Optimizing Docker Container Images using Common Library Packages

Hyungro Lee and Geoffrey C. Fox
School of Informatics and Computing
Indiana University
Bloomington, IN 47408
Email: lee212@indiana.edu

***Abstract*—With advent of Docker containers, an application deployment using container images gains popularity over scientific communities and major cloud providers to ease building environments. While there are needs to create container images efficiently on a stackable union filesystem and provide dependency information to track Common Vulnerability and Exposure (CVE). This paper demonstrates new approaches of building container images using Common Library Packages (CLP) with surveyed data. As a result, building application environments with CLP-enabled container images uses less storage compared to current Dockerfile scripts and dependency information is visible in detail for further developments.**

***Keywords*—Automation, Containers, DevOps, CVE, Dependencies**

I. Introduction

Deployment for modern applications require frequent changes for new features and security updates with a different set of libraries on various platforms such as IaaS, PaaS, SaaS or FaaS. DevOps tools and containers are widely used to complete application deployments with scripts but there are problems to reuse and share scripts when a large number of software packages and libraries are combined to build computing environments for applications. For example, Ansible Galaxy - a public repository, provides scripts (called roles) over ten thousands and Docker Hub has at least 14 thousands container images but most of them are individualized and common libraries and tools are barely shared. This might be acceptable if a system runs only one or two applications without multi tenants but most systems in production need to consider how to run applications efficiently. DevOps provides an automation of a software deployment and a continuous integration yet building identical environments for applications is not ensured unless all required repositories to install are preserved. System administrator and application developer need to review scripts carefully otherwise deployments may encounter a failure at a later time due to dependency hell and unexpected changes from a software repository. Container technology i.e. Docker permits a repeatable build of a application environment using snapshot images i.e. container images but redundant images with unnecessary layers are observed because of a stacked file system. In this paper, we introduce two approaches about building Common Library Packages (CLP) in containers therefore building application environments is optimized and contents are visible in detail for further developments.

Reproducibility is ensured with container images which are stored in a stackable union filesystem and "off the shelf" software deployment is offered through scripts e.g. Dockerfile to automate and share building an equivalent software environment across various platforms. Each command line of scripts creates a directory (called image layer) to store results of commands separately and container instance runs an application on a root filesystem which is merged by these image layers while a writable layer is added on top and other layers beneath it are kept as readable only, known as copy-on-write. The problem is that system-wide shared libraries and tools are placed on an isolated directory and it prevents building environments efficiently over multiple versions of software and among various applications which use same libraries and tools. We use collections of HPC-ABDS (Apache Big Data Stack) [1] and github API to present surveyed data in different fields about automated software deployments. In this case, we collected public Dockerfiles and container images from Docker Hub and github.com and analyzed tool dependencies using Debian package information.

II. Background

A. Software Deployment for dynamic computing environments

Software development has evolved with rich libraries and building a new computing environment (or execution environment) requires set of packages to be successfully installed with minimal efforts. The environment preparation on different infrastructure and platforms is a challenging task because each preparation have individual instructions which build a similar environment, not identical environment. Traditional method of software deployment is using shell scripts to define installation steps with a system package manager command such as apt, yum, dpkg, dnf and make but it is not suitable to deal with large number of packages actively updated and added to community in a universal way. Python Package Index (PyPI) has almost 95,490 packages (as of 12/26/2016) with 40+ daily new packages and github.com where most software packages, libraries and tools are stored has 5,776,767 repositories available with about 20,000 daily added repositories. DevOps tools i.e. Configuration management software supports automated installation with repeatable executions and better error handling compared to bash scripts but there is no industry standards for script formats and executions. Puppet, Ansible, Chef, CFEngine and Salt provide community contributed repositories to automate software installation, for example, Ansible Galaxy has 11353 roles available, Chef Supermarket has 3,261 cookbooks available although there are duplicated and inoperative scripts for software installation

and configuration. Building dynamic computing environments on virtual environments is driven by these DevOps tools and container technologies during last a few years due to its simplicity, openness, and shareability. Note that this effort is mainly inspired by the previous research activities [2], [3], [4], [5], [1], [6].

