A Fog-Cloud Computing Infrastructure for Condition Monitoring and Distributing Industry 4.0 Services

Timo Bayer, Lothar Moedel and Christoph Reich
Institute of Cloud Computing and IT Security, University of Applied Science, 78120 Furtwangen, Germany

Keywords: Fog Computing, Industrial Cyber Physical System, Application Deployment, Condition Monitoring.

Abstract: Data-driven Industry 4.0 applications require low latency data processing and reliable communication models to enable efficient operation of production lines, fast response to failures and quickly adapt the manufacturing process to changing environmental conditions. Data processing in the Cloud has been widely accepted and in combination with Fog technologies, it can also satisfy these requirements.
This paper investigates the placement of service containers and wheater they should be carried out in the Cloud or at a Fog node. It shows how to provide an uniform well-monitored execution environment to automatically distribute services concerning their application-specific requirements. An infrastructure is presented, that utilizes measurement probes to observe the node and environmental conditions, derive and evaluate appropriate distribution algorithms and finally deploy the application services to the node that meets the requirements.

1 INTRODUCTION

Industry 4.0 is a ubiquitous keyword, including automated and data-driven manufacturing technologies, communicating plants and a large set of data sources to collect detailed process information. Basis for this, are optimized processing infrastructures providing computation capabilities and data storage.

Cloud Computing is a typical enabler for such applications. However, Industry 4.0 applications often deal with frequency data and require low latency data processing. Low latency is important to enable efficient operation, fast response to failures and quickly adapt the manufacturing processes to changing environmental conditions (Jiang et al., 2017). Furthermore, they are often built on sensitive data, including intellectual properties or customer data that need to be protected. Low latency and strict privacy requirements make the use of Cloud Computing problematic.

A new trend to overcome these insufficiencies is to combine Cloud Computing and Fog Computing. For instance, Fog Computing can be used to provide low latency data processing or perform preprocessing tasks such as data fusion. Using the results of the Fog nodes, further tasks that require considerable processing power are executed inside the Cloud.

Combining Cloud Computing and Fog Computing poses particular challenges concerning service execution, data integration and communication capabilities. One point of special interest is the decision weather a service is better executed in the Cloud or on a Fog node. To be able to do a founded decision, several characteristics of the involved nodes and their environment need to be considered.

This paper investigates most significant distribution criteria and presents an architecture to measure and collect the required data for service distribution, apply suitable distribution algorithms and finally deploys the services on the most suitable node. The architecture also provides an excelent environment to develop new or optimize existing distribution algorithms to match the individual requirements arising from combining Cloud and Fog Computing.

The paper is structured as follows. Section 2 presents related topics in the field of Fog Computing. In Section 3 criteria and requirements for service distribution are investigated. The architecture to measure these criteria and deploy the services accordingly is presented in Section 4 and evaluated in Section 5. Section 6 summarizes the results.

2 RELATED WORK

Several investigations took place showing possible use cases and approaches to integrating Cloud and Fog. Mahmud et al. (Mahmud et al., 2018) introduced an IoT healthcare solution and explored the in-

Bayer, T., Moedel, L. and Reich, C.
A Fog-Cloud Computing Infrastructure for Condition Monitoring and Distributing Industry 4.0 Services.
DOI: 10.5220/0007584802330240
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 233-240
ISBN: 978-989-758-365-0

tegration of Cloud-Fog technologies. They evaluated the scenario in regards to distributed computing, reducing network latency, optimize communication and power consumption. The investigation showed that using Fog-Cloud solutions results in better task distribution, instance cost, energy usage and network delay. Notably, a significant improvement in the field of real-time processing can be recognized. The benefits are also pointed out by Chakraborty et al. (Chakraborty et al., 2016). They introduced a solution to gather, analyze and store time-sensitive data. To achieve minimum delay while ensuring data accuracy they perform time-sensitive tasks within the Fog nodes and apply further processing and long-term persistence inside the Cloud. We took these design principles and enhanced them with the capability to integrate measurement probes and container virtualization to create an infrastructure focussing on distribute services.

