ISSN 2395-1621

A survey paper on Docker Containers for a consistent IoT development

[#1]Amit Bijave, [#2]Sagar Ippalpalli, [#3]Rajat Kamble, [#4]Rohan Modi
[#5]Prof. Deepali Javale

[1]amit.bijave@gmail.com
[2]cdimal09@gmail.com
[3]rajatkamblemailbox@gmail.com
[4]rohanindia5@gmail.com

[#1234]Department of Computer Engineering M.I.T.College Of Engineering
Savitribai Phule Pune University, Pune, India

ABSTRACT

In this world, developing and deploying new applications quickly is very important, but there is a problem because of number of platforms available in market. Developers need to write different code for different platform. Hence to test them we need to buy different machines, which means more cost. But we can overcome this problem using Virtual Machine (VM), but they are heavy. Thus in this paper we discuss about use of containers and Docker platform to solve porting and other issues.

Keywords— Linux Containers, Internet Of Things, Embedded Systems, Operating-system-level virtualization, Linux Kernel, ARM Architecture, Raspberry Pi, Distributed computing.

ARTICLE INFO

Article History

Received :9th 2015

Received in revised form :

31st December 2015

Accepted :5th January , 2016

Published online :

6th January 2015

I. INTRODUCTION

LXC (Linux Containers) is an operating-system-level virtualization environment for running multiple isolated Linux systems (containers) on a single Linux control host. The goal is to offer a distro and vendor neutral environment for the development of Linux container technologies. Our main focus is system containers. That is, containers which offer an environment as close to possible as the one you'd get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware. This is achieved through a combination of kernel security features such as names- paces, mandatory access control and control groups. [1]

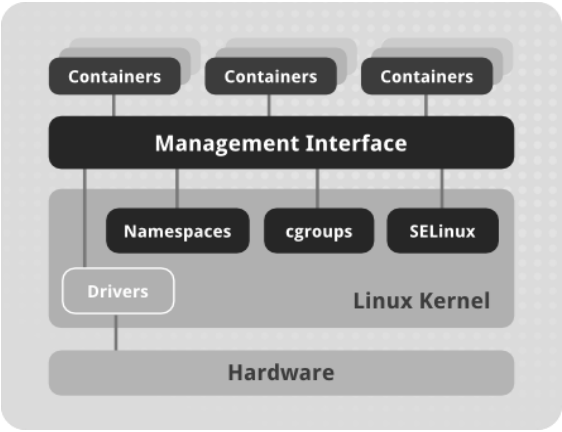


Fig1:Container Arcitecture

II. DOCKER

Docker allows you to package an application with all of its dependencies into a standardized unit for software development. Docker containers wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, and system libraries – anything you can install on a

server. This guarantees that it will always run the same, regardless of the environment it is running in. [3]

Docker Swarm is native clustering for Docker. It allows you create and access to a pool of Docker hosts using the full suite of Docker tools. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts. [5]

III. DIFFERENCE BETWEEN CONTAINER AND VM

Virtual Machine:Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size. [3]

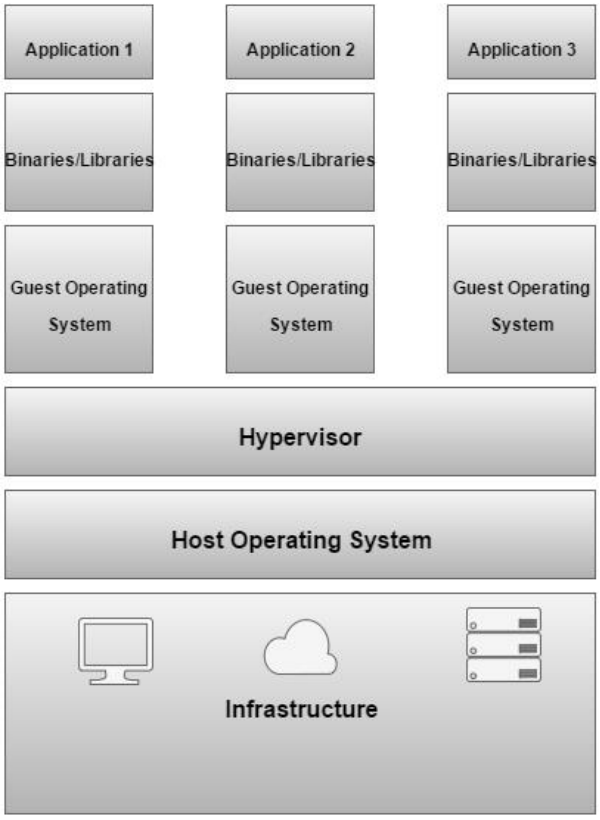


Figure 2: Virtual Machine [3]

Containers: They include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure – Docker containers run on any computer, on any infrastructure and in any cloud. [3]

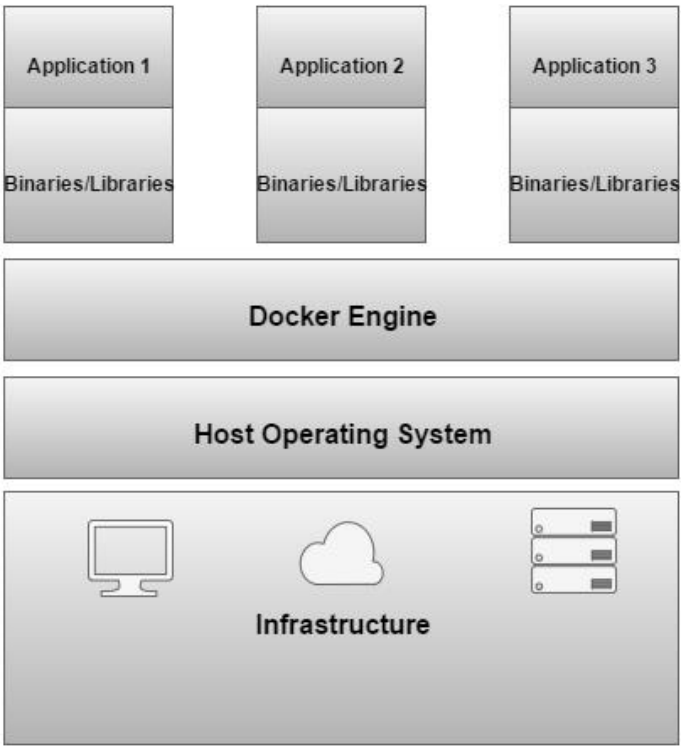


Figure 3: Containers [3]

IV. DOCKER IS NOT REPLACEMENT FOR VIRTUAL MACHINE

Because the Docker platform has become so popular so quickly, some have suggested completely abandoning VMs in favour of application containers in order to take advantage of the cost benefits. It's important to avoid getting caught up in the hype and realize that both VMs and containers are needed. For example, the recommendation is to one run application per Docker container, using a Linux operating system. A VM, in contrast, is capable of supporting virtually any operating system. Also, new VMs are being developed to behave more like containers. Ultimately, the better strategy is to use containers in addition to, and not instead of, VMs.[3]

V. ADVANTAGES OF CONTAINERS

1.Abstraction of the host system away from the containerized application

Containers are meant to be completely standardized. This means that the container connects to the host and to anything outside of the container using defined interfaces. A containerized application should not rely on or be concerned with details about the underlying host's resources or architecture. This simplifies development assumptions about the operating environment. Likewise, to the host, every container is a black box. It does not care about the details of the application inside. [6]

2. Easy Scalability

One of the benefits of the abstraction between the host system and the containers is that, given the correct application design, scaling can be simple and straight-forward. Service-oriented design (discussed later) combined with containerized applications provide the groundwork for easy scalability.

A developer may run a few containers on their workstation, while this system may be scaled horizontally in a staging or testing area. When the containers go into production, they can scale out again. [6]

3. Simple Dependency Management and Application Versioning

Containers allow a developer to bundle an application or an application component along with all of its dependencies as a unit. The host system does not have to be concerned with the dependencies needed to run a specific application. As long as it can run Docker, it should be able to run all Docker containers.

This makes dependency management easy and also simplifies application version management as well. Host systems and operations teams are no longer responsible for managing the dependency needs of an application because, apart from a reliance on related containers, they should all be contained within the container itself. [6]

4. Extremely lightweight, isolated execution environments

While containers do not provide the same level of isolation and resource management as virtualization technologies, what they win from the trade-off is an extremely lightweight execution environment. Containers are isolated at the process level, sharing the host's kernel. This means that the container itself does not include a complete operating system, leading to almost instant start-up times. Developers can easily run hundreds of containers from their workstation without an issue. [6]

5. Shared Layering

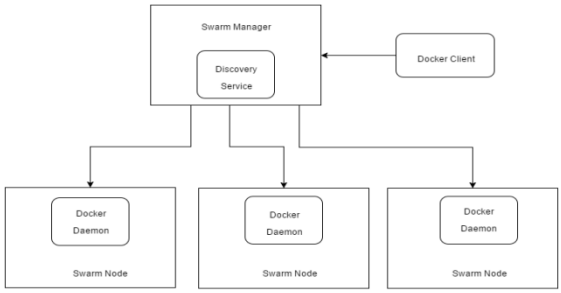
Containers are lightweight in a different sense in that they are committed in "layers". If multiple containers are based on the same layer, they can share the underlying layer without duplication, leading to very minimal disk space utilization for later images. [6]

