The Software Container pattern

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

Abstract
We present here a pattern for a Software Container. 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.

Keywords: software containers, architecture patterns, security, virtualization, security patterns

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.

AKA
Zone, Jail

Example
Our organization has development and testing teams working at distributed locations. We need a quick and easy way to provide a standardized environment to execute and test all kinds of applications. Even if we have application from different companies running on our system, we have to provide each one a standard platform for execution. For a cost-effective operation we need to reduce overhead and improve security. Our applications execute in a few operating systems.

Context
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.

Problem
We want to be able to run applications in self-contained environment in such a way that both (application and 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 processor to another processor, 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.

Solution

Provide a runtime environment that can support the isolated execution of applications on a shared Host OS, this is a *Software Container* (Figure 1). Multiple applications can share one container. They also may share binaries and libraries with other containers. Containers provide isolated execution and extensible services to the application.

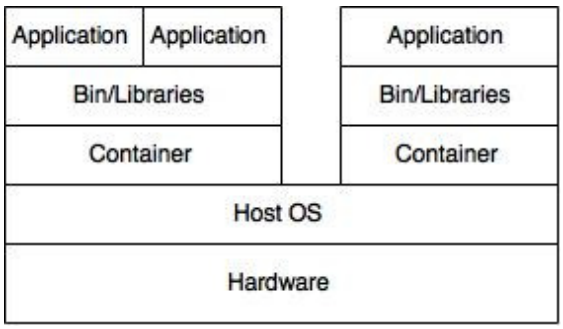

Figure 1. Two containers sharing one OS

Structure

Figure 2 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.

Dynamics

Figure 3 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.

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 (Linux LXC). In this way, resources can be isolated and services restricted to let applications have a specific view of the operating system (Docker).

Known Uses

- Docker provides portable, lightweight containers, using Linux virtualization (Docker).
- Cisco (Cis)--Cisco Virtual Application Container Services automate the provisioning of virtual private data centers and deploy applications with compliant, secure containers.
- Rocket (Rocket)—A product of CoreOS. This container attempts to be composable, secure, open format and runtime components, and simple discoverable images.

Other container providers include Google, Amazon Web Services, IBM, Microsoft, and HP.

