Porting Non Cloud-native Applications across Linux Distributions: A Practical Approach

Sanjeet Kumar[a] and Suvrojit Das[b]
Department of CSE, NIT Durgapur, India

Keywords: Linux, Cloud Computing, Application Portability, Workload Migration.

Abstract: Researchers are trying to solve the problem of application portability ever since we had multiple types of Unix operating systems. The need for a solution to port application between different operating environments has only grown in recent years after the adoption of the cloud. The consumers of IT applications want the freedom of choice when it comes to operating environment. They don't want to be put in a vendor lock-in where they are compelled to use one particular platform inspite of their dissatisfaction of service or availability of a better, economic and more competitive alternative. Sometimes the provider of the operating environment goes out of business or there are other reasons where porting of applications becomes a necessity. Linux being an open source operating system gives the liberty to consumers to first extensively test and use the applications on a free distribution and later port it to a distribution with enterprise support. Portability of cloud native apps are more or less solved by use of containerization tools and their ecosystem however the portability of non cloud-native applications is still an open research problem. This paper provides the solution to the problem of porting an application from one distribution of linux to another using an approach which can further be generalized for porting of application between wider range of operating environments.

1 INTRODUCTION

According to leading industry analysts linux is the fastest growing server operating system but the overall growth of combined server operating system market is only 0.7% (Gartner,) which implies that more and more organizations are moving their applications and workloads from proprietary operating systems to linux. There is another perspective to this migration story, organizations wants to move away from vendor lock-in. They want to use a platform that gives them choice and freedom. Currently there are atleast 4 linux vendors which provide enterprise support for linux viz SUSE, Red Hat, Oracle and Canonical(Ubuntu). Apart from these there are some pretty impressive community driven linux distributions like openSUSE and CenOS which organizations with internal technical skills are adopting without enterprise support.

Porting of application from one OS platform to another OS platform is a very sophisticated task. It needs lots of planning and very granular understanding of the source and target platforms as well as understanding of the required dependencies of application on its platform. Many applications are tightly locked with their platform and require a lot of effort in porting. Many standards that serve as a guide for both OS providers as well as application developers, guides them to provide the capabilities of porting of application source and binary between the platforms that conform to certain standards. Linux Standard Base(LSB) is one such standard. Currently there are about 20 different linux distributions which conforms with LSB 5.0 set of specifications. Still, we see a lot of deviations and porting an application from one linux distribution to another distribution is still a very challenging task.

Porting of applications and workloads between different OS is a very old problem. The growth in adoption of cloud as an infrastructure has only caused this problem to multiply. Cloud brings much more flexibility than that of traditional virtualization infrastructure. Cloud has seen adoption in primarily two types of use cases. One, in a large organization where there are multiple departments or lines-of-business. In such an organization there is a separate IT department which serves all the IT infrastructure requirement of all the other departments. Second use case is

[a] https://orcid.org/0000-0002-8755-8837
[b] https://orcid.org/0000-0001-5414-5019

Kumar, S. and Das, S.
Porting Non Cloud-native Applications across Linux Distributions: A Practical Approach.
DOI: 10.5220/0011083700003200
In *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*, pages 272-279
ISBN: 978-989-758-570-8; ISSN: 2184-5042

where a cloud service provider provides the necessary infrastructure to a consumer who can be an individual or an organization. Traditionally the infrastructure procurement itself used to take weeks, and now, with self service in cloud, the line of business can instantly utilize the infrastructure allocated to them from the pooled resources. The applications can achieve very high scalability as per the demand and load.

Porting of application between different types of infrastructure has its own set of challenges. Standards like Open Virtualization Format(OVF) from the Distributed Management Task Force(DMTF) has worked in this area to make at least the virtual machine image conform to a standard which eases out the process of porting, however, porting from physical to cloud still remains a challenge. There are some proprietary tools which help with physical to virtual/cloud porting but they have limitation that they provide solution primarily for intel/amd based architecture.

The adoption of containerization tools like docker and kubernetes has made the problem of linux applications look solvable. The applications can be modified and made cloud native to be able to run on a container infrastracture and then the porting o such applications becomes very easy. However, just like not all pets can be turned into cattles, similarly, not all applications can be modified into cloud-native applications and therefore the research community has a job to look into solution for portability of non cloud-native applications.