B. DevOps - Ansible

In the DevOps phase, configuration management tool i.e. Ansible automates software deployment to provide fast delivery process between development and operations [7]. Instructions to manage systems and deploy software are written in scripts although different formats i.e. YAML, JSON, and Ruby DSL and various terminologies i.e. recipes, manifests, and playbooks are used. Puppet and Chef are configuration management tools written in Ruby and these tools manage software on target machines regarding to installation, execution in a different state e.g. running, stopping or restarting, and configuration through the client/server mode (also called master/agent). Ansible is also recognized as a configuration management tool but more focusing on software deployment using SSH and no necessity of agents on target machines. With the experience from the class projects at Indiana University and NIST projects [8], a few challenging tasks are identified in DevOps tools, a) offering standard specification of scripts to ease script development with different tools, and b) integrating container technologies towards microservices.

C. Scripts and Templates

Building compute environments needs to ensure reproducibility and constant deployment over time [9], [10]. Most applications these days run with dependencies and setting up compute environments for these applications require to install exact version of software and configure systems with same options. Ansible is a DevOps tool and one of the main features is software deployment using a structured format, YAML syntax. Writing Ansible code is to describe action items in achieving desired end state, typically through an independent single unit. Ansible offers self-contained abstractions, named Roles, by assembling necessary variables, files and tasks in a single directory and an individual assignment (e.g., installing software A, configuring system B) is described as a role. Compute environments are usually supplied with several software packages and libraries and selectively combined roles conduct a software deployment where new systems require environments with needed software packages and libraries installed and configured. Although the comprehensive roles have instructions stacked with tasks to successfully finish a software deployment with dependencies, the execution of applications still need to be verified. In consequence, to preserve an identical results from the re-execution of applications, it is necessary to determine whether environments are fit for the original applications.

D. Containers with Dockerfile

Container technology has brought a lightweight virtualization with a Linux kernel support to enable a portable and reproducible environment across laptops and HPC systems. Container runtime toolkit such as Docker [11], rkt [12] and LXD [13] has been offered since 2014 which uses an image file to initiate a container including necessary software packages and libraries without an hypervisor which creates an isolated environment using a virtual instance but with an isolated namespace on a same host operating system using the Linux kernel features such as namespaces, cgroups, seccomp, chroot and apparmor. Recent research [14] shows that containers outperform traditional virtual machine deployments yet running containers on HPC systems is still an undeveloped area. Shifter [15] and Singularity [16] have introduced to support containers on HPC with a portability and MPI support along with docker images. These efforts will be beneficial to scientific applications to conduct CPU or GPU intensive computations with easy access of container images. For example, a neuroimaging pipelines, BIDS Apps [17], is applied to HPCs using Singularity with existing 20 BIDS application images and Apache Spark on HPC Cray systems [18] is demonstrated by National Energy Research Scientific Computing Center (NERSC) using shifter with a performance data of big data benchmark. Both researches indicate that scientific and big data workloads are supported by container technologies on HPC systems for reproducibility and portability.

Listing 1: Dockerfile Example

```
FROM ubuntu:14.04

MAINTAINER Hyungro Lee <lee212@indiana.edu>

RUN apt-get update && apt-get install -y \\
      build-essential wget git
...
```

Dockerfile (See Listing 1) uses a custom template to describe installation steps of building docker images in a bash like simple format. There are certain directives to indicate particular objective of the commands, for example, FROM indicates a base image to use and RUN indicates actual commands to run. When an image is being generated, each directive of Dockerfile creates a single directory to store execution results of commands and a final image created by Dockerfile has meta-data to merge these directories in a unified logical view. The tag for an image is a reference for stacked image layers. For example in Listing 1, *ubuntu:14.04* is tag to import stacked image layers of building Ubuntu 14.04 distribution and the following directives i.e. *MAINTAINER* and *RUN*, will be stacked in a new branch. This allows to share most common image layers with a tag among other containers e.g. linux distribution.

E. Environment Setup

Preparing environment means that installing all necessary software, changing settings and configuring variables to make your application executable on target machines. Container technology simplifies these tasks using a container image which provide a repeatable and pre-configured environment for your application therefore you can spend more time on an application development rather than software installation and configuration to ensure proper deployments. One of the challenges we found from container technologies is managing dependencies of application libraries. Container users who want to run applications with particular libraries have to find a

