

Automated Dockerization of Python based Web Apps

MSc Research Project
Cloud Computing

Shryni Kedambadi Shreekar

Student ID: X18199011

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Shryni Kedambadi Shreekar
Student ID:	X18199011
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	17/08/2020
Project Title:	Automated Dockerization of Python based Web Apps
Word Count:	6800
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	Shryni Kedambadi Shreekar
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automated Dockerization of Python based Web Apps

Shryni Kedambadi Shreekar
X18199011

Abstract

The race to computing is at the peak and among many ways to achieve it is by using cloud computing. To enable faster deployment of applications and ensuring that environment of deployment remains consistent even after deployment, Docker seems one of the best suited approach. One the challenges which people are facing is the problem associated with ease of use and ease of making Docker based images. This is especially true for people who are naïve or sometimes even experienced individuals who want to explore Docker as a solution. Virtualization To solve the above problem was born an application which can ensure ease of use of making and testing applications using docker. Virtualization also has been trending lately for providing on demand services on the go on cloud. Even though the final aim is to make an application which will allow use of any software programming language, we selected python and especially python based web applications. Why this choice? Majorly driven .by need of time where multiple people are focusing on making applications around python. As mentioned in stack overflow python is seen as most promising and used language from past two years. In our use case we will see how to make use of the power of python itself to develop an application which can perform the pain taking task of making dockerized version of application in few clicks.This simulation is based upon docker toolbox, Windows Operating system, ngnix for load balancing. The dockerized application can be easily uploaded in any docker supportive environment especially if any cloud based environment like Amazon AWS EC2.

Keywords: Python, Docker, Automated, Dockerization, Virtual Machines, nginx, Cloud Services

1 Introduction

The recent boom in the technological development of IoT, serverless computing etc has been creating great scope for Cloud(Buyya (2010)). Within that to improve scalability and portability of any web applications, and to support cross platform the top approaches used in todays world is either containers or virtual machines. Both having their own advantages, at times its essential that we choose the right approach for the situation and the type of application. Virtualisation mainly focuses on optimising the server capacity .But this being packed with the operating system itself cannot be called light weight and with the need of hypervisor for enabling virtualization. Docker works on the principle of containers which can work by sharing the kernel of host operating system side by side other container. Scalability is achieved easily without the need of additional servers. Implementation of either can be decided upon the use case. As per (Kohgadai (n.d.)),

number of organisations who are not dockerizing more than 10 percent of the applications have not advanced as much as the ones that did.

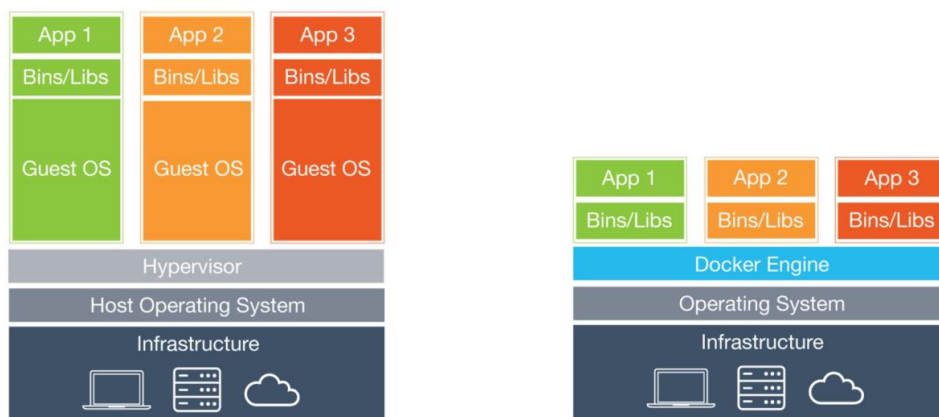


Figure 1: Dockerization vs Virtualization

By standard definition, Docker is “an open-source project which automates the deployment of software applications inside containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux.” With the advancement in technology, docker is highly opted over virtualization mainly because of its open source platform. Docker allows in deployment process and as well as development process. The rather attractive feature of docker is its feature of providing a separation of the infrastructure from the application(Merkel (2014)). With the rise in adaptation of docker the focus of application development is easier rather than having to worry on the deployment process of the application(Naik (2016)).

Because of this the managing of infrastructure is more appealing than ever before. Not alone with Docker but, docker container based virtualization has been trending as one of the most feasible development process in the developer world (Naik (2016)). This feature of building docker images and deploying makes the delay between the code development and Code deployments very less , hence allowing not distracting developer on other menial tasks.Also as opposed to Kubernetes, which runs in cluster, Docker can run as a single node. The use of containers is the main advantage here which when compared to Virtualization is a definite-go. Even in a decoupled manner, the program and the environment of the code can be combined together into a container and then deployed. (Dua et al. (2014)). Hence the applications can be deployed to various environment be it any personal system or on any cloud based environment with ease and effortlessly. Containers being light-weighted, without the Operating System unlike Virtual Machines have attracted many developers. Major leading companies like Uber, BBC, Spotify , Paypal etc are the early adopters of docker platform.

Python has recently emerged as one of the most promising programming language for developing web applications on the go. In web development, the use of Python is abundantly more than what we think. Alone in the year 2018, 122,864 web applications were developed using Python and 93,197 of them were hosted worldwide.

