Docker Container Manager: A Simple Toolkit for Isolated Work with Shared Computational, Storage, and Network Resources

S P Polyakov[1], A P Kryukov and A P Demichev

Skobeltsyn Institute of Nuclear Physics, M.V.Lomonosov Moscow State University (SINP MSU), 1(2), Leninskie gory, GSP-1, Moscow 119991, Russian Federation

[1] s.p.polyakov@gmail.com

Abstract. We present a simple set of command line interface tools called Docker Container Manager (DCM) that allow users to create and manage Docker containers with preconfigured SSH access while keeping the users isolated from each other and restricting their access to the Docker features that could potentially disrupt the work of the server. Users can access DCM server via SSH and are automatically redirected to DCM interface tool. From there, they can create new containers, stop, restart, pause, unpause, and remove containers and view the status of the existing containers. By default, the containers are also accessible via SSH using the same private key(s) but through different server ports. Additional publicly available ports can be mapped to the respective ports of a container, allowing for some network services to be run within it. The containers are started from read-only filesystem images. Some initial images must be provided by the DCM server administrators, and after containers are configured to meet one's needs, the changes can be saved as new images. Users can see the available images and remove their own images. DCM server administrators are provided with commands to create and delete users. All commands were implemented as Python scripts. The tools allow to deploy and debug medium-sized distributed systems for simulation in different fields on one or several local computers.

1. Introduction

If we want to let several users share access to computational, storage, and network resources of a server, we can use several approaches. The most simple one is to let them log in to the server directly. It imposes no overhead, but the server administrators have to configure the server and install any software the users need but cannot install without privileges. This becomes particularly problematic if two or more users need different versions of the same software. Another approach is to give users privileged access to virtual machines. Virtual machines can be configured independently and can even use different operating systems (OS), but the overhead may be significant. Container virtualization [1] provides a middle ground between these two approaches and combines many of their benefits.

We present a set of tools for container virtualization platform Docker that allows users to create and manage their own Docker containers (analogs of virtual machines) with preconfigured SSH access. The users can configure containers, install any software compatible with the server OS kernel, and run network services and computational programs, including microservices. This allows to deploy and debug medium-sized distributed systems that can be used for simulation in different fields.

The rest of the paper is organized as follows: Section 2 outlines some possibilities of container virtualization and features of Docker. Section 3 presents the Docker Container Manager (DCM) toolkit we developed and describes its functionality. Section 4 discusses the security limitations of the tools. Section 5 concludes the paper and outlines some possibilities for further development of the tools.

2. Container virtualization and Docker

Container virtualization is an OS feature that allows to use the same OS kernel to run multiple isolated user-space instances called containers. It was developed from Linux chroot mechanism [2] that allowed to change apparent root directory for a process and its children. Unlike full virtualization, container virtualization does not allow containers to use different operating systems. But it does provide isolation between the processes, allowing each to use its own set of libraries and installed software without dependencies conflict, and imposes relatively small overhead [3]. Many container virtualization methods use copy-on-write technique [4]: resources used by two or more containers are not copied until one of the containers needs to modify it. Thanks to it, a server can run or store large number of containers.

Docker is a rapidly developing and very popular container virtualization platform for Linux. Docker uses copy-on-write technique: file system of a Docker container is based on a read-only image; changes made in a running container can be saved to a new image, resulting in an additional layer. Docker provides a number of tools for creating, monitoring, and managing containers, as well as building images, sharing them via repositories and looking for the images created by other users.

A server administrator can give users access to containers started for them, but if we want users to be able to leverage more of the Docker features, we could let them manage containers on their own, save changes to new images, start new containers from their own images, discard the containers they no longer need and so on. However, giving users full access to Docker capabilities on a server amounts to giving them unrestricted access to the server. In particular, volumes mechanism allows to map any host directory to a container directory (so the users will be able to modify anything from the container they mapped a host root directory to); containers may also be given privileges to access devices of the host. Therefore, to let users create their own containers we need to restrict their access to Docker commands to prevent them from accidentally disrupting the work of the server. Imposing such restrictions is one of the purposes of the toolkit presented in this paper, Docker Container Manager.

3. Docker Container Manager

Docker Container Manager (DCM) is a set of tools which has three primary purposes:

· to allow users to create Docker containers on a server, configure them and save the changes;

· to let users work with the containers remotely, using the computational, storage, and network resources of the server;

· to isolate users from each other and restrict their access to the Docker features that could potentially disrupt the work of the server.

DCM has a simple command-line interface accessible by SSH: when new DCM users are created, their access to the server is configured to redirect them to DCM interface as they log in. The interface allows users to execute a simplified set of Docker commands, with DCM keeping track of their containers and images to make sure only owners can change or see them.

