RESEARCH ARTICLE

DOI: 10.34256/ijcci1918

International Journal of Computer Communication and Informatics

A Novel Approach for Remote Compilation using Docker Containers

G. Selvakumar [a,*]

[a] Associate Professor, Department of Computer Science and Engineering , Sri Shakthi Institute of Engineering and Technology, Coimbatore Tamil Nadu, India.

*Corresponding Author
selva@siet.ac.in
(G. Selvakumar)

Received : 18-04-2019
Accepted : 09-05-2019

ABSTRACT: The number of programming languages is getting more and more and developers are facing a tough time in installing all the compilers, libraries and supporting files for the development activities. Most of the time they want to experiment with new technologies, where the efforts required creating a complete environment to run the programs may not be feasible. On the other hand, several companies have started recruiting developers through their online programming platforms. In such situations, it is essential to protect the resources of the server from malicious programs written by the users by purpose or inadvertently. The client environment has to be as lighter as possible and the server environment must be as secure and efficient as possible. There are several existing solutions to meet this objective with plenty of demerits. In this paper we propose a novel method which overcomes most of the problems in the existing solutions and we have experimented the effectiveness of the proposed solution. In our proposed method we develop a docker based sandbox to run the client programs and display the output. We have developed a complete web interface to test the solution and created a backend to manage the users, sessions, tested programs and the outcomes which can be used for analytics too.

Keywords: Docker, virtualization, containerization, remote compilation, hypervisor.

1. Introduction

Remote compilation of code is required in many occasions such as interviewing developers by an organization or experimentation with new technology or development of a learning platform etc., In these situations, an ideal solutions can be a server side application which compiles the code submitted by the user through a web client and executes the code on the server and displays the results. For this purpose, the server should consists of all the compilers and required libraries to support multiple technologies [1-4]. Frequent update in the libraries and requirement for new languages necessitates cumbersome maintenance activities. Another serious problem is about the quality of the code that the user submits. If the submitted code is malicious it has intentions to control the server or to corrupt the server files the damages would be severe. This can be prevented by adding additional security layers to stop particular commands or statements and by limiting the execution time of the submitted programs. However, this will reduce the efficiency of the server to a greater extent. We can figure out the objectives such as the submitted code should not access the server resources, it should access any ports of the server, should not consume resources beyond a limit and it should get executed beyond a certain time limit. To meet all these objectives in an efficient way we need a more novel idea rather than making our servers more Completely [5].

2. Background and the Existing Solutions

To meet the requirements for online compilation with security and efficiency there are several client server techniques were experimented. This section analyses few of them.

One obvious option is using 'chroot jail'. A chroot jail is a technique to isolate a process and its children from the rest of the system. It cannot be used for the processes that run as root. Because root users has all the privileges and breaking out the jail is not challenging. We can create a file system tree where we store all the system files required. There are specific system calls like chroot() to modify the structure and to run the processes in a controlled environment. Here everything is limited only to a certain specific files and other files of the system should be away from access. Their paths cannot be referenced by any means to make a read or write operation. Though it looks like a sensible option, in recent times there are several successful attempts in breaking the chroot jail and it cannot be relied upon completely [6-10]

Another popular solution is using the platform 'Ideone' which is an online compiler and debugging tool. It supports more than 60 programming language. However it is not an efficient solution and it takes huge time to compile and execute programs.

Using VMware/Virtual Box is another solution that we need to consider. A virtual machine (VM) is a software package (OS) or application setting that's put in on software system that imitates dedicated hardware. This option wins in security front and efficiency. However, there is a possibility that virtual machines can be compromised. In those cases, detected and recreating is a cumbersome practise. It consumes huge amount of time. Moreover, the use of virtual machines results plenty of management concerns, several of which might be addressed through general systems administration best practices and tools that square measure designed to manage VMs. There square measure some risks to consolidation, aswellasovertaxingresourcesorprobablyexperiencingoutages onmultipleVMsbecauseofonphysical hardware outage [11].

3. The Proposed Solution

Dockers are an open source tool that helps us to create containers. Containers are isolated sandboxes where developers can deploy their applications. The container runs in a host operating system. These containers guarantees complete process isolation for any deployed application. The container packages everything required to execute the application within it. This includes the program, dependencies, run time, library files, settings etc., It may be a public cloud or any data centre or it may be even a desktop or laptop computer, if we use Dockers, the application would run uniformly and consistently [12].

Here we need to use the images of supporting environments in our containers. An image is an executable package which has everything required by an application. It can be seen as a blueprint which forms the container. The image has to be hosted in a registry to be found by others and use it for their purpose. Docker Hub is a renowned repository for images. On the other hand, containers are runtime instances of the image. The container is responsible for running the application packaged inside it.

A virtual machine image of Ubuntu can be created with necessary compilers and Dependencies and its instances used to run the code submitted by the user.

