Toward the generation of deployable distributed IoT system on the cloud

Habeeb Ahmed Hassan, Rawaa Putros Qasha
College of Computer Science and Mathematics University of Mosul, Iraq

*habeeb_ahmed@student.uomosul.edu.iq

Abstract. The Internet of Things contains a number of smart devices and tools that collect and communicate sensing data, with different processing services. The diversity of IoT devices and tools in terms of the resources and the required software raises DevOp challenges for the provisioning of IoT applications. In addition, the IoT system requires a number of heterogeneous computations on the devices which form altogether with the diversity of the IoT devices new issues and challenges for the deployment of the distributed IoT applications. In this paper, we propose a new approach to generate a deployable model for the distributed IoT systems that is based on simplified, user-friendly declarative description of the smart devices’ communication, configuration, installation and computation with the IoT system parts. Our approach uses the Ansible-based YAML description for modelling and implementing the deployable IoT system on different infrastructures. The aim of this work is to minimize the efforts required to deploy the distributed IoT applications on various infrastructures including the Cloud by utilizing a simple description of the application and system components of the IoT application and environment. We motivate our work with real use case scenarios including a number of sensors, Raspberry Pi, local machine and AWS Cloud instances to show its feasibility and validity. Finally, an evaluation of our proposed approach is presented by conducting a number of experiments.

1. Introduction
The Internet of things plays a pivotal role in our daily life through network- controlled electronic devices. With the increase in the network spread and the continued appearance of various types of smart sensors with their low cost and the general trend of storing huge data in the cloud [1], it became necessary to find ways to develop and deploy Internet of Things applications and their required tools and libraries [2]. The Internet of Things comprises several smart devices and techniques that perform and communicate a number of run-time environments and process different kinds of data. The diversity of IoT infrastructures and the heterogeneity of their computational components, together with the scaling feature of the IoT systems rises challenges for the deployment of IoT applications [3]. This research provides an integrated framework to cope with these challenges where the process of the deployment of the IoT applications can be performed in an efficient and flexible way.

In this paper, we present an approach that facilitates the modelling of a distributed IoT system and an automated application deployment in various run-time environments. Our goal is to minimize the developer’s effort to deploy the distributed IoT application run on a number of distinct infrastructures including local devices and the Cloud while providing and utilizing a simple description of the
application and system components of the IoT application and environment. The main contributions of the work presented in this paper are:

1. Analyzing the common features of IoT applications and determining their requirements.
2. Providing modelling for the components, tools and application parts of a distributed IoT system based on the Ansible YAML description language including its features, configuration, communication and installation information.
3. Implementing the framework that manages and deploys the developed IoT applications on various infrastructures including Cloud with minimal efforts using Ansible by utilizing the IoT system modelling.
4. Conducting experiments to show the feasibility and validity of the proposed approach.

2. Background

In this section, a common architecture of the IoT systems will be presented to support the distributed IoT applications. Then, we will highlight the smart devices, tools and the run-time environment used in the work presented in this paper.

2.1 IoT Architecture

In this work, the common IoT architecture consists of three tier infrastructures. Such architecture combines various heterogeneous components including sensors, smart devices, computing resources (local PC and Cloud), and networking devices [4]:

a. In the first tier, sensors and actuators are used to interact smart devices with their environment. Smart devices can be static such as Raspberry Pi or mobile to which the sensors are connected to gather the sensed data.

b. In the second tier, the infrastructure manages the connections between the smart devices, gateways and the Cloud. This tier includes the necessary tools to transfer the collected data between the involved devices and infrastructures.

c. The third tier is based on the Cloud to support essential computing and storage services to manipulate, visualise and store the collected data.

2.2 Raspberry Pi

The Raspberry Pi represents the IoT gateway of the system. It functions as an edge device which hides the sensor devices from the other components of the IoT system through its multi-Pin GPIO header. Through Raspberry Pi, the internet and the IoT applications can be linked to the sensor devices without direct access. It uses Linux OS and is frequently used for IoT applications. In addition, the operating system used in Raspberry Pi’s supports Python [5].

2.3 MQTT and Mosquito

Message Queue Telemetry Transport (MQTT) is a lightweight messaging protocol that can be supported by the smallest devices, yet robust enough to ensure that messages transfer to their destinations every time [6]. It is a machine-to-machine (M2M) - Internet of Things communication protocol that was designed as a considerable lightweight publish/subscribe messaging transport. It can be used for the connections with remote places where a small code is required and/or network bandwidth is at a premium [7].

