The Software Container Pattern

MADIHA H. SYED, Florida Atlantic University
EDUARDO B. FERNANDEZ, Florida Atlantic University

A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers with strong isolation between them. Software containers although not new, have become very important to support convenient, secure, and low overhead applications. Containers facilitate application deployment and distribution across computing environments. We present a pattern for a Software Containers which describes advantages and liabilities of the container in addition to examples of existing solutions. Containers have made a significant difference in supporting agile development frameworks like DevOps.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures - Patterns

General Terms: Design

Additional Key Words and Phrases: software containers, architecture patterns, security, virtualization, security patterns

ACM Reference Format:

Syed, M.H. and Fernandez, E.B. 2015. The Software Container Pattern. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (October 2015), 7 pages.

1. INTRODUCTION

A cloud-based computing system involves a variety of users and devices connected to it and provides multiple kinds of services. Typically, clouds provide three levels of service: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). SaaS can be directly used by consumers to access its available applications but also to let these consumers become in turn service providers (SPs). Virtualization is not new term in computing domain but it has undeniably gained popularity in recent times alongside cloud computing. In addition to virtual machines, another form of virtualization has been brought into focus, called operating system virtualization or container based virtualization. It uses a virtualization layer to run as an application in the OS thereby relieving the application from the OS and infrastructure dependencies. A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers with strong isolation between them. Software containers although not new, have become very important to support convenient, secure, and low overhead applications. Containers facilitate application deployment and distribution across computing environments. Containers are being used to facilitate teams using agile development frameworks like DevOps [White 2014].

We present here a pattern for the Software Container. Our audience includes system architects and system designers as well as software developers who are interested in building container solutions. The Container pattern provides virtualization function which can be used in single user single environment scenario as well as in connection to a cloud. It is an important addition to cloud ecosystems. An ecosystem is the expansion of a software product line architecture to include systems outside the product which interact with the product [Bosch 2009]. Figure 1 shows a cloud ecosystem where the Cloud Reference Architecture (RA) is the main pattern (hub) that defines the ecosystem. The Cloud Security RA (SRA) can be derived from a Cloud RA by adding security patterns to control its identified threats. Cloud Web Application Firewalls and Security Group Firewalls provide filtering functions that can be provided as services through Virtual Network Function (VNF) or on their own. The Cloud Compliant Reference Architecture applies patterns to the Cloud RA to comply with regulations. Further elaboration about each of these components can be found in prior work of Fernandez et al. [2015a; 2015b].

Author's address: Madiha H. Syed, email: msyed2014@fau.edu; Eduardo B. Fernandez, email: ed@cse.fau.edu; Dept. of Computer and Elect. Eng. and Computer Science Florida Atlantic University, Boca Raton, FL 33431, USA.

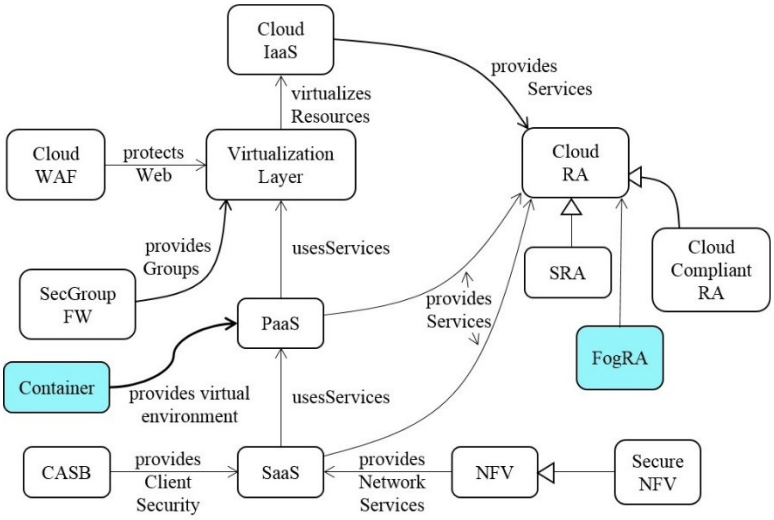

Fig.1. A cloud ecosystem

2. SOFTWARE CONTAINER PATTERN

2.1 AKA

Zone, Jail [Topjian 2013; Wikipedia 2015]

2.2 Intent

A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers. Containers are lightweight, portable, extensible, reliable, and secure.

