Cloud Computing: Definition
The US National Institute of Standards (NIST) defines cloud computing as follows:
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

3-4-5 rule of Cloud Computing
NIST specifies 3-4-5 rule of Cloud Computing

- 3 cloud service models or service types for any cloud platform
- 4 deployment models
- 5 essential characteristics of cloud computing infrastructure

Five characteristics of cloud computing

Cloud computing's characteristics and benefits include on-demand self-service, broad network access, and being very elastic and scalable.

As cloud computing services mature both commercially and technologically, it will be easier for companies to maximize the potential benefits. Knowing what cloud computing is and what it does, however, is just as important. The National Institute of Standards and Technology (NIST) defines cloud computing as it is known today through five particular characteristics.

Editor's note: Control Engineering *has seen recent surge online interest in this article and would like to offer it again to those interested in cloud computing.*

1. On-demand self-service

Cloud computing resources can be provisioned without human interaction from the service provider. In other words, a manufacturing organization can provision additional computing resources as needed without going through the cloud service provider. This can be a storage space, virtual machine instances, database instances, and so on.

Manufacturing organizations can use a web self-service portal as an interface to access their cloud accounts to see their cloud services, their usage, and also to provision and de-provision services as they need to.

2. Broad network access

Cloud computing resources are available over the network and can be accessed by diverse customer platforms. It other words, cloud services are available over a network—ideally high broadband communication link—such as the internet, or in the case of a private clouds it could be a local area network (LAN).

Network bandwidth and latency are very important aspects of cloud computing and broad network access, because they relate to the quality of service (QoS) on the network. This is particularly important for serving time sensitive manufacturing applications.

3. Multi-tenancy and resource pooling

Cloud computing resources are designed to support a multi-tenant model. Multi-tenancy allows multiple customers to share the same applications or the same physical infrastructure while retaining privacy and security over their information. It's similar to people living in an apartment building, sharing the same building infrastructure but they still have their own apartments and privacy within that infrastructure. That is how cloud multi-tenancy works.

Resource pooling means that multiple customers are serviced from the same physical resources. Providers' resource pool should be very large and flexible enough to service multiple client requirements and to provide for economy of scale. When it comes to resource pooling, resource allocation must not impact performances of critical manufacturing applications.

4. Rapid elasticity and scalability

One of the great things about cloud computing is the ability to quickly provision resources in the cloud as manufacturing organizations need them. And then to remove them when they don't need them. Cloud computing resources can scale up or down rapidly and, in some cases, automatically, in response to business demands. It is a key feature of cloud computing. The usage, capacity, and therefore cost, can be scaled up or down with no additional contract or penalties.

Elasticity is a landmark of cloud computing and it implies that manufacturing organizations can rapidly provision and de-provision any of the cloud computing resources. Rapid provisioning and de-provisioning might apply to storage or virtual machines or customer applications.

With cloud computing scalability, there is less capital expenditure on the cloud customer side. This is because as the cloud customer needs additional computing resources, they can simply provision them as needed, and they are available right away. Scalability is more planned and gradual. For instance, scalability means that manufacturing organizations are gradually planning for more capacity and of course the cloud can handle that scaling up or scaling down.

Just-in-time (JIT) service is the notion of requiring cloud elasticity either to provision more resources in the cloud or less. For example, if a manufacturing organization all of a sudden needs more computing power to perform some kind of complex calculation, this would be cloud elasticity that would be a just-in-time service. On the other hand, if the manufacturing organization needs to provision human-machine interface (HMI) tags in the database for a manufacturing project, that is not really just-in-time service, it is planned ahead of time. So it is more on the scalability side than elasticity.

Another feature available for rapid elasticity and scalability in the cloud is related to testing of manufacturing applications. If a manufacturing organization needs, for example, a few virtual machines to test a supervisory control and data acquisition (SCADA) system before they roll it out in production, they can have it up and running in minutes instead of physically ordering and waiting for hardware to be shipped.

In terms of the bottom line, when manufacturing organizations need to test something in the cloud, they are paying for what they use as they use it. As long as they remember to de-provision it, they will no longer be paying for it. There is no capital expense here for computer resources. Manufacturing organizations are using the cloud provider's investment in cloud computing resources instead. This is really useful for testing smart manufacturing solutions.

5. Measured service

Cloud computing resources usage is metered and manufacturing organizations pay accordingly for what they have used. Resource utilization can be optimized by leveraging charge-per-use capabilities. This means that cloud resource usage—whether virtual server instances that are running or storage in the cloud—gets monitored, measured and reported by the cloud service provider. The cost model is based on "pay for what you use"—the payment is variable based on the actual consumption by the manufacturing organization.

Goran Novkovic, MESA International. This article originally appeared on MESA International's blog. MESA International is a CFE Media content partner. Edited by Chris Vavra, production editor, CFE Media, cvavra@cfemedia.com.

Cloud Providers Characteristics
• Provide on-demand provisioning of computational resources
• Use virtualization technologies to lease these resources
• Provide public and simple remote interfaces to manage those resources
• Use a pay-as-you-go cost model, typically charging by the hour
• Operate data centers large enough to provide a seemingly unlimited amount of resources to their clients

Pros and Cons of Cloud Computing
Why is it becoming a Big Deal:
- Using high-scale/low-cost providers,
- Any time/place access via web browser,
- Rapid scalability; incremental cost and load sharing,
- Can forget need to focus on local IT.

Concerns:
–Performance, reliability, and SLAs,
–Control of data, and service parameters,
–Application features and choices,
–Interaction between Cloud providers,
–No standard API – mix of SOAP and REST!
–Privacy, security, compliance, trust…

Scalability and Elasticity in Cloud Computing

Cloud Elasticity :
The Elasticity refers to the ability of a cloud to automatically expand or compress the infrastructural resources on a sudden-up and down in the requirement so that the workload can be managed efficiently. This elasticity helps to minimize infrastructural cost. This is not applicable for all kind of environment, it is helpful to address only those scenarios where the resources requirements fluctuate up and down suddenly for a specific time interval. It is not quite practical to use where persistent resource infrastructure is required to handle the heavy workload.