Mosquito is a lightweight open source MQTT message broker that can be easily to set up publish-clients, broker-servers, and subscribe client sites [8]. The broker is available for several platforms and various programming languages. They will usually put Mosquito on a Raspberry Pi or some other small server that mimics a broker. It simplifies the use of MQTT communication between different systems running different programs [7].

2.4 Ansible
Ansible is an open-source software for facilitating the provisioning and configuration management of the distributed applications. It is a centralized tool that is not based on daemons to manage the remote machines; therefore, it only has to be installed on one machine which will be responsible for controlling the other machines. It configures the target systems by a stand-alone model along with a specific control machine that is responsible for connecting to the remote machines using SSH [9].

Ansible is convenient for managing all environments with different types and scales from small setups with a few number of instances to enterprise environments with large numbers of machines [10]. Ansibles' configuration management language is written in special scripts called play-books which are based in the YAML-format. The used format is very simple, and describes the state that the system should be in the Ansible tool is goal oriented which means that there is no need to write the code to perform changes on the target systems [9].

2.5 AWS Cloud
Cloud computing is a computing paradigm that provides computing resources as a service. It provides resources to store data and host services on a massive scale. One of the essential features of Cloud computing is that it supports the provision of on-demand resources and it offers the facilities to customize different types of computing environments with a charging based on pay-as-you-go model [11]. Therefore, applications are increasingly being moved to the Cloud to exploit the rich set of resources available there. The IoT applications that start with connecting things on the network and in the presence of the cloud leads to high efficiency in the process of storage and analysis, as well as easy access to data and rapid response [12]. Cloud services are provided by a number of providers such as Microsoft, Amazon and Google. In this work, Amazon Web Service is used to host part of the IoT system. Amazon web service is a platform that offers flexible, reliable, scalable, easy-to-use and cost-effective cloud computing solutions [13].

3. The Framework for Modelling and Deployment of Distributed IoT system
In this section, we highlight the design of the framework that supports the modelling of the distributed IoT applications and the overall deployment operation.

3.1 IoT Application
In general, IoT applications are based on several stages: sensing, collecting the data, and processing streams of data within a number of consecutive tasks [16]. The application presented in this paper as a use case is for sensing the temperature and humidity sensor. The collected data is transferred to smart single-board computers (SBCs) such as the Raspberry Pi which acts as the gate to collect data from the sensors and then forwards them to a local device or immediately to a remote Cloud. The local PC represents an edge device to apply some pre-processing on the data then forward the resulted data to the Cloud for further processing, storing and visualizing operations.

3.2 The Units of the Proposed IoT System
In the following, we present the units that form the architecture of the proposed IoT deployment system (see Figure 1) as follows:
1. Sensors Devices: a number of plug-and-play devices have been embedded including different types of sensors. These devices are the fundamental components of the entire IoT system since they are the source of data to be processed.
2. Data Gateway: a single-board computer that is a configurable device and has a number of sensors attached to offer a runtime environment to extract and provision sensor data such as Raspberry Pi. This device has limited processing and storage capabilities [1]. The gateway responsibility is to provide the connectivity between IoT sensor devices, on the one hand, and the on premise PC on the other hand, through reliable and efficient access networks.
3. Deployment Broker: the deployment broker is responsible for the recognition of the YAML-based description of IoT system's components and uses it to automate the deployment of each
component in the specified environment using Ansible platform. In our design, the deployment broker is contained in a local PC.

![Figure1. The Architecture of our Proposed IoT system.](image)

4. The Management and Storage end on the Cloud: the data will be transferred from the local PC to a virtual machine (VM) in the AWS cloud to store, process then visualise it. The sensed data values are stored in MongoDB by implementing a Python application which receives these values using a particular messaging protocol, processes and visualises the results. To transfer the data between the three run-time environments, the MQTT protocol and the Mosquito message brokers are used.

3.3 The Software Stack for the IoT System Units

In this section, we present the details of the software stack required to build each part of the IoT system developed in this work. In order to implement the IoT system, the three run-time environments required a number of tools, libraries and application components to build the software stacks required to make the system run properly as seen in Figure 2.