2.3 Example

Our organization has development and testing teams working at distributed locations. We generally use cloud computing in addition to company's local infrastructure. We need a quick and easy way to provide a standardized environment to execute and test all kinds of applications. Even if we have applications from different companies running on our system, we have to provide each one a standard platform for execution. In addition to this, necessary isolation between these applications is security requirement. We have to furnish frequent releases of our software so the deployment process and distribution is becoming difficult. For a cost-effective operation we need to reduce overhead and improve security. Our applications execute in a few commonly used operating systems, we do not have very specific customized platform requirements.

2.4 Context

Software developers/Institutions developing many applications that will execute in multiple computer systems and/or cloud-based virtual environments and using a few types of operating systems. Security, reliability, and overhead are concerns.

2.5 Problem

We want to be able to run applications in a self-contained environment in such a way that both (application and execution environment) can be treated as a single unit.
The solution to this problem is guided by the following forces:

- *Overhead*: We want the overhead of the execution environment to be as low as possible; otherwise, we can use more flexible solutions such as virtual machines.
- *Portability*: We want applications to be portable across execution environments; that is, they should be able to be moved, for example, from one location to another location, without large modifications.
- *Controlled Execution*: We want to control application execution in a simple and convenient way.
- *Cost*: The cost of running applications should be as low as possible in terms of resources.

- *Isolation*: When multiple applications are running in the same OS we want a strong isolation between them so that if one of them is malicious, compromised, or fails, attacks or errors do not propagate to other applications.
- *Opaqueness*: Applications running on the same OS should not be aware of each other to ensure security.
- *Transparency*: The specific environment should be transparent to the application.
- *Scalability*: The number of applications sharing one type of OS should be scalable.
- *Extensibility*: It should be possible to dynamically provide additional services to the hosted applications like logging/auditing, filtering, persistence, and others
- *Recovery:* We need to be able to recover/replicate the application easily.

2.6 Solution

Provide a runtime environment that can support the isolated execution of applications on a shared Host OS, this is a Software Container (Figure 2). Multiple processes can share one container in special circumstance [Docker 2015b]. They also may share binaries and libraries with other containers. Containers provide isolated execution and extensible services to the application.

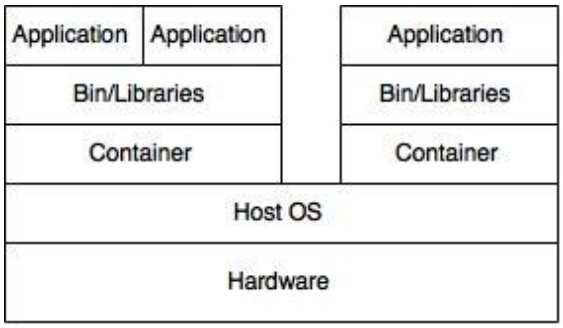

Fig. 2. Two Containers sharing one OS

2.7 Structure

Figure 3 shows the class diagram for this pattern. A *Container* controls a set of *Applications* sharing a *Host OS* that provides a set of *Resources*. An *Interceptor* mediates the services provided to the application by the container. Applications hosted in containers can be accessed remotely through *Proxies*, where the Container acts as a broker. The client interacts with the *Application Proxy*, which represents the application. The application interacts with the *Client Proxy*, which represents the client. The Container provides a set of *Services* to the applications. *Container Images* are stored in *image repositories*.

2.8 Dynamics

Figure 4 shows a use case to execute an application in a container. A remote client executes an application through its proxy. The container transmits client requests to the Application through the Client Proxy. In order to execute the request or access a resource, application issues an OS call. This call goes to the interceptor before it is forwarded to Host OS.

2.9 Implementation

- In a virtualized environment containers can be mixed with virtual machines or bare metal servers. In other words, we can execute applications directly in a physical processor, in a virtual processor, or in a container.
- The Host OS should be able to have primitives to execute each process in strongly isolated execution environment; for example, Linux-based containers such as Docker use kernel namespaces and cgroups to isolate containers. Docker uses the libcontainer library to build implementations on top of libvirt, LXC [LXC 2015; LXD 2015]. In this way, resources can be isolated and services restricted to let applications have a specific view of the operating system [Docker 2015a].

2.10 Known Uses

- Docker provides portable, lightweight containers, using Linux virtualization [Docker 2015a]. Docker was known to have security vulnerabilities but few security feature were added recently including hardware

signing for container image developers, digital signatures for authenticating images in repository and image scanning for known vulnerabilities [Babcock 2015].

