Remote Debugging for Containerized Applications in Edge Computing Environments

Muhammet Oguz Ozcan
Corporate Technology
Siemens A.Ş.
İstanbul, Turkey
muhammet.ozcan@siemens.com

Fatih Odaci
Corporate Technology
Siemens A.Ş.
İstanbul, Turkey
fatih.odaci@siemens.com

Ismail Ari
Computer Science
Ozyegin University
Istanbul, Turkey
ismail.ari@ozyegin.edu.tr

***Abstract*—Edge Computing (EC) became popular again with the rise of IoT, Cloud Computing, and Industry 4.0. In this paper, difficulties of application development in the EC environment are discussed and a container-based solution using remote debugging at the edge is proposed. This container allows application developers to write code in the production environment. Our implementation increases the development speed and facilitates in-place debugging for EC environments.**

***Keywords*- Edge Computing, Remote Debugging, Docker, Container, Internet of Things, Industry 4.0.**

I. INTRODUCTION

The Industrial Internet of Things (IIoT) combined with cloud infrastructures, big data platforms and machine learning can optimize industrial processing [1]. This new revolution is also referred to as *Industry 4.0*. While the cloud computing and storage capacities needed to retain the high volume and velocity of IIoT sensor data have scaled reasonably well within the last decade, the capacity of edge networks haven't improved in most parts of the world [2]. The 5G and Mobile Edge Computing (MEC) deployments are emerging to address this problem [3, 4]. Putting computing resources closer to IoT devices reduces latency and provides higher per device bandwidth at the Local Area Network (LAN)-level; instead of sharing Wide Area Networks (WAN). EC moves parts of the data processing to the edge (*i.e.* fog) to save network bandwidth rather than processing everything in the cloud [1], [7].

Industrial use cases such as "smart manufacturing" are very popular in the EC domain [5]. For example, a large number of sensors in a Computer Numeric Control (CNC) machine could be fused and modelled to measure production quality, analyze machine status, and predict tool breakage events. High speed and high volume data are streamed from these sensors to corresponding analysis channels. However, to gather and analyze data in real time, fast connections and powerful computers are required. These performance issues and the following security-related concerns are increasing popularity of EC for manufacturing. Users of industrial machines produce parts with G-Codes. These program files for various parts and systems represent highly confidential designs of cars, planes, spaceships or military equipment. Many factories are therefore not even allowed to have an internet connection. Nevertheless, to improve their production quality and speed the data have to be analyzed.

Containerization is a very popular technology used both in software development and now in edge computing. One of the most popular application containers is Docker [8], which eases the development, deployment and execution of distributed applications. It is much lighter and faster than Virtual Machine (VM) technologies, because there is no need to launch a full-stack OS for each application. Containers only retain the runtime requirements of applications including libraries, environment variables and external files. Vendor-specific IIoT applications are developed over many different platforms and different OS. To manage this diversity easily, developers can benefit from containerization. However, this ease comes with a cost while trying to debug the applications.

II. DIFFICULTIES OF APP DEVELOPMENT AT THE EDGE

The application development for IIoT differs from the regular desktop application development processes. First of all, applications cannot obtain machine data by themselves; interfaces and applications provided by the vendors are used for security reasons. Since this monolithic approach is difficult to manage at the edge, container-based technologies have been preferred recently. However, containerization has other complications: building, deploying, and testing applications with containers require Continuous Integration and Continuous Delivery (CI/CD) steps to be triggered for every change in the code base. We have identified the following problems of IIoT edge application development:

1. Edge hardware environments differ from the desktop development environments in terms of computational capacities, operating systems used, network connectivity, and power specifications. In software domain, system libraries, available functions as well as the encoding can be different. These ambiguities require extra effort and attention during IIoT application development.
2. It is hard to simulate industrial scenarios with synthetic data; real machine data is required. Moreover, testing with high-speed data serves better in understanding how the system reacts in real cases. Real-time data acquisition from CNC machines operating at microsecond latencies and transferring gigabytes of data per second is difficult.
3. Edge applications usually depend on other vendor applications to obtain the machine data. Engineers have to develop expertise on vendor-specific data acquisition and follow their updates. This is an extra, unwanted

978-1-7281-2708-8/19/$31.00 ©2019 IEEE
DOI 10.1109/EDGE.2019.00021

investment for software developers. Furthermore, when a problem occurs on these vendor applications, developers have to report them and wait for the fixes, which are usually very time consuming and exhausting.