It is most commonly used in pay-per-use, public cloud services. Where IT managers are willing to pay only for the duration to which they consumed the resources.

Example :
Consider an online shopping site whose transaction workload increases during festive season like Christmas. So for this specific period of time, the resources need a spike up. In order to handle this kind of situation, we can go for Cloud-Elasticity service rather than Cloud Scalability. As soon as the season goes out, the deployed resources can then be requested for withdrawal.

Cloud Scalability :
Cloud scalability is used to handle the growing workload where good performance is also needed to work efficiently with software or applications. Scalability is commonly used where the persistent deployment of resources is required to handle the workload statically.

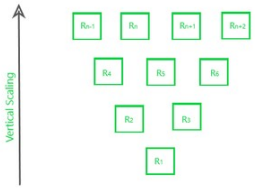
Example :
Consider you are the owner of a company whose database size was small in earlier days but as time passed your business does grow and the size of your database also increases, so in this case you just need to request your cloud service vendor to scale up your database capacity to handle a heavy workload.

It is totally different from what you have read above in Cloud Elasticity. Scalability is used to fulfill the static needs while elasticity is used to fulfill the dynamic need of the organization. Scalability is a similar kind of service provided by the cloud where the customers have to pay-per-use. So, in conclusion, we can say that Scalability is useful where the workload remains high and increases statically.

Types of Scalability :

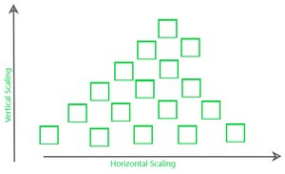
1. Vertical Scalability (Scale-up) –
In this type of scalability, we increase the power of existing resources in the working environment in an upward direction.

upward direction.



2. Horizontal Scalability –
In this kind of scaling, the resources are added in a horizontal row.

3. Diagonal Scalability –
It is a mixture of both Horizontal and Vertical scalability where the resources are added both vertically and horizontally.



Difference Between Cloud Elasticity and Scalability :

	Cloud Elasticity	Cloud Scalability
1	Elasticity is used just to meet the sudden up and down in the workload for a small period of time.	Scalability is used to meet the static increase in the workload.
2	Elasticity is used to meet dynamic changes, where the resources need can increase or decrease.	Scalability is always used to address the increase in workload in an organization.
3	Elasticity is commonly used by small companies whose workload and demand increases only for a specific period of time.	Scalability is used by giant companies whose customer circle persistently grows in order to do the operations efficiently.
4	It is a short term planning and adopted just to deal with an unexpected increase in demand or seasonal demands.	Scalability is a long term planning and adopted just to deal with an expected increase in demand.

There are many aspects of **cloud computing** CIOs, cloud engineers, and IT managers should consider when deciding to add cloud services to their infrastructure. Cost, security, performance, availability, and reliability are some common key areas to consider. Another criterion that has been added to the list recently is cloud scalability and cloud elasticity.
Many have used these terms interchangeably but there are distinct differences between scalability and elasticity. Understanding these differences is very important to ensuring the needs of the business are properly met.

Scalability vs. Elasticity

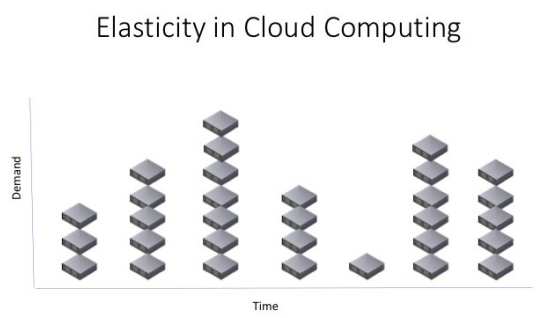
The purpose of Elasticity is to match the resources allocated with actual amount of resources needed at any given point in time. Scalability handles the changing needs of an application within the confines of the infrastructure via statically adding or removing resources to meet applications demands if needed. In most cases, this is handled by adding resources to existing instances—called scaling up or vertical scaling—and/or adding more copies of existing instances—called scaling out or horizontal scaling. In addition, scalability can be more granular and targeted in nature than elasticity when it comes to sizing.

Common use cases where cloud elasticity works well include e-commerce and retail, SaaS, mobile, DevOps, and other environments that have ever changing demands on infrastructure services. Businesses that have a predictable workload where capacity planning and performance are stable and have the ability to predict the constant workload or a growth cloud scalability may be the better cost saving choice.

Cloud Elasticity

Elasticity is the ability to grow or shrink infrastructure resources dynamically as needed to adapt to workload changes in an autonomic manner, maximizing the use of resources. Amazon Web Services defines Elasticity as "The ability to acquire resources as you need them and release resources when you no longer need them." Microsoft Azure defines Elasticity as "The ability to quickly expand or decrease computer processing, memory, and storage resources to meet changing demands without worrying about capacity planning and engineering for peak usage.

Executed properly, capitalizing on elasticity can result in savings in infrastructure costs overall. Not everyone can benefit from elastic services though. Environments that do not experience sudden or cyclical changes in demand may not benefit from the cost savings elastic services offer. Use of “Elastic Services” generally implies all resources in the infrastructure be elastic. This includes but not limited to hardware, software, QoS and other policies, connectivity, and other resources that are used in elastic applications. This may become a negative trait where performance of certain applications must have guaranteed performance. It depends on the environment.



Cloud elasticity is a popular feature associated with scale-out solutions (horizontal scaling), which allows for resources to be dynamically added or removed when needed. Elasticity is generally associated with public cloud resources and is more commonly featured in pay-per-use or pay-as-you-grow services. This means IT managers are not paying for more resources than they are consuming at any given time. In virtualized environments, cloud elasticity could include the ability to dynamically deploy new virtual machines or shutdown inactive virtual machines.