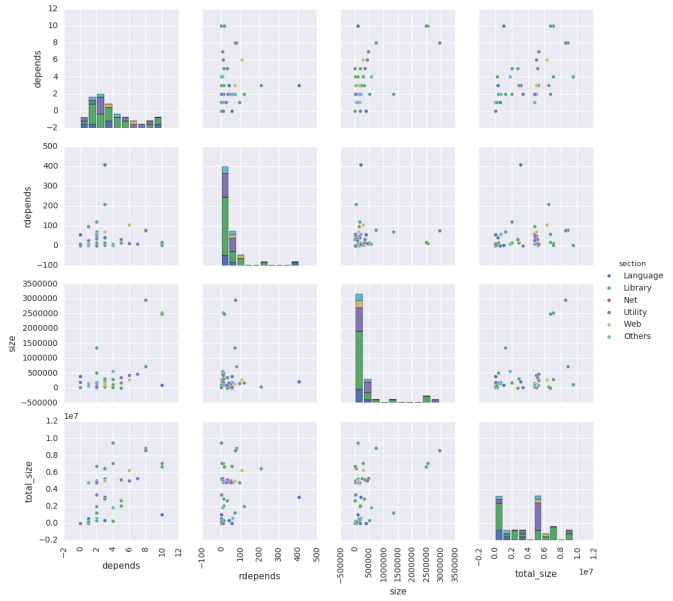
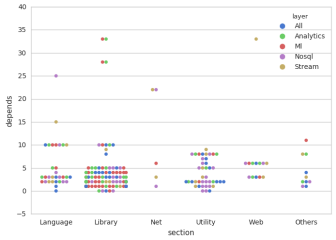
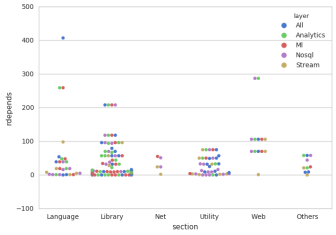


Fig. 1: Debian Package Relations between Dependencies, Reverse Dependencies and Package Sizes (itself and including dependencies) for Popular Docker Images



(a) Dependencies



(b) Reverse Dependencies

Fig. 2: Debian Package Dependencies for HPC-ABDS Layers

proper container images or to create an individual image with required libraries and tools, otherwise jobs for an application will fail. One possible solution for this problem is offering a common library package for an application. We noticed that there is a common list of libraries for particular type of applications based on the survey from Docker images and Dockerfile scripts. The idea is to offer curated environments for domain-specific applications using the surveyed list of libraries from community. For example, libraries for linear algebra calculation i.e. liblapack-dev and libopenblas-dev are commonly used for applications in analytics layer of HPC-ABDS according to the Table I. Additional package installation might be required if suggested list of dependencies does not satisfy all requirement of an application.

F. Package Dependencies

Software packages have many dependencies especially if packages are large and complex. Reverse dependencies reveal which package will break and cause conflicts if current package is removed or changed. Figure 1 shows that one of examples how packages are related to others based on the survey of popular docker files from Debian packages.

G. Infrastructure Provisioning

III. Results

IV. Related Work

V. Conclusion

TBD

Acknowledgment

The authors would like to thank...

References

[1] G. C. Fox, J. Qiu, S. Kamburugamuve, S. Jha, and A. Luckow, "Hpc-abds high performance computing enhanced apache big data stack," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 1057–1066.

[2] G. Fox, J. Qiu, S. Jha, S. Ekanayake, and S. Kamburugamuve, "Big data, simulations and hpc convergence," in *Workshop on Big Data Benchmarks*. Springer, 2015, pp. 3–17.

[3] G. Fox, J. Qiu, and S. Jha, "High performance high functionality big data software stack," 2014.

[4] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, "Towards hpc-abds: an initial high-performance big data stack," *Building Robust Big Data Ecosystem ISO/IEC JTC 1 Study Group on Big Data*, pp. 18–21, 2014.

[5] G. Fox and W. Chang, "Big data use cases and requirements," in *1st Big Data Interoperability Framework Workshop: Building Robust Big Data Ecosystem ISO/IEC JTC 1 Study Group on Big Data*. Citeseer, 2014, pp. 18–21.

[6] G. Fox, J. Qiu, S. Jha, S. Ekanayake, and S. Kamburugamuve, "White paper: Big data, simulations and hpc convergence," in *BDEC Frankfurt workshop. June*, vol. 16, 2016.

[7] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[8] B. Abdul-Wahid, H. Lee, G. von Laszewski, and G. Fox, "Scripting deployment of nist use cases," 2017.

[9] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *Computer*, vol. 40, no. 12, 2007.