4. Containerization is now being used in IIoT applications. It improves platform independence, but brings extra costs for updating certain configuration files. After every change, the application has to be containerized again and the cycle is repeated. Then, the new container has to be sent to the edge device using a custom interface of the remote environment and tested. Even if there is a small bug, the whole procedure has to be repeated and a new version of the application is published. This is a tedious process for the developers who work in the EC environment.

Remote debugging techniques are commonly used in machine dependent developments to increase software quality and observe real case scenarios. However, developers are left with black-box and remote debugging options when dealing with closed industrial systems.

III. Proposed Method

To overcome these problems, we implemented a remote-debugging method. The architecture of the proposed method is presented in Figure 1. A specific Docker container is created and all the dependencies (software packages and libraries) required during development and runtime are placed in this container. Other applications can now be compiled and executed inside the edge device. This remote debugging container, which is also an application by itself, is only used during development.

With this proposed method, the performance of IIoT applications at the production environment can be tested. The security is assured via an encrypted connection to the edge device. Since the applications is debugged and tested in a real device, machine data can easily be obtained by running the correct CNC program. Dependencies to other edge applications are also tested in the production environment and this will save time and effort of the developers.

Finally, the difficulties of the containerized applications are eliminated with the help of the remote debugging. New changes in the applications can be directly tried inside the edge device without creating a new container each time. This can be achieved using a secure copy of the application executable to the corresponding folder inside the remote debugging application container. The container can be mapped to the outside world via any port (we used 2222). SSH server runs inside the container providing a secure connection for secure file sharing. The details of these and the DockerFile are explained in the following section.

A. Remote Debugging

Microsoft Visual Studio (VS) 2017 IDE provides remote debugging for different programming languages and frameworks [12]. It enables developers to implement their applications in Windows and execute them in Linux or vice versa. The source code is moved onto the remote environment, compiled and executed on the device side.

Moreover, features of the remote environment are fetched backed to the development environment so that the developer can see the code auto-completion (like IntelliSense™) and the libraries of the remote environment. This is equivalent to working in the production environment. Developers have more time for developing new features and testing real-life scenarios instead of dealing with issues.

B. DockerFile and Dependencies

As shown in Figure 2, the DockerFile created for our method starts by including a base Linux image (we used debian). Next, other required packages are included: rsync, ssh server, cmake, gdb, and others. Rsync is used to transfer and synchronize source files across computers. Then, the files are compiled and executed inside the edge device. For SSH, any available port can be used. The compiler and debugging tools required in this DockerFile are gdb, g++, cmake, and wget. It is important to understand that our Docker image is not a release image. It is used only for the development and test purposes and it will be used by the application developers. Normally, only the compiled application files would be placed inside the Docker image.

C. Security

Since security is of utmost importance, we carefully enable debug data transfers from the remote container. The *docker-compose.yml* file shown in Figure 3 is used to map the internal port of the container to the outside port. In this example, port 2222 of the application is mapped to port 2222 of the edge device. However, if we only enable security the results of debugging cannot be obtained by the IDE. Therefore, we include `"--security-opt seccomp=unconfined"` option to allow unconfined access to the desktop development environment. Another critical key in this file is "`SYS_PTRACE`" capability. The key is used to return system and memory traces outside the container. This feature is specifically used in C++ application development with gdb.

IV. Related Work

The research field of remote debugging and deployment for EC has not been well-addressed; we aim to fill this gap. Premsankar, *et al.* [9] measure the suitability of edge computing for emerging IoT applications and focus on mobile gaming. They conclude that, since network bandwidth is limited and causes latency in mobile games,

