Containers

**Deployment Environments**

**Development Environment** — This is where the software is developed. In some situations this could be the developer’s desktop, in other situations this would be a server shared by several developers working together on the same project. This environment should resemble the production environment as much as possible to prevent issues where the software acts differently on production.

**Test Environment** — After the application has been developed to an agreed stage it is released to the testing environment. This is where the testers ensure the quality of the application, open bugs and review bug fixes. This environment must resemble the production environment accurately, because this is the last safe place to find and fix environment-related bugs.

**Staging Environment** — The staging site is used to assemble, test and review new versions of a website before it goes into production. The staging phase of the software lifecycle is often tested on hardware that mirrors hardware used in the production environment. The staging site is often different from the development site, and provides a final Quality Assurance zone that is separate from the development or production environments.

**Production Environment** — This is where the application goes out to the world and become production. Content can be updated from the staging environment in to Production Environment, when available, as well as new application functionality and bug fixes release from UAT or staging environment.

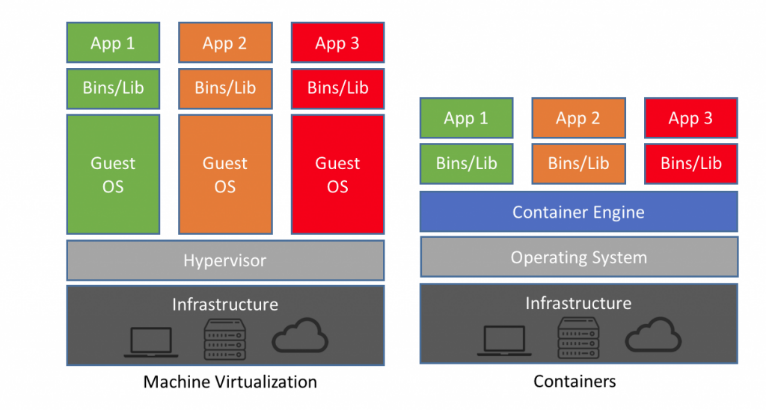
**What is Container?**

A container is similar to an application, which runs as a process on top of the operating system (OS) and isolates with each other by running in its own **address space**. Nevertheless, more than a normal process, a container not only includes the application executable, but also packs together all the necessary software’s that the application needs to run with, such as the libraries and the other dependencies.

Using containers is another way of packing applications in a much lighter weight and with a much faster delivery model. They are a fancy way of running multiple application processes on a single box, regardless of whether that box is a VM or a physical machine.

Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.

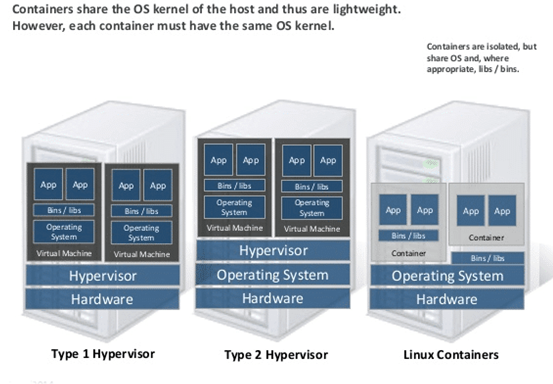
**Role of Host and Guest Operating systems?**

****

**Operating system level virtualization**

Operating-system-level virtualization is a server-virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances, which are sometimes called containers and software containers.

**Hypervisor vs Linux Container**



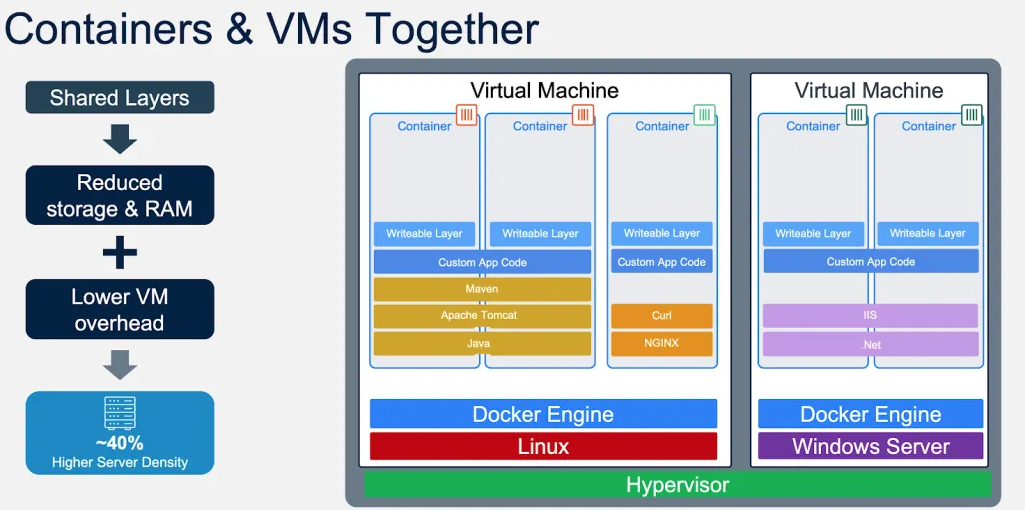
**VMs vs Containers**

In traditional virtualization, a hypervisor virtualizes physical hardware. The result is that each virtual machine contains a guest OS, a virtual copy of the hardware that the OS requires to run, and an application and its associated libraries and dependencies.

Instead of virtualizing the underlying hardware, **containers virtualize the operating system (typically Linux)** so each individual container contains only the application and its libraries and dependencies. Containers are small, fast, and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can, instead, simply leverage the features and resources of the host OS.

Just like virtual machines, containers allow developers **to improve CPU and memory utilization of physical machines.** Containers go even further, however, because they also enable microservice architectures, where application components can be deployed and **scaled more granularly**. This is an attractive alternative to having to scale up an entire monolithic application because a single component is struggling with load.

**Do Vm can carry the multiple containers?**



**Advantages**

**Platform Independent**

The biggest advantage of using Containerization is that the applications are platform-independent. A container will already contain everything that the application needs. It will come with various configuration dependencies and files. This will allow you to run your application on any computer you want. You can run applications on your physical server, local desktop or virtual server. Cloud users can also run their applications on private or public clouds. This offers great flexibility to organizations. Containers will also help you in speeding up the development process. You can easily switch from one cloud provider to another cloud provider. Also, you don’t need to worry about installing a different operating system in your system.

**Efficient**

You **don’t need any separate OS for running your applications**. Thus, containers will use fewer resources. VMs can take a lot of your computer resources**. You need more than 1 gigabyte of hard disk for running your virtual machine.** On the other hand, you only need a few megabytes for running your containerized application. Thus, you can run several containers on a single machine. Containers also have a high utilization level. Hence, containers are more efficient. It will help you in simplifying and reducing your regulatory compliance costs.

**Effective resource sharing**

You can run many containers on one server. Thus, they also use the **same pool of resources**. But, these applications will never communicate with each other. If one of your application crashes, then other applications will still keep running. They won’t face any technical issue. This will also help you in **decreasing the security risks**. Hackers can hack into your whole network by hacking into any application. But, if you are using containerization then hackers can’t use hacked applications for connecting with your network. Thus, it won’t have any effect on other applications.

**Speed**

Containers are lightweight when compared to VMs. You can start them in a fraction of seconds. If you are using **VM, then you need to boot an operating system first**. Also, you need to set up various things before running your VM. You can destroy or **create new containers in seconds**. Also, you can replicate applications in seconds. This will help allow your developers to work more effectively. It will also help you in improving your customer experience. Your developers can act quickly. This will help them in fixing bugs quickly.

**Reproducibility**

If you are using containers, then your application file systems will remain the same throughout the development phase. Version control will replace configuration management. This will help you in managing the different versions of your applications.

This will allow your developers to work more quickly. It will also increase the flexibility and efficiency of your applications. Your IT team doesn’t need to install VMs for testing their applications. Also, they don’t need to debug your application separately for every platform.

**Easy to operate**

In normal virtualization, **you need various VMs for testing your applications**. These VMs consume a lot of your system resources. If you are using container technology, then you don’t need to worry about installing any OS. Your application will be isolated from your operating system. Thus, you can run your applications on your normal systems. Your developers can quickly apply security patches and updates. This will also increase the productivity of your developers.

**Improved productivity**

There are many benefits of using container technology. A container will make sure that your application can run on any platform. You don’t need to code separately for every platform. This will also remove any environmental inconsistency. Thus, the debugging and testing process will become much easier. Your developers don’t need to devote hours to testing and debugging applications. All they need to do is test the application in their local system. Also, they can update your applications. You can destroy and create new containers in second. Thus, you can save a lot of your time. Tools like Docker also offer various features like version control. This will allow you to roll-out new updates.

**Disadvantages**

**????????**

**Terminologies**

**User space / Kernel Space**

Kernel space is strictly reserved for running a privileged operating system kernel, kernel extensions, and most device drivers. In contrast, user space is the memory area where application software and some drivers execute.

**User Space-** It is set of locations where normal user processes run. These processes can't access kernal space directly. Some part of kernal space can be accessed via system calls.These system calls acts as software interrupts in kernal space.

**Kernal Space -** kernal runs in the dedicated part of memory. Role of kernal space is to manage applications/ processes running in user space. It can access all the memory. If a process perform a system call, a software interrupt is sent to kernal which then dispatches an appropriate interrupt handler.