Developing an application that is distributed over Cloud and Fog poses particular challenges such as ensuring interoperability, providing consistent programming models and distribution algorithms. Recently appeared frameworks try to simplify application development. Cheng et al. (Cheng et al., 2018) described a framework for Smart City applications while considering distributed data processing, low latency and reducing network usage. The main characteristic of this approach is a programming model that allows to implement applications using Fog and Cloud in an uniform fashion. Using this approach, they showed three use cases including anomaly detection of energy consumption, video surveillance and smart city magnifier. There are several similarities to the work presented in our paper such as enabling low latency and scalable applications. To increase the flexibility and support heterogeneous nodes, we use a container-based approach instead.

The feasibility of using containerization is investigated by Bellavista et al. (Bellavista and Zanni, 2017). They presented a fog-oriented framework based on extensions of the Kura gateway to investigate the feasibility of Docker-based containerization even over strongly resource limited nodes. The results showed that containerization provides good scalability, limited overhead and high flexibility. Another approach that utilizes containerization is presented by Woebker et al. (Woebker et al., 2018). They described a solution to deploy and manage Fog applications, based on containers. The deployment mainly relies on statically raised distribution criterias such as latency and bandwith. Based on these criteria the nodes are labeled and categorized using the labeling system provided by Kubernetes. Our approach builds upon the solutions presented above but focusses on providing an infrastructure to measure environmental conditions such as the current load, location or security levels to support decision making wheater a service shall be carried out in the Cloud or on a Fog node.

A lot of effort has been made to investigate distribution algorithms in the areas of Cloud and High-Performance Computing. The solutions need to be adapted to the specific requirements of Fog and Cloud integration. Neto et al. (Neto et al., 2017) described the current problems of Fog Computing such as quality of service and load distribution. They described an algorithm to optimize load distribution, that takes criteria such as latency and priorities into consideration. The solution presented in our paper aims to provide a framework to evaluate existing distribution algorithm and apply them in Fog-Cloud environments.

3 DISTRIBUTION CRITERIA

Selecting the most suitable node to distribute a service is one primary challenge while combining Fog and Cloud. The relevance of the criteria highly depends on the optimization goal, like enery optimization, privacy protection or process time minimization. The following section presents static and dynamic criteria that can support the decision making.

Network Connection. Several network characteristics need to be considered to choose a suitable node. Ensuring **low latency** data processing is crucial to provide appropriate query time and fast computation. Industry 4.0 applications require to collect local data, process it and use the processing result to control the manufacturing process. Thus, it's necessary to deploy time-sensitive services to a node with low latency.

Industry 4.0 services often processes large datasets and extract insights on time. Transfering large data sets need a high **network bandwidth** and is critical for deploying services. For instance, it's possible that short running applications perform better deployed in the Cloud due to the delay for the deployment on low bandwidth nodes. The **type of network connection** is also relevant. Applications with high data rates or a high demand for **reliable network connections** require a wired connection, whereas other applications perform well with wireless connections.

Performance & Storage Capabilities. Industry 4.0 services require high computational and storage capabilities. In contrast, Fog nodes are typically resource limited. Therefore, it is necassary to consider the **performance** (e.g. **CPU and RAM**). Besides the statical performance, it is essential to dynamically check the

current load of the nodes before deploying a service. The involved data volume requires high storage capabilities. There is a tradeoff between the total **data amount** required to fulfill the desired task and the demand for **data locality** to match the latency requirements and reduce network traffic.

The number of mobile devices, such as autonomous guided vehicles increase the demand to integrate battery powered devices. Thus, the **energy consumption** of a service and the **available power supply** of the nodes need to be considered. A distinction can be made between services that are continously running or only react to low frequency events. This mainly affects if the service needs to be deployed on a device with a **permanent power supply** or if the **battery level** of the device fits the requirements.

Security & Privacy. Industry 4.0 applications process sensitive information including intellectual properties or customer data. Thus, it is required to provide a **secure infrastructure** and ensure **data privacy**. To achieve this, special characteristics need to be considered. The primary goal is to achieve mutual trust between the involved components. If this can't be ensured due to insecure **communication channels** or missing computation performance to provide advanced **security mechanisms**, it's better to select a different node. In contrast, if the sensitive data processing should only take place in a **trusted network**, using components with high **reputation**, it's appropriate to execute it on a Fog node.

The **node location** also needs to be considered regarding privacy concerns and national regulations. Using Cloud infrastructures, it is required to be aware where the virtual machine is deployed. Due to the possibility to change the location of a virtual machine or a mobile Fog node, it's required to dynamically check the current location before deploying a task and get informed if a **migration** took place.

