Institute for Parallel and Distributed Systems

Institute of Architecture of Application Systems

OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker

Ana C. Franco da Silva[1], Uwe Breitenbücher[2], Kálmán Képes[2], Oliver Kopp[1], Frank Leymann[2]

[1]Institute for Parallel and Distributed Systems,
[2]Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{franco-da-silva,breitenbuecher,kepes,kopp,leymann}@informatik.uni-stuttgart.de

Full ACM reference as provided by ACM : Ana Cristina Franco da Silva, Uwe Breitenbücher, Kálmán Képes, Oliver Kopp, Frank Leymann. 2016. OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker. In *Proceedings of the 6th International Conference on the Internet of Things* (IoT '16). ACM, New York, NY, USA, 181-182.
DOI= 10.1145/2991561.2998464 http://dx.doi.org/10.1145/2991561.2998464

BibTeX:

```
@inproceedings {INPROC-2016-39,
   author = {Ana Cristina Franco da Silva and Uwe Breitenb\"{u}cher and
K\'{a}lm\'{a}n K\'{e}pes and Oliver Kopp and Frank Leymann},
   title = {{OpenTOSCA} for {IoT}: Automating the Deployment of {IoT}
Applications based on the {Mosquitto} Message Broker},
   booktitle = {Proceedings of the 6th International Conference on the
Internet of Things},
   publisher = {ACM},
   pages = {181--182},
   mon = nov,
   year = {2016},
   isbn = {978-1-4503-4814-0/16/11},
   doi = {10.1145/2991561.2998464},
   }
```

OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker

Ana C. Franco da Silva[1], **Uwe Breitenbücher**[2], **Kálmán Képes**[2], **Oliver Kopp**[1], **Frank Leymann**[2]
[1]Institute for Parallel and Distributed Systems, University of Stuttgart, Germany
[2]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{franco-da-silva,breitenbuecher,kepes,kopp,leymann}@informatik.uni-stuttgart.de

ABSTRACT
Automating the deployment of IoT applications is a complex challenge, especially if multiple heterogeneous sensors, actuators, and business components have to be integrated. This demonstration paper presents a generic, standards-based system that is able to fully automatically deploy IoT applications based on the TOSCA standard, the standardized MQTT messaging protocol, the Mosquitto message broker, and the runtime environment OpenTOSCA. We describe a demonstration scenario and explain in detail how this scenario can be deployed fully automatically using the mentioned technologies.

ACM Classification Keywords
K.6 Management of Computing and Information Systems; D.2.12 Software Engineering: Interoperability

Author Keywords
Internet of Things; Cyber-Physical Systems; Sensor Integration; Message Broker; Mosquitto; MQTT; TOSCA.

INTRODUCTION & BACKGROUND
The *Internet of Things (IoT)* paradigm relies on the combination of the physical and virtual world, i. e., on so-called *cyber-physical systems*. The physical part of such systems is implemented by physical devices that typically provide sensors and/or actuators affecting the physical environment. The cyber part consists of software responsible for extracting and processing sensor data in order to react to changes in the real world. To enable IoT applications realizing tasks such as monitoring and reconfiguring physical devices, sensor data needs to be accessible to these applications for further processing. A common approach for this is the use of messaging systems that support the *publish/subscribe interaction model*. Using this interaction model, subscribers register their interest in a certain kind of messages by subscribing to specific *topics* in order to get asynchronously notified when publishers send messages to these topics. A *message broker* is responsible for managing subscriptions and delivering messages to subscribers.

IoT'16 November 07-09, 2016, Stuttgart, Germany

ACM ISBN 978-1-4503-4814-0/16/11.
DOI: http://dx.doi.org/10.1145/2991561.2998464

Eclipse Mosquitto[1] is an open-source message broker that implements the MQTT standard[2], which is a lightweight publish/subscribe messaging protocol. In IoT environments, physical devices can be easily integrated with IoT applications using message brokers such as Mosquitto: Devices publish sensor data to Mosquitto, while IoT applications can act as subscribers and receive these sensor data for further processing. Publishers, e. g., devices with sensors, and subscribers, e. g., IoT applications or devices with actuators, interact with Mosquitto using *MQTT clients*, which are available in different languages, e. g., for Python, Java, and C (cf. *Eclipse Paho*[3]).

This demonstration paper shows how the deployment of such IoT applications can be automated using *OpenTOSCA* [1], a runtime environment for automatically deploying and managing applications based on the TOSCA standard [4].

IOT MOTIVATION SCENARIO
To explain our deployment approach, we first describe an IoT scenario that is based on Mosquitto and two Raspberry Pis: One Raspberry Pi reads temperature sensor values and the other turns a ventilator on/off depending on these values. Mosquitto is installed on a virtual machine hosted on Openstack[4]. Python scripts are employed (i) to read sensor data, (ii) to control the ventilator, and (iii) to communicate with Mosquitto via MQTT. To deploy this scenario, the virtual machine must be provisioned, Mosquitto must be installed, the Raspberry Pis must be configured using SSH connections, and the Python scripts must be copied to the Raspberry Pis and started using SSH connections as well. However, deploying this scenario is not trivial since it requires expert knowledge about the OpenStack API, SSH connections, MQTT and its client and server APIs, and shell scripting. Thus, manually executing these tasks is time-consuming and error-prone.

OVERVIEW ON TOSCA
The IoT deployment automation approach we present in this demonstration is based on the TOSCA standard [4]. We first explain the TOSCA concepts required for understanding this paper and show afterwards how the demonstration scenario can be modeled using TOSCA. TOSCA enables describing the structure of an application to be deployed in the form of a *topology model*, which is a graph of typed nodes and directed, typed edges. Nodes are called *node templates* and correspond to the components of the application while edges are called

[1] https://www.mosquitto.org/ [2] http://www.mqtt.org/
[3] http://www.eclipse.org/paho/ [4] http://www.openstack.org/

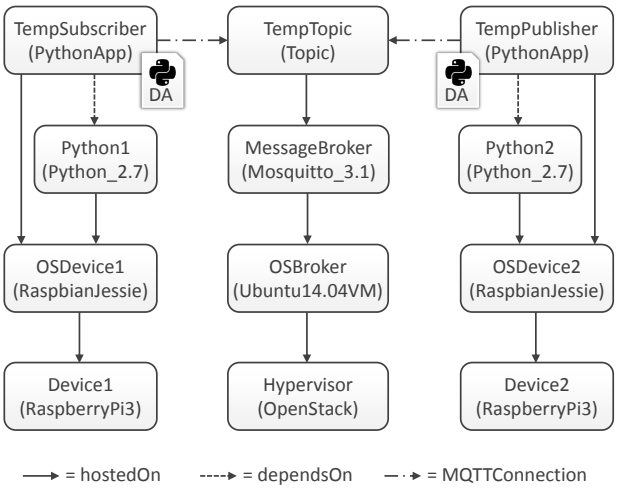

Figure 1. TOSCA topology model of the IoT demonstration scenario

relationship templates and describe the dependencies between the components. TOSCA is a generic language that enables defining arbitrary types of components and dependencies by *node types* and *relationship types*. Business logic implementations of components can be attached to node templates using so-called *deployment artifacts*, which are, for example, binaries or Java Web Archives (WARs). To install or manage a component, *management operations* can be defined for types. The implementations of these operations can be provided using *implementation artifacts*, which are, for example, shell scripts to install a component. TOSCA defines a packaging format called *Cloud Service Archive (CSAR)*, which enables bundling all the described entities in a self-contained manner.

AUTOMATED DEPLOYMENT DEMONSTRATION

Figure 1 depicts the TOSCA topology model of our scenario. The stack in the middle provides the infrastructure for the Mosquitto message broker, which hosts a special topic, to which applications publish and subscribe. The broker and the topic node types provide implementation artifacts in the form of shell scripts that can be executed on Linux systems to install the respective components. The right stack is composed of a Raspberry Pi and its operation system hosting a Python-based publisher application, whose implementation is attached as deployment artifact (DA) containing the respective Python script. The publisher application periodically reads temperature sensor data and sends it to the topic. Finally, the left stack consists of a Raspberry Pi, its operation system, and a Python-based subscriber application, which receives messages from the topic and generates a RF-signal to turn on the ventilator when the temperature passes a threshold value. Also this Python application node provides a deployment artifact that references the actual implementation, i.e., the Python script.

To automatically deploy this TOSCA topology model, we employ the OpenTOSCA runtime environment [1], which is a standards-based open-source management automation system developed at the University of Stuttgart. OpenTOSCA implements the TOSCA standard and, thereby, enables the automated deployment and management of applications. To deploy our demonstration scenario, we developed new node types for Raspberry Pis, RaspbianJessie, Python, Mosquitto, and topics. Our initial setup consists of two physical Raspberry Pis having only the RaspbianJessie image installed. The Raspberry Pis are connected to a physical LAN having static IP-addresses. The IP-addresses and SSH credentials of the Raspberry Pis are also specified in the topology model (not shown in Fig. 1). Moreover, the OpenStack cloud management system is installed and running. Also the user credentials are contained in the topology model. Based on these information, OpenTOSCA is able to automatically deploy and connect the remaining nodes: First, the topology model is processed by OpenTOSCA's *plan generator* [2], which generates a BPEL workflow model. This model specifies all tasks to be executed to deploy the application and is executed fully automatically by a workflow engine (WSO2 BPS[5]) in OpenTOSCA [1, 2].

Thus, to deploy our IoT demonstration scenario, the plan generator needs to know how to deploy these new node types. We achieved this without changing the plan generator's implementation that we have developed originally for Cloud applications [2]: TOSCA defines a so-called *lifecycle interface* [3], which specifies lifecycle operations for managing a component, e. g., *install*, *start*, *stop*. The plan generator is aware of this interface and knows which operations have to be executed in which order to deploy a certain component [2]. Therefore, to add the new node types to our ecosystem, we implemented the TOSCA lifecycle interface for all new node types using SH-scripts, which are supported by the current plan generator implementation. During the plan generation, the generator generates activities that copy these SH-scripts onto the virtual machine or Raspberry Pi, respectively, using SSH and executes them. Thus, the plan generation, which corresponds to the actual deployment, remains completely generic and did not have to be adapted for this demonstration. As a result, by simply replacing the Python deployment artifacts, other business logic, sensors, and actuators can be automatically deployed and connected using this topology template. All files, OpenTOSCA, and the topology model of this demonstration are available as open-source implementation[6].

ACKNOWLEDGMENTS

This work is funded by the BMWi project SmartOrchestra (01MD16001F).

REFERENCES

1. Tobias Binz and others. 2013. OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *ICSOC*. Springer, 692–695.
2. Uwe Breitenbücher and others. 2014. Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *IC2E*. IEEE, 87–96.
3. OASIS. 2013a. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0.*
4. OASIS. 2013b. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0.*

[5] http://wso2.com/products/business-process-server/

[6] http://www.opentosca.org/demos/opentosca_for_iot_mosquitto/