After a container is created, DCM interface informs the user about the server port that can be used to connect to the container by SSH. The interface is not needed to connect to the container and work with it.

The following subsections give the overview of DCM commands in more detail.

3.1. Creating containers

DCM users can create a container using create command. In its most basic form the command takes one argument, an image to create the container from. DCM then invokes a Docker command run with the options that ensure the following:
· port 22 (SSH) of the container is mapped to a random port of the server (the port number is also used as the container ID);
· two storage directories assigned to the user on the server are mapped to home directories /root and /home/<user> of the container.
As a result, the user can access the container by SSH using their own private key both as regular user and as root (the copies of the public key are placed to their storage directories when the DCM user account is created), and any changes made to /root and /home/<user> directories in any container are accessible in all other containers of the user.

Additionally, the user can select one or several container ports to be published on the server. These ports are mapped to the server ports without changing their numbers. More than one container can use the same port or ports if the server has IP aliases: DCM will choose an alias where none of the requested additional ports are currently in use by other containers.

Any other options of the Docker command run that are not available to DCM users by default can be specifically permitted to individual users by the DCM server administrators. Since many of the additional options create security weaknesses, an administrator is supposed to be careful when allowing users to use these options. A user can select one of the permitted options or combinations of options with preset arguments, but not combine them or change arguments at will.

3.2. Saving changes of containers to new images

Administrators of a DCM server must provide users with at least one image with preconfigured SSH access. When creating a container for the first time, a user can select one of these images. After making any changes outside of /root and /home/<user> directories (e.g. installing new software), the user can save these changes to a new image using the commit command.

3.3. Managing containers and images

The following DCM commands to manage containers are available to users: stop – stops a running container (without removing it), start – starts a stopped container, pause – freezes all processes within a container, unpause – unfreezes frozen processes within a container, show – shows all containers belonging to the user and their status, rm – removes a container.

There are also two commands to manage images:
images – shows all images available to the user (this includes basic images provided by the DCM server administrators and the images created by the user), rmi – removes an image.

3.4. Creating and removing DCM user accounts

Administrators of DCM servers are provided with tools to create new DCM user accounts and remove them. Creating new user accounts includes setting up their access to DCM interface, creating storage directories and copying their public keys to these directories. Removing user accounts includes removing their containers, storage directories, and possibly their images.

4. Security

In the earlier versions of Docker, it was considered possible to break out of a container if users were allowed to execute arbitrary commands («Containers do not contain», [5]), even more so if they had root access to the container. Since Docker Engine version 1.10 released in 2016, two key security features are supported: seccomp filtering and user namespaces. These features make Docker considerably more secure (see [6] for analysis and recommendations).

However, user namespaces is not used by default, Docker has some other potential security weaknesses, and a mistake by a DCM server administrator may introduce further weaknesses. Therefore we recommend to allow only users who can be trusted to make no attempts to break out of their containers intentionally. The users should also be instructed to give no access to their accounts or privileged access to their containers to outside parties.

If necessary, DCM can be easily modified to use additional options when creating new containers. With some modifications it is possible to make containers more secure than Docker default, e.g. by further restricting root capabilities.

5. Conclusions and further work

We have presented a set of tools that give users indirect access to some of the Docker commands without full access to the host, allowing them to create Docker containers that can be configured by the user, save and access the changes made to the containers, and manage the containers. The users are isolated from each other. The tools can be accessed via a simple command line interface.

The ideas for further development of the toolkit include creating a Web interface and giving users the option to migrate their images to outside repositories.

Acknowledgements
The authors are grateful to the Russian Foundation for Basic Research (grant No 15-07-09309), with the financial support of which this work was carried out.

References

[1] Soltesz S, Pötzl H, Fiuczynski M E, Bavier A and Peterson L 2007 *Proc. 2nd ACM SIGOPS/EuroSys European Conf. on Computer Systems 2007* Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors (New York: ACM New York) 275–87

[2] McFearin L D 2011 Chroot jail *Encyclopedia of Cryptography and Security* Springer US 206–7

[3] Felter W, Ferreira A, Rajamony R and Rubio J 2015 An updated performance comparison of virtual machines and linux containers *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE Int. Symp. On* IEEE 171–2

[4] Fábrega F J T, Javier F and Guttman J D 1995 Copy on write. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.3144&rep=rep1&type=pdf

[5] Walsh D 2014 Are Docker containers really secure? [Electronic resource]. URL: https://opensource.com/business/14/7/docker-security-selinux (accessed 10.09.2017)

[6] Grattafiori A 2016 Understanding and hardening Linux containers *NCC Group whitepaper* URL: https://www.nccgroup.trust/uk/our-research/understanding-and-hardening-linux-containers/