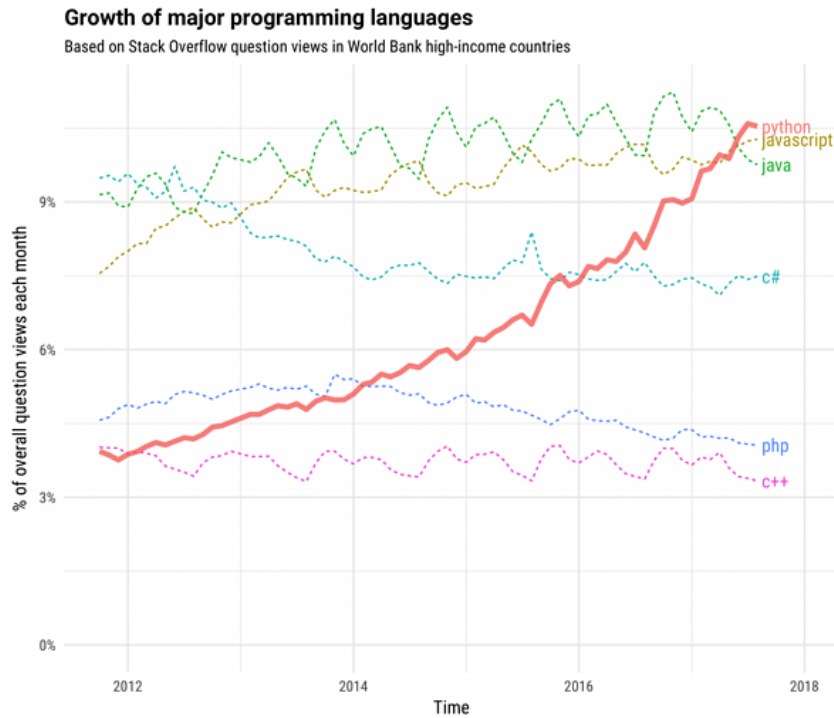


Figure 2: Rise of Python for web application development (Oberoi (n.d.))

The above graph clearly shows that number of questions raised in python over years has been increasing and trend continues to increase. Used majorly for machine learning , the second best use case of Python was in developing web applications.

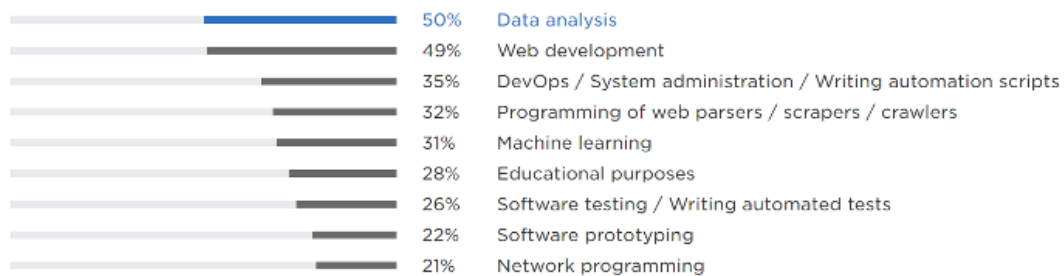


Figure 3: Usage of python in Web application development (Oberoi (n.d.))

The data is collected from opensource.com which indicates the usage of python in web application.

Two fore runners in this area within python is Django and Flask. Django provides a full blown spectrum of web application while flask provides a micro level rest framework. Number of adopters of web application using Django is far more than other frame works. A full blown web application includes ability to serve web pages at the same time allow developers to make use of features like templating which is advantageous.

To evaluate our implementation we have worked using Docker Toolbox on same operating systems instead of multiple hosts and clouds. For ease of use, developed python web apps have been used from GitHub to apply our implementation on them and evaluate the dockerization performed. Further we have attempted at installing nginx as well within the virtual environment created within the execution.

The proper identification of the objectives of the research by the researcher are considered as one of the vital stats. The objective of this research is,

- To see any existing solution which can improve the usability of docker adoption even to users not very familiar with Docker.
- Reducing the complexities associated with Docker commands help in adoption of web app migration.

Research Question: Will hiding complexities with Docker commands and providing an easy to use user interface improve adoption rates and increase the number of applications which are dockerized?

The remainder of this paper has been organized as follows. Section 2, Related Work takes us through the major study and researches done in the field of Virtualization and Dockerization, where we critically analyse the progress and depict the areas lagging concerns. Further section 3, Methodology explains our approach on how the python web apps have been dockerized in django framework. Next section, Design specifications, depicts the techniques and framework that supports our implementation. Further in the Implementation section, we discuss the implementation of our proposal. In Evaluation section we show the results of our implementation achieved. We conclude with a summary and future work and scope on which this proposal can be advanced.

2 Related Work

Docker has been in the market for quite sometime and multiple research has been done in the field. A study on the difference between what docker offers and what the precession of it offered and how they are different has been conducted by (Bashari Rad et al. (2017)). Elaborating the measures on measuring the performance from application perspective prior and post adoption of Docker. Our research proposal is greatly influenced by few performance metrics proposed here.

Disruption of cloud services around the world leads to opportunities in terms of development of container based solutions which can be easily managed (Docker (n.d.)). Unified operations standardized technological practices and operational patterns helps companies in better in the field of technological evolution. This leads something like single source of cluster for various environments like development to Production by leveraging existing teams and processes. Iterative or repeatable action causes faster delivery and makes use of existing team and environment to do better. Consistent security due to constantly improving container platforms offers good support and reduces the risk due to threats. Data center utilization is increased which again falls back to standard patterns and allows capabilities be generated across consolidating services. With redundancy removed and better performance will lead to reduced IT cost .

Over time docker has expanded more than the normal role of earlier virtualization done by VMWare (Harzog (2014)). There will be a possible clash of application developed in different programming languages if not properly segregated. (Nardelli (2017)) mentions few interesting steps to get from legacy to cloud but at the same time does not say how to adopt the same easily. It also does not take into account the practical problems which would be faced by organisation while moving to docker. This is where the automated tool proposed by us comes into play.

(Erdenebat and Kozsik (2019)) discusses from PRISMA perspective and the advantages PRISMA offers over others. The solution which we are implementing will be compatible on PRISMA platform but they won't have custom monitoring as we made our solution generic and not tied to particular vendor. (K91107541 (n.d.)), discusses on dockerising F5 based products, our solution implemented does not use F5 products but surely will go with future scope due to high demand of F5 products. Also dependency injection of mule which is in its core part been dockerized and the image of same can be used for deployment. The core runtime which forms the heart of mule based applications have been successful in this transformation (Seth (2018)).

2.1 Virtual Machines and Containers

Designing and systems on the present day cloud with the aid of either dockerization or virtualizations is not as straight forward as it seems (Cunha and Laranjeiro (2018)). There are several parameters to be considered such as isolation, communications, ability to scale and portability. Cloud computing has started invading the consumer area where it attempts to offer computing needs based on virtualization and dockerization (Kazi et al. (2012)) (Marinescu et al. (2014), Seo et al. (2014), Felter et al. (2015))

The maturity level of container application starting from no container to all the way to container integrated process are in stages .