1. The software stack of the Data Gateway includes: Python package to run the developed publisher application for the sensors, MQTT and Mosquito agent.
2. The software stack for the Deployment Broker includes a Python package to run the developed subscriber application, MQTT, Mosquito and Ansible platform.
3. AWS VM software stack includes: Python package to run the python application to process and store data, the MongoDB database and the MQTT and Mosquito agents.
4. Components Modelling and the deployment of the IoT System

Our proposed approach for modelling and deploying the IoT system consists of two stages: 1) Configuration Modelling and 2) Deployment Implementation. In order to make the IoT system described above workable and ready to use, all the libraries, tools, and the application parts mentioned in section 3.3 need to be well described through the first stage which will be used in the second stage to install, configure and deploy them properly. In order to model the configuration, characteristics and setup information for each component of the IoT system we use YAML format to define their specifications. Afterwards, the Ansible Engine in the Deployment Broker device utilize the components specifications to effectively and automatically install, configure and run each component in the destination IoT infrastructures.

4.1 Configuration Modelling

The first stage is modelling all the components of the IoT system to make them ready to use in the deployment stage performed using Ansible. In this stage, the modelling of the three run-time infrastructures is described using YAML as follows:

1. Describe the Data Gateway infrastructure: The YAML format describes the executable Ansible play-book of the Raspberry Pi. The description comprises a set of descriptive steps that will be performed sequentially as shown in figure 3 which is the play-book for Raspberry PT configuration. We are using YAML for describing the overall application and its components. In the second line, the modelling includes (- hosts: Raspberry Pi) which is a key/value pair that specifies run-time environment to run commands on. It means, applying this play-book to the host (Raspberry Pi).
The main function of this play-book is to specify the IoT application part to be run on the described infrastructure which is the publisher’s application of the sensed data. This application is stored in GitHub repository to be dynamically downloaded during the deployment time.

2. The modelling of Deployment Broker Describe: The YAML format describes the executable play-book in PC which represents the deployment broker listed in descriptive steps that must be implemented sequentially as can be seen in figure 4. They represent the “hosts” field which refers to localhost as the current infrastructure to host the execution of the Subscriber application. The play-book specifies the component of the IoT application to be run in the local PC which is also resided in a GitHub repository and dynamically downloaded as needed during the deployment process.

```
---
- hosts: localhost
  become: yes
  become_method: sudo
  tasks:
    - name: Download the file
      get_url:
        url: https://raw.githubusercontent.com/habeeb1999/ansible_store2/master/sub.py
        dest: '/home/pi'
        register: res
      - debug:
        var: res.dest
      - name: "run the python application"
        command: python sub.py
---
```

Figure 4. The Description of Deployment Broker Environment.

3. Modelling of the AWS Virtual Machine: In this part, a YAML file is created to describe the AWS VM as a host infrastructure for the tools and some of IoT components. The file lists a set of descriptive steps that must be implemented sequentially as shown in figure 5 where the host field refers to the AWS Cloud on which MongoDB and Python-based applications for processing and visualizing the data are hosted. The description also contains information about the application to be dynamically downloaded from GitHub repository during the deployment process.

Since Ansible uses no agents, all the deployment operations will be performed from the Deployment Broker on the local PC where the Ansible commands will push the configurations to the rest of the system.
4.2 Deployment Implementation

After completing the modelling stage, we can now implement application deployment in the parts as mentioned previously through which we can obtain an integrated deployable application system within the Internet of Things. In these environments, it is important to provide synchronous communication for the class of distributed systems that operate in a loosely-coupled and autonomous fashion. This requirement has been implemented using the Ansible tool. To start the process of deployment on the three environments synchronously, a bash script file shown in figure 6 has been implemented. The commands in the script file refer to the three Ansible play-books that have been described in the previous section and represent Raspberry Pi, PC, AWS VM respectively. By running the script file, the three play-books will run in a synchronous way and all included tasks will be run with the defined order on each target environment. So you execute tasks on all servers without waiting order on each target environment. So you execute tasks on all servers without waiting target environment. So you execute tasks on all servers without waiting.

```bash
#!/bin/bash
Ansible-playbook -i R-PI.yaml > R-PI.log &
Ansible-playbook -i PC.yaml > PC.log &
Ansible-playbook -i AWS-VM.yaml > AWS-VM.log &
```

**Figure 6.** The Play-books for the IoT System

4.2.1 Implementation and Experiments

To validate the proposed approach, we conducted a number of experiments in which we used the modelling files described in the previous section to deploy the IoT system shown in Figure 7. The aim was to investigate and analyse several aspects regarding the performance of our modelling and deployment approach.