- Cisco - Cisco Virtual Application Container Services automate the provisioning of virtual private data centers and deploy applications with compliant, secure containers [Cisco 2015].
- Rocket - A product of CoreOS. This container attempts to be composable, secure, open format and runtime components, and simple discoverable images [Polvi 2014].

In addition to these, Windows server containers and Hyper V containers supported for Windows Server 2016. [Bisson 2015]. Other container providers include Google, Amazon Web Services, IBM, Microsoft, and HP.

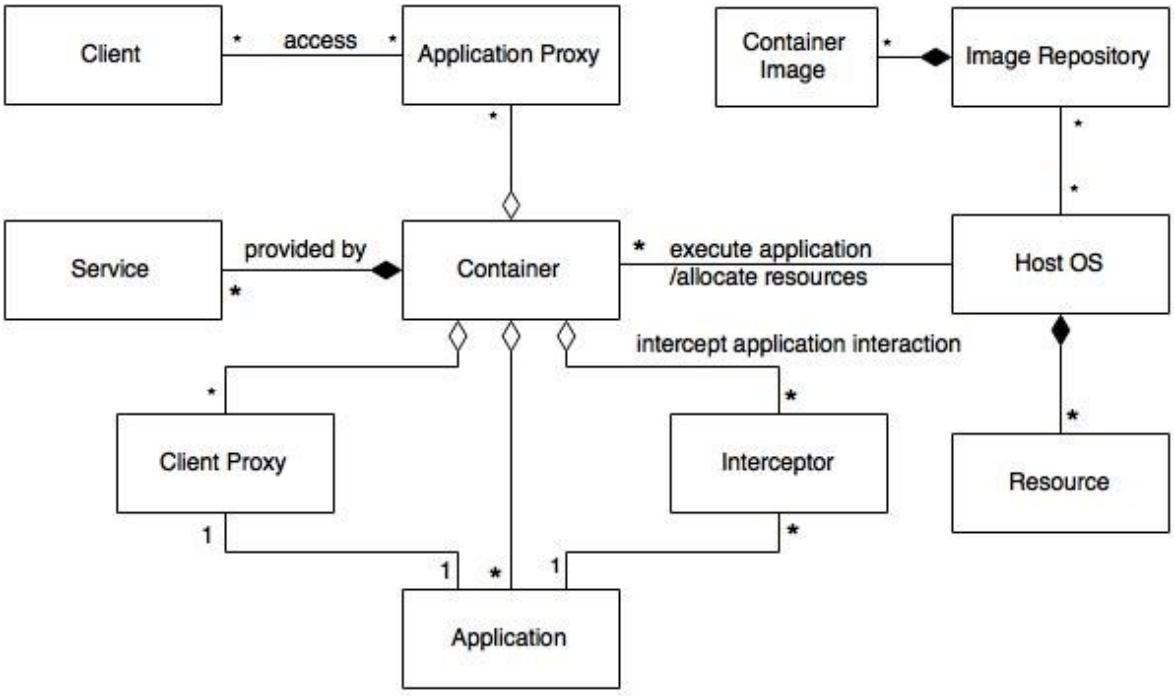

Fig. 3. Class diagram of the Container pattern

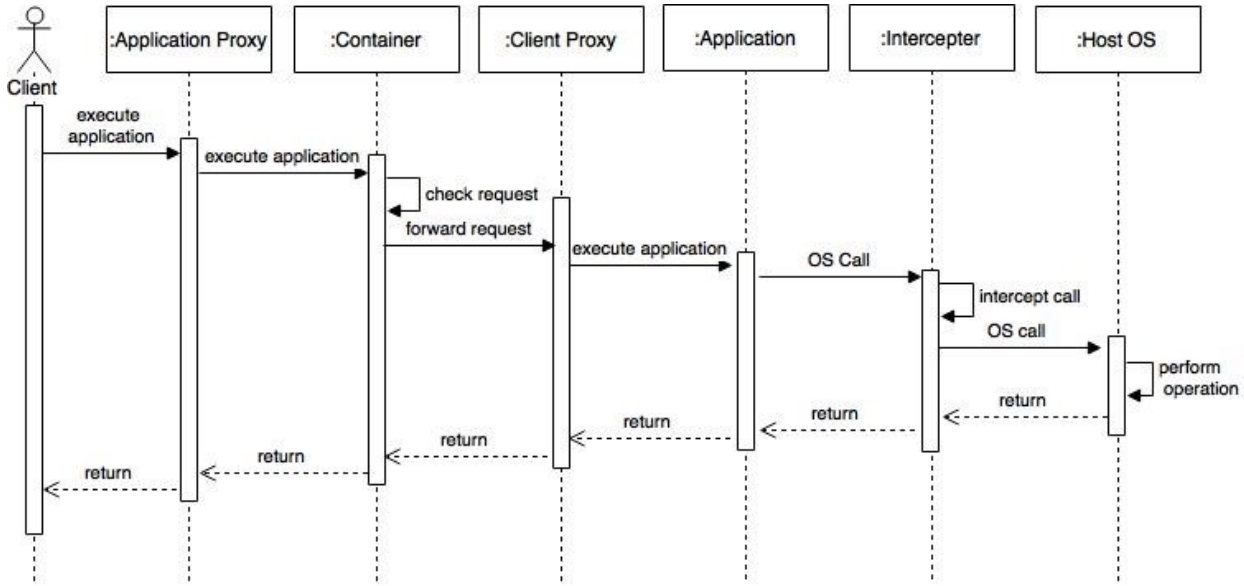

Fig. 4. Sequence diagram for the use case 'Execute an application in container'

2.11 Consequences

This pattern presents the following advantages:

- *Overhead*: Containers are more efficient since they do not require separate guest OSs as for the case of VMs (Compare Figure 5 with Figure 2). They will however, be slower as compared to directly executing an application on Host OS without virtualization.
- *Portability*: Containers can be executed in any processor and they can relieve application developers and testers of worrying about application distribution.
- *Controlled Execution*: Containers can control application execution as they can control and filter interactions with the Host OS.
- *Cost*: Host OS is shared by multiple containers, so unlike VMs we do not need to purchase separate licenses for Guest OSs on each VM.
- *Isolation*: The Container can use the facilities of the host OS to provide isolation between applications running on the same OS. This feature protects the other applications of attacks or errors in applications running in different containers.
- *Opaqueness*: Applications running in separate containers on the same OS are not aware of each other, which can prevent attacks.