- No Container: Container not been adopted
- Container Engine : Container is been used in some places which are easy to deployed over
- Orchestrated Containers : Applications which talk to each other have been converted to docker
- Container Platform Integrated Processes: Seamless flow of new and old application happens through end point. It is at this point the accelerated development/improvement becomes visible in the organisation.

,

Also, the cloud applications are prone to crashes and the response time is still slow, especially for many web based applications. According to (Carvalho and Kim (2017)) because of the complexity involved in the virtualization in physical hosts, and the interference between multiple virtual machines complicate the response and resurrection time.

Parameter	LXC	Warden	Docker	OpenVZ
Process Isolation	Uses pid namespace	Uses pid namespace	Uses pid namespace	Uses pid namespace
Resource Isolation	Uses cgroups	Uses cgroups	Uses cgroups	Uses cgroups
Network Isolation	Uses net namespace	Uses net namespace	Uses net namespace	Uses net namespace
Filesystem Isolation	Using ch-root	Overlay File system using overlayfs	Using ch-root	Using ch-root
Container Lifecycle	Tools lxc-create, lxc-stop, lxc-start to create, start, stop a container	Containers are managed by running commands on a warden client which talks to warden server	Uses Docker daemon and a client to manage the containers	uses vzctl to manage container lifecycle

Figure 4: Implementation of Containers Compared(Dua et al. (2014))

IaaS being the basic cloud computing model, offers physical or virtual machines. This works along with hypervisor such as KVM etc, hence running the VMs as guests. But to support many virtual machines and to be able to support scalability as per user requirements pools of hypervisors might need be used (Upadhyay and Lakkadwala (2014)). To be able to deploy and applications in this case, the initial VM set up is overwhelming.

To deploy their applications, cloud users install operating-system images and their application software on the cloud infrastructure.

A study on providing support for container life cycle management with Virtual machines from within the IaaS layer itself has been performed by (Dua et al. (2014))(Zhanikeev (2017)). But a very few PaaS implementations support this approach.

An arm based embedded platform that supports containerization was discussed and implemented by (Bin et al. (2019)). In accordance with hardware assisted virtualization, AMD introduced virtualization support extension in the ARM v7 and 8.

2.2 Containerizing or Dockerizing an application

Every web application usually has external dependencies such as libraries, modules, framework etc. There might be situation where there exists two or more applications on the same framework but run on different versions which could be compatible with each other or might not be (Ijtihadie et al. (2017)). In today's fast paced world, everyone targets at reducing the deployment time and also multiple deployments within a short period of time. DevOps is gaining popularity on this term. Meanwhile containers come to the rescue, where the dependencies are packed into the docker file. They are securely consolidated and isolated so cross platform issues and version issue are sorted out. The similar steps of deployment can be followed in developing, staging, and deployment process. The containers that use Linux primitives sandbox applications and hence increasing the security too (Bin et al. (2019)).

PaaS has bridged the gap between the developer community and complex IT infrastructure in cloud. But for every developer to be able to develop an application irrespective of the target platform is an advantage to increase productivity in development. Devops environment with Google App Engine, Docker, Kubernetes etc make this possible (Li et al. (2017)). A virtualization approach based on the concept of double flowvisors for production where the hardware which is underlying and the the apps have been given unified access was researched. This proved to be better in terms of resource utilization (Xingbin Yin et al. (2013)). The apps for which there is always conflict in the flowtable entries, multipath routing can be introduced. Flowvisor has been implemented similar to how an abstraction layer operates in between the forwarding paths and user control.

One step ahead of this, an orchestration based on the software has also been explored. This works in between the ontology and protocol based systems. There are several middleware approaches in attempt of dockerizing the applications for cloud. But using the ontology approach, the creation of ontologies and implementing them in a way such that the mapping between layers is simple enough is not achieved (Pahl and Carle (2013)).

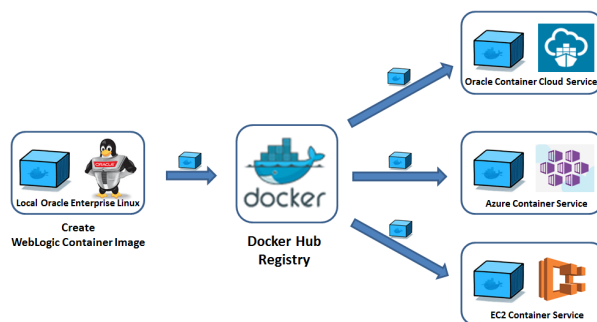


Figure 5: Dockerizing web apps

Docker assists in leveraging big data analytic in a efficient manner. But with overlay networks used, stability of network transfer is increased, however this comes with the cost of reduced performance of 10percent (Bin Xie et al. (2016)).

2.2.1 Microservices

With the advancement in technologies and Devops, many companies have moved into Microservices architecture as well. Mainly E-commerce applications which makes it easier to manage and reduce the down time of entire application . DevOps continuous integration and continuous deployment using RunDeck, Docker, as well as Kubernetes have aided this transformation. The docker images pushed in Docker hub, can be a source from which docker images can be pulled but once concern is to maintain the flow between the micro services so that it doesnt break. Recently ”-link” command from Docker has been deprecated. For this purpose , a single yaml file was introduced which runs along the docker compose, which in turn keeps the containers connected. Still the emphasis was on the creation of yml file. In Docker-compose , in production environment doesn’t require any manual configuration(Rajavaram et al. (2019)). While in case of Kubernetes supports calling multiple pods, our approach sets up the portability in much lesser time.

Virtualization and automation have also been used in cases such as integrating the mobile terminals into the cloud services which is then considered as a resource called as Testing as a Service(TaaS) (Tao et al. (2015)). Here the apps and their vulnerability can be tested as well. This aids in building flexible testing environments in different cases(apps). This approach can be integrated with our proposal to further enhance the adaptability hence providing both faster containerization and in the long run providing secure environments on the hosted cloud for the applications.

To summarise, while existing approaches have addressed the concerns of security of apps hosted on cloud, introduced flowvisors, most dont address the complexity of this migration. In comparision with Virtual Machines, Containerization is the most appropriate approach for web apps which need to migrated which reduces the overhead of configuration set up. We aim to ease the process by automated dockerization of web apps, to increase adoptability. How can we cause docker adoption easier and simpler.? Especially in the stages of no container to Container engine stage what can be done so that even teams hesitant to get started with docker be able to see the power and ease of getting started with user interface is missing. This is what we are trying to achieve.