First, we measured the time required to deploy all parts of the system on the three infrastructures Raspberry Pi, local PC and the Cloud. We also wanted to see and minimise the efforts required to use our approach and compare it with the manual one.
5.1 System Setup
In our experiments we used two sensors: a 3-pin extremely reliable temperature detector named DS18B20 SENSORS (its temperatures vary from -55°C to +125) and the temperature and humidity sensor DHT11. It is made of two parts, a capacitive humidity sensor and a thermistor [14].

Gateway was implemented using the open-source hardware Raspberry Pi 3 Model B+ running the Linux-based Raspbian OS (version). For transferring data between the gateway and the broker, the MQTT and the Mosquito version 1.6.8 are used. In addition, python's version 2.7.17 was used for implementing the subscriber and publisher applications. The deployment broker has been developed on a local PC with Linux Ubuntu version 18.04.4 LTS installed on it. An AWS computing instance was launched with Ubuntu Server 18.04 LTS (HVM), SSD Volume Type-ami-0fc20dd1da406780b. Further, MongoDB version v4.2.3 is installed in the instance and used to store the data.

5.2 The deployment and Running of the IoT application on the AWS Cloud
To evaluate the performance of our proposed approach, we used the description files created in the modelling stage to deploy all parts of the developed IoT application and all required software stack as mentioned previously on the three environments. Prior to the deployment process, all the three environments should only contain the operating systems and Ansible platform on the local PC.

Based on the YAML descriptions for all IoT system components, Ansible will automatically start configuration, installing and deploying all components starting from Data Gateway with the required software stack, then the local PC and finally the AWS VM. By the end of the deployment process, the IoT application components will be run and start working so that the Raspberry Pi will collect the sensed data from the two sensors and forward them to the local PC to perform a simple preprocessing operation. After that, the resulting data will be transferred to the Cloud to be stored on MongoDB and visualised.

The deployment process has been executed fifteen times and the elapsed time for the full process has been measured with Standard Deviation Errors. Figure 8 shows the average time of the executions with the Standard deviation errors. The deployment time in each environment represents the time required to install all the libraries and tools dynamically downloading all IoT application components (in Python) from the GitHub repository and running them on a particular run-time environment. The differences of the deployment times shown on the three run-time environments are due to the number of operations required to deploy a full stack for each environment as shown in Figure 2 and the time required to download the IoT application components.
The automation of distributed application deployment is beneficial for the improvement of the whole process correctness (by reducing human errors), pace (by in-parallel applying of the time-consuming installations), reduce human efforts, in addition to improve the process documentation using a proper description. Using our approach to deploy the above described IoT system shows that the number of operations required to setup and deploy the system components is reduced from sixteen operations (listed in table 1) using manual approach to only one command using our approach which is to start Ansible engine to execute the described play-books. In addition, our proposed approach facilitates the whole process to enable non IT professional users to efficiently re-use the same YAML-based system descriptions and our proposed approach to set up a similar IoT system even if he/she is not a professional IT developer.

### Table 1. Manual Deployment Operations on the Run-time Environments

| Operation                  | Raspberry-Pi | local PC | AWS VM   |
|----------------------------|--------------|----------|----------|
| install openssh-server     | install openssh-server | install openssh-server |
| install python-pip         | install python-pip | install python-pip |
| install paho-mqtt          | install paho-mqtt | install paho-mqtt |
| install mosquito           | install mosquito | install mosquito |
| install mosquito-clients   | install mosquito-clients | install mosquito-clients |

#### 5.3 Discussion

The presented results observe that the manual implementation of the deployment operations is probably convenient for simple, small scale applications because it has a small barrier to entrance. While for the heterogeneous multi-components system, automated deployment should present benefits over the manual approach which requires additional investments to recognize the heterogeneity of the system to be deployed and developing the installation and configuration descriptions consisting of all required tools and dependencies. In addition, using manual methods consume a lot of time and effort to install and configure all the required packages, applications, and software libraries.

Further, an application on the remote environment may require regular updates; therefore, this is another problem of using manual installation. Therefore, a precise description and the automation of...
the deployment process for all required tools, packages and application components can be installed and configured on the remote environments efficiently. In our approach, using Ansible to carefully modelling the management steps and the required components helps during the first deployment and even more on the subsequent deployments. Therefore, the automatic distributed system deployment can be repeated in an easy and efficient way.