- *Transparency*: The specific environment becomes transparent to the application when executed within a container. Changes made to the OS or Application can be handled by container modification, without affecting other containers.
- *Scalability*: Containers make the system more scalable since the number of applications sharing one OS can be increased provided enough hardware resources are available.
- *Extensibility*: Interceptors allow adding services such as logging/auditing, security, or others to an application.

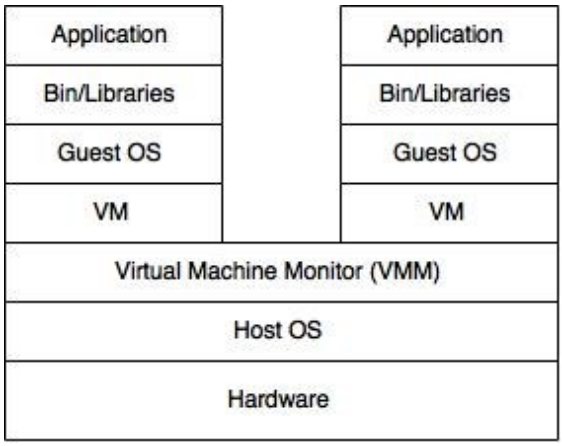

Fig. 5. A Virtual Machine

Liabilities of the pattern include the following:

- Use of containers can slow down application execution since we are using an additional layer of interceptors for indirection of messages between OS and application. However, this overhead should be smaller than using separate virtual machines.
- Containers are meant to provide isolation between applications, so if there is a need for collaboration between applications, the task becomes more difficult if they are executing in separate containers. This can be still be achieved for example, Docker provides an option to securely link containers through 'bridge' network [Links 2015].
- We can only use one OS in each container. If we need different operating systems we can have separate sets of containers or use virtual machines.
- Security or reliability flaws in the common OS affect all the applications running on it.

2.12 Example Resolved

The company started using containers on their servers. They built images for their applications so that each could run in a separate container. This provided lower overhead and more security for the whole system. In addition the use of containers and application images made distribution and deployment of applications very easy.