[10] A. Goodman, A. Pepe, A. W. Blocker, C. L. Borgman, K. Cranmer, M. Crosas, R. Di Stefano, Y. Gil, P. Groth, M. Hedstrom *et al.*, "Ten simple rules for the care and feeding of scientific data," *PLoS computational biology*, vol. 10, no. 4, p. e1003542, 2014.

[11] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[12] "Coreos/rkt: a container engine for linux designed to be composable, secure, and built on standard," https://github.com/coreos/rkt, 2016, [Online; accessed 09-November-2016].

[13] "Ubuntu lxd: a pure-container hypervisor," https://github.com/lxc/lxd, 2016, [Online; accessed 09-November-2016].

[14] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in

Name	PCT1	PCT2	PCT3	PCT4	Description	Section	CT1	CT2	Dependencies	Size	Important
software-properties-common	0.01	0.06	0.02	0.03	manage the repositories that you install software from (common)	admin	8	4	python3-dbus, python-apt-common, python3-software-properties, gir1.2-glib-2.0, ca-certificates, python3:any, python3-gi, python3	9418 (630404)	optional
build-essential	0.14	0.16	0.03	0.05	Informational list of build-essential packages	devel	5	32	dpkg-dev, libc6-dev, gcc, g++, make	4758 (2705548)	optional
g++	0.15	0.06	0.02	0.01	GNU C++ compiler	devel	4	57	cpp, gcc, g++-5, gcc-5	1506 (22034848)	optional
gcc	0.03	0.05	0.02	0.01	GNU C compiler	devel	2	57	cpp, gcc-5	5204 (6735366)	optional
groovy	-	-	**0.01**	-	Agile dynamic language for the Java Virtual Machine	universe/devel	14	10	libbsf-java, libservlet2.5-java, antlr, libxstream-java, libcommons-logging-java, libjline-java, libasm3-java, libjansi-java, libregexp-java, libmockobjects-java, junit4, default-jre-headless, ivy, libcommons-cli-java	9729202 (3257906)	optional
libatlas-base-dev	-	**0.06**	-	-	Automatically Tuned Linear Algebra Software, generic static	universe/devel	2	8	libatlas-dev, libatlas3-base	3337570 (2690424)	optional
liblapack-dev	-	**0.03**	-	-	Library of linear algebra routines 3 - static version	devel	2	22	liblapack3, libblas-dev	1874498 (2000176)	optional
ruby	-	0.01	0.01	0.01	Interpreter of object-oriented scripting language Ruby (default version)	interpreters	1	987	ruby2.1	6026 (73880)	optional
maven	-	-	0.02	0.01	Java software project management and comprehension tool	universe/java	2	5	default-jre, libmaven3-core-java	17300 (1441844)	optional
libffi-dev	-	0.03	-	**0.01**	Foreign Function Interface library (development files)	libdevel	2	11	libffi6, dpkg	162456 (2101914)	extra
libssl-dev	0.12	0.07	0.01	0.03	Secure Sockets Layer toolkit - development files	libdevel	2	70	libssl1.0.0, zlib1g-dev	1347070 (1258956)	optional
net-tools	0.01	0.02	0.03	0.05	NET-3 networking toolkit	net	1	51	libc6	174894 (4788234)	important
chrpath	-	-	-	**0.05**	Tool to edit the rpath in ELF binaries	utils	1	0	libc6	12932 (4788234)	optional
git	0.33	0.21	0.06	0.07	fast, scalable, distributed revision control system	vcs	8	75	perl-modules, liberror-perl, libpcre3, libcurl3-gnutls, git-man, zlib1g, libc6, libexpat1	2951026 (8563378)	optional
nodejs	0.01	0.04	-	0.02	evented I/O for V8 javascript	universe/web	6	287	libssl1.0.0, libc6, libstdc++6, zlib1g, libv8-3.14.5, libc-ares2	683742 (7551922)	extra

TABLE I: Most Common Debian Packages from Dockerfile Survey Samples
(PCT1: Percentage by General Software, PCT2: Percentage by Analytics Layer, PCT3: Percentage by Data processing Layer, PCT4: Nosql Layer, CT1: Count of Dependencies, CT2: Count of Reverse Dependencies)

