Software Deployment and Self-adaptation of IoT Systems

Iván Alfonso[1][0000−0003−3940−0629]

Department of Systems and Computing Engineering, Universidad de los Andes, Bogotá, Colombia
id.alfonso@uniandes.edu.co

Abstract. Traditional software engineering methods bring challenges to the construction of IoT systems. In particular, software deployment in distributed systems is not trivial. Resource limitations at the edge/fog layer nodes and heterogeneity of technologies and communication protocols are some of the challenges that can generate failures in deployment and operation. It is necessary to make intelligent decisions on the location of the deployed software in the system infrastructure to favor the quality of service (QoS). Additionally, the operation of IoT systems with distributed architectures becomes complicated due to the dynamic environment and the events produced that can impact QoS. We present Ph.D. Thesis proposal for intelligent decision making of software re-deployments in the edge/fog layer devices of a distributed IoT system, and a method for self-adapting the system architecture.

Keywords: Internet of Things · Software deployment · Self-adaptive architecture · Edge computing · Fog computing.

1 Introduction

About twenty years ago the term Internet of Things (IoT) was publicly used by Professor Kevin Ashton [2]. From that moment, the interest around IoT has been growing and a lot of startups and companies working in this area have emerged. Gartner Inc. envisions that by 2020, about 20 billion IoT endpoints will be in use [4]. However, this growth in the use of IoT introduces challenges in software development and systems operation. Designing IoT systems using traditional architectures and software development methodologies present problems such as slow upgrades, security issues, increased costs, low performance, and reduced platform reliability [10].

Traditionally, software engineering has addressed these issues by introducing new architectures, techniques, and development methodologies that have been adopted by vendors and software development teams. For example, DevOps methodology promises to improve the performance of work teams and streamline the tasks of software development and operation of the system. DevOps combines development and operation in a single process consisting of several stages in a loop. In the literature, the use of DevOps predominates in web development mainly in SaaS applications, but there are few advances in the use of this methodology in the domain of IoT. Furthermore, there are some problems to adopt DevOps for IoT systems. For example, the complexity and limitations of creating test environments due to heterogeneity of device hardware [7]. Also, the complexity to remotely deploy software on tiny devices that do not support containerization or have no Internet connection [11], and the complexity to deploying and orchestrating software on systems with distributed architecture. Additionally, in the operation state the dynamic environment in an IoT system induce changes that impacts quality of service (QoS). In this research, we propose an approach to deploying and orchestrating software on IoT systems with distributed architectures, as well as a strategy for self-adapting the architecture to dynamic events that affect QoS.

The conventional IoT architecture consists of three main layers: (1) the cloud layer, which has unlimited processing and storage resources, hosts the application logic and perform the processing tasks; (2) the device layer which contains the system devices such as sensors and actuators; (3) and the network layer –composed of several types of communication technologies– which enables communication between physical layer devices and cloud layer servers. This centralized architecture reduces maintenance costs and application development efforts, but it has limitations such as low latency response, security risks, and high bandwidth consumption [5]. Distributed IoT architectures have emerged to address these problems. Edge and fog computing are technologies introduced as layers in a decentralized IoT architecture. Both technologies offer advantages in terms of latency, security and bandwidth usage. Because these allow move computation, storage, communication, control, and decision making closer to the network edge where data is being generated. This research addresses distributed architectures for IoT systems, to take advantage of edge and fog computing.

2 Related Work

In order to identify and to analyze relevant studies, we conducted a Systematic Literature Review (SLR). The SLR helped us systematically reach a comprehensive and fair assessment of the topic. The main goal of the SLR was to identify the dynamic environmental events in the physical and edge/fog layers of an IoT system that could impact its QoS. In addition, we classified the strategies to achieve this self-adaptation and we identified weaknesses.