The decision how secure a node is can be arbitrarily complex. Besides some easy examples such as checking for a encrypted communication, more complex or application specific methods are required. Security checks can comprise autonomous security tasks that run on each involved node and collect evidence about security or privacy concerns. These checks need to be applied on heterogeneous environments, such as Cloud nodes, Hypervisors and Fog nodes.

Inventory & Environmental Constraints. Distributing Industry 4.0 services also requires more application-specific distribution criteria, such as the **node inventory**. For instance, the node inventory can include the **data sources/actuators** attached to a specific node or the available **software libraries** needed to fulfill the desired task. Concerning the node inventory before deploying a service can also ensure **data locality** because the inventory can be used to determine where the required data is collected.

Another type of application-specific distribution criteria are **environmental constraints**. A environmental constraint can be a specific **type of node**, that is required by the service or a certain **level of reputation** a node must provide. Especially, if mobile nodes are involved, the constraints can also include the **availability of collaboration partners**.

4 ARCHITECTURE

The architecture presented in this section aims to overcome the challenges of combining Fog and Cloud Computing by providing uniform functionality to monitor advanced deployment criteria and consistently apply distribution algorithms to find the most suitable node. This section describes the requirements and the high level architecture.

4.1 Requirements

Develop a Industry 4.0 Service

- R1: Support distributed data processing tasks between Cloud and Fog nodes
- R2: Provide all required functionalities to provide suitable node management, inter-node communication, and task execution

Integrating Measurement Probes

- R3: Determine node conditions by integrating and deploying measurement probes on the nodes
- R4: Provide support for centralized and decentralized measurement probes
- R5: Provide a uniform way to execute the measurement probes and services on all nodes regardless of the actual technology they use
- R6: Provide a consistent execution environment that is suitable to migrate measurement probes and services during runtime

Integrate Distribution Algorithms

- R7: Integration of distribution algorithms that make use of the measurement results to deploy an application service accordingly
- R8: Provide a common interface to apply and configure different distribution algorithms

4.2 Overview & Containerization

The following section provides an overview about the architecture and how container virtualization is utilized to provide state of the art node and task management, design and deploy interoperable application services and how to execute measurement probes.

Infrastructure Services. According to Figure 1, the architecture can be divided into management and worker layers. The management layer provides functionality for node and task management, control the measurement probes and deploy services based on different distribution algorithms. These responsibilities are realized using infrastructure-specific implementations, which are described later.

Kubernetes Master. The service deployment, management functionality and low-level measurement probes are realized using *Kubernets*. The management layer includes a *Kubernetes* cluster, that is interfaced by the infrastructure services using *Kubectl*. *Kubectl* allows to add nodes, initiate the deployment or request data provided by the cluster. These requests are processed by the *API Server*. All common services such as the *Kubernetes Scheduler* for service deployment, the key-value store *etcd* as well as the *Controller Manager* deamon responsible for lifecycle management are contained in the *Kubernetes Master* and can be utilized by requesting the *API Server*. One component of particular interest is the *Metrics Server*. The *Metrics Server* is an add-on for the *Kubernetes* cluster and provides measurement capabilities. Within this architecture, the *Metrics Server* is used to collect low-level environmental conditions, such as the actual CPU or memory usage.

Kubernetes Nodes. The *Worker Layer* contains all nodes responsible for executing services and collecting the information required to decide how to deploy a specific service. On a low-level perspective all nodes are managed by the *Kubernetes Cluster*. Hence, all nodes are registered within the *Kubernetes Master* and contain the *Kublet* agent, responsible for monitor the *API server* and execute the desired containers. One central part of the *Kubelet* is the *cAdvisor*. The *cAdvisor* collects, processes and aggregates metrics about the containers. This information is used by the *Metrics Server*. All *Kubernetes Nodes* include a *Service Proxy* that enables inter-container communication by forwarding requests to the desired target. Besides these services, all worker nodes include *Measurement Pods* as well as several *Application Pods*. A

