Docker - Introduction

1. Virtualization

- 1.1. What is Virtualization?
- 1.2. How does virtualization work?
- 1.3. Types of hypervisors
- 1.4. Benefits of virtualization

2. Containerization

- 2.1. What is containerization?
- 2.2. Benefits of Containerization

3. Virtualization vs Containerization

4. DevOps

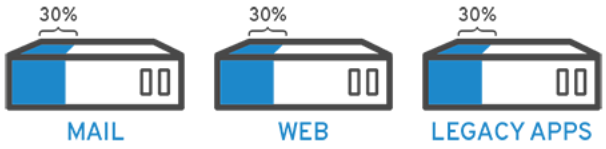
- 4.1. What is DevOps?
- 4.2. How does DevOps work?
- 4.3. What problems does DevOps solve?

1. Virtualization

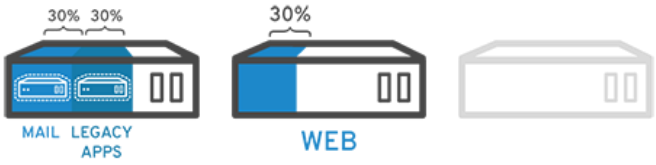
1.1. What is Virtualization?

Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware. It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments.

imagine you have 3 physical servers with individual dedicated purposes. One is a mail server, another is a web server, and the last one runs internal legacy applications. Each server is being used at about 30% capacity—just a fraction of their running potential. But since the legacy apps remain important to your internal operations, you have to keep them and the third server that hosts them.



It was often easier and more reliable to run individual tasks on individual servers: 1 server, 1 operating system, 1 task. It wasn't easy to give 1 server multiple brains. But with virtualization, you can split the mail server into 2 unique ones that can handle independent tasks so the legacy apps can be migrated. It's the same hardware, you're just using more of it more efficiently.



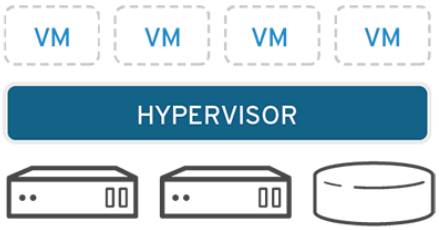
Keeping security in mind, you could split the first server again so it could handle another task—increasing its use from 30%, to 60%, to 90%. Once you do that, the now empty servers could be reused for other tasks or retired altogether to reduce cooling and maintenance costs.

1.2. How does virtualization work?

Software called hypervisors separate the physical resources from the virtual environments—the things that need those resources. Hypervisors take your physical resources and divide them up so that virtual environments can use them.

Resources are partitioned as needed from the physical environment to the many virtual environments. Users interact with and run computations within the virtual environment (typically called a guest machine or virtual machine). The virtual machine functions as a single data file. And like any digital file, it can be moved from one computer to another, opened in either one, and be expected to work the same.

When the virtual environment is running and a user or program issues an instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes—which all happens at close to native speed.



1.3. Types of hypervisors

There are 2 different types of hypervisors that can be used for virtualization: type 1 and type 2 hypervisors.

1.3.1. Type 1

A type 1 hypervisor, also referred to as a native or bare metal hypervisor, runs directly on the host's hardware to manage guest operating systems. It takes the place of a host operating system and VM resources are scheduled directly to the hardware by the hypervisor .

This type of hypervisor is most common in an enterprise data center or other server-based environments.

KVM, Microsoft Hyper-V, and VMware vSphere "Esxi", Openstack, ProxMox are examples of a type 1 hypervisor.