A use case that could easily have the need for cloud elasticity would be in retail with increased seasonal activity. For example, during the holiday season for black Friday spikes and special sales during this season there can be a sudden increased demand on the system. Instead of spending budget on additional permanent infrastructure capacity to handle a couple months of high load out of the year, this is a good opportunity to use an elastic solution. The additional infrastructure to handle the increased volume is only used in a pay-as-you-grow model and then “shrinks” back to a lower capacity for the rest of the year. This also allows for additional sudden and unanticipated sales activities throughout the year if needed without impacting performance or availability. This can give IT managers the security of unlimited headroom when needed. This can also be a big cost savings to retail companies looking to optimize their IT spend if packaged well by the service provider.

Cloud Scalability

Scalability includes the ability to increase workload size within existing infrastructure (hardware, software, etc.) without impacting performance. These resources required to support this are usually pre-planned capacity with a certain amount of headroom built in to handle peak demand. Scalability also encompasses the ability to expand with additional infrastructure resources, in some cases without a hard limit. Scalability can either be vertical (scale-up with in a system) or horizontal (scale-out multiple systems in most cases but not always linearly). Therefore, applications have the room to scale up or scale out to prevent a lack of resources from hindering performance. There are cases where the IT manager knows he/she will no longer need resources and will scale down the infrastructure statically to support a new smaller environment. Either increasing or decreasing services and resources this is a planned event and static for the worse case workload scenario.

For example, there is a small database application supported on a server for a small business. Over time as the business grows so will the database and the resource demands of the database application. If the IT manager knows based on the growth rate of the business and/or the database he may purchase provisioned infrastructure (compute, network, and storage) so that the database application has

the room to grow to its maximum performance and capacity expected. In other words, scale up performance without having to worry about not meeting SLAs in a steady pay-as-you-grow solution.

Another use case is virtual desktop infrastructure (VDI). There are an expected number of desktops based on employee population. To ensure the ability to support the maximum number of users and meet SLAs, the amount of services purchased must be enough to handle all users logged in at once as a maximum use case. In short, the amount of resources allocated are there to handle the heaviest predicted load without a degradation in performance.

Where Elasticity and Scalability Cross Paths

Some cloud services are considered adaptable solutions where both scalability and elasticity are offered. They allow IT departments to expand or contract their resources and services based on their needs while also offer pay-as-you-grow to scale for performance and resource needs to meet SLAs. Incorporation of both of these capabilities is an important consideration for IT managers whose infrastructures are constantly changing. Do not fall into the sales confusion of services where cloud elasticity and scalability are presented as the same service by public cloud providers.

There are distinct differences between elasticity and scalability. It depends on the business need or use case whether elastic or scalability services will be the best choice. A rule of thumb can help you make that decision: Cloud scalability is generally delivered more readily in private cloud environments while cloud elasticity is generally delivered more readily in public cloud environments.

Turbonomic allows you to effectively manage and optimize both cloud scalability and elasticity. It not only optimizes infrastructures for performance and efficiency while upholding compliance in an autonomic manner, it also keeps infrastructures right sized for scalability and elasticity in real time for on-premises, hybrid, and cloud environments.

What is OpenStack?

OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by some of the biggest companies in software development and hosting, as well as thousands of individual community members, many think that OpenStack is the future of cloud computing. OpenStack is managed by the OpenStack Foundation, a non-profit that oversees both development and community-building around the project.

Introduction to OpenStack

OpenStack lets users deploy virtual machines and other instances that handle different tasks for managing a cloud environment on the fly. It makes horizontal scaling easy, which means that tasks that benefit from running concurrently can easily serve more or fewer users on the fly by just spinning up more instances. For example, a mobile application that needs to communicate with a remote server might be able to divide the work of communicating with each user across many different instances, all communicating with one another but scaling quickly and easily as the application gains more users.

More on OpenStack

- The OpenStack Foundation
- The RDO Project
- Red Hat Stack: An OpenStack Blog

And most importantly, OpenStack is open source software, which means that anyone who chooses to can access the source code, make any changes or modifications they need, and freely share these changes back out to the community at large. It also means that OpenStack has the benefit of thousands of developers all over the world working in tandem to develop the strongest, most robust, and most secure product that they can.

How is OpenStack used in a cloud environment?

The cloud is all about providing computing for end users in a remote environment, where the actual software runs as a service on reliable and scalable servers rather than on each end-user's computer. Cloud computing can refer to a lot of different things, but typically the industry talks about running different items "as a service"—software, platforms, and infrastructure. OpenStack falls into the latter category and is considered Infrastructure as a Service (IaaS). Providing infrastructure means that OpenStack makes it easy for users to quickly add new instance, upon which other cloud components can run. Typically, the infrastructure then runs a "platform" upon which a developer can create software applications that are delivered to the end users.

What are the components of OpenStack?

OpenStack is made up of many different moving parts. Because of its open nature, anyone can add additional components to OpenStack to help it to meet their needs. But the OpenStack community has collaboratively identified nine key components that are a

part of the "core" of OpenStack, which are distributed as a part of any OpenStack system and officially maintained by the OpenStack community.

- **Nova** is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.
- **Swift** is a storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy, as developers don't have the worry about the capacity on a single system behind the software. It also allows the system, rather than the developer, to worry about how best to make sure that data is backed up in case of the failure of a machine or network connection.
- **Cinder** is a block storage component, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This more traditional way of accessing files might be important in scenarios in which data access speed is the most important consideration.
- **Neutron** provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.
- **Horizon** is the dashboard behind OpenStack. It is the only graphical interface to OpenStack, so for users wanting to give OpenStack a try, this may be the first component they actually "see." Developers can access all of the components of OpenStack individually through an application programming interface (API), but the dashboard provides system administrators a look at what is going on in the cloud, and to manage it as needed.
- **Keystone** provides identity services for OpenStack. It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud, which they have permission to use. It provides multiple means of access, meaning developers can easily map their existing user access methods against Keystone.
- **Glance** provides image services to OpenStack. In this case, "images" refers to images (or virtual copies) of hard disks. Glance allows these images to be used as templates when deploying new virtual machine instances.
- **Ceilometer** provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud. It also keeps a verifiable count of each user's system usage of each of the various components of an OpenStack cloud. Think metering and usage reporting.
- **Heat** is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure needed for a cloud service to run.