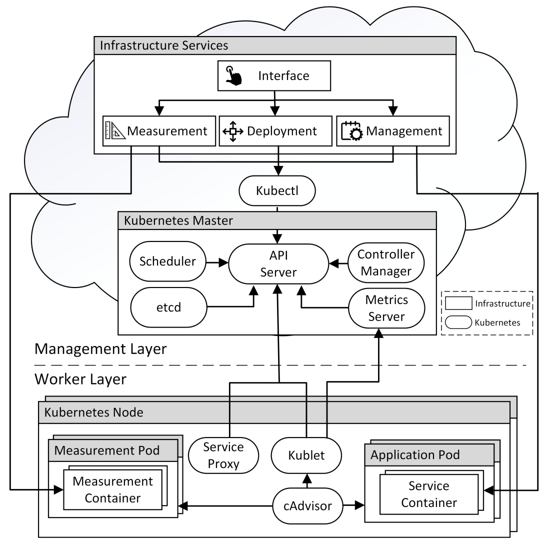


Figure 1: Architecture & Containerization.

Pod is the smallest execution unit that can be orchestrated by using *Kubernetes*. The architecture utilizes *Pods* to execute the services and provide distributed measurement capabilities. A *Pod* can be a whole application including several containers or just a single container. The *Measurement Pods* include all measurement tasks mentioned in Section 3. Which measurement probes are executed on a node can be configured by using the *Infrastructure Service*. The application services are executed within a *Application Pod*. Both kind of *Pods* are monitored by the *cAdvisor*.

Networking and Communication. Another challenge is to provide the networking capabilities while minimizing the administration effort. We realized the communication between nodes, *Pods* and containers using the *pod-overlay network* provided by *Flannel*. *Flannel* creates a network between the host environment, all *Pods* and all containers. The main benefit of using *Flannel* is the high flexibility concerning public IPs. In contrast to many other technologies, the IP addresses are not required to belong to the same network. Primarily, in geographically distributed systems this approach results in higher flexibility. The overlay network creates a private network that belongs to all *Kubernetes Nodes*. Within the nodes, each *Pod* receives an IP assigned from *Docker*. Communication between *Kubernetes Nodes* is also possible by using the *Flannel Kernel Route Table* and UDP encapsulation. This approach allows requesting components by just using the service name, forwarded by the *Service Proxy*. This benefit is utilized by the architecture in regards to migrating services dynamically based on changing environmental conditions.

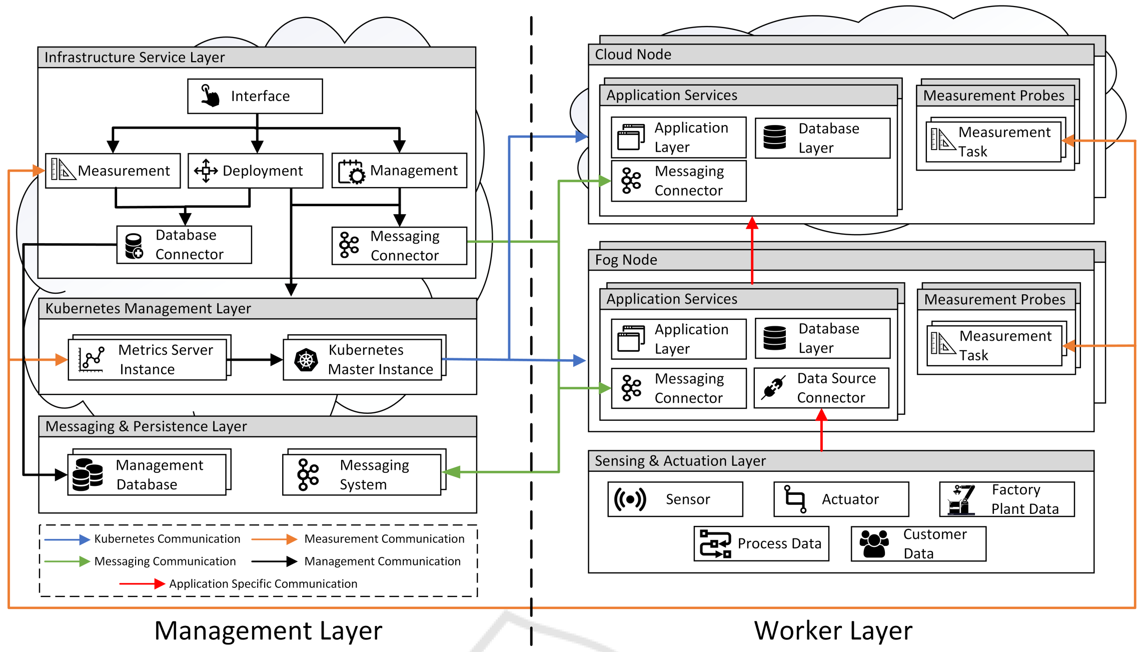


Figure 2: Architecture Overview.