6. Related Work
In recent years, with the increasing publicity of the IoT systems, they have gained considerable concern by the researchers in many directions. One of these significant directions is the design of approaches to facilitate the deployment of the IoT application's parts which has been investigated in a number of researches [15][16][4]. Michael Vogler et al[16] presented a framework to dynamically dynamic generate the optimized topologies of deployment for IoT cloud applications. This approach used a declarative, constraint based model for the deployment of the desired application, to enable the provisioning of application components on both cloud infrastructure, and the IoT devices deployed in the IoT infrastructure. In [4], the authors proposed using TOSCA, a Standard for the management of Cloud Service, to define the IoT application components.

Where it is shown that using TOSCA, application templates can be reused, and deployments can have applied in heterogeneous IoT system environments. While in this work, we proposed the first attempt to address the deployment of IoT application using Ansible platform to facilitate and automate the IoT system modelling and deployment by utilizing the YAML-based description for the distributed IoT application on heterogeneous IoT environments.

7. Conclusion
In this paper we have presented a new approach that supports an effective and simplified modeling for the components of an IoT system that can be used to deploy these components in number heterogeneous IoT environments including public cloud. We identified a number of common requirements in many IoT applications that have been taken into consideration during the design of our deployment approach. As a proof of concept we have developed a distributed IoT application with different usages and apply our approach to deployed it on a number of infrastructures including Raspberry Pi board, local PC and AWS Cloud instance.

The evaluation of our approach shows that the feasibility of using it to facilitate the deployment of a full IoT system with distributed components on heterogeneous environments and the ability to repeat the process efficiently even by nonprofessional developers as the required effort has been minimised to get the IoT system.

References
1. L. Hou, S. Zhao, X. Xiong, et al., “Internet of things cloud: architecture and implementation,” IEEE Communications Magazine, vol. 54, no. 12, pp. 32-39, 2016.
2. Hirmer, P., Breitenbücher, U., et al., “Automating the Provisioning and Configuration of Devices in the Internet of Things”, CSIMQ 9, pp.28-43, 2016.
3. V. Nundloll, B. Porter, G. S. Blair, et al., “The design and deployment of an end to end IoT infrastructure for the natural environment,” Future Internet, vol. 11, no. 6, p. 129, 2019.
4. F. Li, M. Vogler, M. Claeens, and S. Dustdar, “Towards automated IoT application deployment by a cloud-based approach,” in 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications. IEEE, pp. 61-68, 2013.
5. http://www.raspberrypi.org, accessed: 2020-03-26.
6. M. L. Pham, T. T. Nguyen, and M. D. Tran, “A benchmarking tool for elastic MQTT brokers in IoT applications,” International Journal of Information and Communication Sciences, vol. 4, no. 4, pp. 70-78, 2019.
7. D. B. Ansari, A. Rehman, and R. Ali, “Internet of things (IoT) protocols: A brief exploration of mqtt and coap,” International Journal of Computer Applications, vol. 975, p. 8887, 2018.
8. http://mosquitto.org, accessed: 2020-03-26.
9. S. Thakur, S. C. Gupta, N. Singh, and S. Geddam, “Mitigating and patching system vulnerabilities using Ansible: A comparative study of various configuration management tools for IaaS cloud,” in Information Systems Design and Intelligent Applications. Springer, pp. 21-29, 2016.
10. K. Torberntsson and Y. Rydin, “A study of configuration management systems: Solutions for deployment and configuration of software in a cloud environment,” 2014.
11. J. W. Rittinghouse and J. F. Ransome, Cloud computing: implementation, management, and security. CRC press, 2016.
12. A. Hosseinian-Far, Mand C. L. Slack, “Emerging trends in cloud computing, big data, fog computing, IoT and smart living,” in Technology for Smart Futures. Springer, pp. 29-40, 2018.
13. Varia, J., “Best practices in architecting cloud applications in the AWS cloud," in Cloud Computing: Principles and Paradigms. Wiley Online Library, vol. 18, pp. 459-490, 2011.
14. https://components101.com/dht11-temperature-sensor, accessed: 2020-04-10.
15. P. Hirmer, U. Breitenbucher, et al., “Automating the provisioning and configuration of devices in the internet of things." CSIMQ, vol. 9, pp. 28-43, 2016.
16. M. Vogler, J. M. Schleicher, C. Inzinger, and S. Dustdar, “Diane-dynamic IoT application deployment," in 2015 IEEE International Conference on Mobile Services. IEEE, pp. 298-305, 2015.