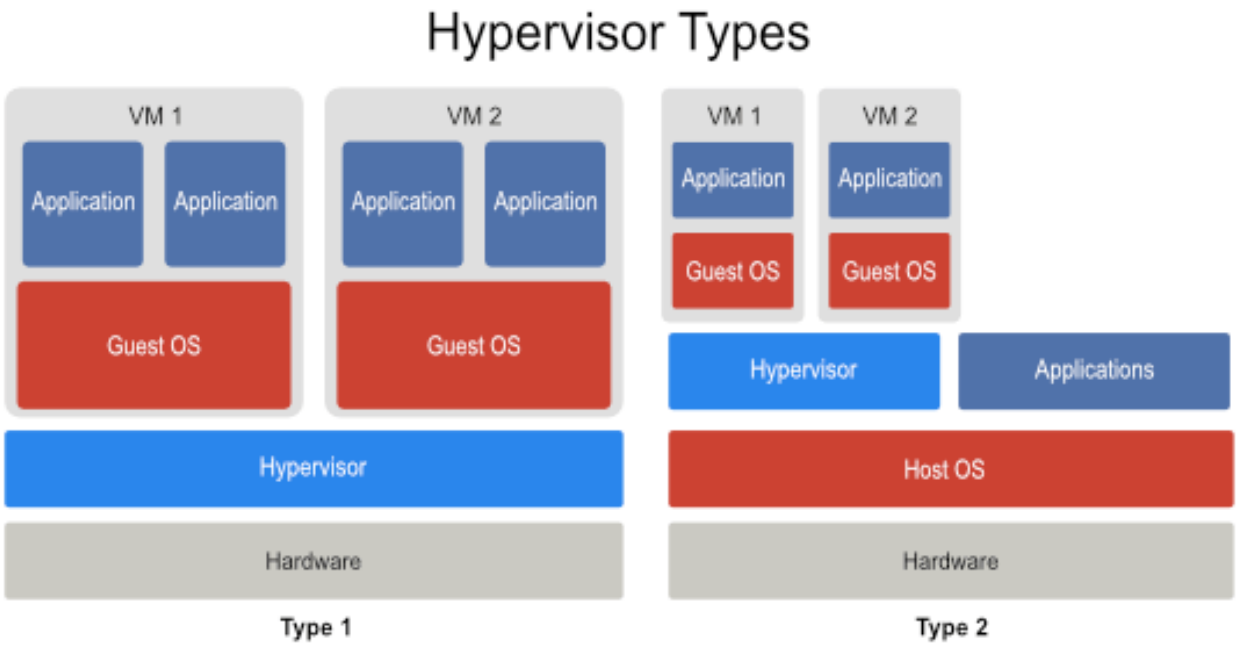
1.3.2. Type 2

A type 2 hypervisor is also known as a hosted hypervisor, and is run on a conventional operating system as a software layer or application.

It works by abstracting guest operating systems from the host operating system. VM resources are scheduled against a host operating system, which is then executed against the hardware .

A type 2 hypervisor is better for individual users who want to run multiple operating systems on a personal computer .

VMware Workstation and Oracle VirtualBox are examples of a type 2 hypervisor.



1.4. Benefits of virtualization

Virtualization brings several benefits to data center operators and service providers:

1.4.1. Resource efficiency: Before virtualization, each application server required its own dedicated physical CPU—IT staff would purchase and configure a separate server for each application they wanted to run. (IT preferred one application and one operating system (OS) per computer for reliability reasons.) Invariably, each physical server would be underused. In contrast, server virtualization lets you run several applications—each on its own VM with its own OS—on a single physical computer without sacrificing reliability. This enables maximum utilization of the physical hardware's computing capacity.

1.4.2. Easier management: Replacing physical computers with software-defined VMs makes it easier to use and manage policies written in software. This allows you to create automated IT service management workflows. For example, automated deployment and configuration tools enable administrators to define collections of virtual machines and applications as services, in software templates. This means that they can install those services repeatedly and consistently without cumbersome, time-consuming. and error-prone manual setup. Admins can use virtualization security policies to mandate certain security configurations based on the role of the virtual machine. Policies can even increase resource efficiency by retiring unused virtual machines to save on space and computing power.

1.4.3. Minimal downtime: OS and application crashes can cause downtime and disrupt user productivity. Admins can run multiple redundant virtual machines alongside each other and failover between them when problems arise. Running multiple redundant physical servers is more expensive.

1.4.4. Faster provisioning: Buying, installing, and configuring hardware for each application is time-consuming. Provided that the hardware is already in place, provisioning virtual machines to run all your applications is significantly faster. You can even automate it using management software and build it into existing workflows.

2. Containerization

2.1. What is containerization?

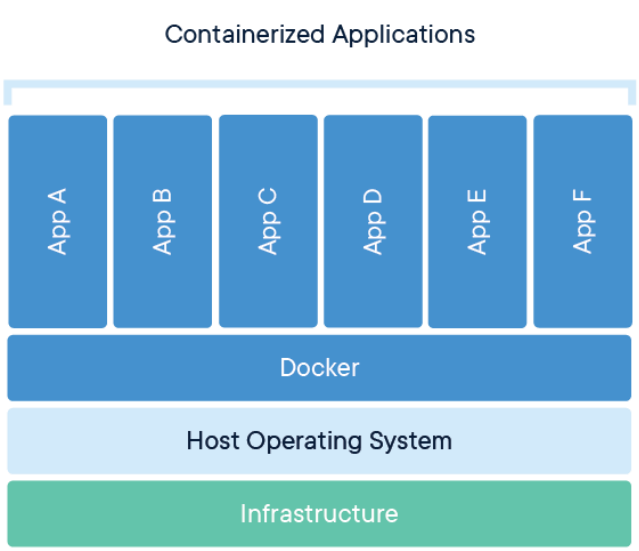
Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.