Who is OpenStack for?

You may be an OpenStack user right now and not even know it. As more and more companies begin to adopt OpenStack as a part of their cloud toolkit, the universe of applications running on an OpenStack backend is ever-expanding.

How do I get started with OpenStack?

If you just want to give OpenStack a try, one good resource for spinning the wheels without committing any physical resources is TryStack. TryStack lets you test your applications in a sandbox environment to better understand how OpenStack works and whether it is the right solution for you.

Ready to learn more? Every month, we publish a collection of the best new guides, tips, tricks, tutorials for OpenStack.

OpenStack is always looking for new contributors. Consider joining the OpenStack Foundation or reading this introduction to getting started with contributing to OpenStack.

How do I follow progress in OpenStack?

Because OpenStack is not owned by one single company, getting information about OpenStack can be a little confusing. Opensource.com is working to help bring you up to date information about OpenStack in a format that helps provide answers to common questions from end users, developers, and decision makers seeking to deploy OpenStack at their organizations. To get started with this content, take a look at our OpenStack tag here on Opensource.com. Or check out the following articles, which may help you to learn more about OpenStack and the community behind it:

- The Women of OpenStack talk outreach, education, and mentoring
- Copyright statements proliferate inside open source code

- How OpenStack differs from Amazon and must rise to the occasion
- Open source private cloud storage with OpenStack Swift
- How to analyze corporate contributions to open source projects

Introduction to virtualization and resource management in IaaS

The Rise of Resource Overcommitment

In this world of constant digitization, the rise of large-scale cloud computing has given the users a choice to receive computing resources on-demand and with flexible pricing models. The cloud vendors pool in their massive hardware resources to provide virtual machines on top of it to its users. “Resource overcommitment” is best used to make use of the resources present in a virtualized cloud infrastructure. But what exactly is “Resource overcommitment”? In general terms, it is the allocation of more virtual resources to a machine or a group of machines than are physically present.

As most applications will never use all the resources allocated to them at all times, most of the resources provided by a provider will remain idle without “overcommitment.” Therefore, this approach is less wasteful and more profitable. Also, other than the public cloud offerings such as Microsoft Azure and Amazon Web Services, organizations of today are virtualizing their IT infrastructures to create private clouds. Although private clouds may have less strict SLAs, it still needs resource scheduling to improve the performance of virtual machines. Hence, overcommitment without resource management might lead to poor performance.

Virtualization in IaaS

IaaS or Infrastructure as a Service is driven by one of the critical technologies known as Virtualization. It enables multiple operating systems with different configurations to run on a physical machine at the same time.

A software layer called virtual machine monitor (VMM) or hypervisor is required to run the virtual machines on a system. This hypervisor controls all the hardware resources and can take resources from one VM to give them to another. The hypervisor manages the state of all the VMs at all times. It does so by catching all the privileged directions implemented by the guest VM and copying the resource they access. The hypervisor is also responsible for emulating all the hardware devices and implementing suitable resource isolation between multiple machines operating on the corresponding physical machine.

Types of Virtualization

Based on how they trap the privileged directions given by the guest kernel, there are primarily three techniques utilized for virtualization.

Complete Virtualization with Binary Translation. In this method, the user-mode code operates directly on the CPU without any translation. The non-virtualizable directions in the guest kernel-code are transcribed on the fly-to-code, which has the intended impact on the virtual hardware.

Hardware-driven Full Virtualization. The hardware vendors have developed new features to support virtualization and make virtualization simpler. AMD-V and Intel VT-x are two such technologies developed by AMD and Intel to provide specific directions in their ISA (Instruction Set Architecture) for virtual machines and a new ring privilege level for VM. Sensitive and privileged calls are set up to automatically trap the VMM, eliminating the need for either paravirtualization or binary translation. It also has a modified MMU with support for tagged TLBs and multi-level page tables.

Paravirtualization. This approach needs modification of the guest kernel. The non-virtualizable/privileged directions in the guest kernel source code are replaced with hypercalls that directly call the hypervisor. The hypervisor presents hypercall interfaces for kernel operations such as interrupt handling, memory management, and communication to devices. It varies from full virtualization, where the unmodified guest kernel is utilized, and the guest OS is not aware that it is operating in a virtualized environment.

Virtualization Benefits for IaaS

Virtualization presents numerous benefits other than just resource isolation. This makes it a driving technology behind IaaS success. Here are a few benefits of virtualization:

1. It offers the capability to treat disks of a virtual machine as files which can be snapshotted for quick backup and restore.
2. Virtual machines can be easily relocated or migrated if the physical machines require maintenance or develop some failure.
3. Ease in expanding the resource capacity (RAM or CPU cores) of the machine at runtime by CPU or memory hotplug.

4. It presents the ease of creation of new machines and deployment of applications via pre-built images of the filesystem of the machine.
5. As the hypervisor emulates hardware resources, there is an opportunity for overcommitment of CPU and memory resources.

Resource Management in IaaS

Resource management is an indispensable way to make use of the underlying hardware of the cloud effectively. A resource manager oversees physical resources allocation to the virtual machines deployed on a cluster of nodes in the cloud. Also, the resource management systems have differing purposes depending upon the requirements.

Using physical machines reduces operational costs and can be accomplished through the overcommitment of resources. However, resource overcommitment comes with new challenges such as removal of the hotspot and the dilemma of where to schedule new incoming VMs to reduce the chances of the hotspot. For example: if the total capacity of the virtual machines operating on a physical machine is more than the full capacity of the physical machines, a situation may appear where the VMs might want to use a sum total of more resources than that are currently present. Not meeting those resource specifications can lead to violation of SLAs and poor performance of the VMs. This situation is called a hotspot.

