# Container Based Data Processing Management on IoT Gateways

Prashant Kumar
Department of Electronics and Communication Engineering
Faculty of Engineering, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India

***ABSTRACT:*** ***IoT is characterized by its diversity and heterogeneity with respect to the hardware and software components which shape its systems. This in effect would stress the delivery and upgrading phase of hardware components and their corresponding software elements. In addition, these devices are continuously producing data and transmitting it to advanced-processing central cloud systems, with billions of devices anticipated to be linked in the coming years, which will affect network performance. A recent trend to mitigate this issue is to process these data locally close to their origins at the edge of the network. However, edge devices are likely to be resource-constrained which restricts their ability to perform processing tasks in some cases. This work is part of the effort to provide a solution to enable effective resource management at the edge devices and facilitate the deployment and updating of heterogeneous software modules. The solution proposed is based on containers which are a lightweight virtualization technology. It takes advantage of some container concepts, particularly the concept of orchestration, to allow for task forwarding and resource sharing within the same cluster between IoT gateway devices. Our experimentation showed promising results and revealed the potential of the existing platforms for virtualisation.***

***KEYBORDS:*** ***Containers, Docker, Docker Swarm, Internet of Things (IoT), Resource Management.***

## INTRODUCTION

Nowadays, IoT networks have become pervasive and an integral part of our existence, through the growth of linked objects. IoT networks will connect tens of trillions of devices across the globe by 2020, according to. Such networks are designed to support a wide variety of applications including hospitals, smart homes and automation in the industry. IoT implementations rely on an infrastructure that is distinguished by its complexity in terms of the stack of technology used in different layers of these systems. In addition, IoT networks are composed of device spectrum ranging from low-cost, tiny devices with very limited computing resources to resource-rich central cloud servers [1]. Billions of devices are expected to be connected to the Internet, resulting in the generation of a vast amount of data, mostly to be transmitted to the cloud for processing in traditional IoT architectures; whereas results are needed in some cases to be sent back to the sensor nodes for actuation purposes [1], [2]. This cycle causes the network infrastructure to be placed under tremendous pressure due to the rising volumes of traffic. Furthermore, this is likely to increase the cost of data transformation and degenerate the network performance.

To lower the load on IoT networks, a recent tendency is to process data at the edge of the network close to their resources. By delegating certain cloud activities to distributed edge computers, edge computing is deemed to increase the overall efficiency of IoT networks. Our methodology targets the level of the gateway devices as part of the IoT edge networks in the context of this research review. A significant effort has recently been made to boost their computational and communication capabilities; In this way, allowing them to perform some rational data processing tasks close to their network edge tools. This, in turn, is likely to lighten network traffic, minimizing latency and improving network performance overall. However, in some cases where gateway devices receive data packets that may not be compatible with their limited resources, this could lead to overhead resources, leading to network failures in turn [3]. This could be crucial especially in some situations such as applications for healthcare and surveillance. In addition, the complexity of gateway-connected sensors impedes the process of deploying new services or upgrading existing ones. These issues have been the focus of several research projects to propose a solution for efficient resource management provided by IoT gateway devices, and to develop strategies

for enabling the independent delivery of services on such heterogeneous devices. In this paper our main goal is to suggest a solution that covers both resource management and deployment [5].

Thus, this work aims to provide an efficient framework for resource management by allowing resource sharing between IoT devices as well as simplifying and speeding up the process of deploying services on IoT devices. The solution proposed in this work is based on lightweight virtualization technologies, such as containers from Docker to overcome the challenges of resource limitations and deployment of services. This solution takes advantage of the container technology in terms of deployment which is reusable, easy to duplicate and easy to migrate. Containers are lightweight as regards resource management, practical in terms of resource use. Additionally, container-based solutions may contribute significantly to alleviating heterogeneity issues in IoT environments, as they allow deployment and communication between different software modules regardless of the underlying context. Containers can also be configured to cope with the available resources on the target node. Additionally, container orchestration techniques such as Swarm allow resource sharing between IoT devices and provide means of container communication across virtual networks.