4.3 Management & Measurements

The following section provides an overview about the architecture in regards to infrastructure management and measurement probes. According to Figure 2 the architecture is divided into management layer and worker layer. The *Infrastructure Service Layer* contains the management capabilities of the architecture. Using the *Interface* service, the infrastructure can be administrated, measurement probes can be chosen and deployment algorithms can be defined.

One major responsibility is to introduce uniform measurement capabilities for heterogeneous Cloud/Fog nodes. The measurements can be divided into low-level infrastructure and application-specific measurement probes, which are both controlled by the *Measurement Infrastructure Service*. The infrastructure measurements are provided by the *Metrics Server*. To be able to use these measurements, the *Measurement Infrastructure Service* periodically requests the *Kubernetes API Server*, converts the measurement data and persists it using the *Database Connector*. Due to the consistent execution environment all infrastructure measurements can be directly utilized requesting the *Metrics Server* and can be applied to each *Pod* within the *Kubernetes Cluster*. Furthermore, they can be configured e.g. to adapt the measurement interval. This low-level resource monitoring performs best regarding scalability and efficiency similar as can be seen in (Chang et al., 2017) and (Tsai et al., 2017). According to the creators, the *Metrics Server* can be scaled up to 5000 nodes with 30 *Pods* per node, assuming a 1 minute metrics granularity (Kubernetes, 2018). The *Metrics Server* can be scaled individually. Both services are located in the *Kubernetes Management Layer*.

While the *Metrics Server* already covers low-level performance metrics, more sophisticated measurements are required to optimize service deployment. Therefore, the *Measurement Infrastructure Service* is capable of executing central and controlling distributed measurements. Decentralized measurements come with higher communication efforts and the need to correlation the measurements. On the other hand, centralized measurements are limited in measurement capabilities and result in higher load of the infrastructure services. For instance, the centralized measurement probes include monitoring the network connection or determining the actual location. They also can be configured to either performed by the *Measurement Infrastructure Service* or in a distributed fashion.

Decentralized measurement probes are located at the worker nodes and controlled by the *Measurement Infrastructure Service*. The probes are executed within a *Pod*, which can either contain a single task such as monitoring the energy consumption or include multiple containers with more complex measurement probes such as enhanced security checks. The infrastructure allows deploying whole security frameworks such as the *Audit Agent System* which is able to check if the node complies to security policies such as the use of secure sockets or restrictions concerning data retention (Rübsamen et al., 2016). The required probes can either be configured while adding the node or can be changed during runtime to match changing requirements. Depending on the probe, data collection can be configured regarding collection periods, aggregation intervals or task-specific thresholds. Measurement results are stored in the *Management Database* and are made available for service deployment. To ensure scalability and loose coupling, the communication between the *Measurement Infrastructure Service* and the measurement probes is realized using a messaging system. The architecture supports hybrid probes with decentralized measurement probes and a centralized probe backend.

4.4 Service Deployment

Distributed Industry 4.0 applications include services using different programming languages or execution environments. Therefore, the *Deployment Service* provides a uniform mechanism to deploy measurement probes and services. According to Figure 3 the user uploads a deployment package and a deployment configuration (1), using the *Interface Service* (2-3). To address the high variability, the deployment package can assume several formats such as python scripts or web application archives. Depending on the format, the required interpreter or application server is automatically detected and a corresponding container image is created (4) and uploaded to the repository (5). The image creation utilizes Dockerfiles to be able to reproduce the image if the service needs to be redeployed due to changing conditions.