3 Methodology

The core logic is written in a way that it become easily extensible to include new type of programming languages in future and also support non web application. Eventually keeping the core logic same and be reused to create a complete web application which can perform these operations on fly.

Our approach to implementing the proposal embraces a mixture of various ideas and frameworks as below for the successful execution.

- The ability of docker to be run as docker client using cli or shell.
- The ability to download all dependencies used by python using pip.
- The ability to run command line from python with the help of sub process module.
- A simple user based interface to take various inputs from user to perform validation and run the process

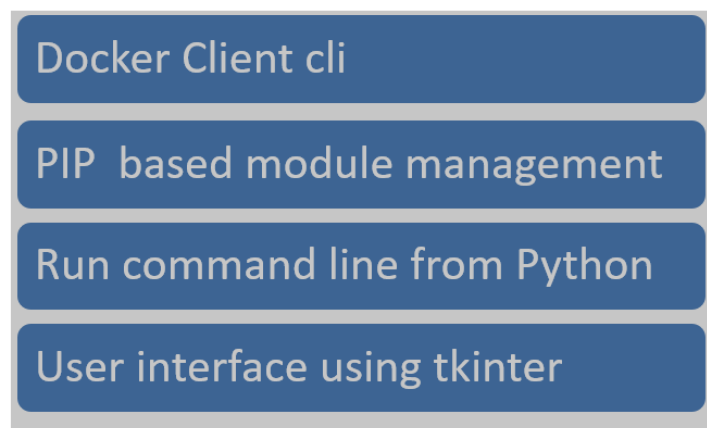


Figure 6: Main keypoints of implementation

The approach used here to harness the advantages of python programming language to call shell commands and corresponding stacks around it.

The methodology used was agile in nature with features being added as and when found , as well as use new features when required. This was majorly due to research nature of the project that led to agile mode. With pip package management system we were able to install and manage packages written in python. While python has a variety of graphical user interface frameworks, tkinter framework was found more appropriate for our purpose mainly because its built into python's standard library. One rather important thing is because of the cross platform nature of tkinter which workks equally well with any Operating Systems, which greatly supported outr research proposal. Being a lightweighted framework, it was more compelling for the cause.

Another module Subprocess deals with spawning process in various operating system from python. This is the back bone module for the application. Using this module helps us in ensuring various docker related commands and execution commands to generate

docker artifacts becomes possible. Ability of it to run in back ground in back ground is another cool feature. Nginx was selected as a web server to provide production based environment with ability of the web applications to deal with large number of requests. The memory consumption and ability to deal with large number of requests parallel and consistently makes it one of the best choices for web server. Ability to use with a simple configuration file was another reason for selecting this

The experiments were carried out and tested by picking python based web applications from public source code repositories such as GitHub, and compiling them within our setup to create a docker file and then being able to run the image on our targeted platform.

1. Write the python application code.
2. Containerize/Build Docker file
3. Deploy to target of choice (local system/cloud)

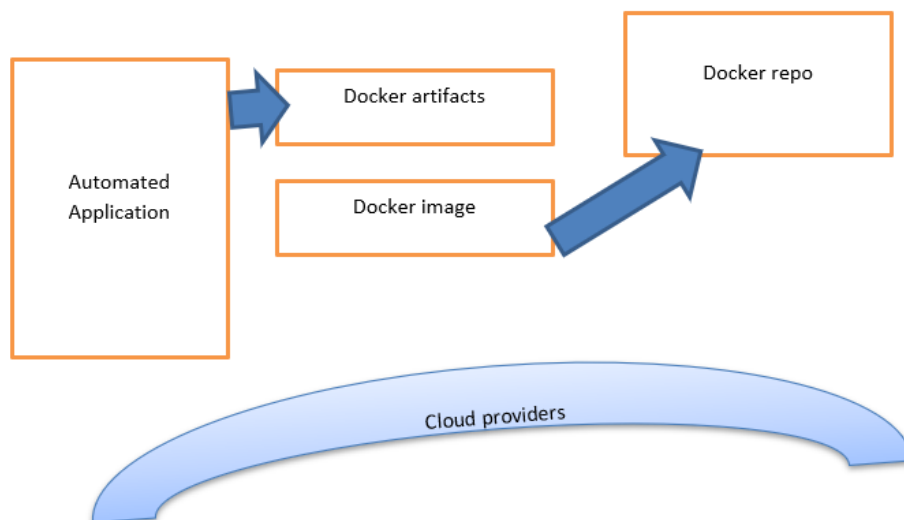


Figure 7: Flow of automated application

The Environment requirements are as stated below:
For setting up our implementation we had the following setup done:

- Python 3.x installed. Any version of Python 3.x installed most preferred being the latest one.
- Docker Desktop installed. This is required for running docker commands. The challenge with above is that it only supports windows professional version onwards and hence cannot be used if using windows home.
- In case of inability of docker desktop to be installed, we can make use docker client for lower Operating system versions

4 Design Specification

4.1 Input Specification:

Input for this application is selecting the directories like location of python and the corresponding location of source code. Also input like name of project and application is also given as input. First level of input specification is given in the form of table below:

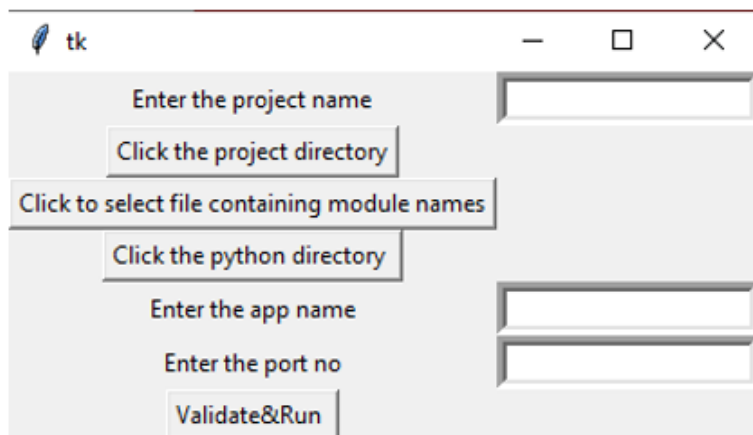


Figure 8: Input Tab

Table 1: Input Specification

Description	Price (\$)
Project directory	Directory holding source code to be dockerized
requirements.txt	File containing all the dependencies of python which can be used by pip to install in virtual env which can be later used in docker
Python installation directory	Allows selecting different version of project based on requirement
Docker directory	Allows selecting different docker directory as preferred by client
Project name	Generally refers to name of project
Application name	Refers to name of application which would be used in tagging
Port number	Refers to the port number to be exposed by application from docker

The input specifications are explained as below:

- Project directory : A project consisting of python based web frameworks. Most targeted ones are Django due to wide usability and higher adoption rate

- requirements.txt : A file which is known as requirements.txt which contains all the modules to be used by project. This file will contain modules listed one below other so that they can be installed in virtual environment. This can then be transferred in container to used when the docker application is started.
A sample containing all modules of python is shown below
 - Django
 - Markdown
 - gunicorn
 - requests
 - martor
- Python installation directory: This may look a bit absurd, but the aim is to support different versions if user wants so. We have given the ability to select python version of choice.
- Docker directory: This allows user to select the location where docker is installed. Again this is used to provide ability to select docker version of choice.
- Project name: This indicates the name of the project which needs to provided by end user. This is generally same as the final directory where user stores the project.
- Application name: This is the name which will be used while building the tag using docker command. This allows project name and application name to be different even though they can be same also.
- Port number: This allows user to enter port number which can be exposed from docker when the application is run so that they can use access the application from browser.

4.2 Output Specifications:

There are lot of variations of output which can be produces as a part of this application. The most common one is the build artifacts which are generated using docker build command. This is the lower level output produced by the application. The second level is the ability to run the docker application and ability to access it from web browsers. Though not implemented, eventually we will have the ability to push the docker artifacts to central repository. The artifacts thus uploaded can be easily uploaded on any public cloud network or kubernetes platform.

5 Implementation

For the purpose of implementation we have selected Python as the programming language to develop user interface and logic handling. The major modules of Python which were:

- Tkinter which is the GUI part of python.
- Subprocess deals with spawning process in various operating system from python
- Nginx was selected as a web server to provide production based environment with ability of the web applications to deal with large number of requests

The main advantage of tkinter is that it can be used as compatible with different operating systems like windows, unix. Contains wide variety of widgets to select from , though for the project we made use of input box which is also known as entry ,file dialog and button. Also, two variations of file dialog used are normal file dialog which allows selecting file and directory dialog which allows selecting directory. This is required since as specified in input specification we deal with files as well directories. 4) Call back are way for achieving event action pattern. This allows controlling certain behaviour which is required either to capture name of files or directory or deal with validating input parameters which are specified by user.

We have used subprocesses in our implementation. The advantages being:

- Can be included in build module within python. No separate installation required for this.
- Allows executing shell or command prompt commands.
- Allows getting status of run status
- Allows provides ability to get task statistics like error description in case of error occurrence.
- Allows running commands even after activating virtual environment. This was very much essential since we would run a host of commands post activating virtual environment

Along with above we also made use of nginx which acts as a reverse proxy to hit the actual calls. The main reason to adopt nginx in our implementation is

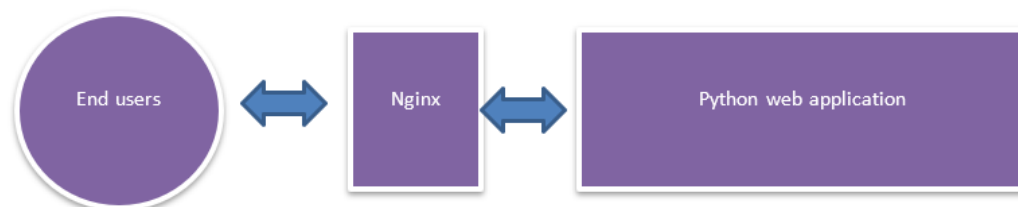


Figure 9: NGNIX explained

- Ease of installation both from docker and non docker perspective
- Allows easy serving of static files including images , javascript etc.
- Ability of caching thus giving us increased performance.
- Ability to handle more concurrent connections as compared to other competitors.
- Less memory usage as compared to competitors.
- Excellent compatibility with various web related frameworks.

- Allows traffic distribution with the help of excellent load balancing capabilities.
- Excellent community support and low risk related features.

From code perspective we have following levels of handling.

5.1 Handling User interface interactions

Tkinter of python has widgets and call backs. Widget serves as input or action items while callback is used for corresponding action to be executed on event. In our case we used more than one call backs

- The first type of callback is associated with handling requirement file. This deals with selecting or getting the location of requirements file which contains details of all modules which are used by application

```
def handleSelectRequirement():
    requirementFilePath= fd. askopenfilename( initialdir = "/", title =
        "Select file")
    print ( requirementFilePath )
```

In the above given code we make use of askopenfilename function of filedialog module of tk. This allows the user to select a file and the corresponding path is returned back which can be stored. The above call back is associated with requirement button i.e. as soon as user clicks on this the above call back gets invoked.

```
tk.Button ( text = 'Click to select file containing module names',
            command = handleSelectRequirement ).grid( row = 2 , column = 0)
```

- The second type of call back is associated with handling project directory. This deals with selecting or getting the location of project directory which contains source code of the project..

The corresponding call back is slightly different since it makes use of directory function to get the details of directory.

```
def handleSelectProject():
    global projectDirPath
    projectDirPath = fd.askdirectory()
```

Please note the use of global variable is to indicate that the variable of global scope is given the value.

- The third type of call back associated is with selecting python directory . Again call back is used for this purpose. Since python path is also directory hence we make use of ask directory function to capture the path.
- We also made use of Label and Entry to capture values.To retrieve value from input boxes also known as entry we define variables which can be used within input like StringVar and so on.

- We make use of grid layout so that each item can be arranged in the form of rows and columns Row number is specified by row attribute and column is specified by column attribute.

5.2 Construction of bat file/shell script

After filling values when the user clicks on Validate and Run button, we have a logic which generates bat file. The making of bat file is based on the input filled by user. The bat file consists of series of commands . The steps can be organised as below.

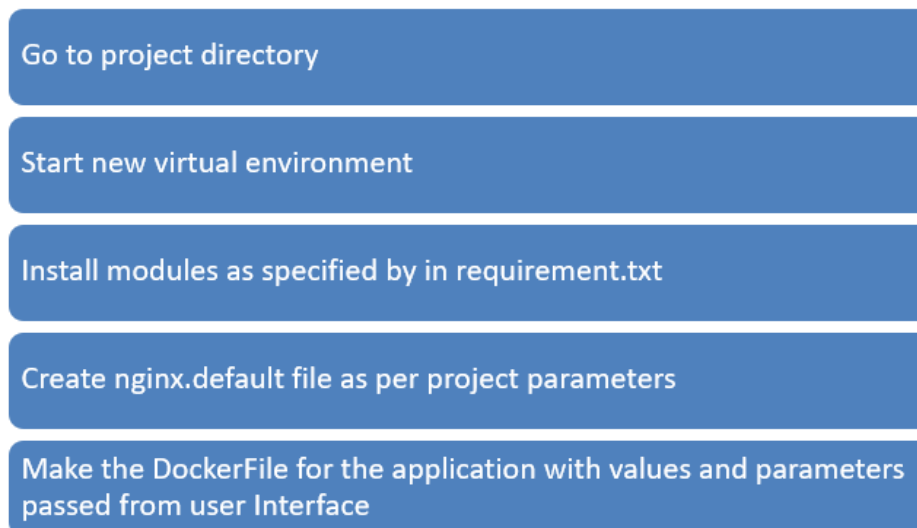


Figure 10: Bat file creation

```

f =open ( 'run.bat', 'w' )
f.write ( "cd " + projectDirPath + "\n" )
f . write ( pythonPath + 'python -m venv venv' + "\n" )
f . write ( "cd venv \n" )
f . write ( 'if exist ' + projectDirPath + "\\" + projectName + "\\static" +
' ( rmdir /s/q ' + projectDirPath + "\\" + projectName + "\\static ) \n" )
f . write ( 'if exist ' + projectDirPath + "\\" + ". pip_cache" + " ( rmdir
/s/q " + projectDirPath + "\\" + ". pip_cache ) \n" )
f . write ( "call . \Scripts\activate && echo martor >> requirements . txt
&& echo gunicorn >> requirements . txt && pip install -r requirements .
txt && cd " +
projectDirPath + "\\" + projectName + " && python manage . py collectstatic
&& cd . . && mkdir . pip_cache && docker build -t " + appName + " . " )
f . close ( )
  
```

Which can be explained as:

- First create a file with name run.bat
- Post going to project directory we start a new virtual environment and activate it.

- We remove some of the directories if it exists. They are cache directory of pip module as well as static folder. The static folder is used for collecting html / javascript
- Install the modules as specified in the requirements.txt.
- Create a new set of static and cache files.

We also create a nginx file for the purpose of load balancing. The two important parameters which are used are port number specified by appPort and projectName.

5.3 Running the constructed file and capturing the corresponding output

This allows creation of DockerFile which will be used finally for dockerization of application. Steps are as below:

1. Install Python 3.7 or above
2. Install nginx and generate files for nginx server like server.log and error.log
3. We also install all the pip module inside docker
4. We expose the port which needs to be made available outside and start the server

6 Evaluation

On running the bat file with the help of subprocess module, Subprocess module allows executing bat file and corresponding gives execution status and execution description. In case of error it also shows error description with complete details of error.

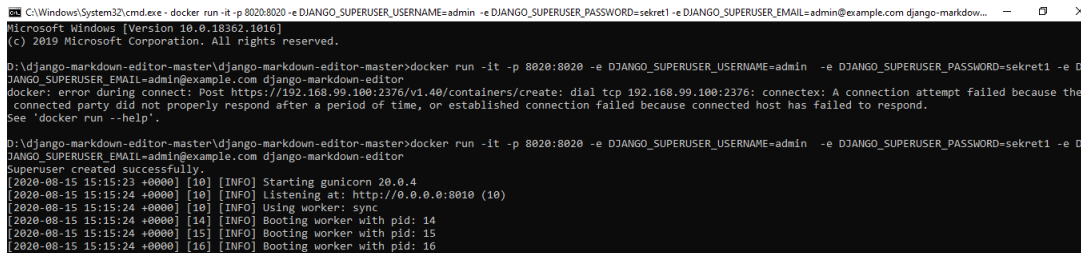
The validate method plays a very important role in making sure that all the pr required softwares for docker execution and running is covered before start of any process. In its absence users would not be informed of any missing pieces which would have caused a problem and hence an adverse impact in the adoptability. To achieve the same subprocess . run(['docker']), is called. The output of these process is captured using stdout = subprocess . PIPE , . universal_newlines = True also ensures different types of new lines due to difference in operating system is covered. We check the output of above command with the help of process . returncode. If process . returncode return 0 it implies process completed successfully implying docker is installed. In this case we invoke the methods of runCommand file which will cause file generations to take place. If the above process fails then we halt any progress on same. Corresponding message is given back

Post successful run we can do a sample run using docker command.

```
docker run -it -p 8020:8020 -e DJANGO_SUPERUSER_USERNAME=admin
-e DJANGO_SUPERUSER_PASSWORD=sekret1 -e
DJANGO_SUPERUSER_EMAIL=admin@example.com django-markdown-
edito
```