## RELATED WORK

The use of containers has recently become an enticing research subject for industry and academia alike. Some research initiatives have been suggested in this context to assess the feasibility of using containers at the edge of IoT networks, taking into account their limited resources. For example, research studies have been carried out to explore container virtualization technologies and their rule as an efficient means to promote the creation and scale-up of IoT applications. To assess the effect of containers on the resources of IoT systems, IoT frameworks were developed and deployed in the form of Docker containers. Some assessment criteria, such as service deployment and management, re-source consumption, and fault tolerance, were considered. Results show good efficiency in using Docker containers. However, no specific method for effective management of resources is defined [4], [5]. Though in an IoT sense, Docker brings advantages for resource management, it is just a working tool. Therefore, an appropriate method for this reason needs to be identified. We are investigating some recent work as compared to the approach presented in this study.

A data processing solution architecture is proposed in, it is edge oriented and its functional components are designed as Docker reusable containers. This facility provides a flexible atmosphere for IoT applications and allows for the modular provision of distinctive system and data management resources as well as orchestration. Although their implementation is calculated by evaluating the use of CPU, Memory, and Network, there is a lack of practical evaluation of the orchestration and data processing modules, which is considered extensively in our work. In order to increase the use of resources offered by IoT devices and reduce the generated network traffic a container-based resource allocation model was proposed. The proposed approach allows various applications and users to dynamically distribute the resources provided by IoT apps, thereby optimizing data processing at source level, rather than sending them to cloud. The feasibility of this method has been tested in terms of performance, as IoT devices are resource constrained. The case study considered in this work, however, focuses on presenting the approach's viability and not on improving the efficiency of resources such as the CPU and memory [12]. In support of autonomic data stream processing applications on the Fog layer, a resource management and orchestration approach based on containers is proposed. Fog nodes (FNs) are fitted with three main components, namely Fog Node Controllers (FNCs), Devices, and Application Controllers (ACs). Apps are encapsulated as docker containers; they each run an AC. The AC communicates with the corresponding FN's FN to allow the scaling up / down of the collection of resources allocated to the Docker container (e.g., number of cores, CPU power, and bandwidth). In contrast to our research, their implementation relies on rich-resource infrastructure and does not take into account situations where the Fog node is resource-limited, as is the case with a large portion of IoT applications.

## THE PROPOSED APPROACH

A solution based on container virtualization technology for addressing system limitation and deployment issues has been proposed in this study. This approach relies on the concepts of certain container clusters

to promote the sharing of resources between devices and enable the hosting infrastructure to be deployed easily and independently. IoT topology is usually built at the top of three critical layers, namely sensor nodes, gateways and the cloud. The sensor nodes continuously send the data to the gateways where it is stored and processed, and then send it to the cloud for use as needed. Mostly, the gateway collects data from several sensor nodes, and some gateways are overwhelmed due to IoT system resource limitations and can no longer process data. Though it is likely to find some gateways inside the same cluster that are not overloaded gateways and have some resources available. As a consequence, the work on re-source management is aimed at the distribution of workload between the gateways which are the central element in our approach.

*Overall Architecture:*

They will be grouped into clusters to completely take advantage of resources available on IoT gateway computers. Each cluster consists of a number of gateways with one as manager, while the remainder are workers. As suggested by its name, the manager node will take care of managing the other worker nodes, as they can only use their resources to implement the services for the other gateways. That gateway is equipped with a container system and its orchestration mechanisms which are responsible for deploying and communicating services within the same cluster. This decomposition of the framework would allow the gateways belonging to the same community to share the resources and to communicate via a virtual network. Typically each gateway receives data from one or more sensor nodes, but in the case of overload, the data processing may be transferred to another gateway within the same cluster in order to spread the load equally according to our approach. Gateway of the system will contain three services, namely reading service, processing service, and sending service, which constitute the data processing workflow coming from sensor nodes. To facilitate the deployment of services on the gateways, their images are created with all their dependencies (libraries, version, etc.) and stored in a registry, the latter being a centralized service directory shared among all the gateways. In order to deploy a service in a gateway, we only need to access the registry and draw the corresponding picture for the program. With such a procedure, the problem of gateway heterogeneity could be overcome, as the services are created with all their dependencies and are ready to be executed in containers regardless of the host's infrastructure. A centralized repository would also simplify the upgrade, as users just need to download a new image construct instead of going through all the gateways and updating it.

