain View, CA, USA, Tech. Rep., 2020. [Online]. Available: https://www.confluent.io/white-paper/optimizing-your-apache-kafka-deployment/

[61] *Timescaledb: Distributed Hypertables*. Accessed: Mar. 28, 2023. [Online]. Available: https://docs.timescale.com/timescaledb/latest/how-to-guides/hypertables%/distributed-hypertables/

[62] J. Kovács and P. Kacsuk, "Occopus: A multi-cloud orchestrator to deploy and manage complex scientific infrastructures," *J. Grid Comput.*, vol. 16, no. 1, pp. 19–37, Mar. 2018.

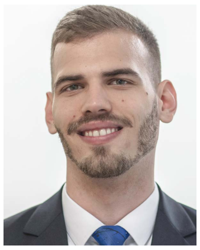
[63] A. Ullah, H. Dagdeviren, R. C. Ariyattu, J. DesLauriers, T. Kiss, and J. Bowden, "MiCADO-edge: Towards an application-level orchestrator for the Cloud-to-edge computing continuum," *J. Grid Comput.*, vol. 19, no. 4, p. 47, Dec. 2021.

ATTILA CSABA MAROSI received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics. He is currently a Research Fellow with the Laboratory of Parallel and Distributed Systems (LPDS), Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He has over 20-years of research and development experience involving both industry and academia that encompasses a wide range of distributed and parallel systems. He has contributed to many national and international research and development projects. He is the coauthor of over 50 scientific papers. His research interests include large-scale time-series data collection and inference.

MÁRK EMŐDI received the B.Sc. and M.Sc. degrees in computer science engineering from the John von Neumann Faculty of Informatics, Óbuda University. He is currently pursuing the Ph.D. degree in cloud trouble-shooting. He is a Research Associate with the Laboratory of Parallel and Distributed Systems (LPDS), Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He has been involved in COLA, CloudiFacturing, and DIGITbrain European H2020 projects. His research interests include software engineering, parallel computing, clouds, and machine learning.

ATTILA FARKAS received the B.Sc. and M.Sc. degrees in computer science engineering from the John von Neumann Faculty of Informatics, Óbuda University. He is currently pursuing the Ph.D. degree in distributed deep learning. He is a Research Associate with the Laboratory of Parallel and Distributed Systems (LPDS), Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He has been involved in COLA and NEANIAS European H2020 projects. His research interests include parallel computing, clouds, container technologies, and machine learning.

RÓBERT LOVAS received the Ph.D. degree in informatics from the Budapest University of Technology and Economics. He is currently the Deputy Director of the Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He is also a Habilitated Associate Professor and the Founder of the John von Neumann Faculty of Informatics, Institute for Cyber-Physical Systems, Óbuda University. His research and development experience in wide range of application fields of distributed and parallel systems has been gained in various global, EU, and national collaborations with academic organizations, universities, and enterprises focusing on computational chemistry, numerical meteorological modeling, bioinformatics, agriculture, connected cars, and industry 4.0. He has been coordinating EU FP7/H2020 projects and the ELKH Cloud research infrastructure. His latest cloud, big data, the IoT, and AI-related research achievements contribute to the recently launched Artificial Intelligence and Autonomous Systems National Laboratories. He is a member of the Committee on Information Science at the Hungarian Academy of Sciences.



RICHÁRD BEREGI graduated in mechatronical engineering from the Faculty of Mechanical Engineering, Budapest University of Technology and Economics, in 2016, where he is currently pursuing the Ph.D. degree. He is a Research Associate with the Research Laboratory on Engineering and Management Intelligence (EMI), Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He directs the development, education, and research activities of the SmartFactory Demonstration System for Cyber-Physical Production situated in the laboratory for the past eight years. His research interest includes the agent-based control of cyber-physical production systems.

GIANFRANCO PEDONE received the Ph.D. degree in computer science from Eötvös Lóránd University, Budapest, Hungary, in 2012. He is currently an IT Engineer and a Research Associate with the Research Laboratory on Engineering and Management Intelligence (EMI), Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH). He is involved in several national and international projects mainly related to smart cyber-physical systems applications, operational services in circular economy, industry 4.0 research innovation center of excellence, and the manufacturing execution systems of the future. He is the author or the coauthor of more than 30 scientific articles, with more than 360 independent references. His main research interests include multi-agent systems, languages, and cloud and web computing for semantic interoperability in scientific and business applications, more recently cyber-physical system applications and interoperability, cross-sectorial, and circular economy digital platforms.

BALÁZS NÉMETH (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Faculty of Transportation and Vehicle Engineering, Budapest University of Technology and Economics, Budapest, Hungary, in 2009 and 2013, respectively. Since 2007, he has been a Senior Research Fellow with the Systems and Control Laboratory, Institute for Computer Science and Control, Eötvös Loránd Research Network (ELKH). His current research interests include energy-optimal control of autonomous road vehicles, in-vehicle intelligent transportation systems, driver-assistance systems and human factors, and nonlinear analysis and synthesis of control systems.

PÉTER GÁSPÁR received the M.Sc. and Ph.D. degrees from the Faculty of Mechanical Engineering, Budapest University of Technology and Economics, Budapest, Hungary, in 1985 and 1997, respectively, and the D.Sc. degree in control from the Hungarian Academy of Sciences (HAS), Budapest, in 2007. Since 1990, he has been a Senior Research Fellow with the Systems and Control Laboratory (SCL), Institute for Computer Science and Control, Eötvös Loránd Research Network (ELKH), where he has been a Research Advisor, since 2007. He is currently the Head of the Vehicle Dynamics and Control Research Group, SCL. He is also a Full Professor with the Control and Transport Automation Department, Budapest University of Technology and Economics. His current research interests include linear and nonlinear systems, robust control, multiobjective control, system identification, identification for control, mechanical systems, vehicle structures, and vehicle control. Since 2016, he has been a Corresponding Member of the Hungarian Academy of Sciences. He is a member of the IFAC Technical Committee on both the automotive control and the transportation systems.