After the screening and study quality assessment stages, we obtained 20 relevant studies to the research. We found six dynamic events that impact the QoS of the IoT system: (1) mobility client, (2) dynamic data transfer rate, (3) dynamic network connectivity, (4) attacks from the traffic sensor, (5) failures and software aging, and (6) important event detected by sensors. Some of the least explored events by the community are 5 and 6 (failures and software aging and important event detected by sensors). For example, studies that addressed *important event detected by sensors* –as [16]– proposed an IoT architecture that detects a physical event (e.g., when it starts raining) and changes the priority of the sensor data that is sent for analysis in the cloud. This dynamic architecture was designed using Flow-Based Programming (FBP), and the adaptive decisions were made by analyzing the data in an edge layer device. [14] proposed an architecture for a Video Surveillance System that horizontally scales virtual machines deployed in fog layer nodes when emergency surveillance was detected by the cameras. This architecture was implemented in an OpenStack based virtualization environment using Network Functions Virtualization (NFV) and Software-Defined Networking (SDN). Studies that addressed the issue of *failures and software aging* –as [15] and [9]– propose strategies for software deployment in the edge and fog layers of the architecture. For instance, [15] proposed Foggy, a framework used to automate application deployment in Fog Computing architectures. Foggy was based on the use of Docker containers and deployment rules to facilitate dynamic resource provisioning.

We found four techniques or strategies used to address these dynamic events: (1) data flow reconfiguration, (2) auto-scaling of services and applications, (3) software deployment and upgrade, and (4) task prioritization in edge/fog nodes. Some of these strategies require tasks of deploying, updating, or moving software artifacts on the edge/fog layer nodes and devices. Most of the analyzed studies semi-automate these tasks but the allocation decisions in the nodes are taken by DevOps engineer. Intelligent deployment and location decisions must be made by analyzing the current and historical metrics information about nodes resources consumption and QoS.

We propose an approach for smart re-deployment in distributed architecture IoT and to self-adapt the system to two dynamic events that impact the QoS (failures and software aging and important event detected by sensors). In contrast to these studies, we are interested in a method of monitoring and logging metrics about the node resources consumption and QoS. One of the components of our proposed solution analyzes this information to take intelligent re-deployment decisions.

3 Problem Statement

Most of the strategies we found in the literature to adapt the architecture IoT systems require deploying, updating, or moving software artifacts between nodes of the three layers: physics, edge/fog, and cloud. However, managing software deployments on IoT systems poses challenges due to the heterogeneity of communication devices and protocols. Unlike the cloud layer, at the edge/fog layer resources are not centralized and the network environment is heterogeneous, making it difficult to manage resources to balance the load and dynamically scale [3]. In addition, edge nodes have limited computational resources unlike cloud servers. Because of these reasons, the orchestration strategies and tools currently used in the cloud must be adapted to work together between the edge/fog and cloud layers.

We address two dynamic events mentioned in the SLR chapter 2:

Failures and software aging IoT systems require frequent software deployments, software updates, or movement of software artifacts, either to solve problems, improve performance, improve system security, or expose new services to the end-user. However, automating software deployment in IoT systems is a challenge given the heterogeneity and the large number of devices that must be handled. Intelligent allocation decisions are crucial to efficiently deploy software at the edge/fog layer of the system [13]. Kubernetes[1] (an open-source tool for automating the management of containerized applications) has provisioning capabilities, but these only consider the number of requested resources (CPU, RAM) on each node. Other factors such as time, energy consumption, network latency, reliability, bandwidth usage, and mobility should be taken into account when choosing the nodes or the best location to deploy.

In addition, the update process may fail, or the software displayed may have errors. One of the basic solutions, when an upgrade error occurs, is to roll back to an earlier stable software version. However, there are deployment patterns primarily used in web development to improve reliability and reduce the probability of failure in a production environment (e.g., rolling deployment, blue-green, or canary) [1]. These patterns must be explored to facilitate the process of deployment and to give reliability to software deployments on IoT systems.