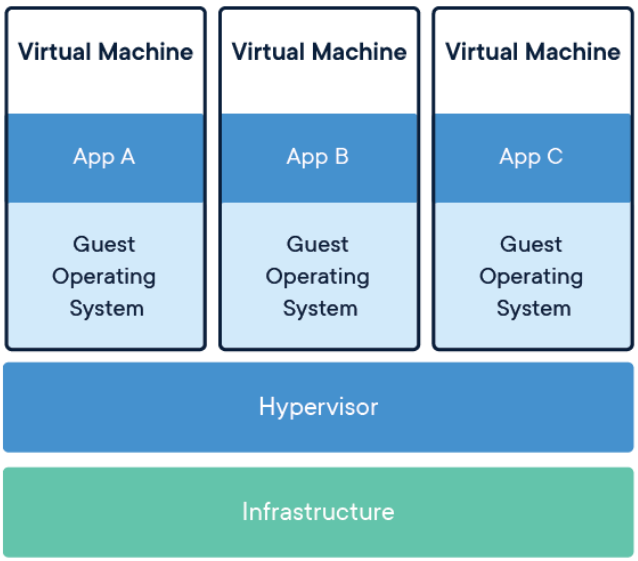
Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from 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 actually decades old, but the emergence in 2013 of the open source Docker Engine—an industry standard for containers with simple developer tools and a universal packaging approach—accelerated the adoption of this technology. Today organizations are using containerization increasingly to create new applications, and to modernize existing applications for the cloud. In a recent IBM survey (PDF, 1.4MB), 61% of container adopters reported using containers in 50% or more of the new applications they built during the previous two years; 64% of adopters expected 50% or more of their existing applications to be put into containers during the next two years.

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.

Perhaps most important, containerization allows applications to be "written once and run anywhere." This portability speeds development, prevents cloud vendor lock-in and offers other notable benefits such fault isolation, ease of management, simplified security and more.





2.2. Benefits of Containerization

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

2.2.1. 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 .

2.2.2. 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.

2.2.3. 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.

2.2.4. 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.

2.2.5. 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.

2.2.6. 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. 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.

2.2.7. 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.

3. Virtualization vs Containerization

Area	Virtualization	Containerization
Isolation	Provides complete isolation from the host operating system and the other VMs	Typically provides lightweight isolation from the host and other containers, but doesn't provide as strong a security boundary as a VM
Operating System	Runs a complete operating system including the kernel, thus requiring more system resources such as CPU, memory, and storage	Runs the user-mode portion of an operating system, and can be tailored to contain just the needed services for your app using fewer system resources
Guest compatibility	Runs just about any operating system inside the virtual machine	Runs on the same operating system version as the host
Deployment	Deploy individual VMs by using Hypervisor software	Deploy individual containers by using Docker or deploy multiple containers by using an orchestrator such as Kubernetes
Load balancing	Virtual machine load balancing is done by running VMs in other servers in a failover cluster	An orchestrator can automatically start or stop containers on cluster nodes to manage changes in load and availability.

