IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Remote Debugging in Edge Computing Environments for Containerized Applications

Nakul Ashok Gade
MVPS's Rajarshi Shahu Maharaj Polytechnic, Nashik

Vikas Pralhad Gawai
MVPS's Rajarshi Shahu Maharaj Polytechnic, Nashik

Ankita Kamlakar Pangavhane
MVPS's Rajarshi Shahu Maharaj Polytechnic, Nashik

Sachin Ashok Surywanshi
MVPS's Rajarshi Shahu Maharaj Polytechnic, Nashik

Priya Gulabrao Deshmukh
MVPS's Rajarshi Shahu Maharaj Polytechnic, Nashik

ABSTRACT

Abstract—With the development of IoT, Cloud Computing, and Industry 4.0, the term edge computing (EC) acquired popularity once more. The difficulties of developing applications in the EC environment are described in this paper, and a container-based approach leveraging remote debugging at the edge is suggested. Application developers can now write code in the production environment thanks to this container. Our solution speeds up development and makes in-place debugging easier for EC setups.

Keywords

Edge computing, remote debugging, Docker, containers, the Internet of Things, and Industry 4.0 are a few associated terms.

INTRODUCTION

Machine learning, big data platforms, cloud infrastructures, and the Industrial Internet of Things (IIoT) may all be utilized to optimize industrial processing [1]. The term "Industry 4.0" is also used for this new revolution. The capacity of edge networks has not grown in the majority of the world's regions over the previous decade, despite the cloud computing and storage capacities required to maintain the large volume and velocity of IIoT sensor data scaling rather effectively [2]. In order to solve this issue, 5G and mobile edge computing, or MEC, deployments are growing [3, 4]. Instead of sharing a wide-area network (WAN), placing computer resources closer to IoT devices minimizes latency and increases per-device bandwidth at the LAN level. Instead of processing all the data in the cloud, EC moves some of it to the edge (i.e., fog) to reduce network bandwidth [1], [7].

In the EC sector, industrial use cases like "smart manufacturing" are highly popular [5]. For instance, a large number of sensors in a Computer Numeric Control (CNC) machine might be combined and modeled to assess machine status, forecast tool breakage events, and estimate the quality of the output. These sensors send rapidly and huge quantities of data to the appropriate analysis channels. However, quick connections and potent processors are necessary for real-time data collection and analysis. Following security-related difficulties and these performance issues are making EC more and more appealing for manufacturing. G-Codes are used by users of manufacturing equipment to produce parts. These program files for various systems and parts indicate highly secret designs for vehicles like cars, planes, spacecraft, or military gear. As a result, many factories are not even permitted to have internet access. However, analysis of data is required to increase both the quality and speed of their output.

The technology of containerization is widely employed in edge computing as well as software development. Docker [8], which facilitates the development, deployment, and operation of distributed applications, is one of the most widely used application containers. Since there is no requirement to start a full-stack OS for each application, it is much smaller and quicker over virtual machine (VM) technologies. Only the running-time requirements of applications, like as libraries, environment variables, and external files, are retained by containers. Vendor-specific IIoT apps are created for a variety of platforms and operating systems. Containerization can help developers manage this variability more readily. However, whereas attempting to debug the programs, this ease has a cost.

DIFFICULTIES OF APP DEVELOPMENT AT THE EDGE

IIoT application development is different from the creation of desktop applications in general. Application interfaces and apps offered by manufacturers are employed for security concerns because programs cannot get machine data on their own. These monolithic methods are challenging to handle at the edge, hence container-based systems have lately gained popularity. However, containerization has additional drawbacks, including the need for continuous enhancement and Continuous Deployment (CI/CD) processes to be triggered for any change to the code base while creating, releasing, and evaluating apps using containers. The issues in developing IIoT edge applications that we have seen are as follows:

1. There are several aspects of edge hardware settings that set them apart from desktop development environments, including their computing capabilities, operating systems, connectivity to networks, and power requirements. System libraries, available functionality, and encoding can all vary in the software domain. When creating IIoT applications, these ambiguities demand particular care and attention.
2. Synthetic data is difficult to model industrial circumstances; actual machine data is needed. Additionally, testing with data at high speeds helps to better understand how the system responds in actual situations. It is challenging to acquire real-time data from CNC machines that transport gigabytes of information per second and have microsecond latency.
3. In order to access machine data, edge apps typically rely on those of other vendors. Engineers must get knowledge about and keep up with vendor-specific data gathering. For software engineers, this is an added expense that they don't need. Additionally, developers must report issues with these vendor programs and wait for patches, which are frequently tiresome and time-consuming.
4. Applications for the IIoT are now using containerization. Platform independence is increased, although there are additional expenses associated with changing some configuration files. The cycle must be repeated after each modification, requiring another containerization of the application. The new container must then be transferred to the edge devices via a customized remote environment interface before being evaluated. The entire process must be performed in the event of a minor bug, and an updated version of the program must be released. The developers who operate in the EC context find this to be a tiresome task.
5. To improve software quality and examine real-world case studies, remote debugging methods are frequently employed in machine-dependent development. When working with closed industrial systems, developers are only left with the options of remote and black-box debugging.

PROPOSED METHOD

We built a remote-debugging mechanism to solve these issues. Figure 1 shows the suggested method's architectural layout. All the requirements (software packages and modules) needed for development and runtime are added to a specific Docker container that has been constructed. Now, other programs can be built and run within the edge device. This application-by-itself remote debugging container is exclusively employed when developing.