If there is any malicious code which tries to corrupt server resources or perform unauthorised activities, it would run only inside the container and it could never access anything beyond the container. And the containers are created whenever required and destroyed if their task is completed. So they don't have to be alive all the time. We can have all the required compilers and dependencies inside the same image and use it which is a very efficient option.

The Experiment and Results

To experimentally evaluate the proposed solution

A Docker was installed and compiler images were created. A client server web application was developed to get the programs from the user. A complete client interface with options to select programming language, to enter program and to compile and run was created. The server part was developed in Node.js. The server is responsible for delegating the code to the image and to obtain the result.

From the previous discussions, we have identified that Docker could be used as remote build environment which can reduce the complexities of remote compilation from the server side. We can use the Docker beyond the possibilities of a mere deployment environment and to add these new services with some simple steps. It is very challenging and a slow process to develop and maintain build machines in the production environments.

The developer should create an Ubuntu (it can be any distribution of Linux. In this paper Ubuntu has been taken for experimentation and analysis). After that, the dependencies like libraries and tools required have to be installed. Combined together all these will be responsible for compilation or interpretation. Any developer can do the above steps quickly, however management of the created build system is a complicated one.

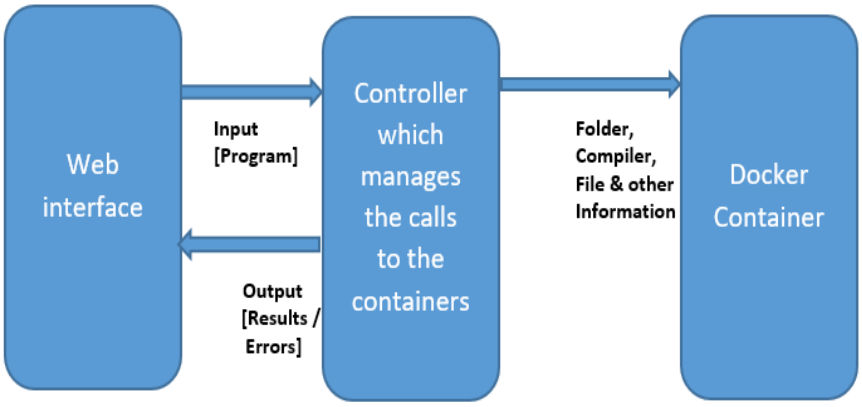

Fig1. Architecture for remote compilation using containers

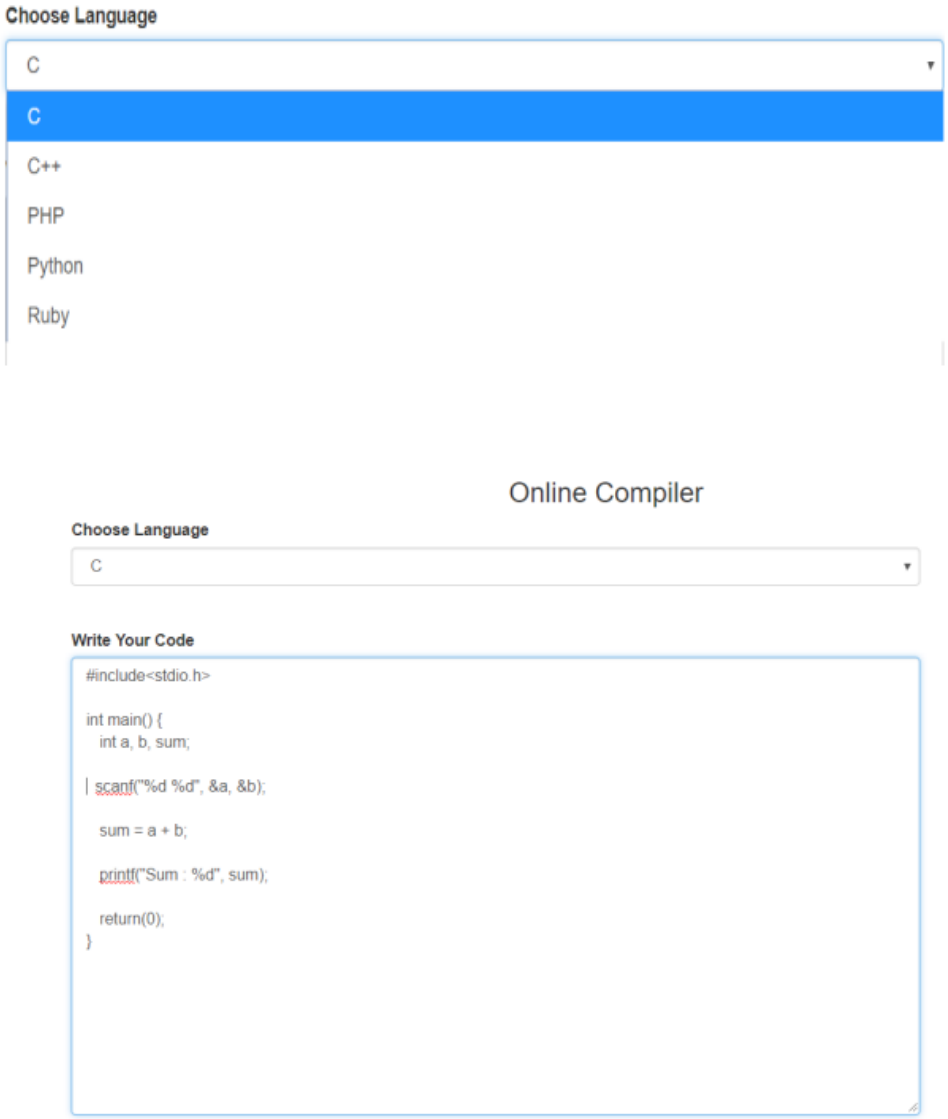

Fig 2. The web interface for entering the code

Because, in this changing technology environment, when there is a requirement for update in a library for a module it poses new challenges in maintaining the system. The development team will be demanded to back up the previous code and data before making new changes. The problems arise when there are several teams are working on the same project and they keep on adding multiple copies for the build environment. Very quickly there will be multiple versions of the code with different variants of the build.

Maintaining the build machine is not practically sustainable and we need solutions which is similar to virtualization where Docker comes to play. Migration of applications to Docker has already gained lots of attention in the industrial sector. In our experiment we wanted to build compilers on remote machines deployed using Docker.

We use C, C++, Java and Python in the application containers. The process of creating the build machine has been given below for Python. Similarly build machines are created for every language and the images are developed.

The steps followed are

Create a virtual machine with the required operating system and Ubuntu is preferred.

1. The compiler tools are installed along with all necessary dependencies and libraries required to compile from sources.
2. Compile OpenSSL and install it at a custom location.
3. Install the required libraries and -dev packages needed to build Python from sources. In other cases libraries for the respective language has to be installed.
4. Compile Python from sources, pointing it to our custom-installed OpenSSL.
5. Compile all needed Python modules and dependent libraries that are linked against OpenSSL, again pointing it to our custom OpenSSL.

6. Build the application and verify that it is correctly linked to our custom OpenSSL.

Manually installing the libraries from the sources and pre compiled system packages are a very long process and error prone. Along the way, some libraries are installed via sources, some pre-compiled and some via the system packages.

Docker is considered as a deployment environment in most of the cases. It is used for the toolset around that environment. It has greater flexibility and interactive nature to support the build process. The images are created with the help of a file which is like a windows batch file. This is known as Docker file and this file is processed line by line in the process of creating an image. Image metadata is created using the Docker file. Here we provide a sample Docker file.

#This is a sample Image

FROM ubuntu

MAINTAINER demousr@gmail.com

RUN apt-get update

RUN apt-get install –y nginx

CMD [“echo”,”Image created”]

FROM gcc: 8.2

COPY.

/usr/src/myapp
WORKDIR

/usr/src/myapp
RUN gcc -o myapphello.c

EXPOSE 3000

CMD ["./myapp]

Cpp
FROM gcc:8.2

COPY.
/usr/src/myapp
WORKDIR
/usr/src/myapp
RUN g++ -o myapp hello.cpp

EXPOSE 3000

CMD ["./myapp]

Python
FROM python:3

WORKDIR
/usr/src/app
COPY requirements.txt

./

RUN pip install --no-cache-dir–r requirements.txt
COPY. .
CMD ["python", "./hello.py"]

Php
FROM php:7.2-cli

COPY .
/usr/src/myapp
WORKDIR /usr/src/myapp CMD ["php", "./hello.php"]

Ruby
FROM ruby: 2.5

RUN bundle config --global frozen

WORKDIR
/usr/src/app
COPY GemfileGemfile.lock.

RUN bundle instal

COPY. .
CMD ["./hello.rb"

Fig3. The web interface for input and output

The above is sample Docker file contents for multiple images. We have experimented with C, C++, Python, Ruby and PHP. The base machine layer is the first part and we start describing about the further layers based on that.

Fig.2 and 3 are depicting how the user interface functions to run the compilers on the Docker backend. It is a simple web page where the user has options to enter the program and input parameters. The user would never have an idea what happens in the backend.