Mitigating the Challenge of Hotspot

Live migration and memory ballooning of VM help in minimizing hotspots. Ballooning can be used if a VM is low on memory to take away some memory from one guest on the same host, which has some free memory, and provide it to the needy guest. But, if none of the guests have enough free memory, then most of the time, the host is overloaded. In that case, a guest has to be migrated from the current host to a different host while keeping an account of the complete load of the cluster.

Final Thoughts

This article gives a fair insight into the different technologies behind IaaS in cloud computing and how they are used. Resource overcommitment helps in making use of the hardware effectively, but resource management is necessary for overcommitment to work. However, there are numerous problems and challenges related to resource management and virtualization in IaaS, but a conscious approach along with proper research can empower us to reap the benefits of IaaS.

What are Linux containers?

Linux containers, in short, contain applications in a way that keep them isolated from the host system that they run on. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. And they are designed to make it easier to provide a consistent experience as developers and system administrators move code from development environments into production in a fast and replicable way.

Technology brief:
Learn to secure your container platform
across the stack
In a way, containers behave like a virtual machine. To the outside world, they can look like their own complete system. But unlike a virtual machine, rather than creating a whole virtual operating system, containers don't need to replicate an entire operating system, only the individual components they need in order to operate. This gives a significant performance boost and reduces the size of the application. They also operate much faster, as unlike traditional virtualization the process is essentially running natively on its host, just with an additional layer of protection around it.

And importantly, many of the technologies powering container technology are open source. This means that they have a wide community of contributors, helping to foster rapid development of a wide ecosystem of related projects fitting the needs of all sorts of different organizations, big and small.

Why is there such interest in containers?

Undoubtedly, one of the biggest reasons for recent interest in container technology has been the Docker open source project, a command line tool that made creating and working with containers easy for developers and sysadmins alike, similar to the way Vagrant made it easier for developers to explore virtual machines easily.

Docker is a command-line tool for programmatically defining the contents of a Linux container in code, which can then be versioned, reproduced, shared, and modified easily just as if it were the source code to a program.

Containers have also sparked an interest in microservice architecture, a design pattern for developing applications in which complex applications are broken down into smaller, composable pieces which work together. Each component is developed separately, and the application is then simply the sum of its constituent components. Each piece, or service, can live inside of a container, and can be scaled independently of the rest of the application as the need arises.

How do I orchestrate containers?

Simply putting your applications into containers probably won't create a phenomenal shift in the way your organization operates unless you also change how you deploy and manage those containers. One popular system for managing and organizing Linux containers is Kubernetes.

Kubernetes is an open source system for managing clusters of containers. To do this, it provides tools for deploying applications, scaling those application as needed, managing changes to existing containerized applications, and helps you optimize the use of the underlying hardware beneath your containers. It is designed to be extensible, as well as fault-tolerant by allowing application components to restart and move across systems as needed.

IT automation tools like Ansible, and platform as a service projects like OpenShift, can add additional capabilities to make the management of containers easier.

How do I keep containers secure?

Container add security by isolating applications from other applications on a host operating system, but simply containerizing an application isn't enough to keep it secure. Dan Walsh, a computer security expert known for his work on SELinux, explains some of the ways that developers are working to make sure Docker and other container tools are making sure containers are secure, as well as some of the security features currently within Docker, and how they function.

Linux Containers vs Docker - What is the Difference and Why Docker is Better

Containers have gained traction in enterprise IT for the past several years, and most developers seem to be very interested in this technology. Many are using them to develop and deploy applications to the public. Containers are useful for many reasons.

They isolate applications and operating systems from the rest of the system, thus bringing in speed and flexibility. They are portable and easy to clone and move to other operating systems. Thus, containers can be more or less made up of a virtualized software environment that the application or operating system runs on a host computer.

There are several kinds of container technologies that you can use. There are Docker containers, Kubernetes containers, and Linux containers (LXC). This article will go over and compare LXC over Docker Containers.

What is LXC, and what does it do

LXC is a userspace interface for the Linux kernel containment features. It uses a sophisticated API and simple tools to make it easier for Linux users to develop and maintain systems or application containers.

To understand LXC well, let’s first discuss what a virtual machine is.

The concept of Virtual Machines (VM)

Typically, in a virtual machine, you have a host machine where you install your operating system. The operating system could be Windows, MacOS, or Linux. When you use a virtual machine, you add or install hypervisors in your host operating system like VirtualBox, VMware, and ESXi.

You use the hypervisor to install a guest operating system to run on top of your host operating system. In this case, the guest OS is running as a virtual machine/container. Each guest operating system you add is a complete operating system.

All the hardware resources (RAM, CPU, HDD/SSD) attached to these guest virtual machines are virtualized and the hypervisor is responsible for allocating the guest OS and hardware resources needed from your base/host machine in a virtualized fashion.

You create an instance of an emulator that can interact with the hardware just like an ordinary computer. The image below depicts this concept.

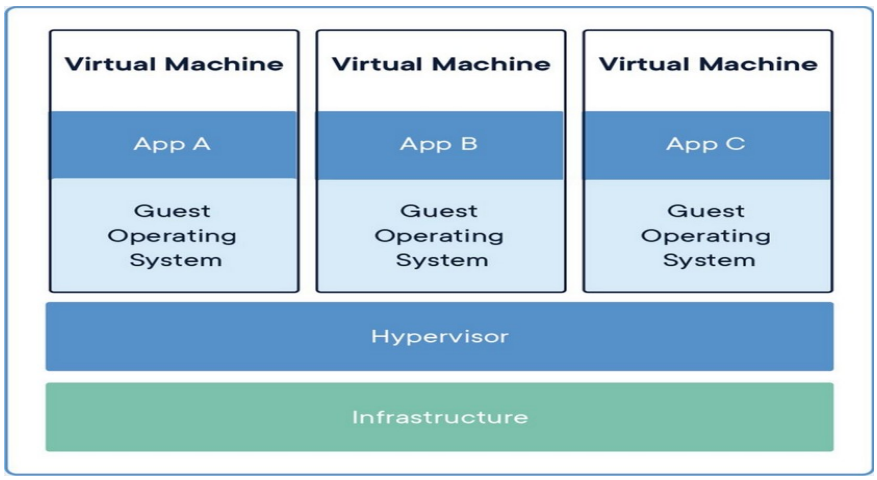


Image source

The main issue with that mentioned above is that every instance running will need allocated resource, which can lead to over-allocation of the available resources. Every instance of an operating system must have these resources allocated to it, and those resources will be used even if the server is idle.

If you have 16GB of RAM and start 4 Guest machines/servers, they will use all of the RAM that is currently available, i.e., each is allocated 4GB of ram. Even if the individual server only uses 1GB of RAM. Assuming each can run on 1GB of RAM, you just need 4 GB of RAM to boot up these four virtual hosts.

Yet because the instances (they are in) have been given 4 GB each, they are using 16GB of RAM within your virtual environment. If you were to add another virtual host, you wouldn't have enough RAM to start it. Every time a VM starts, all of the resources that have been assigned to it are automatically given to it. Those resources can't be assigned to another guest host. In the end, you'll have a lot of resources wasted.

Another issue is that all the guest operating instances are running on a single machine. In case of any boot-up processes, or whenever you want to restart the service for any reason, you have to reboot the entire computer. Depending on what environment you are you're in, a couple of extra seconds can add up.

LXC virtual environments (VE)

LXC (a Linux container) is a virtualization solution on the operating system-level that enables the creation and operation of many isolated Linux virtual environments (VE). These can spun up on a single centralized host. Containers, which are isolation levels, can isolate certain apps or simulate a fully different host.

LXC mitigates VM disadvantages. Linux containers enable the host CPU to effectively divide memory allocations into confinement levels known as namespaces. In comparison, a VM contains the whole operating system and machine setup/emulators, such as the hard disk, virtual CPUs, and network interfaces. The entire (enormous) virtualized environment typically takes some time to boot and consumes a large amount of RAM and CPU.

LXC virtual environment has no hardware preload emulation. Each virtual environment (an OS or an application) is loaded in a container and executes without any additional overhead and no hardware emulation. This means no penalty from software with limited memory. In the end, LXC will improve the performance of the bare metal as it only bundles the OS/application that is required.

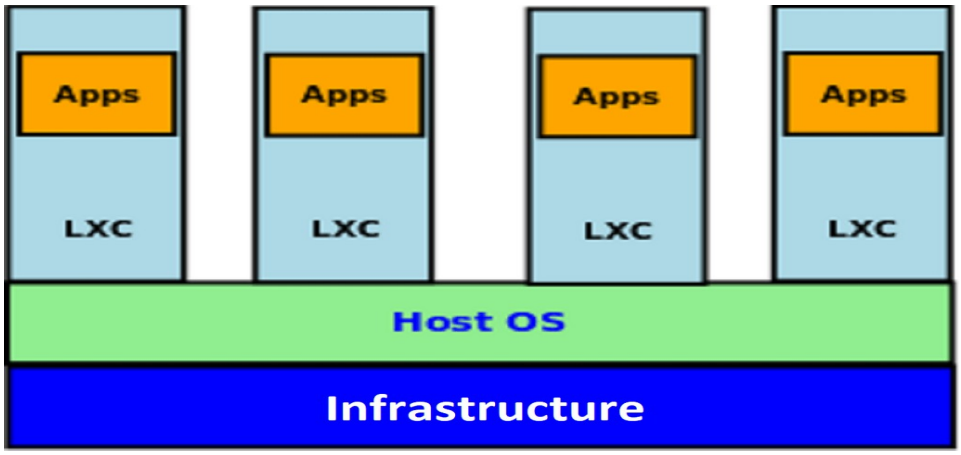


Image source

In simpler terms, within the LXC virtual environments (VE), you create containers for the service, the virtual OS, or the application. At the same time, the underlying hardware resources and kernel is shared by all the containers. For example, let's say you have a container running a LAMP server (Linux, APACHE, MYSQL PHP).

In this case, the software installed, the configuration files, and the IP address are all virtualized. However, the container doesn't have the kernel and the RAM (hardware resources) directly associated with it. It is operating on the lower level of the base metal.

In other terms, let's say you have three different containers, i.e., a LAMP server, a DHCP container, and a DNS container. In this case, all the configuration files are contained within this container. And they all share the underlying resources. In this case, resource allocation will be much easier when compared to VMs.

If the LAMP server needs a lot of resources, it can be allocated to it. So if the DNS container doesn't need a lot of RAM, and the LAMP server is demanding additional RAM at the moment, that additional memory not used in the DNS server can be assigned to the LAMP server to speed up this container's services. Resources are allocated according to the need of every container.

Another important thing to note is that each container is contained within itself with its configuration files. Each container can have its IP address and its network configuration. You can also go ahead and change those network configurations.

Setting up a Linux server, you need a Linux operating system such as Ubuntu. Then you install the Linux Container software. Once installed, you can log into it, and you will get a prompt that looks like any other Linux client.

Docker containers

Docker is a containerized virtual environment that makes it easier to develop, maintain, and deploy applications and services. Docker containers are incredibly light, and you don't have to configure or set up virtual machines and environments, meaning you have hypervisors in between.

Docker containers work off directly from the host OS. Essentially it doesn't have a separate kernel to run its containers. It utilizes the same resources as the host OS. Docker exploits namespaces and control groups to let you use host OS resources more efficiently.

Let's interpret this with a diagram.

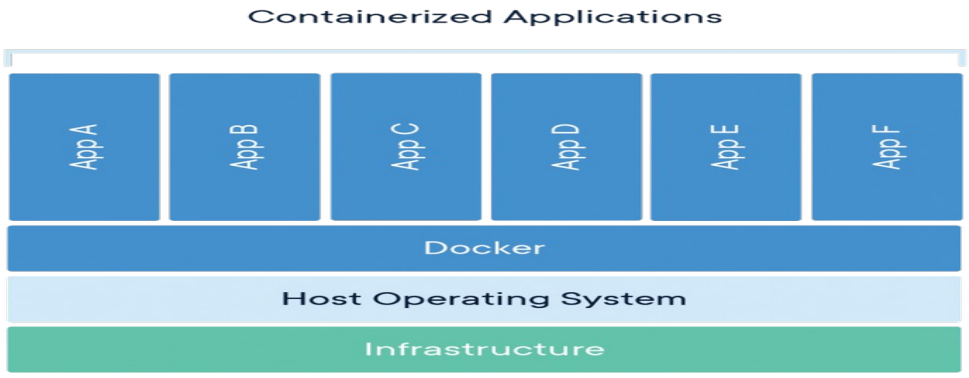


Image source

The above containerization concept has the infrastructure and an operating system added to it. In contrast to LXC, there is no guest operating system. Instead, it has a Docker demon that runs directly on the operating system. Docker demons facilitate creating, running, and the managing of containers.

Each instance of a container runs one process (application). Each application is isolated and runs without affecting other applications. So, in this case, each app is containerized with its configuration files.

Docker demons allow containers to ping back resources from the host machine. In this case, the demons allocate resources to this container depending on how much a container needs to run.

What is the major difference between Linux and Docker containers?

LXC focuses on OS containerization, while Docker thrives on application containerization. Docker is single-purpose application virtualization, and LXC is multi-purpose operating system virtualization.

In this case, LXC specializes in deploying Linux Virtual machines. A container is like a VM with a fully functional OS environment. However, the OS has to support and handle the features and capabilities of a Linux environment. You can SSH into an LXC container, operate it as an operating system, and install any application or services, and everything will work as expected.

This is not the case with Docker. Docker specializes in deploying applications. Docker containers aren't lightweight virtual machines. Thus they can't be considered as such.

Docker containers are limited to a single application due to their architecture. Although Docker runs natively in a Linux environment, it is not entirely dependent on Linux, and it supports other operating systems such as Windows and MacOS.

Why Docker is better?

Portability

You have probably heard a phrase it works on my computer but not yours.

Consider a case where a company hires a developer to develop an app on Oracle's WebLogic software. This means the developer has Oracle WebLogic installed on the computer. When the application is fully developed, the developer will share this application with the team, testing the code.

The team will then have to repeat the process of installing and configuring the Oracle WebLogic. The same application is shared with the production team, and the cycle repeats itself.

Due to the environmental difference on these three levels, the installation was done separately. This is one of the most comprehensive solutions that Docker container offers. Docker will containerize this single application with all the configuration the application needs to run.

The other team only needs to have the Docker demons installed. Then the developer can share this application without having to install additional software regardless of the operating system the other teams are in.

Docker goes hand in hand with versioning. When sharing this application, the version of the services used don't change. Thus, the whole application will be shared without breaking the code.

Flexibility

As we have explained above, Docker knows no environmental boundaries. Linux containers only support containerization with a Linux-based operating system. Thus, a container only runs in one environment. That's not the case with Docker. If it works on your computer, it will work on my computer as well.

Docker engine is well set to work constantly across all environments. If you set an application on a Windows-based server, it can be easily be deployed in a Linux server without compatibility issues.

With Docker containers, applications will work efficiently and correctly in different computer environments. It can be easily deployed and executed on other computer environments regardless of their host operating system or configurations

Efficient use of system resources

Docker containers have no kernels of their own or virtual memory and CPU, making it easier to utilize resources directly from the host infrastructure. There is no added resource allocation complexity. Resources are served as demanded. An application only uses resources if it's up and running.

If it's running, it is the role of the Docker engine to assign resources to this application directly from the physical resources of the host. You would be able to spin up many more Docker containers than the Linux containers in a single host OS.

The Docker engine also allows you to run multiple containers while sharing resources and still maintain isolation. In Docker, an application is specified using a Docker image. Docker allows you to build or stack an application on top of previously created packages, which provides a greater possibility for component usability.

Assume you need to create many instances, each of which requires Apache and MySQL. In such a situation, you can create a "base image" that includes these two pieces, then construct and generate more instances that already have them installed.

Lightweight and fast

You don't have to set up a hypervisor or configure virtual machine images that are hard to set up and deploy. And there's no need to put up a guest OS. Thus, Docker is quicker and easier to set up. To get to your application, you don't have to start up a complete operating system.

Docker engine has embedded configuration management commands that let you start/stop or restart a container within seconds. Docker containers take up less disk space with no complex configuration to set, making them relatively faster in executing applications.

Popularity

If popularity was the only factor to consider when choosing between these two containerization solutions, Docker would easily defeat LXC. Docker was launched in 2013. In its early stages, Docker used LXC but has subsequently modified its codebase to create a completely new container architecture.

Many IT titans, including Netflix, Twitter, Google, and other web-scale organizations, have embraced Docker's application containerization strategy for its scalability benefits. Docker's popularity is booming, according to ZDNet, with over 3.5 million container-based applications and billions of container orchestration distributed using Docker.

One of the reasons for Docker's popularity is the approach it used to align itself with its target market. Containers were designed to go beyond the LXC Operating system and into the more detailed realm of an application that fits many organizations and enterprises.

What is LXD?

LXD ([lɛks'di:]) is a next generation system container and virtual machine manager. It offers a unified user experience around full Linux systems running inside containers or virtual machines.

LXD is image based and provides images for a wide number of Linux distributions. It provides flexibility and scalability for various use cases, with support for different storage backends and network types and the option to install on hardware ranging from an individual laptop or cloud instance to a full server rack.

When using LXD, you can manage your instances (containers and VMs) with a simple command line tool, directly through the REST API or by using third-party tools and integrations. LXD implements a single REST API for both local and remote access.