The efficiency of IIoT applications in a production setting can be evaluated using the suggested method. An encrypted link to the edge device ensures security. Running the appropriate CNC program makes obtaining machine data simple because the application has been tested and debugged on a real device.

Finally, the use of remote debugging helps to resolve the issues with containerized apps. Without having to repeatedly create new containers, new modifications in the applications may be directly tested inside the edge device. A secured backup of the application executable to the appropriate directory inside the remote debugging program container can be used to accomplish this. Any port (we used 2232) can be used to map the container to the outside world. A safe connection for safe sharing of files is provided by an SSH server running inside the container. The following section provides more information on these and the DockerFile.

A. Remote Debugging

For various programming languages and frameworks, remote debugging is offered by the Microsoft Visual Studio (VS) 2017 IDE [12]. It enables programmers to create their apps in Windows and run them in Linux, or the other way around. The original code is transferred to the remote surroundings, where it is then built and run on the device.

Additionally, aspects of the remote environment are fetched back to the development environment so that the developer may see the remote environment's libraries and code auto-completion (like IntelliSenseTM). It's the same as working in a production setting. Instead of fixing problems, developers now have more time to work on creating new features and testing them in real-world situations.

B. DockerFile and Dependencies

As seen in Figure 2, the DockerFile made using our technique begins by including a debian-based base Linux image. Then, more necessary packages are included, such as rsync, ssh server, cmake, gdb, and others. To move and synchronize source files between machines, use Rsync. The files are then assembled and run inside the edge device. Any open port can be utilized for SSH. Gdb, G++, Cmake, and Wget are the compiler and debugging tools needed for this DockerFile. The fact that our Docker picture is not a release image must be understood. It is only utilized for the application developer for testing and development purposes. Typically, the Docker image would contain simply the compiled application files.

C. Security

We cautiously enable debug messages to and from the remote container because security is of the utmost concern. The internal port of the container is mapped to the external port using the docker-compose.yml file, as seen in Figure 3. In this illustration, the edge device's port 2222 is mapped to port 2222 of the application. However, the outcomes of debugging cannot be acquired by the IDE if we merely enable security. The "--security-opt seccomp= unconfined" option is therefore included to permit unrestricted access to the desktop environment for development. The "SYS_PTRACE" capability is a key that is also important in this file. System and memory records are returned outside the container using the key. The development of C++ applications using gdb especially makes use of this functionality.

RELATED WORK

We seek to fill a gap in the literature about remote debug and deployment for EC. Premsankar, et al.'s [9] assessment of edge computing's viability for new IoT applications focuses on mobile gaming. Since network capacity is constrained and causes delay in mobile games, they draw the following conclusion:

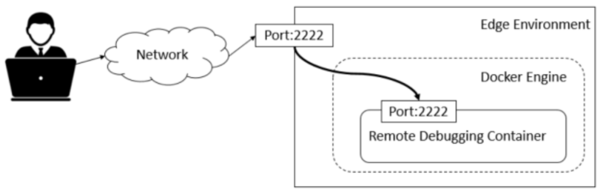

Figure 1. The architecture of the proposed container method.

```
FROM debian:stretch

# Install dependencies to compile and execute C++
appsRUN 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.shRUN 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
loginRUN 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
CMakeRUN wget --no-check-
certificate

https://cmake.org/files/v3.13/cmake-3.13.0-
rc2.tar.gzRUN 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

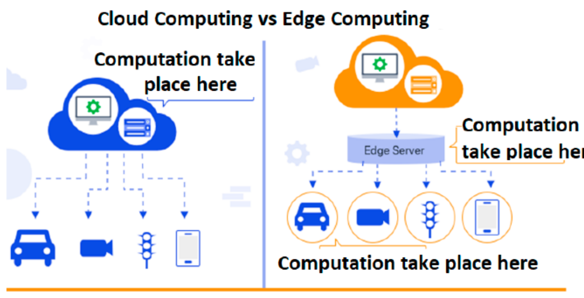
Cloud Computing vs Edge Computing
Computation take place here
Computation take place here
Edge Server
Computation take place here

Figure 4. edge computing for speech recognition in container

Even a small quantity of EC usage enhances the quality of the gaming experience. Plastiras, et al. [10] address the advantages and disadvantages of edge intelligence. They demonstrate an application scenario where a correctly constructed Convolutional Neural Network (CNN) can execute computer vision tasks in real time on edge computing devices. Benchmarking amongst cloud providers' new Edge offerings becomes required as they are released. EdgeBench was created by Das, et al. [11] to compare the Amazon AWS Green Glass and Microsoft IoT Edge services. For CPU light tasks, they discovered that these services offer an appealing alternative to cloud computing.

Conclusion and Future Work

The new standard for IIoT app development is edge computing (EC), however its implementation presents new difficulties. In this research, we recognized such difficulties and suggested a brand-new approach to remote debugging to get around them. Our approach requires no additional configuration work but offers significant development time and security gains. All of the application dependencies needed during compilation and runtime are included in the remote debugging container. Any program may be privately and securely debugged within the edge device by opening a specified port and launching an SSH server within the container. This can address the issues with edge application development. Remote C++ application development was made possible by this study.

ACKNOWLEDGEMENT

The Siemens Turkey Corporate Technology division funded this research project. We appreciate their ongoing assistance and amenities

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

13) /en-us/visualstudio/debugger/remote-debugging-cpp?view=vs-2017