6.1 Experimentation

On submitting the input parameters, our implementation starts creating a virtual environment and start packing the web app into docker file.



```
C:\Windows\System32\cmd.exe - docker run -it -p 8020:8020 -e DJANGO_SUPERUSER_USERNAME=admin -e DJANGO_SUPERUSER_PASSWORD=sekret1 -e DJANGO_SUPERUSER_EMAIL=admin@example.com django-markdown-editor
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\django-markdown-editor-master\django-markdown-editor-master>docker run -it -p 8020:8020 -e DJANGO_SUPERUSER_USERNAME=admin -e DJANGO_SUPERUSER_PASSWORD=sekret1 -e DJANGO_SUPERUSER_EMAIL=admin@example.com django-markdown-editor
docker: error during connect: Post https://192.168.99.100:2376/v1.40/containers/create: dial tcp 192.168.99.100:2376: connect: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
See 'docker run --help'.

D:\django-markdown-editor-master\django-markdown-editor-master>docker run -it -p 8020:8020 -e DJANGO_SUPERUSER_USERNAME=admin -e DJANGO_SUPERUSER_PASSWORD=sekret1 -e DJANGO_SUPERUSER_EMAIL=admin@example.com django-markdown-editor
superuser created successfully.
[2020-08-15 15:15:23 +0000] [10] [INFO] Starting gunicorn 20.0.4
[2020-08-15 15:15:24 +0000] [10] [INFO] Listening at: http://0.0.0.0:8010 (10)
[2020-08-15 15:15:24 +0000] [10] [INFO] Using worker: sync
[2020-08-15 15:15:24 +0000] [14] [INFO] Booting worker with pid: 14
[2020-08-15 15:15:24 +0000] [15] [INFO] Booting worker with pid: 15
[2020-08-15 15:15:24 +0000] [16] [INFO] Booting worker with pid: 16
```

Figure 11: Output of Docker Run

We can access the application from a standard web browser

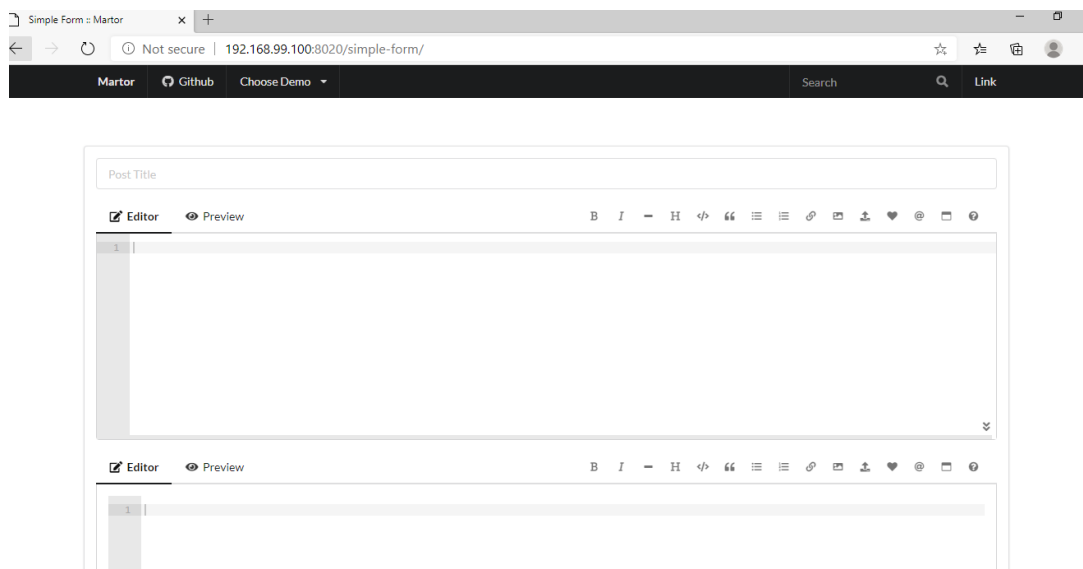
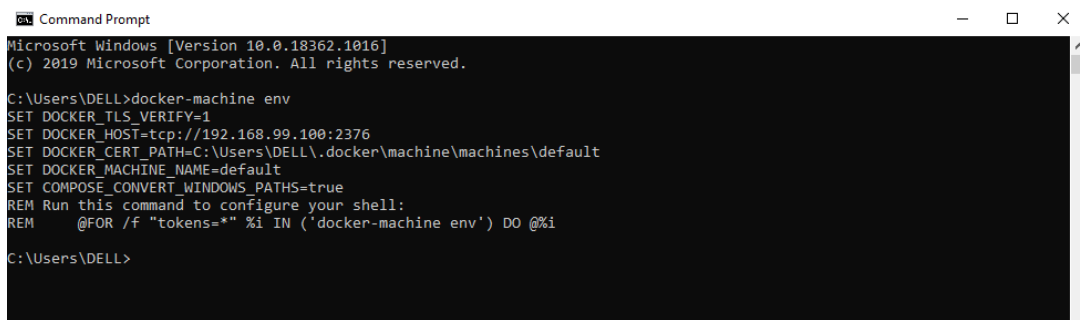


Figure 12: Page exposed by URL

We made use of URL: <http://192.168.99.100:8020/>. We used above internet protocol address by using docker-machine env command.

The application was applied on variety of python based web applications falling within frameworks like Django and Flask. The application was able to easily produce images in the case of flask based application because of simplicity of application developed in flask. Django based applications which are one of the widely used also were easily dockerized using the above approach. Speed of execution ranged from 5 min to a maximum 15 minutes based on the size of the applications being dockerized.. In the entire above cases output image was generated successfully. There were no inconsistency and all the application did run on docker console.

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window content shows the following text:

```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\DELL>docker-machine env
SET DOCKER_TLS_VERIFY=1
SET DOCKER_HOST=tcp://192.168.99.100:2376
SET DOCKER_CERT_PATH=C:\Users\DELL\.docker\machine\machines\default
SET DOCKER_MACHINE_NAME=default
SET COMPOSE_CONVERT_WINDOWS_PATHS=true
REM Run this command to configure your shell:
REM   @FOR /f "tokens=*" %i IN ('docker-machine env') DO @%i

C:\Users\DELL>
```

Figure 13: Exposed IP address

The aim of this work is to provide a user interface using which end users can create docker images of their applications. The type of application which are targeted are basically python based web applications like Django and Flask. Even though the final objective is to provide integration with most of the programming language, the scope in this case is reduced to producing docker images with python based web applications. Post generating the image a trial test to see how the application works will be the test of effectiveness of the solution proposed. With our implementation we have effectively dockerized python based web apps within minutes and got it up and running in the target will lesser hassle than what would have taken tedious setup in the Virtual machine environments. The entire process being automated, is achieved with few click of the user interface developed with tkinter, and the app is ready to be deployed into any cloud environment. Future scope would be to push the docker image built into docker hub and then extracting it from the hub to further make the migration simple.

