Containerization for Layman!

Published on September 7, 2018


Aayush Shrut
Consultant at Deloitte

11 articles + Follow

In our previous articles of Cloud Computing, we discussed at length about Virtualization. We also deep dived into the most used (and free!) private cloud provider called OpenStack.

In this article, we will do a brief overview of the next evolution of Virtualization called "**Containerization**". We will go deep down into the basic building blocks of Containers, and analyze the differences between Containers and Virtual Machines. If you are new to Cloud computing and virtualization, please read this to establish a basic primer about the terminologies to be used in this article. As has been the USP of this series, we will try to use the power of analogies with a sprinkle of humour to get through the technical "terminologies" (or fancy words).

What are Containers?

Every new concept in Computer Science is aimed to solve problems posed by previous concepts. So to understand Containers, let's first understand what were the problems with Virtual Machines or Virtualization?

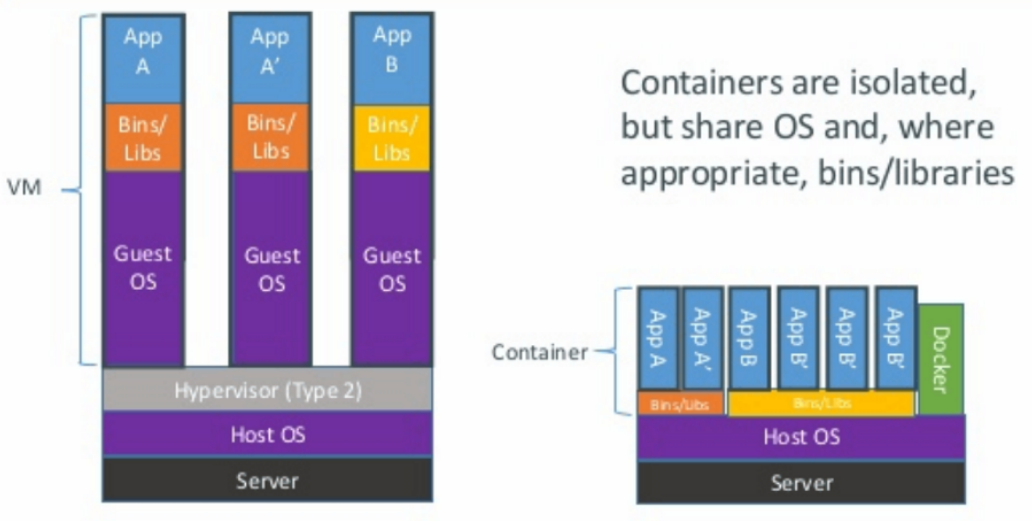
We know that Virtual Machines uses software called H

emulate a new system. In simple terms, this means that each VM creates its own kernels (cloning OS), which uses Hypervisors to translate virtual kernel calls to real system calls.

While VM works fine if we are running every component on a single platform or OS (*monolithic software architecture*). However, modern application software needs to run on a tablet to windows to even your refrigerators. They are also created in a way that a group of loosely coupled lightweight services are integrated to create complex software (*micro-service software architecture*).

Testing microservices using VMs leads to issues related to a.) **Logistics** (each VM requires lots of CPUs, Memory, Disk etc. for emulation to work), b.) **Performance** (due to associated context switching) and c.) **Efficiency** (as multiple applications running on different VMs will require complex synchronization and dependencies resolution).

To solve these problems, Containers come in the picture. Containers are alternates to hypervisor driven virtualization. Instead of cloning the kernel, containers use the host OS's kernel itself to create virtual environments. Hence in a way, Containers are "**operating system level virtualization**". They provide a fast and easy way to run our software without the associated problems of logistics, performance or efficiency.