The decision-making can be influenced by providing a deployment configuration. Depending on the requirements, a configuration can define the desired deployment algorithm or can specify thresholds for specific node criteria. This information is passed to the *Decision Making Unit* (6), which applies and parameterizes the deployment algorithm. To cover current conditions and historical data, the *Decision Making Unit* makes use of the measurement data persisted in the *Management Database* (7). Due to the high diversity regarding deployment criteria and the heterogeneous technologies, combining Cloud and Fog requires to adapt existing and develop new deployment mechanisms. To investigate this research topic, the infrastructure can apply different algorithms (8), including basic round-robin scheduling, threshold-based decision making, prioritized list of weighted distribution criteria or more complex mechanisms such as rule-based or model-based machine learning approaches. Regardless of the deployment algorithm used, the architecture provides a consistent interface for initiating the deployment. The uniformity allows to investigate new kinds of deployment algorithms, without changing the infrastructure.

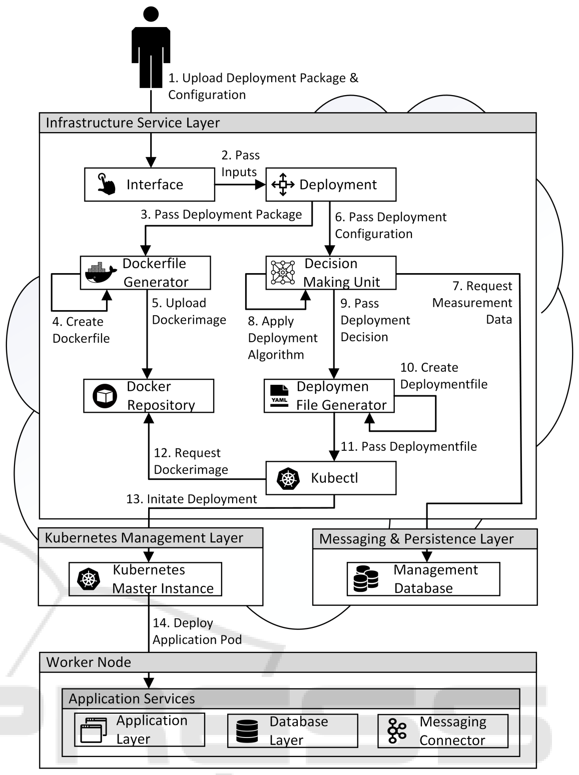

1. Upload Deployment Package & Configuration
Infrastructure Service Layer
Interface
2. Pass Inputs
Deployment
3. Pass Deployment Package
6. Pass Deployment Configuration
Dockerfile Generator
Decision Making Unit
7. Request Measurement Data
4. Create Dockerfile
5. Upload Dockerimage
8. Apply Deployment Algorithm
9. Pass Deployment Decision
10. Create Deploymentfile
Docker Repository
Deploymen File Generator
YAML
11. Pass Deploymentfile
12. Request Dockerimage
Kubectl
13. Initate Deployment
Kubernetes Management Layer
Kubernetes Master Instance
Messaging & Persistence Layer
Management Database
14. Deploy Application Pod
Worker Node
Application Services
Application Layer
Database Layer
Messaging Connector

Figure 3: Deployment Architecture.

Data-driven applications usually include utility tasks that need to be included in each worker node. To match this requirement, two deployment types can be chosen. This includes the default case in which one service is deployed to one specific node or a deamonset. A deamonset is deployed to any node in the cluster. The deployment decision is passed to the *Deployment File Generator* (9), which is responsible for creating a *YAML* file including the deployment type, network configuration and authorization information (10). With this file, the *Pods* are created by the *Kubectl* service using the repository (11-12) and automatically deployed on a specified node (13-14).

5 EVALUATION

The following section evaluates the architecture by using the scenario-based software architecture anaysis method. This includes an example containing monitoring probes and a basic deployment algorithm.

According to Figure 4 the example includes a subset of an automated manufacturing process. The process starts with an incoming order. Based on the order