4. DevOps

4.1. What is DevOps?

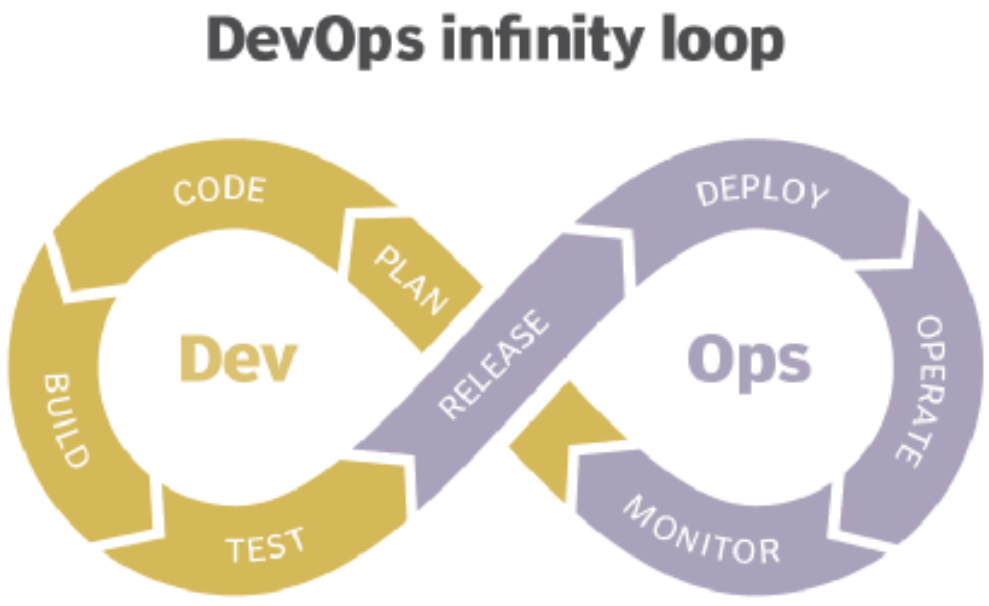
The word DevOps is a combination of the terms development and operations, meant to represent a collaborative or shared approach to the tasks performed by a company's application development and IT operations teams. In its broadest meaning, DevOps is a philosophy that promotes better communication and collaboration between these teams -- and others -- in an organization. In its most narrow interpretation, DevOps describes the adoption of iterative software development, automation and programmable infrastructure deployment and maintenance. The term also covers culture changes, such as building trust and cohesion between developers and systems administrators and aligning technological projects to business requirements. DevOps can change the software delivery chain, services, job roles, IT tools and best practices.

4.2. How does DevOps work?

DevOps is a methodology meant to improve work throughout the software development lifecycle. You can visualize a DevOps process as an infinite loop, comprising these steps: plan, code, build, test, release, deploy, operate, monitor and -- through feedback -- plan, which resets the loop.

Ideally, DevOps means that an IT team writes software that perfectly meets user requirements, deploys without any wasted time and runs optimally on the first try. Organizations use a combination of culture and technology to pursue this goal.

To avoid wait times, IT teams use CI/CD pipelines and other automation to move code from one step of development and deployment to another. Teams review changes immediately and can enforce policies to ensure releases meet standards.



DevOps dissected

Here's what DevOps methods do and how they can help

DEVOPS PRACTICE	DESCRIPTION	BENEFITS
Build Automation	Automated code preparation for deployment to a live environment; the tool used is tied to the programming language selected	■ Fast, consistent, repeatable ■ Portable ■ More reliable than a poorly done manual build
Continuous Integration (CI)	Frequent code merging and unit testing	■ Early bug detection (e.g., compilation errors) ■ Maintains code in a state that can be deployed to production with minimal effort ■ Encourages use of modular code
Continuous Deployment (CD)	Deployment of small code changes to production in a routine and frequent process	■ Faster time to market ■ Dependable deployment process ■ Reliable rollbacks
Infrastructure as Code	Manages and provisions IT infrastructure through code and automation	■ Consistent resource creation and management ■ Reusable ■ Self-documenting infrastructure ■ Simplifies complex infrastructure (DB, authentication, application servers)
Configuration Management	Manages and changes state of infrastructure in constant and maintainable ways	■ Saves time ■ Provides insight (documentation) on infrastructure ■ Maintainability with system changes ■ Minimizes configuration drift, especially in large server environments
Orchestration	Automation that supports processes and workflows (e.g., resource provisioning)	■ Scalability ■ Stability, especially with automatic response to problem detection ■ Saves time
Monitoring	Collects and presents data about the performance and stability of services and infrastructure; detects problems	■ Fast recovery ■ More data to analyze for better root-cause analysis ■ Cross-team visibility ■ Automated response
Microservices	Service architecture that breaks an application into a collection of small, loosely collected services	■ Modularity reduces complexity ■ Flexibility in choice of technology for each service ■ Cost-effective when used with containerization

4.3. What problems does DevOps solve?

Each company faces its own challenges, but common problems include releases that take too long, software that doesn't meet expectations and IT that limits business growth.

Without wait times, manual processes and lengthy reviews, a DevOps project moves from requirements to live software faster. Shorter cycle times can keep requirements from shifting so that the product delivers what customers want.

DevOps solves communication and priority problems between IT specializations. To build viable software, development teams must understand the production environment and test their code in realistic conditions. A traditional structure puts development and operations teams in silos. This means developers are satisfied when their code delivers functionality. And if the release breaks in production, it's up to the operations team to make the fixes.

With a DevOps culture, developers won't have the "It worked on my machine" response when a problem arises. Changes rolled out to production are small and reversible. Plus, the whole team understands the changes, so incident management is greatly simplified.

With a faster process from idea to live software, companies can capitalize on market opportunities. In this way, DevOps provides a competitive advantage for businesses.