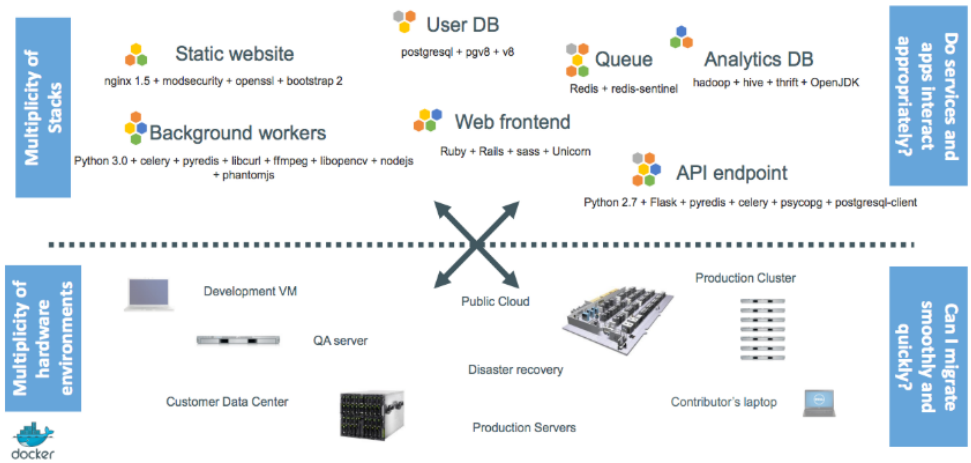
Think of containers as **VMs, but without the hypervisors**. So your software running on containers will behave as if running on a new computer (same like VM). But instead of using hypervisors to isolate resources as CPUs, Network cards or HDDs, containers will use OS concepts such as Cgroups and Namespaces to isolate

means that while we won't need OS to create VMs, **we need OS to create containers**.

So this means Virtualization is dead?

No. Virtualization has a more mature and industry accepted ecosystem. In fact, in a production environment, containers are rarely used (although Amazon has started using it recently).

But think of a use case, where, as a simple software developer, all I want to is to write code and expect my software to work in every environment possible (whether tablet, iPhone, or Linux). Now, instead of creating VM images for each test environment (Go, Swift, iOS, Django, Node.js, React, <insert fancy framework/OS> etc.), what if I just ship all the dependencies in a "container", and expect it to run everywhere. The huge business cost of tedious manual platform testing (along with associated resource constraint) is saved. The **compatibility nightmare and testbed preparation** that Ops team went through during integration testing (famously called *integration hell*) can finally be over and we can all go home early to our family and friends!



This **robust application portability across different stacks and platforms** is the USP of containers. In fact, it has started a whole new culture of application development called *DevOps*, where the code is rapidly deployed, tested, and redeployed by the developers itself, bridging the interface between Developers and Operations.

However, the other side is, that for large enterprise systems that rely more on security and complex networking (NFV, SDN,

virtualization still serves the purpose. This is because **containers rely on OS to isolate resources**. So anyone with a root access to OS can exploit your containers. VMs, since they emulate their own OS, are isolated from the host OS.

What is happening now in the industry is that a mix of containers + VM systems are being used. Containers are being run on top of VMs (as containers require only OS, virtual or really doesn't matter). Different microservices are being run as containers, which are being run on VMs, which are supporting other monolithic complex functions (like Network virtualization).

That being said, we simply can't ignore the advantages of Containers. It provides super fast deployment (milliseconds), is easily scalable, has no infrastructure overhead (can be run on virtually anything), uses less memory and CPU, and leaves virtually zero footprints.

I can go home early?? Quickly show me how do Containers work!!

At the heart of container technology are three OS level concepts called **Cgroups, Namespaces and Union Filesystems**. These have been existent for a long time, and in fact, we have used them from time to time (like jailbreaks to run toxic processes in an isolated environment or coding platforms running your codes). Let's try to analyze each of them.