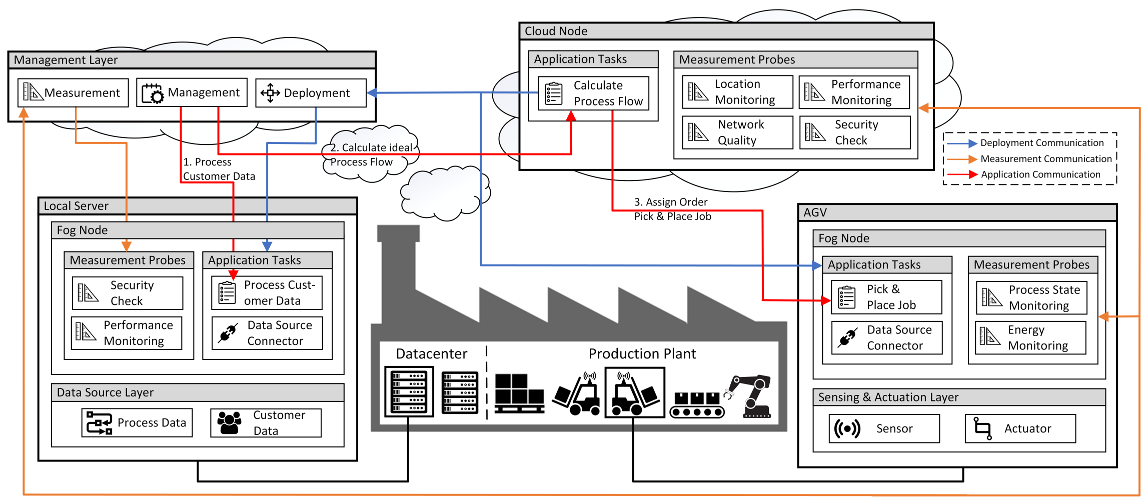


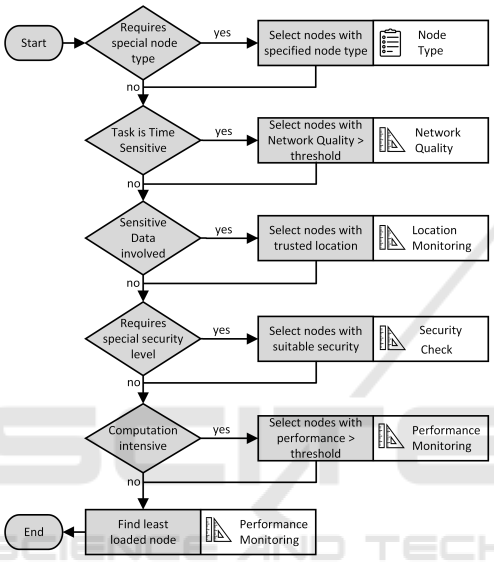
Figure 4: Application Example.

the industrial feedstock is picked and conveyed to the manufacturing areas by using Autonomous Guided Vehicles (AGV). Each AGV contains a processing unit that is representing a mobile Fog node. The production plant contains a local datacenter that is considered as a Fog node. To provide further computation capabilities the plant is connected to a public Cloud.

Develop a Industry 4.0 Service (R1-R2). The example includes three services with individual requirements (R1). The order information, including sensitive customer data, is processed by the *Process Customer Data* service. The requirement of this service is a high security level (e.g. data locality, encryption) to ensure data privacy. Based on the order the process flow is calculated using the *Calculate Process Flow* service. This service aims to optimize the pick order by performing comprehensive calculations. Thus, strong computation capabilities (e.g. CPU, Memory) and a appropriate network connection (e.g. bandwith, latency) are required for fast response time. The AGVs use the results from the previous services and start operation, controlled by the *Pick & Place Job*. To ensure a efficient process the services need to be deployed in regards to the process state of the AGVs (e.g. current location, loaded goods) and their energy level (e.g. remaining battery power). The tasks are orchestrated by the *Management Service* (R2).

Integrating Measurement Probes (R3-R6). In the following it is described which measurement probes are utilized to fulfill the requirements (R3). To be able to choose the most suitable Cloud node, each Cloud node contains four measurement probes responsible for monitoring the location, network quality, security level and performance (R4). These measurement probes are made available by the infrastructure and linked to the node type. The Fog nodes located in the datacenter contains two measurement probes, performance metrics and security checks (ensuring an encrypted communication channel). The performance metrics are determined by using the *Metrics Server* (R4). The mobile Fog nodes include two measurement probes, monitoring the battery level and locate the vehicle inside the production hall to determine the actual process state. To integrate a new probe, an executable need to be created using the infrastructure framework and its blueprint implementations (R5). In this example, the implementation consists out of requesting a GPS modul and keep track of the process state by interpreting the current and subsequent pick orders. Using the *Interface Service*, the executable is uploaded and a corresponding *Pod* is created (R6).