Figure 1 shows our system's high-level architecture, details of the critical components will be given in the following sections. The gateways are grouped into a series of clusters as illustrated. There is a man- ager gateway (in green) in each cluster which manages the other gateways. The gateways are communicated directly by the sensor nodes (in red). The same cluster's gateways interact with each other over an overlay network.

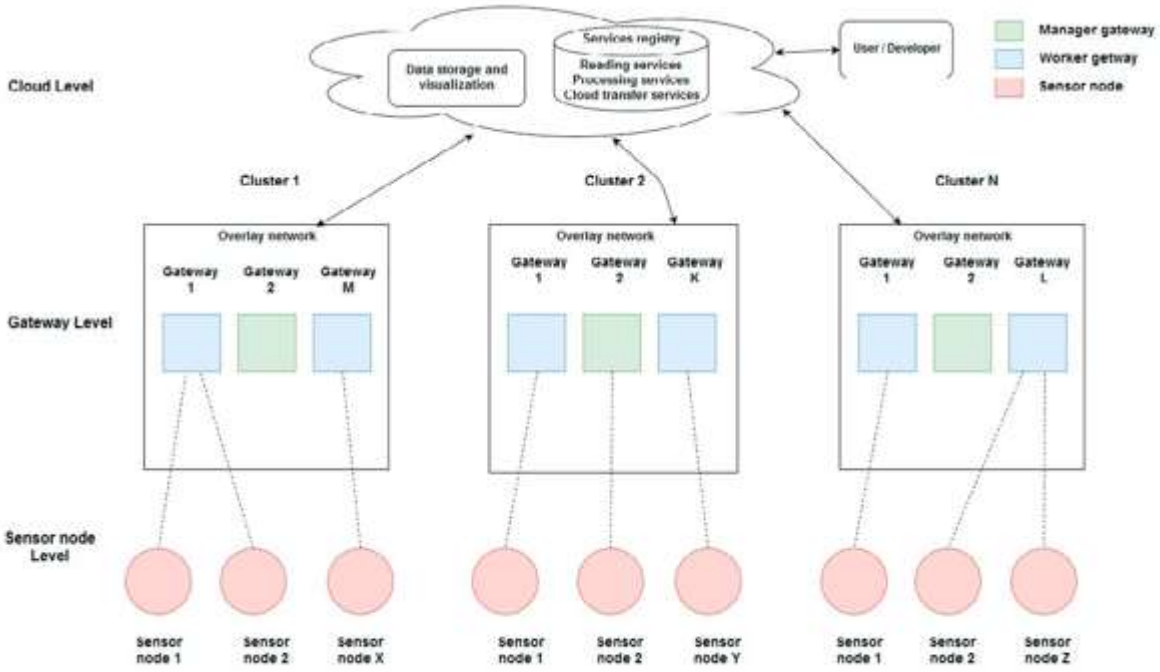


**Figure 1. The Overall Architecture**

*Gateway level:*

Throughout this job, the gateway is given some importance since it represents the core element of our solution. A gateway may be either a boss or a worker, but all of them share the same architecture. The only difference is the advantage assigned to the manger gateway; so commands can be executed. Each gateway has a container engine that allows services to operate as packaged containers. The engine chosen must be able to support composition of the cluster. Each gateway is also equipped with a load balancer, the latter allowing the load to be distributed on the same service instances located in the same cluster gateways. A gateway comprises three services that operate continuously as containers and constitute the workflow from the sensor nodes to data processing. Those facilities are:

Data Read Service: This service receives data packets from sensor nodes at a predefined time interval; it will then store them in a specified volume directory shared between the services deployed on the gateway. Received data packet includes SensorID, Calculated value, and Timestamp.

Data Processing Service: This service communicates with the volume directory to access the data obtained and to execute user-requested data processing tasks. In our case, we find the semantic annotation as the method of processing as set out in Section 4.

Cloud data transfer service: this service's main objective is to promote contact between the level of the gateway and the cloud. This contact aims to allow for the exchange of messages between the two levels. It includes tasks such as sending processed data to the cloud and receiving Cloud requests for deployment and updating.

*Cluster Functionality:*

Figure 2 demonstrates the distribution of requests in a cluster consisting of three gateways. An IP-address identifies each gateway. As we have described in the gateway architecture, the data handling workflow is formed by three services. Because the data reading service is the first component of the data processing services chain, it is implicitly realizable that the distribution of reading tasks will inevitably result in a decrease in the amount of data to be handled individually on each gateway in the following services. When the device is initialized, the gateways start collecting data from the sensor nodes. The load balancer at each gateway receives requests from the sensor nodes, then transfers them to the same gateway's data read service, or redirects them to some other gateway's read service case. This task divide is based on an overlay network.

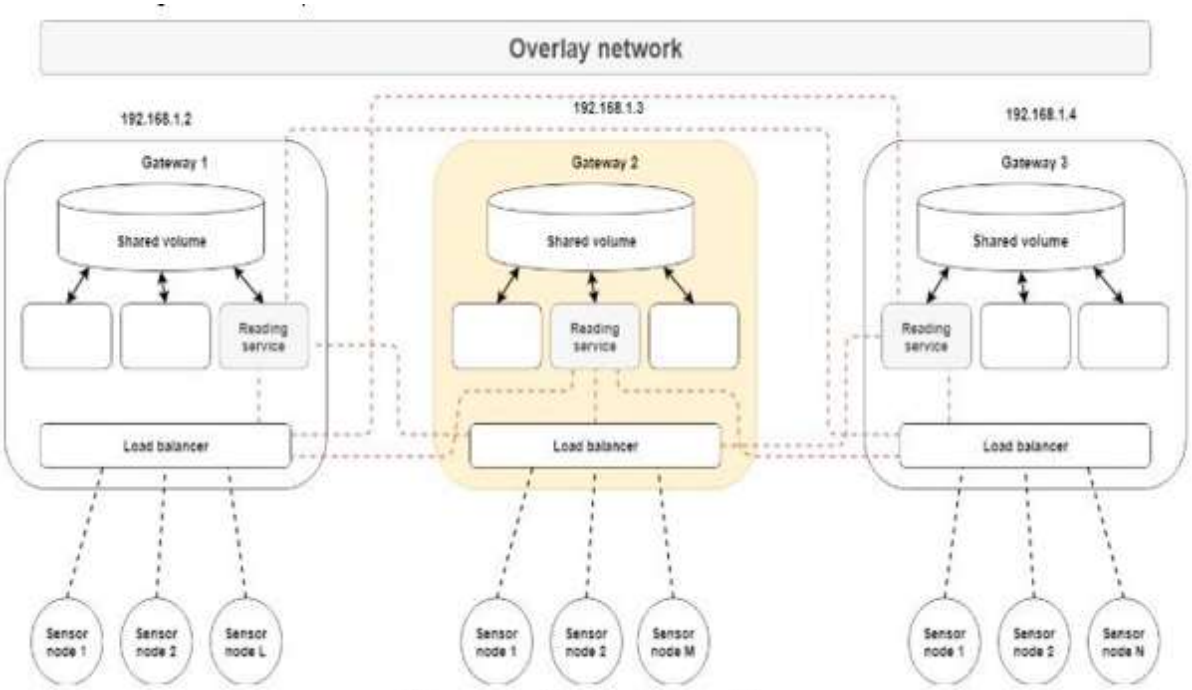


**Figure 2. Cluster Operation.**

A gateway's data processing service accesses the shared volume to retrieve the data supplied by the same gateway's read service, then it processes it and reinserts it into the shared volume. The cloud data transfer service then accesses the shared volume to get the data processed by the same gateway's data processing service, and then transfers it to the cloud. Thus, in the case of the IoT framework which consists of clusters of gateway devices, each cluster will function independently following the above steps. Getting a balanced structure is the goal behind this division of tasks; thereby eliminating circumstances where certain gateways are overloaded; while others are fairly unloaded. The next section tells us how this approach affects gateway capital.

## CASE STUDY

To test the viability and efficacy of our resource management strategy, we have chosen a case study in the IoT context where data processing is very resource-intensive. In this job, the semantic annotation is treated as the function of data processing. It involves adding some concrete definition to the raw data collected by different sensor nodes; thus, different machines and applications become easily consumable. In addition, the annotation process promotes interoperability between different IoT systems relying on a collection of heterogeneous sensors to collect the data. This contributes significantly to the development of new technologies irrespective of the technology underneath which leads to the mitigation of the problems of vertical IoT data silos.

We also discussed issues relating to the data annotation process in the sense of IoT in our previous works. The key problem that has been explored is the method of annotation which is too resource demanding; this is not ideal for IoT devices characterized by restricted resources. The first cause of this high resource use is the use of Semantic Web technologies, commonly adopted for modelling and integrating data from various sources on the Web. These technologies are considered heavy for IoT purposes, because they often require considerable computational power. This situation may get worse in situations that require a large number of sensors (data sources), as is often the case with IoT applications like the smart city. The key goal of our previous research falls within the annotation process's attempts to reduce resource consumption. However, this minimisation is extended to each gateway independently. As a result, some gateways will obtain greater volumes of data than others, which means that the data processing tasks consume more power. Therefore some gateways would no longer be able to process data from the sensors in such a situation over time. Having other gateways accessible in terms of resources may be an alternative in leveraging their resources to solve this problem. This then highlights the need to use a load balancing strategy to fairly allocate the annotation function between the various gateways within the same cluster.

The implementation of our container-based resource allocation (i.e. resource management) methodology will be seen and evaluated in the following section.

## IMPLEMENTATION AND EVALUATION

We used Docker technology and its Docker Swarm definition to implement the case study mentioned in Section 4 to determine the effect of our approach on the resources IoT devices offer. Docker is used to build the containers, while Docker swarm is used to orchestrate the Docker-containing devices by creating Swarms. Table 1 demonstrates the mapping of the principles of architecture and of the technological concepts of implementation:

**Table 1. Mapping Between Architectural and Technical Concepts.**

| Architecture concept | Technical concept |
|---|---|
| Container | Docker container |
| Cluster | Docker swarm |
| Manager | Manager node |
| Worker | Worker node |
| Service registry | Docker Hub |
| Overlay network | Ingress network |
| Container engine | Docker container engine |
| Load balancer | Docker swarm load balancer |
| Service | Python program with its dependencies in a container Docker. |

*Implementation and configuration:*

To check our method, we conducted experiments using five virtual machines spread on two physical computers, configured as follows:

A Linux operating system Ubuntu 18.04 and Docker Engine 18.01 is mounted on each of the virtual machines. A Docker Swarm is used to build a cluster of five virtual machines; one of them is called a manager node (VM5), while the rest are staff. A network of overlays was built between the Swarm machines to allow them to interact in the same cluster. Three services were installed on each server, including a data reading service, a data annotation service and a cloud data sending service. A volume has been generated on each node to allow data sharing among services. A Python program was created to simulate the process by which sensor nodes generate and transmit data.

The tests were carried out on two conditions. Firstly, the simulation programs were designed to send data only to VM1 and VM2 with a specified load of data and without using the method. Although the same data load was placed on VM1 and VM2 in the second example, but using our method. Other systems were also deployed to capture and store output data for each (i.e., CPU, RAM, and Network metrics used). And in the first case of the some modifications were made to initialize the system, and others were made to allow the export of performance data from different machines.