For our experiment we took help of an open source tool named machinery. Machinery is aimed at configuration discovery, system validation and service migration. Machinery is based on an idea of universal system descriptor. Although, this project is in a very nascent stage and currently the feature and functionalities are very limited, the idea of universal system descriptor should be valid for all kind of operating system but the current implementation only focuses on linux OS. The current implementation captures the system description for rpm and deb family of linux distributions. However, for the purpose of exporting the system description as a response file for automatic description, currently it only supports SUSE and openSUSE and in one of our previous work (Kumar and Das, 2021) we have contributed to the project so that it can export the system-description of redhat family of linux distribution aswell. Secondly, although it has a base prepared for porting of application and services but the implementation for the same has not been done.

We identified that this could be a very useful tool for application portability use cases. Therefore, we have began contributing to this project https://github.com/ksanjeet/machinery/tree/port_task and solved the problem of portability of non cloud-native linux application across linux distribution. In this paper we are proving the concept using experimental setup and based on our contribution to this project for portability of non cloud-native application from one linux platform to another linux platform.

This paper is further divided into a section on literature review where we have referenced theoretical and experimental work on application portability in general as well as in cloud since 1985 till now. In next section we explain real life problems which became our motivation for this work and our scope and objectives that we want to achieve through this paper. After that we are introducing our approach and methodologies for this experimental work, the challenges we faced and the ideas and algorithms that we used to overcome the challenges. In the next section we discuss the results from our experiment and limitations that we find in our approach and what we can do to remove these limitation in future work. Finally we conclude our paper and explain how this work can help in future research in application portability in cloud.

2 LITERATURE REVIEW

Application portability was first considered as a problem in mid 1980s when there were several variations of Unix that ran on various hardware architecture. Mooney (Mooney, 1990) defines application portability as an application is portable across a class of environments to the degree that the effort required to transport and adapt it to a new environment is less than the effort of redevelopment.

In 1985 X/Open Portability Guide (Taylor, 1987) was released which gave guidelines for a common application environment(CAE) and served as a reference on creating application that can be ported across operating systems.

In 1988 ISO/IEEE POSIX (Group, 1988)(Group, 2018) was released which had a limited scope of source code portability across various conforming operating systems.

Hankinson (Hankinson, 1990) writes from a US govt federal agencies perspective that there was a need to establish strategies and plans for acquiring information technology products and services based on open system standard which supports application portability and interoperability. His organization, National Institute of Science and Technology(NIST) developed a standard called Open System Environment(OSE) towards this. Similarly Application Porta-

bility Profile(APP) which is an OSE standard profile was developed by NIST to define US federal agencies requirement of application. This was probably the first instance when requirement of a software application was defined in the form of standard profiles, an idea that 'll later be seen in application portability solutions and approaches for cloud systems.

After the growth in adoption of linux operating system, The Linux Foundation drafted its first version of Linux Standard Base(LSB) (Foundation, b) in 2001 to develop and promote open standard and increase compatibility among various distributions. LSB gave guidelines for common libraries, commands, utilities, runlevels, printing systems etc. It extensively adopted guidelines from POSIX and in particular it incorporated Filesystem Hierarchy Standard(FHS) (Foundation, a) that was published in 1994.

When virtualization in x86 machine grew adoption of technology like VMware and XEN and later KVM, researchers (Ribiere, 2008) saw virtualization as an important tool for application portability. It was during this time that Distributed Management Task Force(DMTF) came up with a virtual machine image standard called Open Virtualization Format(OVF) (DMTF, 2017). Before this there were different image formats like VMDK, VHD, VDI, etc. Now with a common image format even if the source image was something else it could be converted to either OVF or OVA(open virtualization appliance) and could be imported at the target virtual machine. This helped a lot for the cause of application portability, however one couldn't port an image to a bare metal server,

With the growth in adoption of cloud infrastructure, researchers tried to find the solution for the problem for application portability in this new kind of environment. Cretella and Martino (Cretella and Martino, 2012) suggested a method for automated analysis of cloud vendors.

Research in application portability in cloud are focused in two levels of abstraction:

One, at a higher level of abstraction where standard profiles are created for application where the capabilities required by the application from the platform is defined in a descriptive language. The capabilities of the platforms can be queried and a mapping can be done whether application can run on a particular platform or not. For example, an application profile might mention the requirement as an http server running on port 80. This requirement can be fulfilled by platform running either Apache, lighttpd or nginx. This type of application portability research focuses more on cloud native applications, applications with microservices architecture, applications that are highly scalable in design.