Integrate Distribution Algorithms (R7-R8). The distribution can be categorized into load balancing between similar nodes and choosing the preferred node type. Whereas the *Pick & Place Job* is linked to a specific node type (Mobile Fog Node) and only load balancing is required, the *Process Customer Data* and *Calculate Process Flow* services can be deployed to a Cloud or a Fog node. To illustrate decision-making, we use a rule-based algorithm for each service (R7). Figure 5 shows the decision-tree. For instance, the *Process Customer Data* doesn't requires a special node type, is not time sensitive, but involves sensitive data. Therefore the location monitoring measurement probe is used to choose a node that is located in a trusted area (e.g the local plant). The further decisions are made on the resulting subset. The service also requires a node that provides an encrypted communication channel. This is validated by using the security check measurement probe. Afterwards, a least loaded algorithm is used to determine the most suitable node within the resulting subset. Using the blueprint implementations, it is straight-forward to implement a distribution algorithm (R8). The framework predefines distribution logic and provides the functionality to request the measurement data. In the example, it is only required to implement nested conditional statements that create a set of suitable nodes. Afterward, this set is passed to a predefined least-loaded algorithm. Using this approach existing blocks can be combined or enhanced with additional distribution logic.



Start
Requires special node type
yes
Select nodes with specified node type
Node Type
no
Task is Time Sensitive
yes
Select nodes with Network Quality > threshold
Network Quality
no
Sensitive Data involved
yes
Select nodes with trusted location
Location Monitoring
no
Requires special security level
yes
Select nodes with suitable security
Security Check
no
Computation intensive
yes
Select nodes with performance > threshold
Performance Monitoring
no
Find least loaded node
Performance Monitoring
End

Figure 5: Example Distribution Algorithm.

6 CONCLUSIONS

This paper investigated how to provide a consistent environment to combine Fog and Cloud Computing using container virtualization. The architecture is focussing on develop new deployment algorithms for distributing services between worker nodes. To be able to do so, this paper described how to integrate centralized and decentralized measurement probes to collect node conditions such as the available network connection, performance and storage capabilities as well as security and privacy checks. Another research question targeted in this paper is to provide uniformity in terms of applying different deployment algorithms and combine services using different technologies.

The architecture enables further research to increase efficiency of data-driven applications, reduce power consumption and overcome the security and privacy challenges. As a next step, we'll integrate further distribution algorithms such as machine learning approaches and integrate additional measurement probes for more complex security checks.

REFERENCES

Bellavista, P. and Zanni, A. (2017). Feasibility of fog computing deployment based on docker containerization over raspberrypi. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, pages 16:1–16:10. ACM.

Chakraborty, S., Bhowmick, S., Talaga, P., and Agrawal, D. P. (2016). Fog networks in healthcare application. In *IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 386–387.

Chang, C., Yang, S., Yeh, E., Lin, P., and Jeng, J. (2017). A kubernetes-based monitoring platform for dynamic cloud resource provisioning. In *GLOBECOM IEEE Global Communications Conference*, pages 1–6.

Cheng, B., Solmaz, G., Cirillo, F., Kovacs, E., Terasawa, K., and Kitazawa, A. (2018). Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, 5(2):696–707.

Jiang, X., Fischione, C., and Pang, Z. (2017). Low latency networking for industry 4.0. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, pages 212–213.

Kubernetes (2018). Metrics Server documentation. https://github.com/kubernetes/community/blob/master/contributors/design-proposals/instrumentation/metrics-server.md.

Mahmud, R., Koch, F. L., and Buyya, R. (2018). Cloud-fog interoperability in iot-enabled healthcare solutions. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ICDCN '18, pages 32:1–32:10. ACM.

Neto, E. C. P., Callou, G., and Aires, F. (2017). An algorithm to optimise the load distribution of fog environments. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE.

Rübsamen, T., Pulls, T., and Reich, C. (2016). Security and privacy preservation of evidence in cloud accountability audits. In *Cloud Computing and Services Science*, pages 95–114. Springer International Publishing.

Tsai, P., Hong, H., Cheng, A., and Hsu, C. (2017). Distributed analytics in fog computing platforms using tensorflow and kubernetes. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 145–150.

Woebker, C., Seitz, A., Mueller, H., and Bruegge, B. (2018). Fogernetes: Deployment and management of fog computing applications. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE.