Containers vs Virtual Machines: What’s The Difference?

Virtualization transformed networked computing in the early 70’s, paving way for unprecedented paradigm shifts such as cloud computing in recent decades. Not so long ago, the Container revolution emerged with a similar goal toward improvements in data center technologies and application development. The technology isn’t entirely new. Linux container solutions including LXC and Solaris Zones have been in the industry for over a decade. Enterprises including Google have been using their own container technologies for several years.

However, it wasn’t until Docker launched in 2013 with its developer-friendly container solutions and ecosystem that the technology truly gained traction in the enterprise IT industry. In fact, we’re potentially heading toward an era
where traditional virtualization methodologies will give way to containerization. Before organizations embrace container-based development solutions for their app development and software release processes, it is important to understand the concepts and operation of virtual machines, containers and the key differentiators between them.

**BMC Helix - The Future of Service and Operations Management**

**BMC Helix** is the first and only end-to-end service and operations platform that’s integrated with 360-degree intelligence. Built for the cloud, this reimagined service and operations experience is unrivaled, giving you:

- Single pane of glass for ITSM and ITOM functions
- **BMC Helix ITSM** optimized for **ITIL® 4**
- Enterprise-wide service including IT, HR, Facilities, and Procurement
- An omni-channel experience across Slack, Chatbot, Skype, and more
- Cloud native micro-services platform for your enterprise
- Automation with conversational bots and RPA bots
Virtual Machine (VM) can be described as a software program that emulates the functionality of a physical hardware or computing system. It runs on top of an emulating software called the hypervisor, which replicates the functionality of the underlying physical hardware resources with a software environment. These resources may be referred to as the Host Machine, while the VM that runs on the hypervisor is often called a Guest Machine. The virtual machine contains all necessary elements to run the apps, including the computing, storage, memory, networking and hardware functionality available as a virtualized system. It may also contain the necessary system binaries and libraries to run the apps, while the OS is managed and executed using the hypervisor.

The virtualized hardware resources are pooled together and made available to the apps running on the VM. An abstraction layer is created to decouple the apps from the underlying physical infrastructure. As a result, the physical hardware can be changed, upgraded or scaled without disrupting the app performance. A VM will operate as an isolated PC and the underlying hardware can operate multiple independent, isolated VMs for different workloads. VMs operations are typically resource-intensive and do not allow individual app functionality to run in isolated PC-like virtualized environments unless a separate VM is used for different modular elements of the app. If an app workload needs to migrate between different virtual machines or physical data center locations, the entire OS needs to migrate along with it. Furthermore, rarely ever does a workload operation consume all of the resources made available to the associated VM. As a result, the remaining unused resources many not be used incorporated in capacity planning and distribution across all VMs and workloads. This leads to inaccurate planning and significant resource wastage even though the concept of
virtualization was developed specifically to optimize the usage and distribution of hardware resources within a data center.

Modern apps and IT services are developed in several modular chunks in order to facilitate faster development and release, high scalability and the flexibility to evolve application development in response to changing business and market needs. Monolithic app development practices are losing popularity and organizations are pursuing infrastructure architecture solutions to further optimize hardware utilization. This is precisely why containerization was invented and gained popularity as a viable alternative.

**Containerization** creates abstraction at an OS level that allows individual, modular and distinct functionality of the app to run independently. As a result, several isolated workloads can dynamically operate using the same physical resources. Containers can run on top of bare metal servers, hypervisors, or in the cloud infrastructure. They share all necessary capabilities with the VM to operate as an isolated OS environment for a modular app functionality with one key difference. Using a containerization engine such as the Docker Engine, containers create several isolated OS environments within the same host system kernel, which can be shared with other containers dedicated to run different functions of the app. Only bins, libraries and other runtime components are developed or executed separately for each container, which makes them more resource efficient as compared to VMs.

Containers are particularly useful in developing, deployment and testing of modern distributed apps and microservices that can operate in isolated execution environments on same host machines. With containerization, developers don’t need to write application code into different VMs operating different app components to retrieve compute, storage and networking resources. A complete application component can be executed in its entirety within its isolated environment without affecting
other app components or software. Conflicts within libraries or app components do not occur during execution and the application container can move between the cloud or data center instances efficiently.

The figure below provides a visual representation of the architectural difference between VMs and containers:

The architectural difference offers the following key value propositions for IT personnel and businesses:

- **Continuous Integration, Deployment and Testing:** In DevOps-driven organizations, organizations can leverage containers to facilitate processes in the CI/CD pipeline. Containers operate as consistent infrastructure environment such that developers don’t need to perform complex configuration tasks for every SDLC sprint as workloads migrate across the physical resources.

- **Workload Portability:** IT workloads can switch between different infrastructure instances and virtual environments without significant configuration changes or rework on the application code.

- **Software Quality and Compliance:** Transparent
collaboration between Devs and testing personnel in delivering operating chunks of the application leads to better software quality, faster development cycles and improved compliance.

- **Cost Optimization**: Containers maximize resource utilization within their own isolated virtualized environments. This allows organizations to plan for infrastructure capacity and consumption accurately.

- **Infrastructure Agnostic**: Containers make the app components infrastructure agnostic, allowing organizations to move workloads between bare metal servers to virtualized environments to cloud infrastructure in response to changing business needs.

These value propositions justify the growing interest and spending containerization technologies. According to a recent survey, nearly a third of the responding organizations are spending at least $500,000 on container licenses for the year 2017. According to the same survey from the previous year, only 5 percent of organizations invested as much. Another study by [451 Research](https://451research.com) found that by the year 2020, spending on containerization technologies will grow at a 40 percent compounded annual rate to reach the $2.7 billion mark.

For DevOps-driven organizations that focus on faster and continuous release cycles of distributed, microservices-based app functions, containerization will continue to attract investments, especially in areas where virtualization failed to deliver.

*Original reference image:*
Virtual Machines

Containers