Second type of research focuses on a slightly lower level of abstraction where the application along with its required dependencies is ported from one infrastructure to the other. The infrastructure here can be Physical, virtual, containers or cloud. This type of application portability research focuses more on application with traditional monolithic and N-tier architecture.

Different researchers have suggested different approach to solve the problem of application portability in cloud. Petcu et al (D. Petcu and Crciun, 2013) solution creates a higher layer of abstraction which abstracts different APIs of different cloud so that the application finds a common API to work with. The problem with this approach is that it is not useful when a set of API is not covered under the abstraction layer of this solution.

OASIS provided a different approach, a standard called TOSCA for portability and interoperable operations on cloud applications. Binz et al (T. Binz and Leymann, 2014) introduces TOSCA as a standard which provides new way to enable portable automated deployment and management of composite applications, TOSCA uses a modeling language to formalize the application structure as typed topology graph. TOSCA has since been very popular with researchers with implementation done by Katasaros et. al. (G. Katsaros and Eberhardt, 2014) and Antoniades et. al. (D. Antoniades and Kornmayer, 2015).

There are however other approaches or notable experimental work done by:

Gonidis reported an experimental work with different PaaS platforms. Munnisso demonstrates a proof-of-concept for application portability of cloud application that consumes restful APIs using an algorithm for mapping the application to its cloud resource.

3 OBJECTIVE AND MOTIVATION

There are several use cases that are the objective of this work like upgradation of an existing linux system, moving of workload from physical/virtual to cloud, regulatory and compliance requirements. However, the main motivation is to find a solution to avoid vendor lock-in for a paid linux distribution. The linux distributions like Amazon Linux, Canonical Linux, Oracle Linux RHEL, SLES provides great services and support to their customers but sometimes when the consumer wants to port from one distro to another there is no easy way to do that.

In december 2020 Redhat announced that they will discontinue centOS in its original format used by

thousands of consumers running millions of servers (RedHat,) forcing them either use paid version of RedHat or use a version that is of no use to them. This was a real life situation which motivated us to build a solution to port workloads from one linux distribution to another. In this work we are demonstrating how the above problem can be solved in an automated way.

4 METHODOLOGY

To port non cloud-native application from one linux distribution to another we have used machinery tool and developed additional features in this tool that helps us to achieve the application portability. We had to answer three questions in order to build our solution.

What Constitute an Application? A clear definition of application is very necessary to perform its migration. What may be an application for one user may be an Linux distribution feature or dependency for another user. Defining what is application is also necessary as not every application comes in the packaging format of the linux distribution package manager. For example a python application may not have been installed by RPM or DEB and instead it may have been installed by ”pip”. Similarly a ruby application may have been installed by ”gem” instead of the distribution specific package manager. A tomcat or apache applicaion may simply be a collection of files in their document root. We have therefore arrived at following definition that can be put into a program logic to identify the application from the rest of the operating system:

1. It has to be RPM package that is not part of the base OS. Please see section 4.1.1 to see what we have assumed as the base OS.
2. It has to be a ”pip” installed python application
3. It has to be a ”gem” installed ruby application
4. It has to be a ”npm” installed javascript application
5. It has to be tomcat application stored in its standard document root as per its linux distribution
6. It has to be apache web application in its standard document root as per its linux distribution.

Above definition is used for a working boundary and it can be extended with additional criteria for including more sets of applications.

How to Optimize Machinery System Descriptions So That It only Captures the Application and Not the Rest of the OS? To filter only the contents of applications (rpm and other packages, configuration, managed and unmanaged files etc) we first compare the system descriptions of the source system and the base of the source system. A base system is a plain vanilla installation of that particular linux distribution which doesn't have installed on it. The comparsion gives us a diff of the two system description. This is further optimized to remove any distrubution specific content and finally only the application related contents are left. This is also explained in Algorithm 1.

How to Convert from Source System Description to Target i. e. RPM Names etc? In different linux distributions the names of many packages are different. Even for the distributions using same package manager for example centOS and openSUSE uses RPM package manager the names of the package related to apache web servers are different. On CentOS the name for apache web server is ”httpd” while on openSUSE it is ”apache2”. There are many many more packages that have different names. Our method for conersion involves using tools like ”zypper” and ”dnf” to match the binaries and libraries from the target distribution repository. Once the match is found then the system description content is replaced accordingly from source content to target content.

4.1 Experiment: Portability of Application from One Linux Platform to Another Linux Platform

Automated porting of application from linux distribution to another is a big challenge. We have chosen one distro of RedHat family as the source and one distro of SUSE family as the target. There are many difference between the two distros such as software packages names are different in Red Hat family of Linux and SUSE family of linux. Users and groups may have different UIDs, GIDs. Configuration files for the same packages may have different locations for these two families of linux. A simple free command shows slightly different output when run on these two flavour of linux. Therefore an automated porting has to be done very carefully.

4.1.1 Assumptions

To come up with an automated solution we have to make a very basic assumption. The assumption is that a minimal OS installation of these two flavours are equal. By minimal installation we mean only OS has been installed without any additional application server or database services installation. More granularly, a minimal installation is: Linux Standard Base(LSB), X-Window environment, Gnome or KDE desktop environment, other basic utilities to provide

Algorithm 1: Port system description of one version or distribution type to another version or distribution type.

Command=”port”, argument1=”sname” (for source name), argument2=”tname” (for target name)
Legends:
← = assignment, <action> direction
: = action
:: = derives subclass
.function(arg) = executes function using argument arg

```
1: "dsname" ← system_description(sname)
2: "sbase" ← name(Search : {OS, architecture} ← source)
3: "dsbase" ← system_description(Search : {OS, architecture} ← source)
4: Input : target ← choices
5: "tbase" ← name(target)
6: "dtbase" ← system_description(target)
7: Machinery::PortTask.new()
8: Machinery::PortTask(sname, dsname, sbase, dsbase, tbase, dtbase, tname)
9: Create(Dir) : {name : tname, parent_dir : DEFAULT_CONFIG_DIR}
10: json_parse({sname, sbase, tbase})
11: "s_diff" ← "dsname" − "dsbase"
12: "plist" ← rpm_names(Run_remote_command : sname(rpm − group : {"SystemEnvironment/Daemons","Applications/Engineering"}))
13: c ← 0
14: l1 ← length of array s_diff["packages"]["_elements"]
15: l2 ← length of the array "plist"
16: for i ∈ {1,...,l1 − 1} do
17:     h ← hash of s_diff["packages"]["_elements"][c]
18:     name ← h["name"]
19:     if plist includesname? then
20:         c=c+1
21:         next i
22:     else
23:         delete h from s_diff["packages"]["_elements"]
24:         next i
25:     end if
26: end for
27: "flist" ← unmanaged files to be deleted from target
28: c ← 0
29: l1 ← length of array s_diff["unmanaged"]["_elements"]
30: l2 ← length of the array "flist"
31: for i ∈ {1,...,l1 − 1} do
32:     q ← 0
33:     h ← hash of s_diff["unmanaged"]["_elements"][c]
34:     name ← h["name"]
35:     for j ∈ {1,...,l2 − 1} do
36:         if name starts_with flist[j]? then
37:             s_diff["unmanaged"]["_elements"][h].delete()
38:             next j
39:         else
40:             q ← 1
41:         end if
42:     end for
43:
44:     if q = 0? then
45:         c=c+1
46:         next i
47:     else
48:         next i
49:     end if
50: end for
51: for all s_diff["packages"]["_elements"] as s do
52:     spname ← s["name"]
53:     "tpname" ← Run_remote_command : "tbase"(rpm_name = "spname")
54:     s["version"] ← tpname["version"]
55:     s["release"] ← ""
56:     s["arch"] ← tpname["arch"]
57:     s["vendor"] ← tpname["vendor"]
58:     s["cheksum"] ← ""
59: end for
60: tdes ← dtbase + s_diff
```

a graphical interface to user. It can also be understood in this way that when we install RedHat/CentOS manually at one point it gives us a choice for software selection. There if we choose server with GUI, that is our definition of minimal. Similarly in case of SUSE/OpenSUSE it gives a choice of interface during manual installation and if we choose Gnome Environment, that is our minimal. The reason for choosing a graphical environment in minimal system is that in every practical installation we see graphical environment. If we do not chose a graphical environment then our difference from this minimal system and our source system which we want to port will be huge.

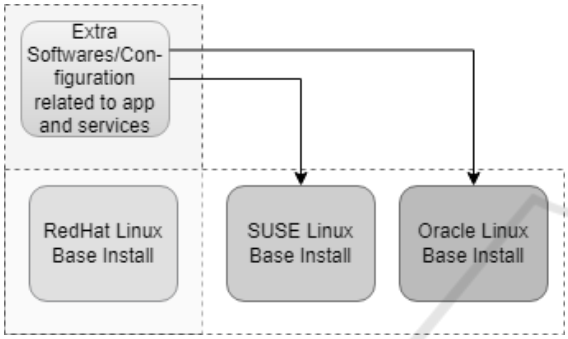

Figure 1: Conceptual Depiction of problem of linux application portability

4.1.2 Settings

Now that we have defined what is a minimal OS installation, we take a machinery system description of this minimal system which will become our base system description for reference and for calculating difference. Secondly we take a system description of our source system. Lets suppose our source system is a CentOS and we want to port it to openSUSE. We find out the difference between CentOS minimal and CentOS source by compare command of the machinery tool and store it into a file.

Now, we had to create another script which will read the content of this ”diff” file and check from the rpm package file list which are the equivalent rpm packages containing the same binaries and libraries and then substitute those rpm names with equivalent names of the target system. In this experiment’s case openSUSE system.

```
desc "Move system description"
long_desc <<-LONGDESC
  Move a system description.

  The system description name is
     ↪ changed to the provided
     ↪ name.
LONGDESC
arg "FROM_NAME"
arg "TO_NAME"
command "move" do |c|
  c.action do |_global_options,
     ↪ _options, args|
    from = shift_arg(args, "
       ↪ FROM_NAME")
                to = shift_arg(args
                   ↪ , "TO_NAME")
    task = MoveTask.new
    task.move(
       ↪ system_description_store,
       ↪  from, to)
    end
  end
desc "Port System Description"
long_desc <<-LONGDESC
Port a system description.
The source system description is
   ↪ ported to target linux
   ↪ platform (e.g. CentOS ->
   ↪ openSUSE )
LONGDESC
arg "SOURCE_NAME"
arg "SOURCE_BASE"
arg "TARGET_BASE"
arg "TARGET_NAME"
command "port" do |c|
  c.action do |_global_options,
     ↪ _options, args|
    sname = shift_arg(args, "
       ↪ SOURCE_NAME")
    dsname = SystemDescription.load
       ↪ (sname,
       ↪ system_description_store)
    sbase = shift_arg(args, "
       ↪ SOURCE_BASE")
    dsbase = SystemDescription.load
       ↪ (sbase,
       ↪ system_description_store)
    tbase = shift_arg(args, "
       ↪ TARGET_BASE")
    dtbase = SystemDescription.load
       ↪ (tbase,
       ↪ system_description_store)
    tname = shift_arg(args, "
       ↪ TARGET_NAME")
    task = PortTask.new
    task.port(sname, dsname, sbase,
       ↪  dsbase, tbase, dtbase,
       ↪ tname)
  end
end
desc "Deploy image to OpenStack
   ↪ cloud"
long_desc <<-LONGDESC
```

Once the conversion of the diff or extra system description is complete we can add this diff or extra system description to the base or minimal install of the target distribution and then export the final system description as either autoyast profile if the target distribution belongs to SUSE family or as kickstart profile if the target distribution belongs to RedHat family. Since in our experiment's the target distribution is openSUSE therefore we exported the final system description as autoyast profile. These autoyast/kickstart profile are further used to build a deployable ISO image which can be used to deploy on the target infrastructure which can be physical, virtual or cloud.

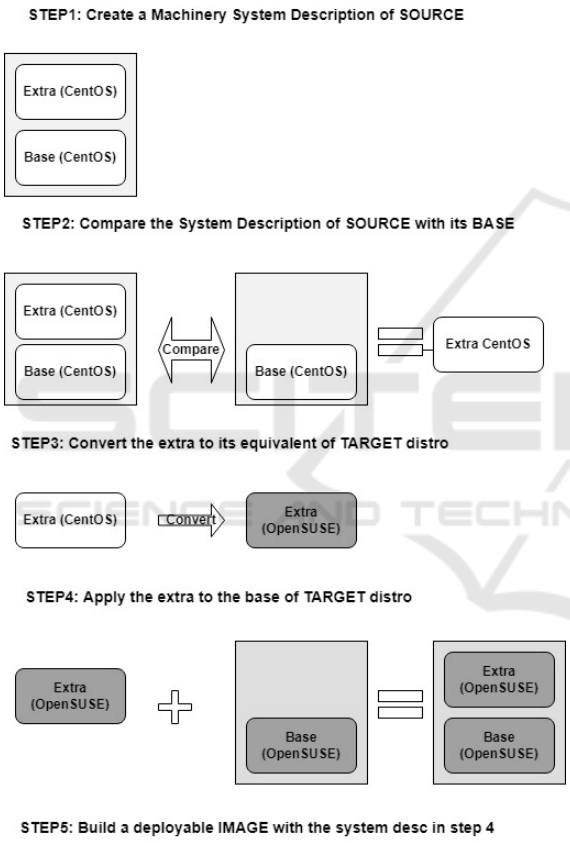

Figure 2: Conceptual Depiction of solution of linux application portability.