7 Conclusion and Future Work

”Will hiding complexities with Docker commands and providing an easy to use user interface improve adoption rates and increase the number of applications which are dockerized?”

We have successfully demonstrated how using a simple python application we can easily create a wrapper around complex dockerization process to generate artifacts which can be easily deployed on cloud. The major use of the application is for new people/developers who want to make use of docker but have limited knowledge on the topic but wanted to get started quickly. The use of above solution is not only limited to usability but at the same time improve efficiency of the end developers and users in making deployment using docker quite easy.

What we demonstrated is a just a piece of entire cake. The final aim of this project is to a common user interface using which user can deploy any type of application, using any programming language with the ability to change versions as per need and have deployment done in minutes. The following scopes can be highlighted for future work: Scope to increase non web applications of python, Scope to increase number of programming languages supported, Ability to test docker, Deploy to docker registry, Convert the automation application from stand alone to web application. As a further enhancement the implementation can be extended to support other programming languages such as

Java, C++ and databases.

From our work we can say that containers have a upper hand when compared to Virtual Machines mainly because of the performance and also the faster start up and deployment time. While there are multiple and variety of flavours to implement containers, each comes with its own advantages and disadvantages. With the use of Docker we can add layers in Linux Containers as well which we can say will be most suitable for Cloud PaaS services. With further improvement in abstraction, we see a promising future for containers in PaaS.

References

- Bashari Rad, B., Bhatti, H. and Ahmadi, M. (2017). An introduction to docker and analysis of its performance, *IJCSNS International Journal of Computer Science and Network Security* **173**: 8.
- Bin, N., Zhihua, B., Dejian, L., Sheng, L. and LiXin, Y. (2019). Containerization of intelligent terminal application in ubiquitous power internet of things, *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, pp. 1821–1826.
- Bin Xie, Guanyi Sun and Guo Ma (2016). Docker based overlay network performance evaluation in large scale streaming system, *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 366–369.
- Buyya, R. (2010). Cloud computing: The next revolution in information technology, *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*, pp. 2–3.
- Carvalho, T. and Kim, H. S. (2017). App-centric and environment-aware monitoring and diagnosis in the cloud, *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7.
- Cunha, M. and Laranjeiro, N. (2018). Assessing containerized rest services performance in the presence of operator faults, *2018 14th European Dependable Computing Conference (EDCC)*, pp. 95–100.
- Docker (n.d.). Definitive guide to enterprise container platforms.
URL: <https://goto.docker.com/rs/929-FJL-178/images/Whitepaper-Definitive-Guide-to-Enterprise-Container-Platforms.pdf>
- Dua, R., Raja, A. R. and Kakadia, D. (2014). Virtualization vs containerization to support paas, *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614.
- Erdenebat, B. and Kozsik, T. (2019). Adoption of cloud and containerization technologies in mongolia, *2019 IEEE 15th International Scientific Conference on Informatics*, pp. 000249–000254.

- Felter, W., Ferreira, A., Rajamony, R. and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers, *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172.
- Harzog, B. (2014). Managing applications in docker containers, *The Virtualization Practiccy* **17**: 9.
- Ijtihadie, R. M., Santoso, B. J., Fablius, D. and Nusawan, I. D. P. A. (2017). Multi criteria decision system for distribution provisioning and resource optimization using analytical hierarchy process, *2017 11th International Conference on Information Communication Technology and System (ICTS)*, pp. 197–202.
- K91107541 (n.d.). Application containerization and orchestration.
URL: <https://support.f5.com/csp/article/K91107541>
- Kazi, R., Zhang, X. and Deters, R. (2012). Supporting apps in the personal cloud: Using websockets within hybrid apps, *2012 Second Symposium on Network Cloud Computing and Applications*, pp. 110–115.
- Kohgadai, A. (n.d.). Container adoption trends.
URL: <https://www.stackrox.com/post/2020/03/6-container-adoption-trends-of-2020/>
- Li, Z., Zhang, Y. and Liu, Y. (2017). Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications, *Tsinghua Science and Technology* **22**(01): 1–9.
- Marinescu, P., Hosek, P. and Cadar, C. (2014). Covrig: A framework for the analysis of code, test, and coverage evolution in real software.
- Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment, *Linux Journal* **2014**.
- Naik, N. (2016). Building a virtual system of systems using docker swarm in multiple clouds, *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–3.
- Nardelli, M. (2017). Elastic allocation of docker containers in cloud environments.
- Oberoi, A. (n.d.). Software technology insights - learn, develop, grow.
URL: <https://insights.daffodilsw.com/blog/top-10-python-frameworks-for-web-application-development>
- Pahl, M. and Carle, G. (2013). The missing layer — virtualizing smart spaces, *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 139–144.
- Rajavaram, H., Rajula, V. and Thangaraju, B. (2019). Automation of microservices application deployment made easy by rundeck and kubernetes, *2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CON-ECCT)*, pp. 1–3.
- Seo, K.-T., Hwang, H.-S., Moon, I.-Y., Kwon, O.-Y. and Kim, B.-J. (2014). Performance comparison analysis of linux container and virtual machine for building cloud, pp. 105–111.

- Seth, M. (2018). Mulesoft – salesforce integration using batch processing, *2018 5th International Conference on Computational Science/ Intelligence and Applied Informatics (CSII)*, pp. 7–14.
- Tao, D., Lin, Z. and Lu, C. (2015). Cloud platform based automated security testing system for mobile internet, *Tsinghua Science and Technology* **20**(6): 537–544.
- Upadhyay, A. and Lakkadwala, P. (2014). Performance evolution of higher reliability task in cloud computing, *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pp. 1–3.
- Xingbin Yin, Shanguo Huang, Shouyu Wang, Di Wu, Yuming Gao, Xiaobing Niu, Mingyan Ren and Heng Ma (2013). Software defined virtualization platform based on double-flowvisors in multiple domain networks, *2013 8th International Conference on Communications and Networking in China (CHINACOM)*, pp. 776–780.
- Zhanikeev, M. (2017). Theory and practice for fog infrastructure based on standalone cloudified iot boxes, *2017 TRON Symposium (TRONSHOW)*, pp. 1–8.