Important event detected by sensors There are changes that can be detected by system sensors that force the reaction or adaptation of the IoT system architecture. For example, when the cameras of a video surveillance system detect motion, the video is transferred to fog nodes or cloud servers in real-time to be processed. Data sent from devices to application servers increases, therefore the bandwidth consumption and response latency also increase, and the QoS of the system will be affected. The IoT system should detect and adapt the architecture to support these events, e.g. scaling the application containers to meet the increased load.

According to the identified problems, we propose a general research objective: to propose and develop an approach for smart software re-deployments in IoT systems and the self-adaptation of the system architecture in response to dynamic events detected by system sensors. We propose four sub-objectives: i) to provide a mechanism to smart redeploy and orchestrate software in a distributed architecture IoT systems; ii) to provide a mechanism to monitor and log metrics in IoT systems; iii) to design (or reuse if existing) strategies to self-adapt the IoT system architecture when a deployment failure or event is detected by system sensors; and iv) to validate our approach by simulating a realistic case study and evaluating the response to the dynamic events addressed.

In order to achieve these goals, we define three research questions and two sub-questions: (1) How to take intelligent allocation decisions to redeploy and orchestrate software containers in distributed architecture IoT systems? (1.a) Which metrics should be analyzed to make intelligent allocation to software re-deployment? (1.b) How to implement deployment patterns in a distributed architecture IoT systems? (2) How to adapt the architecture of an IoT system

[1] https://kubernetes.io

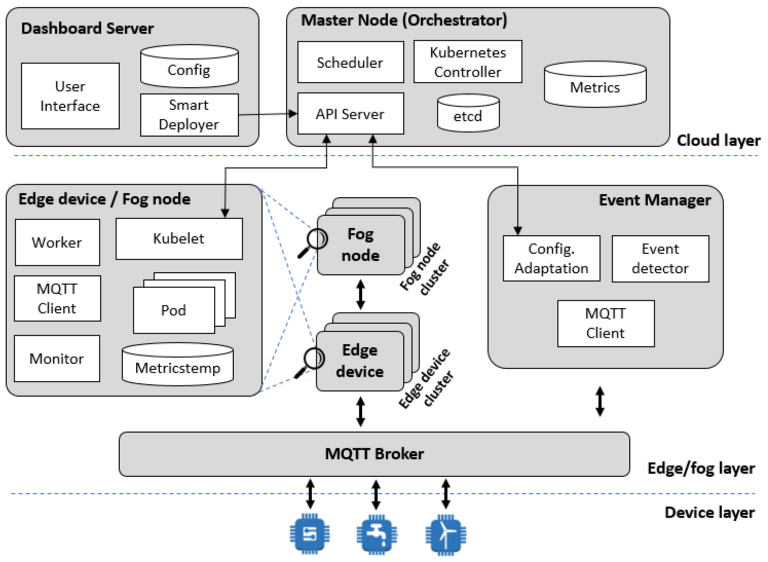

Fig. 1. High level architecture

to ensure QoS in response to dynamic events detected by system sensors? and (3) How to monitor and log service quality metrics for an IoT system?

4 Proposed Solution

We propose an approach that has three main contributions: (1) it allows developers to deploy, manage, and monitor software version deployments on edge/fog layer; (2) suggests intelligent re-deployment decisions after analyzing metrics about node resources consumption and QoS; and (3), analyzes the sensor data to detect dynamic events that induce adaptations in the architecture.

We designed a Kubernetes-based approach to the deployment and orchestration of edge/fog layer devices and nodes. We chose Kubernetes because it is infrastructure-agnostic and allows developers to simplify their DevOps practices by reducing the time spent at integrating with heterogeneous operating environments. The basic control and management unit of Kubernetes is the Pod, which represents the collection of containers and storage (volumes). Figure 1 shows the proposed high-level architecture which is composed of three layers: the device layer contains the sensors and actuators of the system, the edge/fog layer contains devices and nodes to perform tasks on the edge, and the cloud layer contains the deployment orchestrator and the user interface. The main components of the architecture are described in detail below.

Dashboard Server: the *Dasboard Server* provides the user interface and enables the developer or DevOps engineers to perform the following tasks: (1) configure and deploy software versions on the system's edge devices and fog nodes; (2) configure the adaptation rules composed of a stimulus and an action; (3) visualize in real-time the status of the Pods deployed on the nodes. The *Config database* stores the deployment configuration information and the adaptation rules created by the user. When the DevOps engineer configures and runs a new software deployment, *Smart Deployer* component sends the information to the master node to start the deployment. **Master Node (Orchestrator):** the Orchestrator is responsible for the software deployment and the management of resources (nodes). The *Master Node* has the following components: *etcd* is a lightweight, distributed key-value data store that reliably stores cluster configuration data; the *Scheduler* is a component that checks the available infrastructure resources to locate the pods. the *Kubernetes Controller* is a daemon responsible to ensure the desired state of the cluster; the *API Server* is the gateway to the Fog/edge node clusters (this component implements a RESTful API over HTTP); and finally, the *Metrics* component is a time series database responsible for storing the metrics about nodes resources consumption and QoS. We store this information to make an analysis of this data, give valuable feedback to DevOps engineers, and suggest smart re-deployment decisions in future releases.

Edge Device and Fog Node: edge devices and fog nodes make up the node clusters that are managed by the *Master Node*. Each *Edge/fog* node has the following components: The*Kubelet* which connects to the *APIServer* to synchronize information and execute instructions sent from the *MasterNode*; the *Kubelet* ensures that the containers described by the *MarterNode* are functioning and healthy on each node; the *Monitor* component is responsible for collecting and storing the resource consumption metrics of each Pod in a *Metrics temp* database (a lightweight key-value database to temporarily store the metrics); the *Worker* component periodically stores the metric information from the temp database in the Metrics database of the cloud layer; and the *MQTTClient* component is a client capable of interacting with MQTT servers through the publish/subscribe pattern.

MQTT Broker: the *MQTT Broker* is the communication bridge between the device layer devices and the edge/fog layer nodes. The MQTT communication protocol is highly used in IoT systems due to the advantages of the publish/subscribe pattern regarding scalability, asynchronism, decoupling between clients, low bandwidth, and power consumption.

Event Manager: the controller component is responsible for applying the adaptation rules defined by the DevOps engineer. An adaptation rule consists of a stimulus and an action. The stimulus can be detected by analyzing sensor data, for example when a temperature sensor exceeds a limit or when a surveillance camera detects movement. The action is the adaptation of the system in response to the stimulus, such as scaling containers on edge devices, stopping the execution of containers, or moving pods from one node to another. The *Event Detector* component analyzes the sensors data to detect dynamic events. When a dynamic event is detected, the *Config Adaptation* component makes the necessary adaptations to the architecture according to the configured adaptation rule.

5 Methodology and Preliminary Results

Our research began with an exploratory stage, in which we conducted a systematic literature review to understand the needs and challenges in the operation of IoT systems. The SLR is based on the methodology proposed by Kitcheham et al. [6], and it consists of six main steps (planing, search process, inclusion and exclusion criteria, study quality assessment, data collection, and data analysis). For the development stage, we use the Design Science Research methodology (DSR) [8] which has as a fundamental block the design, construction, and evaluation of artifacts. Following the DSR methodology, we designed an initial approach for semi-automating deployment, monitoring and visualization of the impact of software updates on edge devices of the IoT system. We used Docker as the containerization technology and designed an orchestrator to manage software version deployments. We monitored the availability and consumption of computer resources of the edge devices and stored the information in a time-series database. The historical information monitored is presented to the user through metric-centered visualizations. Finally, we experimentally validated our first development using a prototype of an IoT temperature monitoring system and presented the work at MODELS Conference 2019 [12]. After analyzing the results obtained with the first approach, we redesigned the architecture proposed to use Kubernetes because of the advantages in automating the deployment of software, orchestration, and management of distributed systems. We added the components needed to monitor the system, analyze the monitoring data to suggest re-deployment decisions, and make self-adaptations to the architecture. Currently, we are in the construction stage of the software components of the architecture. Shortly, we plan to evaluate the artifacts by making a case study that experimentally validates the built prototypes. After analyzing these results, we will make the necessary adjustments and improvements incrementally until obtaining the expected results.

6 Conclusion and Future Work

We presented an approach for software deployments on IoT systems with a distributed architecture that monitors operation and self-adapts the system architecture. We propose the use of Kubernetes as a tool for deployment and orchestration in edge/fog layer devices and nodes. Pods and containers are periodically monitored and its metrics are logged to analyze them and make intelligent software re-deployment decisions. Our approach allows configuring adaptive rules to support dynamic events detected by system sensors.

Future work is devoted to the development and validation of the architecture components. In particular, the *Config Adaptation* and *Smart deployer* are the components that make the adaptations in the architecture to maintain or improve the QoS of the system. Additionally, we will perform the validation using the test scenarios for the identified case study.

References

1. Ahmadighohandizi, F., Systä, K.: Application development and deployment for iot devices. In: European Conference on Service-Oriented and Cloud Computing. pp. 74–85. Springer (2016)
2. Ashton, K., et al.: That 'internet of things' thing. RFID journal **22**(7), 97–114 (2009)
3. Giang, N.K., Lea, R., Blackstock, M., Leung, V.C.: Fog at the edge: Experiences building an edge computing platform. In: 2018 IEEE International Conference on Edge Computing (EDGE). pp. 9–16. IEEE (2018)
4. Hung, M.: Leading the iot, gartner insights on how to lead in a connected world. Gartner Research pp. 1–29 (2017)
5. Jiang, Y., Huang, Z., Tsang, D.H.: Challenges and solutions in fog computing orchestration. IEEE Network **32**(3), 122–129 (2017)
6. Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering–a systematic literature review. Information and software technology **51**(1), 7–15 (2009)
7. Lwakatare, L.E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H.H., Bosch, J., Oivo, M.: Towards devops in the embedded systems domain: Why is it so hard? In: 2016 49th hawaii international conference on system sciences (hicss). pp. 5437–5446. IEEE (2016)
8. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision support systems **15**(4), 251–266 (1995)
9. Morabito, R., Beijar, N.: A framework based on sdn and containers for dynamic service chains on iot gateways. In: Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems. pp. 42–47. ACM (2017)
10. Motta, R.C., de Oliveira, K.M., Travassos, G.H.: On challenges in engineering iot software systems. In: Proceedings of the XXXII Brazilian Symposium on Software Engineering. pp. 42–51 (2018)
11. Nguyen, P., Ferry, N., Erdogan, G., Song, H., Lavirotte, S., Tigli, J.Y., Solberg, A.: Advances in deployment and orchestration approaches for iot-a systematic review. In: 2019 IEEE International Congress on Internet of Things (ICIOT). pp. 53–60. IEEE (2019)
12. Prens, D., Alfonso, I., Garcés, K., Guerra-Gomez, J.: Continuous delivery of software on iot devices. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 734–735. IEEE (2019)
13. Santos, J., Wauters, T., Volckaert, B., De Turck, F.: Resource provisioning in fog computing: From theory to practice. Sensors **19**(10), 2238 (2019)
14. Wang, J., Pan, J., Esposito, F.: Elastic urban video surveillance system using edge computing. In: Proceedings of the Workshop on Smart Internet of Things. p. 7. ACM (2017)
15. Yigitoglu, E., Mohamed, M., Liu, L., Ludwig, H.: Foggy: a framework for continuous automated iot application deployment in fog computing. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS). pp. 38–45. IEEE (2017)
16. Young, R., Fallon, S., Jacob, P.: A governance architecture for self-adaption & control in iot applications. In: 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT). pp. 241–246. IEEE (2018)