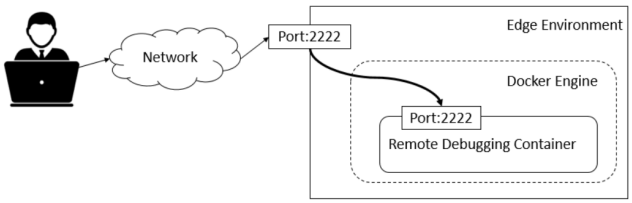

Figure 1. The architecture of the proposed container method.

```
FROM debian:stretch

# Install dependencies to compile and execute C++ apps
RUN apt-get update && apt-get -y upgrade && apt-get
install -y -f --no-install-recommends \
   cmake wget gdb g++ rsync build-essential openssh-
server && rm -rf /var/lib/apt/lists/*

#Prepare SSH server settings and credentials
RUN echo "mkdir /var/run/sshd" >> /usr/bin/runme.sh
RUN echo "echo \"root:root\" | chpasswd" >>
/usr/bin/runme.sh
RUN echo "echo \"PermitRootLogin yes\" >>
/etc/ssh/sshd_config" >> /usr/bin/runme.sh

# SSH login - Otherwise user is kicked out after login
RUN echo "sed
's@session\s*required\s*pam_loginuid.so@session
optional pam_loginuid.so@g' -i /etc/pam.d/sshd"  >>
/usr/bin/runme.sh

RUN echo "echo \"export VISIBLE=now\" >> /etc/profile"
>> /usr/bin/runme.sh
#Install latest version of the CMake
RUN wget --no-check-certificate
https://cmake.org/files/v3.13/cmake-3.13.0-rc2.tar.gz
RUN tar -xvf  cmake-3.13.0-rc2.tar.gz
RUN cd cmake-3.13.0-rc2 && bootstrap && make && make
install

# Start SSH server on port 2222
CMD ["/usr/sbin/sshd", "-p 2222", "-D"] -D
```

Figure 2. The DockerFile of the proposed container method

```
version: '3'
services:
   remotedebugapplication:
            build: .
            security_opt:
                - seccomp:unconfined
            ports:
                - "2222:2222"
            cap_add:
                - SYS_PTRACE
```

Figure 3. The docker-compose.yml of the proposed method

using even a limited amount of EC helps to improve the quality of gaming experience. The challenges and opportunities of the edge intelligence are discussed by Plastiras, *et al.* [10]. They present a use case where a carefully designed Convolutional Neural Network (CNN) has a real-time performance on edge computing devices for computer vision tasks. As cloud providers release new Edge products benchmarking among them becomes necessary. Das, *et al.* [11] developed EdgeBench for comparing of Amazon AWS Greenglass and Microsoft IoT Edge services. They found that these services provide a promising alternative to cloud computing for CPU light workloads. In the container domain, Docker-pi [6] combines three different optimizations to reduce deployment times by a factor of four: sequential image layer downloading, multi-threaded layer decompression and I/O pipelining. All of these studies are complementary to our current work in progress.

V. Conclusions and Future Work

Edge Computing (EC) is the new norm for IIoT app development, but its implementation comes with new challenges. In this study, we identified those challenges and proposed a novel remote debugging method to overcome them. Our method adds only a small amount of configuration effort, but provides significant returns in development time and security. The remote debugging container includes all the required application dependencies during compile and runtime. By opening a specific port and running an SSH server inside the container, any application can be remotely and securely debugged inside the edge device. This can solve the problems of the application development at the edge. This study enabled remote development of C++ applications. We plan to support commonly used languages such as Python, Java, and NodeJS in the future.

Acknowledgement

This research effort was supported by the Siemens Turkey Corporate Technology unit. We thank for the continuous support and facilities they provide.

References

[1] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues and M. Guizani, "Edge Computing in the Industrial Internet of Things Environment: Software-Defined-Networks-Based Edge-Cloud Interplay," in *IEEE Communications Magazine*, vol. 56, no. 2, pp. 44-51, Feb. 2018.

[2] P. Corcoran and S. K. Datta, "Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network," in *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 73-74, Oct. 2016.

[3] W. Shi and S. Dustdar, "The Promise of Edge Computing," in *Computer*, vol. 49, no. 5, pp. 78-81, May 2016. doi: 10.1109/MC.2016.145

[4] W. Yu *et al*., "A Survey on the Edge Computing for the Internet of Things," in *IEEE Access*, vol. 6, pp. 6900-6919, 2018.

[5] OpenFog Consortium Arch. Working Group, "OpenFog reference architecture for fog computing." OPFRA001 20817 (2017): 162.

[6] A. Ahmed, G. Pierre. "Docker Container Deployment in Fog Computing Infrastructures". IEEE EDGE 2018 - IEEE International Conference on Edge Computing, Jul 2018, San Francisco, CA, United States. IEEE, pp.1-8, 2018.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016.

[8] Docker Inc., "Docker: Build, ship, and run any app, anywhere," https://www.docker.com/

[9] G. Premsankar, M. Di Francesco and T. Taleb, "Edge Computing for the Internet of Things: A Case Study," in *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275-1284, April 2018.

[10] G. Plastiras, M. Terzi, C. Kyrkou and T. Theocharidcs, "Edge Intelligence: Challenges and Opportunities of Near-Sensor Machine Learning Applications," *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Milan, 2018, pp. 1-7.

[11] A. Das, S. Patterson and M. Wittie, "EdgeBench: Benchmarking Edge Computing Platforms," *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich, Switzerland, 2018, pp. 175-180.

[12] "Remote Debugging a Visual C++ Project in Visual Studio", Microsoft Docs, Accessed Feb. 2, 2019, https://docs.microsoft.com /en-us/visualstudio/debugger/remote-debugging-cpp?view=vs-2017