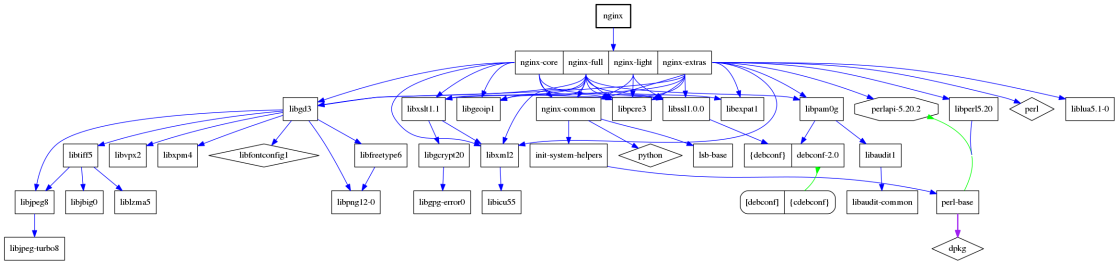


Fig. 3: Nginx Debian Package Dependencies

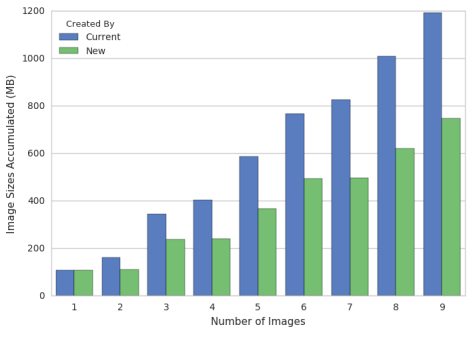


Fig. 4: Comparison of Container Images for Nginx Version Changes
(Current: Built by Official Dockerfiles, New: Built by Common Library Packages)

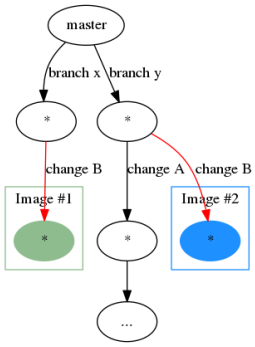


Fig. 5: Union File System Tree

Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE, 2015, pp. 171–172.

[15] D. M. Jacobsen and R. S. Canon, "Contain this, unleashing docker for hpc," *Proceedings of the Cray User Group*, 2015.

[16] G. M. Kurtzer, "Singularity 2.1.2 - Linux application and environment containers for science," Aug. 2016. [Online]. Available: https://doi.org/10.5281/zenodo.60736

[17] K. J. Gorgolewski, F. Alfaro-Almagro, T. Auer, P. Bellec, M. Capota, M. M. Chakravarty, N. W. Churchill, R. C. Craddock, G. A. Devenyi, A. Eklund *et al.*, "Bids apps: Improving ease of use, accessibility and reproducibility of neuroimaging data analysis methods," *bioRxiv*, p. 079145, 2016.

[18] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, "Scaling spark on hpc systems," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and*

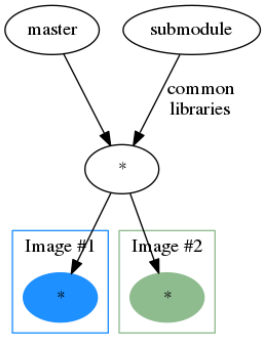


Fig. 6: Common Library Packages by submodules

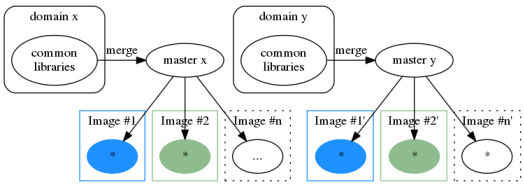


Fig. 7: Common Library Packages by merge

Distributed Computing. ACM, 2016, pp. 97–110.

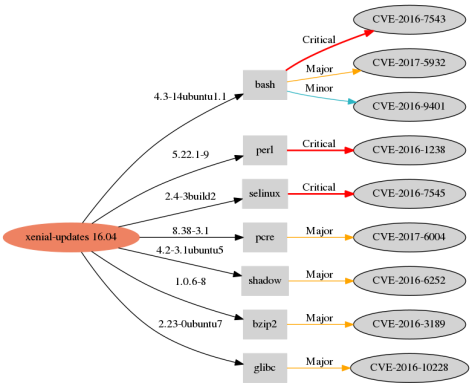


Fig. 9: Example of Security Vulnerabilities for Ubuntu 16.04 based on Libraries

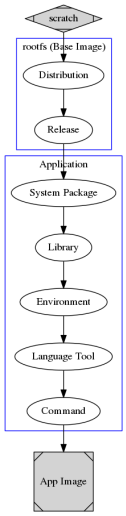


Fig. 8: Dockerfile Workflow