The backend part has been implemented by Node.js. Node.js is an open-source, cross-platform JavaScript run-time environment. Usually JavaScript is embedded in the HTML files and they are executed in the client side, thus called client side scripting. However Node.js enables us to develop server side scripting for dynamic web pages. The following section has a fragment of Node.js code which is responsible for understanding the language choice from the

user and to save the program file in the server with proper extension.

```
app.post('/get_code', function (req, res) {
const data = {};
data["lang"] = req.body.lang;
data["code"] =req.body.code;
data["input"] = req.body.input;
res.json(data);
varlang_string = data.lang;
varcode_string = data.code;
varinput_string = data.input;
if (lang_string === 'c') {
varlang_string = data.lang;
varcode_string = data.code;
varinput_string = data.input;
if (lang_string === 'c') {
fs.writeFile('hello.c', code_string, function (err) {
if (err) {
console.log(err);
}
else{
console.log('hello.c was updated.');
fs.writeFile('input_c.txt', input_string, function (err) {
if (err) {
console.log(err);
}
else {
console.log('input_c.txt was updated.');
}
var spawn = equire('child_process').spawn,
ls = spawn('cmd.exe', ['/c', 'build.bat']);
ls.stdout.on('data', function (data) {
console.log('stdout: ' + data);
}
ls.stderr.on('data', function (data) {
console.log('stderr: ' + data);
}
ls.on('exit', function (code) {
console.log('child process exited with code ' + code);
}}
}}
else if (lang_string === 'cpp')
fs.writeFile('hello.cpp', code_string, function (err) {
if (err) {
console.log(err);
}
else {
console.log('hello.cpp was updated.');
fs.writeFile('input_cpp.txt', input_string,function (err) {
if (err) {
console.log(err);
}
else {
console.log('input_cpp.txt was updated.');
}
var spawn = require('child_process').spawn,
ls = spawn('cmd.exe', ['/c', build_cpp.bat']);
ls.stdout.on('data', function (data) console.log('stdout: '+ data);
});
ls.stderr.on('data', function (data)
console.log('stderr: ' + data);
});
ls.on('exit', function (code) {
console.log('child process exited with code ' + code);
});
});
}
});
```

The server side scripting extracts the program content and saves it with appropriate .c, .cpp, .py or other extensions. Based on that the suitable compiler image is invoked and the code is getting executed. The output generated in the backend is displayed in the front page as illustrated in the figure 3.

5. Conclusion

The developed system has been thoroughly tested with malicious code, improper arguments for the function calls and attempts to access the files inside the server. However, none of these actions made the server to crash or lose its efficiency. All the server requests were handled only inside the container and the user could never identify any difference that their code is running inside a sandbox. The server side scripting was done with node.js which is very

flexible in controlling the backend operations. With the help of framework created, it can be extended for any number of programming languages. This can be used for any web based programming interviews or competitions without worrying about security and it provides better control of entire workflow.

References

[1] BabakBashari Rad, Harrison John Bhatti, Mohammad Ahmad, An Introduction to Docker and Analysis of its Performance, Inter. J.Com. Sci. Net. Sec. 17 (2017) 228-235.

[2] B.Varghese, L. T.Subba, L.Thai, A. Barker, Container-Based Cloud Virtual Machine Benchmarking, 2016 IEEE Intern. Conf. Cloud Eng., (2016) 16039489.

[3] A. M. Joy, Performance comparison between Linux containers and virtual machines, Paper presented at the Computer Engineering and Applications, 2015 Intern. Confe. Adv. Com. Eng. App. (2015).

[4] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer, Cedric Notredame, The impact of Docker containers on the performance of genomic pipelines, Peer J. 3 (2015) e1273.

[5] Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu, Wei Zhou, A Comparative Study of Containers and VirtualMachines in Big Data Environment, 2018 IEEE 11th Inter. Confe. Cloud Comp. (2018)

[6] S. Abdulla, S. Iyer, S. Kutty, Cloud based compiler, Inter.J. Stu. Res. Tech. Man. 1 (2016), 308-322.

[7] Xiaolin Geng, Xuewen Zeng, Linlin Hu and Zhichuan Guo, An novel Architecture and Inter-process Communication Scheme to Adapt Chromium Based on Docker Container, Inter. Congress Info. Comm. Tech. (2017) 691-696.

[8] E.N. Preeth,Jaison Paul Mulerickal, Biju Paul, Yedhu Sastri, Evaluation of docker containers based on hardware utilization, 2015 Inter. Confe. Con. Comm. & Comp. India, (2015) 19-21.

[9] Fawaz Paraiso, Stéphanie Challita, Yahya Al-Dhuraibi, Philippe Merle, Model Driven Management of Docker Containers, 2016 IEEE 9th Inter. Confe. Cloud Comp. (2017).

[10] David Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing, 1(2014)

[11] Krishan Kumar, Manish Kurhekar, Economically Efficient Virtualization over Cloud Using Docker Containers, IEEE Inter. Conf. Cloud Comp. Emer. Mark. (2017).

[12] Pankaj Saha, Piotr Uminski, Evaluation of Docker Containers for Scientific Workloads in the Cloud, Proceedings of the Practice and Experience on Advanced Research Computing (2018)

About The License