2.13 See also (related patterns)

- Interceptor [Schmidt et al. 2000]--allows services to be added transparently to a framework and triggered automatically in the present of specific events.
- Broker [Buschmann et al. 1996]--the Broker structures distributed systems with separate components that interact by remote service calls. A broker coordinates communications, including forwarding requests and sending back results and exceptions.
- Reference Monitor [Fernandez 2013]. In a computational environment in which users or processes make requests for data or resources, this pattern enforces declared access restrictions when an active entity requests resources. It describes how to define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations
- Virtual Machine Operating System [Fernandez 2013]--provides a set of replicas of the hardware architecture (Virtual Machines) that can be used to execute (maybe different) operating systems with a strong isolation between them.
- Controlled Virtual Address Space (Sandbox) [Fernandez 2013]. How to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access rights? Divide the VAS into segments that correspond to logical units in the programs. Use special words (descriptors) to represent access rights for these segments.

A formal analysis of component containers is presented in [Sridar 2006].

3. ACKNOWLEDGEMENTS

We thank our shepherd, Robert Hanmer, for his careful and insightful comments that have significantly helped to improve this paper. We also appreciate the valuable feedback we received from writer's workshop participants.

REFERENCES

Charles Babcock. 2015. Docker Tightens Security Over Container Vulnerabilities. (November 2015). Retrieved December 1, 2015 from http://www.informationweek.com/cloud/platform-as-a-service/docker-tightens-security-over-container-vulnerabilities/d/d-id/1323178.

Simon Bisson. 2015. First look: Run VMs in VMs with Hyper-V containers. (December 2015). Retrieved December 9, 2015 from http://www.networkworld.com/article/3013224/virtualization/first-look-run-vms-in-vms-with-hyper-v-containers.html.

Jan Bosch. 2009. From software product lines to software ecosystems. In Proceedings of the 13th International Software Product Line Conference (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, 111-119.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. 1996. Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. J. Wiley & Sons, Inc.

Cisco. 2015. Virtual Application Container Services. (June 2015). Retrieved June 22, 2015 from http://www.cisco.com/c/en/us/products/switches/virtual-application-container-servicesvacs/index.html

Docker. 2015a. Docker (April 2015). Retrieved April 10, 2015 from http://www.docker.com/.

Docker. 2015b. Using Supervisor with Docker. (December 2015). Retrieved December 17, 2015 from https://docs.docker.com/engine/articles/using_supervisord/

Eduardo B. Fernandez. 2013. *Security patterns in practice: Designing Secure Architectures Using Software Patterns*. J. Wiley & Sons, Inc.

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2015a. Building a security reference architecture for cloud systems. *J. Requirements Engineering* (June 2015), 1–25. DOI:10.1007/s00766-014-0218-7

Eduardo B. Fernandez, N. Yoshioka and H. Washizaki. 2015b. Patterns for Security and Privacy in Cloud Ecosystems. In *2nd International Workshop on Evolving Security and Privacy Requirements Engineering* (ESPRE 2015), IEEE (August 2015), 13-18. DOI:10.1109/ESPRE.2015.7330162

Container Links. 2015. Legacy container links. (December 2015) Retrieved December 1, 2015 from https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/#container-linking.

Linux LXC. 2015. Linux Containers - LXC - Introduction. Retrieved June 9, 2015 from https://linuxcontainers.org/lxc/introduction/

Linux LXD. 2015. Linux Containers - LXD - Introduction. Retrieved June 9, 2015 from https://linuxcontainers.org/lxd/introduction

Alex Polvi. 2014. CoreOS is building a container runtime, rkt. (December 2014). Retrieved April 25, 2015 from https://coreos.com/blog/rocket/.

Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. 2000. Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects. J. Wiley & Sons, Inc.

Nigamanth Sridar and Jason Hallstrom. 2006. A behavioral model for software containers. In *Proceedings of 9th International ConferenceFundamental Approaches to Software Engineering* (FASE 2006). Springer. 139-154. http://dx.doi.org/10.1007/11693017_12.

Joe Topjian. 2013. Contain your enthusiasm - Part Two: Jails, Zones, OpenVZ, and LXC. (November 2013). Retrieved December 17, 2015 from http://www.cybera.ca/news-and-events/tech-radar/contain-your-enthusiasm-part-two-jails-zones-openvz-and-lxc/.

Langdon White. 2014. Containers for Development. (September 2014). Retrieved November 30, 2015 from http://www.drdobbs.com/architecture-and-design/containers-for-development/240168801?pgno=1.

Wikipedia. 2015. Operating-system-level virtualization. Retrieved December 18, 2015 from https://en.wikipedia.org/wiki/Operating-system-level_virtualization.

Received May 2015; revised September 2015; accepted February 2016