INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

Team id: PNT2022TMID45514

Containerize the app in cloud

Explore the history of containerization technology, the benefits and advantages of utilizing the technology, and how it related to virtualization.

Containerization has become a major trend in software development as an alternative or companion to virtualization. It involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure. The technology is quickly maturing, resulting in measurable benefits for developers and operations teams as well as overall software infrastructure.

Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a virtual machine (VM) or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or "container" is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.

The concept of containerization and process isolation is decades old, but the emergence of the open-source Docker Engine in 2013, an industry standard for containers with simple developer tools and a universal packaging approach, accelerated the adoption of this technology. Research firm Gartner projects that more than 50% of companies will use container technology by 2020. And results from a late 2017 survey conducted by IBM suggest that adoption is happening even faster, revealing that 59% of adopters improved application quality and reduced defects as a result.

Containers are often referred to as "lightweight," meaning they share the machine's operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.

Put simply, containerization allows applications to be "written once and run anywhere." This portability is important in terms of the development process and

vendor compatibility. It also offers other notable benefits, like fault isolation, ease of management and security, to name a few. Click here to learn more about the benefits of containerization.

Application containerization

Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open-source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.

Other container layers, like common bins and libraries, can also be shared among multiple containers. This eliminates the overhead of running an operating system within each application and makes containers smaller in capacity and faster to start up, driving higher server efficiencies. The isolation of applications as containers also reduces the chance that malicious code present in one container will impact other containers or invade the host system.

The abstraction from the host operating system makes containerized applications portable and able to run uniformly and consistently across any platform or cloud. Containers can be easily transported from a desktop computer to a virtual machine (VM) or from a Linux to a Windows operating system, and they will run consistently on virtualized infrastructures or on traditional “bare metal” servers, either on premise or in the cloud. This ensures that software developers can continue using the tools and processes they are most comfortable with.

One can see why enterprises are rapidly adopting containerization as a superior approach to application development and management. Containerization allows developers to create and deploy applications faster and more securely, whether the application is a traditional monolith (a single-tiered software application) or a modular microservice (a collection of loosely coupled services). New cloud-based applications can be built from the ground up as containerized microservices, breaking a complex application into a series of smaller specialized and manageable services. Existing applications can be repackaged into containers (or containerized microservices) that use compute resources more efficiently.

Benefits

Containerization offers significant benefits to developers and development teams. Among these are the following:

- **Portability:** A container creates an executable package of software that is abstracted away from (not tied to or dependent upon) the host operating system, and hence, is portable and able to run uniformly and consistently across any platform or cloud.
- **Agility:** The open-source Docker Engine for running containers started the industry standard for containers with simple developer tools and a universal packaging approach that works on both Linux and Windows operating systems. The container ecosystem has shifted to engines managed by the Open Container Initiative (OCI). Software developers can continue using agile or DevOps tools and processes for rapid application development and enhancement.
- **Speed:** Containers are often referred to as "lightweight," meaning they share the machine's operating system (OS) kernel and are not bogged down with this extra overhead. Not only does this drive higher server efficiencies, it also reduces server and licensing costs while speeding up start-times as there is no operating system to boot.
- **Fault isolation:** Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.
- **Efficiency:** Software running in containerized environments shares the machine's OS kernel, and application layers within a container can be shared across containers. Thus, containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies, reducing server and licensing costs.
- **Ease of management:** A container orchestration platform automates the installation, scaling, and management of containerized workloads and services. Container orchestration platforms can ease management tasks such as scaling containerized apps, rolling out new versions of apps, and providing monitoring, logging and debugging, among other functions. Kubernetes, perhaps the most popular container orchestration system available, is an open source technology (originally open-sourced by Google, based on their internal project called Borg) that automates Linux container functions originally. Kubernetes works with many container engines, such as Docker, but it also works with any container system that conforms to the Open Container Initiative (OCI) standards for container image formats and runtimes.
- **Security:** The isolation of applications as containers inherently prevents the invasion of malicious code from affecting other containers or the host system. Additionally, security permissions can be defined to automatically block unwanted components from entering containers or limit communications with unnecessary resources.

Types

The rapid growth in interest and usage of container-based solutions has led to the need for standards around container technology and the approach to packaging software code. The Open Container Initiative (OCI), established in June 2015 by Docker and other industry leaders, is promoting common, minimal, open standards

and specifications around container technology. Because of this, the OCI is helping to broaden the choices for open source engines. Users will not be locked into a particular vendor's technology, but rather they will be able to take advantage of OCI-certified technologies that allow them to build containerized applications using a diverse set of DevOps tools and run these consistently on the infrastructure(s) of their choosing.

Today, Docker is one of the most well-known and highly used container engine technologies, but it is not the only option available. The ecosystem is standardizing on container and other alternatives like CoreOS rkt, Mesos Containerizer, LXC Linux Containers, OpenVZ, and crio-d. Features and defaults may differ, but adopting and leveraging OCI specifications as these evolve will ensure that solutions are vendor-neutral, certified to run on multiple operating systems and usable in multiple environments.

Microservices and containerization

Software companies large and small are embracing microservices as a superior approach to application development and management, compared to the earlier monolithic model that combines a software application with the associated user interface and underlying database into a single unit on a single server platform. With microservices, a complex application is broken up into a series of smaller, more specialized services, each with its own database and its own business logic. Microservices then communicate with each other across common interfaces (like APIs) and REST interfaces (like HTTP). Using microservices, development teams can focus on updating specific areas of an application without impacting it as a whole, resulting in faster development, testing, and deployment.

The concepts behind microservices and containerization are similar as both are software development practices that essentially transform applications into collections of smaller services or components which are portable, scalable, efficient and easier to manage.

Moreover, microservices and containerization work well when used together. Containers provide a lightweight encapsulation of any application, whether it is a traditional monolith or a modular microservice. A microservice, developed within a

container, then gains all of the inherent benefits of containerization—portability in terms of the development process and vendor compatibility (no vendor lock-in), as well as developer agility, fault isolation, server efficiencies, automation of installation, scaling and management, and layers of security, among others.

Today's communications are rapidly moving to the cloud where users can develop applications quickly and efficiently. Cloud-based applications and data are accessible from any internet-connected device, allowing team members to work

remotely and on-the-go. Cloud service providers (CSPs) manage the underlying infrastructure, which saves organizations the cost of servers and other equipment and also provides automated network backups for additional reliability. Cloud infrastructures scale on demand and can dynamically adjust computing resources, capacity, and infrastructure as load requirements change. On top of that, CSPs regularly update offerings, giving users continued access to the latest innovative technology.

Containers, microservices, and cloud computing are working together to bring application development and delivery to new levels not possible with traditional methodologies and environments. These next-generation approaches add agility, efficiency, reliability, and security to the software development lifecycle—all of which leads to faster delivery of applications and enhancements to end users and the market.

For a deeper dive into microservices, check out "Microservices: A Complete Guide" and watch the following "What are Microservices?" video (6:38):

Security

Containerized applications inherently have a level of security since they can run as isolated processes and can operate independently of other containers. Truly isolated, this could prevent any malicious code from affecting other containers or invading the host system. However, application layers within a container are often shared across containers. In terms of resource efficiency, this is a plus, but it also opens the door to interference and security breaches across containers. The same could be said of the shared Operating System since multiple containers can be associated with the same host Operating System. Security threats to the common Operating System can impact all of the associated containers, and conversely, a container breach can potentially invade the host Operating System.

But, what about the container image itself? How can the applications and open source components packaged within a container improve security? Container technology providers, such as Docker, continue to actively address container security challenges. Containerization has taken a “secure-by-default” approach, believing that security should be inherent in the platform and not a separately deployed and configured solution. To this end, the container engine supports all of the default isolation properties inherent in the underlying operating system.

Security permissions can be defined to automatically block unwanted components from entering containers or to limit communications with unnecessary resources.

For example, Linux Namespaces helps to provide an isolated view of the system to each container; this includes networking, mount points, process IDs, user IDs, inter-process communication, and hostname settings. Namespaces can be used to limit

access to any of those resources through processes within each container. Typically, subsystems which do not have Namespace support are not accessible from within a container. Administrators can easily create and manage these “isolation constraints” on each containerized application through a simple user interface.

Researchers are working to further strengthen Linux container security, and a wide range of security solutions are available to automate threat detection and response across an enterprise, to monitor and enforce compliance to meet industry standards and security policies, to ensure the secure flow of data through applications and endpoints, and much more.

Learn about a strategy for scaling container security across organizations of any size.

Virtualization vs. containerization

Containers are often compared to virtual machines (VMs) because both technologies enable significant compute efficiencies by allowing multiple types of software (Linux- or Windows-based) to be run in a single environment. However, container technology is proving to deliver significant benefits over and above those of virtualization and is quickly becoming the technology favoured by IT professionals.

Virtualization technology allows multiple operating systems and software applications to run simultaneously and share the resources of a single physical computer. For example, an IT organization can run both Windows and Linux or multiple versions of an operating system, along with multiple applications on the same server. Each application and its related files, libraries, and dependencies, including a copy of the operating system (OS), are packaged together as a VM. With multiple VMs running on a single physical machine, it’s possible to achieve significant savings in capital, operational, and energy costs.

For more of an overview on virtualization, check out the "Virtualization in 2019" video and "Virtualization: A Complete Guide."

Containerization, on the other hand, uses compute resources even more efficiently. A container creates a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Unlike VMs, however, containers do *not* bundle in a copy of the OS. Instead, the container runtime engine is installed on the host system’s operating system, becoming the conduit through which all containers on the computing system share the same OS.

As noted, containers are often referred to as “lightweight”—they share the machine’s OS kernel and do not require the overhead of associating an OS within each application (as is the case with a VM). Other container layers (common bins and libraries) can also be shared among multiple containers, making containers

inherently smaller in capacity than a VM and faster to start up. Multiple containers can then run on the same compute capacity as a single VM, driving even higher server efficiencies, further reducing server and licensing costs.

Containerization and IBM

In a nutshell, virtualization eliminates the need for an entire server for one application. Containerization eliminates the need for an entire OS for each application.

Portability, agility, fault isolation, ease of management, and security are among the advantages of utilizing containerization technology.

Click here to learn about the security and container orchestration capabilities of the IBM Cloud Kubernetes Service.

For an overview of how managed Kubernetes can help you in your cloud journey, watch our video, "Advantages of Managed Kubernetes" (3:15):

To learn more about best practices to enable and expedite container deployment in production environments, see the report "Best Practices for Running Containers and Kubernetes in Production."

Sign up for an IBM Cloud account and start provisioning your first cluster for free.

Featured products

- Red Hat OpenShift on IBM Cloud
- IBM Cloud Satellite