## RESULTS DISCUSSION

We will only present the results of VM1 and VM3 due to page cap, as they should have a different behaviour in both experiment scenarios. Presentation and interpretation of the findings are divided into subsections; each reflects a metric of the assessment.

*CPU:*

Since the data load is performed only on VM1 and VM2, these two machines process all the data without spreading it to other machines before using the method. After using our method, the CPU consumption of VM1 decreased, because the data is spread over all devices in this scenario, so VM1 processes less data. The data are handled only by the VM1 and VM2 before using the method, so we note that there is only a

small consumption of the CPU returning to the monitoring services. The CPU usage of VM3 increased after using the solution, because it participates in data processing in this case.

*Memory:*

Monitoring services produce the largest memory usage, and are resource-intensive. These resources are mainly used to gather performance measurements for the purposes of the assessment and will not be part of the final program. VM1 resource usage marginally decreased after the approach using the solution, since data processing is spread to all devices within the cluster. Conversely, shortly after using the method, the memory usage of VM3 increased, as it is involved in processing data from sensor nodes. Data processing is done on VM1 and VM2 only prior to using the method.

*Network:*

There is a constant network traffic arising from the continuous data received by the monitoring services. In the first scenario the network traffic of VM1 containers is higher, as in this time all data load was implemented only on VM1 and VM2, while in the second scenario traffic decreased, because the load was spread to all Swarm devices. Throughout the experiment's first scenario, VM3 network traffic remained constant, as all data from the sensor nodes are processed by VM1 and VM2 only before using the method.

## CONCLUSION

Edge devices have transmitted major benefits to IoT networks by taking the burden of performing certain cloud activities, thus alleviating the workload placed on the network. However, their existence has placed new problems around resource management and service delivery in terms of resource constraint and the variety of technologies employed. Our research studies in this field have led us to propose a solution focused on lightweight container-based virtualization technologies to enable efficient resource management and delivery of services on IoT gateway apps. The deployment of containers renders this operation independent of the infrastructure of the host, as well as containers promote the upgrade process as they allow changes to their corresponding images stored in a centralized directory. It introduced the architecture of the proposed solution, its key components and their functionalities. To validate the efficacy of the approach in managing system resources, the results of an experiment that showed the values of a few metrics with and without using the approach were presented. Implementing the method is based on Docker system principles and the container formation and orchestration process Swarm. The results obtained show a decrease in use of CPU, memory and network traffic by the gateways as requests from sensors are spread among all the gateways that make up the cluster, resulting in a balanced system that can process as much data as possible. However, the network activity of the cluster is increased, as the separation of tasks between the cluster's gateways generates new communications. To evaluate the solution's true efficacy, it would be important to apply it on a wide scale in the industry's actual situation. It would also be useful to expand the solution with rules for request distribution to mitigate this operation and thereby reduce network traffic on the cluster.

## REFERENCES

[1] F. Ramalho and A. Neto, “Virtualization at the network edge: A performance comparison,” 2016, doi: 10.1109/WoWMoM.2016.7523584.

[2] R. Morabito and N. Beijar, “Enabling Data Processing at the Network Edge through Lightweight Virtualization Technologies,” 2016, doi: 10.1109/SECONW.2016.7746807.

[3] M. Helmy, E. S. M. El-Rabaie, I. M. Eldokany, and F. E. A. El-Samie, “3-D Image Encryption Based on Rubik’s Cube and RC6 Algorithm,” *3D Res.*, 2017, doi: 10.1007/s13319-017-0145-8.

[4] T. Kondo, H. Watanabe, and T. Ohigashi, “Development of the Edge Computing Platform Based on Functional Modulation Architecture,” 2017, doi: 10.1109/COMPSAC.2017.108.

[5] V. Shubha Rao and M. Dakshayini, “Priority based optimal resource reservation mechanism in constrained Networks for IOT applications,” 2016, doi: 10.1109/WiSPNET.2016.7566332.