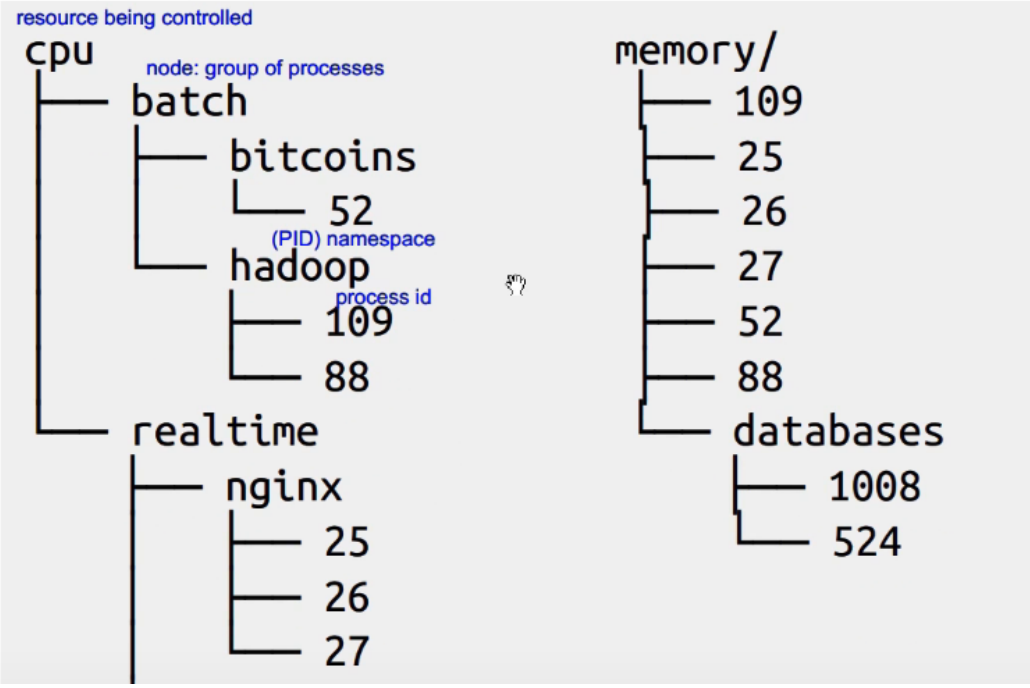
Cgroups:

Control Groups provide a mechanism to organize processes hierarchically and impose limits on how much resources they can use. So for example, a resource such as Firefox can have a limit of using 100MB disk space (using blkio cgroups). Another resource such as an admin user can have a limit of using 2 CPUs only (using CPU cgroups). In layman terms, think of Cgroups as something which defines "*how much to use*".

NameSpaces:

They are the core OS level concept behind Containerization and the absolute fundamental on which containers work. NameSpaces allow us to an abstract global system resource such as network interfaces (net) or mount points (mnt). This creates the illusion of a process running in the namespace that they have the resources all by themselves. Pretty much like virtualization, except we use OS, instead of

create this virtualization possible. In layman terms, think of NameSpaces as something which defines "*what to use*".



Union File Systems (UnionFS):

UnionFS are a way of linking different file systems in a layered manner. Essentially, it allows different file system directories and files (called branches), to be overlaid on top of each other to create an entirely new virtual filesystem. For example, in Linux, we can mount CD ROMS (which are read-only), and can write on them. This is possible by Union FS. In layman terms, think of it as something that "unifies" different file systems (or directory structures) in one coherent manner.

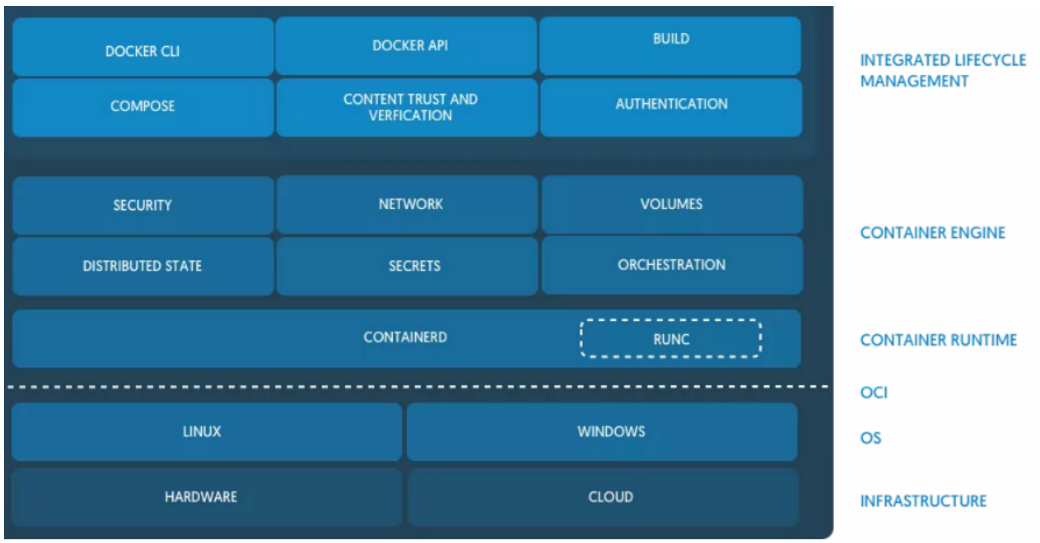
Container Run Times:

NameSpaces, Cgroups and UnionFS, they all have been in existence for a long time. So how come containers came recently into the picture (2013 to be precise)? This is because using these features of Linux Kernels to create containers is not an easy process. Hence, to save us from nightmares and facilitate easy creation of containers, **Container Run times act as the middleware software**. They use these complex kernel features to create ready to deploy containers.

Now, as we know, container run times use OS features. It may so happen that some vendors can launch their own OS+Container Run Time combo, and start WW3: Container Wars in the process

for our container run times or vice versa). This will beat the purpose of containers in the first place, which was to facilitate easy deployment and remove dependency issues.

Hence, to prevent war and make peace, a group of IT companies came together and formed an open governance structure, called *The Open Container Initiative*, under the auspices of *The Linux Foundation*. The governance body came up with specifications to create standards on Operating System process and application containers. **runC** is the CLI tool for spawning and running containers according to these specifications.



The most popular **implementation of runC tool is Containererd**, which is an OCI compliant container runtime. And the most popular software, using Containererd, is **Docker.**

Conclusion

This concludes part 1 of Containers for layman. We discussed containerization, and why it was needed to solve performance and compatibility issues in virtualization. We also did a comparison of containers with VMs, and the different use cases for both. We then discussed the fundamental building blocks of Containers: NameSpaces, UnionFS and Cgroups. We then saw how Container Run times uses these features to create our containers.

In the next part, we will discuss the industry standard container software called "Docker", and the enterprise container orchestration software called "Kubernetes" in detail.

Writer's Note: *If you like this post, do take a minute to like, share or comment :). You can also read some of my previous work here.* As always, *sharing is caring!*

Report this

Published by

Aayush Shrut
Consultant at Deloitte
Published • 1yr

11 articles + Follow

#kubernetes #docker #containers #microservice #cloud #computing #containerization #virtualization #cgroups #kernel #namespaces #unionfs #introduction #architecture #overview #layman #dummies #help

Like Comment Share 10 · 5 Comments

Reactions

5 Comments

Most Relevant

Add a comment...

Kinga Kięczkowska • 3rd+
Security Engineer at Fortinet
11mo ...

Thank you for your article - it's really insightful and a great starter for people like myself who have never worked with containers before. That said, could you please consider using more inclusive language in the future? '[...] we can all go home early to our wife/children/friends!' sounds as if you assume your audience is entirely male. Again, thank you for the content :)

· 3 Likes · 2 Replies

Aayush Shrut • 2nd
Consultant at Deloitte
11mo ...

Thanks for the praises, and the insight. Apologies for being sexist, it was surely unintentional xD. Edited it! We can all go home early to our family and friends :).

· 1 Like

Kinga Kięczkowska • 3rd+
Security Engineer at Fortinet
10mo ...

Thank you so much :)

Vikram Ranabhatt • 3rd+
Sr Engineer at GE Healthcare
1y ...

As you said containers share underlying os. So one container can access items saved (let us say file) by other container in hte os?

· 1 Reply

Aayush Shrut • 2nd
Consultant at Deloitte
1y ...

Nope, they are separated by namespaces, and have resource restriction by Cgroups. Pretty much like one VM can't access another VM's data, but share the same hypervisor.

Messaging

1 Like

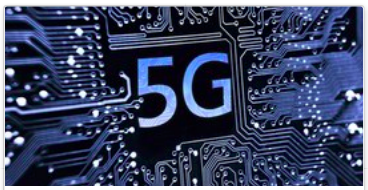
Aayush Shrut
Consultant at Deloitte

+ Follow

n Aayush Shrut

Layman
ut on LinkedIn

Docker and Kubernetes for Layman
Aayush Shrut on LinkedIn



5G and Network Slicing for Layman
Aayush Shrut on LinkedIn

LTE for Layman (part 3) - The Complete Picture!
Aayush Shrut on LinkedIn

rticles

Messaging