5 RESULT AND FUTURE WORK

This experiment is a proof-of-concept that non cloud-native applications and services can be ported between different flavours of linux. We were able to improve the capabilities of the machinery tool in a way that it can be used for porting of whole application workloads from one operating environment to another.

Table 1: Results of contribution in Machinery tool.

S.No.	Features	Before Contribution	After Contribution
1	Inspect system and create system description	Yes	Yes
2	Compare two system descriptions	Yes	Yes
3	Store difference after comparison	No	Yes
4	Convert difference to the equivalent of target linux distro	No	Yes
5	Apply the difference to a target base system description	No	Yes
6	Export system description to a autoinstall profile	Only Autoyast	Autoyast and Kickstart

Successful application portability requires support both from the infrastructure as well as applications. While working on this we realised that the challenge of identifying the application layer of software or the software or services that needs to be ported can be solved a small information is added to the rpm packages, therefore, we recommend that porting of linux applications can become easier if while creating rpms of applications we set a portable flag in the rpm information e.g. instead of rpm-group name System Environment/Daemon we may give the rpm-group name as System Environment/Daemon:Portable so that it can easily be identified as an rpm package that can easily be ported to a new version or a different linux distribution.

We made several enhancement to the original machinery tools to achieve the target of application portability. Here is a list of contributions done.

In the enterprise segment the type of operating systems used as platform for hosting applications are very few. Apart from linux, which is the most growing operating system, windows and unix flavours like HP UX, IBM AIX and Solaris are the only other operating systems that are predominantly used.

REFERENCES

Cretella, G. and Martino, B. D. (2012). Towards automatic analysis of cloud vendors apis for supporting cloud application portability. 375:61–67.

D. Antoniades, N. Loulloudes, A. F. C. S. D. T. G. P. M. D. and Kornmayer, H. (2015). Enabling cloud application portability, proceedings of the 8th international conference on utility and cloud computing. pages 354–360.

D. Petcu, G. Macariu, S. P. and Crciun, C. (2013). Portable cloud applications-from theory to practice. 29(6):1417–1430.

DMTF (2017). Information technology open virtualization format (ovf) specification. pages 1–61.

Foundation, T. L. Filesystem hierarchy standard (fhs). https://refspecs.linuxfoundation.org/FHS3.0/fhs-3.0.pdf. last accessed: Dec 24, 2021.

Foundation, T. L. Linux standard base (lsb). https://wiki.linuxfoundation.org/lsb/start. last accessed: Dec 24, 2021.

G. Katsaros, M. Menzel, A. L. J. R. R. R. S. and Eberhardt, J. (2014). Cloud application portability with tosca, chef and openstack. 302:295–385.

Gartner. Gartner market share analysis server operating system, worldwide 2016. https://www.gartner.com/doc/3731017/market-share-analysis-server-operating. last accessed: Oct 20, 2020.

Group, P. (1988). Ieee standard portable operating system interface for computer environments. pages 1–40.

Group, P. (2018). Ieee standard for information technology–portable operating system interface (posix(r)) base specifications. pages 1–3951.

Hankinson, A. (1990). Open system standards: A us government perspective. 10:207–211.

Kumar, S. and Das, S. (2021). An open source and practical approach to x2x linux workload migration. page 1791–1796.

Mooney, J. (1990). Strategies for supporting application portability. *Computer*, 23(11):59–70.

RedHat. Centos stream: Building an innovative future for enterprise linux. https://www.redhat.com/en/blog/centos-stream-building-innovative-future-enterprise-linux. last accessed: Jan 16, 2022.

Ribiere, A. (2008). Using virtualization to improve durability and portability of industrial applications. pages 1545–1550.

T. Binz, U. Breitenbcher, O. K. and Leymann, F. (2014). Tosca: Portable automated deployment and management of cloud applications.

Taylor, C. (1987). The x/open group and the common application environment. 5(4):665 – 679.