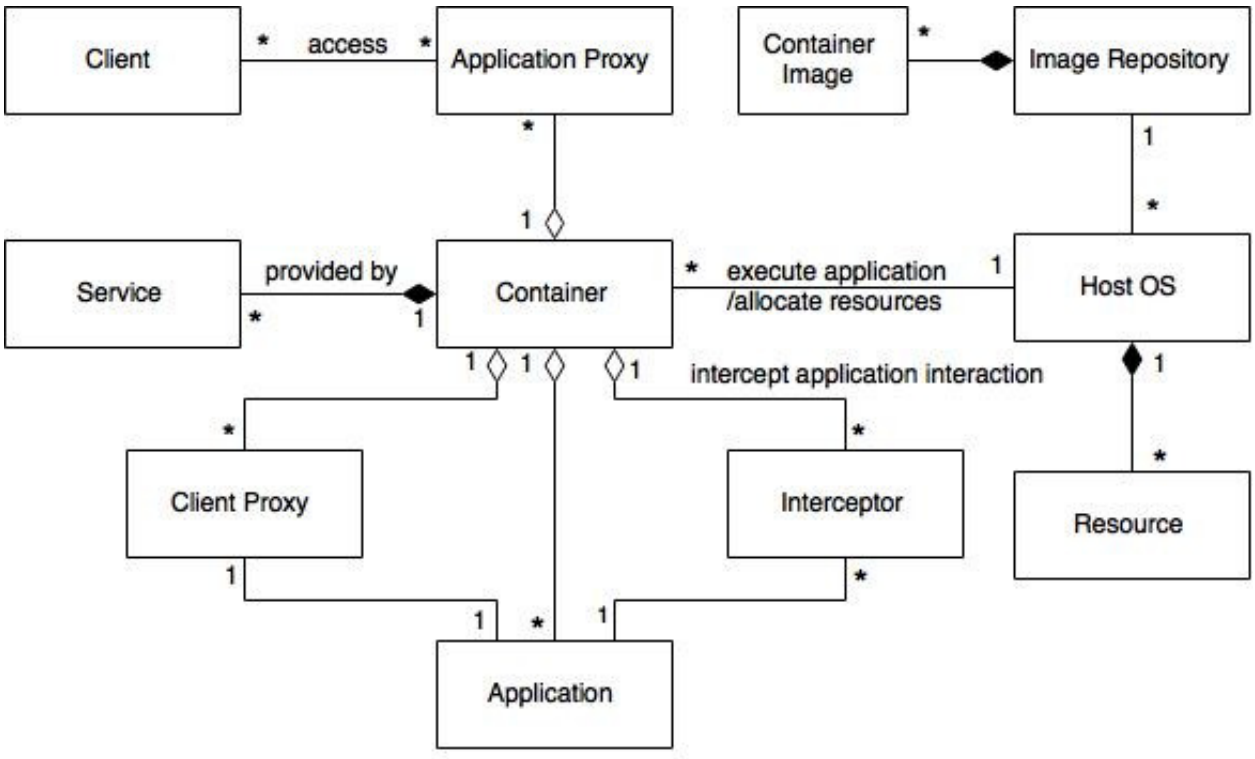

Figure 2. Class diagram of the Container pattern

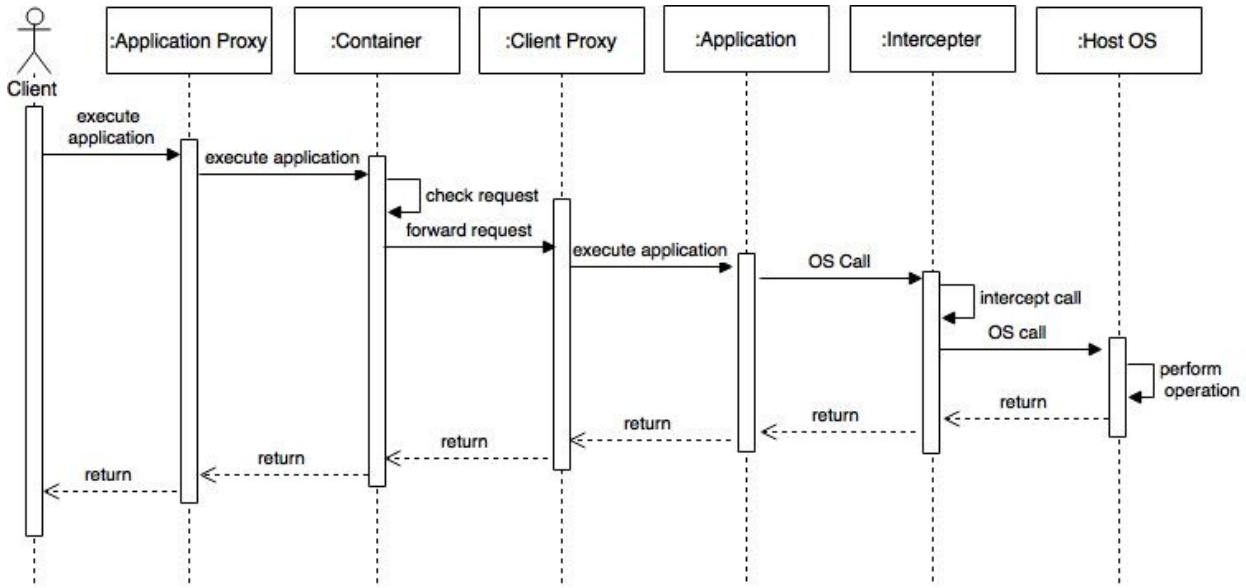

Figure 3. Sequence diagram for the use case 'Execute an application in container'

Variants

Component Containers are used in Java EJBs (Sridar and Hallstrom, 2006). A pattern for Component Containers is given in (Kobryn00).

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 4 with Figure 1).
- *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.

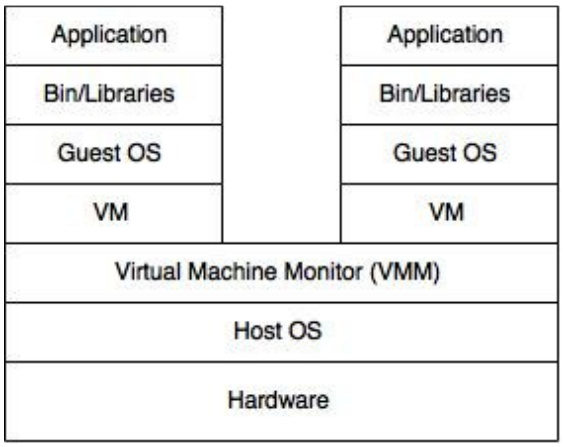

Figure 4. A Virtual Machine

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.
- 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.

Example resolved

The company installed a container manager on their server's hardware, and then installed various applications such that each could run in a separate container. This provided lower overhead and more security for the whole system.

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 and Hallstrom, 2006).

Acknowledgments
We thank our shepherd, Robert Hanmer, for his careful and insightful comments that have significantly helped to improve this paper.

References

Cisco, Virtual Application Container Services, http://www.cisco.com/c/en/us/products/switches/virtual-application-container-services-vacs/index.html

Docker, http://www.docker.com/

E. B. Fernandez, *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*, J. Wiley Sons, May 2013.

FreeBSD, FreeBSD Jails (https://www.freebsd.org/doc/en/books/handbook/jails.html)

C. Kobryn, "Modeling components and frameworks with UML"**.** Commun. ACM 43(10): 31-38 (2000)

http://www.jot.fm/issues/issue_2004_04/article9.pdf

Linux LXC, https://linuxcontainers.org/lxc/introduction/

Linux LXD, https://linuxcontainers.org/lxd/introduction/

Rocket, "CoreOS is building a container runtime, rkt", https://coreos.com/blog/rocket/

Anand M. Sankar, "Containers (Docker): A disruptive force in cloud computing", March 28, 2015. http://anandmanisankar.com/posts/container-docker-PaaS-microservices/

Douglas Schmidt, Michael Stal, Hans Rohert, and Frank Buschmann, *Pattern-oriented software architecture: Concurrent and networked objects*, John Wiley and Sons, 2000

Nigamanth Sridar and Jason O. Hallstrom. "A behavioral model for software containers." *Fundamental Approaches to Software Engineering*. Springer Berlin Heidelberg, 2006, 139-154. http://selab.csuohio.edu/~nsridhar/research/Papers/PDF/abfms.pdf