6. Composability and Predictability

Docker files allow users to define the exact actions needed to create a new container image. This allows you to write your execution environment as if it were code, storing it in version control if desirable. The same Docker file built in the same environment will always produce an identical container image. [6]

VI. TOOLS AND COMPLEX WORKFLOWS

Regarding Orchestration it is necessary to have a convenient management of a large number of containers, so to manage applications across multiple containers and hosts. There are several tools from Docker to orchestrate your application. Docker Compose allows you to run your stack with only one command and describe your multi-app with one file. Docker Swarm helps to cluster your application in a distributed environment. It pools several Docker Engines and exposes them as a single virtual Docker engine. Beware that these tools Docker provides are still in development (beta) and therefore should not be used in production yet. But there are tons of alternative implementations from third party developers out there that you could easily plug into your dev environment. If you are interested in Compose alternatives, check out Core OS, Amazon Web Services or Google Compute Engine. For Swarm i.e. Apache Mesos, Google Kubernetes or Core OS Fleet. [4]



Swarm Architecture Design

Figure 4: Docker Swarm

VII. MINIMALISTIC OPERATING SYSTEM

Instead of just using Ubuntu as base image for docker containers or host operating system, there are other distributions available - specialized for the use with containers. Examples are: Core OS, Ubuntu snappy core, project atomic, rancher OS, busybox, staticpython and much much more (just google a little or look in docker hub to find tons of them). Most of those distributions are smaller than typical Linux distros and reduce the overhead by delivering only tools that are necessary in container context. Some like Core OS even deliver clustering solutions and improved stability and security or they are intended to provide the best environment for one particular use case, such as being the smallest distribution that can still handle python apps. [4]

VIII. ADVANTAGES OF DOCKER

Docker is more lightweight than VMs and offers therefore a minimal overhead and resource usage. This makes it cheaper as well because you can run several Docker containers on only one VM so you do not need several VMs. Another point is the speed of starting a container. Running multiple containers on one host is easy and the containers itself are isolated from each other. It offers many advantages for DevOps such as sharing of containers, e.g. for a local development with a GUI for a container. Another big advantage is the tremendous ecosystem that has developed around docker. The public repository DockerHub offers many pre-built images that you can use for your own purpose. And there are lots of tools available that can complement your docker experience with customized solutions. It is very positive to note that most of the third party developers share Dockers' idea of "openness" and provide pluggable modules that can be combined with others.[4]

IX. DISADVANTAGES

Personally we could not find many disadvantages of Docker. But because there are many beta tools so far (e.g. Swarm), orchestration needs to be provided by other tools to make your application scalable. Docker has got a big growing ecosystem, so it can take a while for you to find the right tools to combine Docker with. Although Docker is easy to learn, we needed some time to be really into it.

But there is a need to say that there are some serious security concerns as e.g. the possibility of escaping the 'jail'.

There are many posts from Docker as well and they seem to work hard on fixing all the security concerns many people have mentioned. Another point is the problem with security bugs in existing images on DockerHub. If you use an existing image, please check it first and make sure it can't be affected by Heartbleed or any other bugs anymore. So at the moment you still have to be very careful to run Docker containers in production environments. [4]

X. CONCLUSION

1. Docker provides the fundamental building block necessary for distributed containerdeployments. By packaging application components in their own containers, horizontal scaling becomes a simple process of spinning up or shuttingdown multiple instances of each component.
2. Docker provides the tools necessary to build containers and manage and sharethem with new users or hosts.While containerized applications provide thenecessary process isolation and packaging to assist in deployment, there aremany other components necessary to manage adequately and scale containersover a distributed cluster of hosts.

REFERENCES

[1]https://linuxcontainers.org/

[2]https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/version-7/getting-started-with-containers/

[3]https://www.docker.com/what-docker

[4]https://learning-continuous-deployment.github.io/docker/conclusion/2015/06/14/conclusion/

[5]https://docs.docker.com/swarm/

[6]https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-overview-of-containerization

[7]https://engineering.riotgames.com/news/thinking-inside-container

[8]https://www.ibm.com/developerworks/library/iot-docker-containers/

[9]https://github.com/LalatenduMohanty/container-workbook

[10]http://www.slashroot.in/difference-between-hypervisor-virtualization-and-container-virtualization