The LXD project was founded and is currently led by Canonical Ltd with contributions from a range of other companies and individual contributors.

What's LXC?

LXC is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.

Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)

LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.

Components

LXC is currently made of a few separate components:

- The liblxc library
- Several language bindings for the API:python3 lua Go ruby Haskell
- A set of standard tools to control the containers
- Distribution container templates

Licensing

LXC is free software, most of the code is released under the terms of the GNU LGPLv2.1+ license, some Android compatibility bits are released under a standard 2-clause BSD license and some binaries and templates are released under the GNU GPLv2 license.

The default license for the project is the GNU LGPLv2.1+.

Support

LXC's stable release support relies on the Linux distributions and their own commitment to pushing stable fixes and security updates.

Based on the needs and available resources from the various distributions, specific versions of LXC can enjoy long term support with frequent bugfix updates.

Other releases will typically be maintained on a best effort basis which typically means until the next stable release is out.

Commercial support for LXC on Ubuntu LTS releases can be obtained from Canonical Ltd.

Extended support

LXC 5.0 and 4.0 are long term support releases:

- LXC 5.0 will be supported until June 1st 2027
- LXC 4.0 will be supported until June 1st 2025

This is thanks to Canonical Ltd and Ubuntu who include the long term support releases of LXC into their own LTS releases and work closely with LXC upstream to maintain our stable branches.

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

What can I use Docker for?

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and execute automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

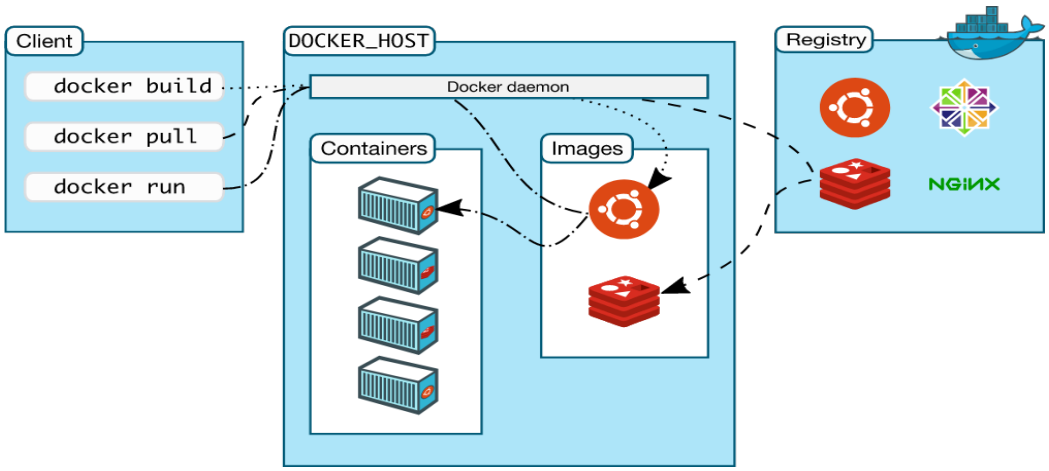
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

Docker architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a

Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Example docker run command

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

```
docker run -i -t ubuntu /bin/bash
```

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run `docker pull ubuntu` manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

The underlying technology

Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker is an open-source software designed to facilitate and simplify application development. It is a set of platform-as-a-service products that create isolated virtualized environments for building, deploying, and testing applications.

Although the software is relatively simple to master, there are some Docker-specific terms that new users may find confusing. Dockerfiles, images, containers, volumes, and other terminology will need to be mastered and should become second nature over time.

It is a good idea to try to comprehend the basic roles of these elements. It will speed up learning on how to work with them.

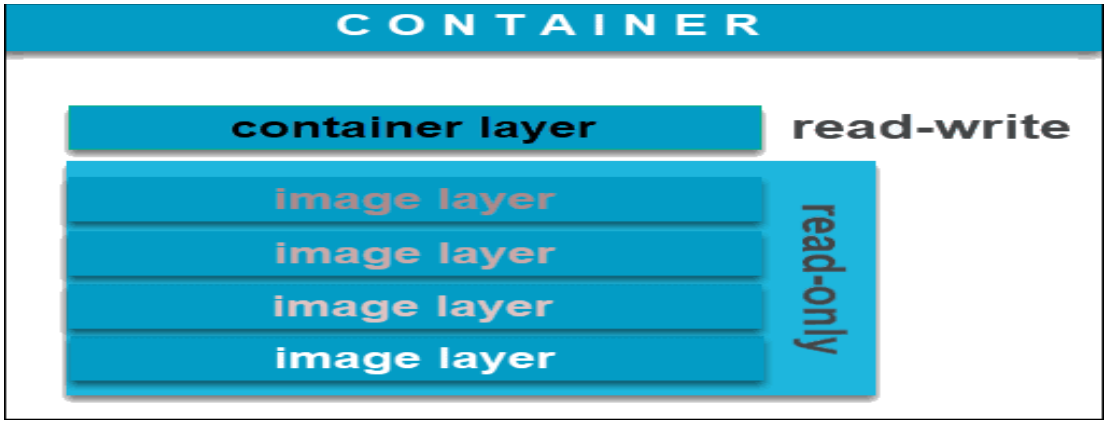
What is a Docker Image?

A **Docker image** is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

Due to their **read-only** quality, these images are sometimes referred to as snapshots. They represent an application and its virtual environment at a specific point in time. This consistency is one of the great features of Docker. It allows developers to test and experiment software in stable, uniform conditions.

Since images are, in a way, just **templates**, you cannot start or run them. What you can do is use that template as a base to build a container. A container is, ultimately, just a running image. Once you create a container, it adds a writable layer on top of the immutable image, meaning you can now modify it.

The image-base on which you create a container exists separately and cannot be altered. When you run a containerized environment, you essentially create a **read-write copy** of that filesystem (docker image) inside the container. This adds a **container layer** which allows modifications of the entire copy of the image.



You can create an unlimited number of Docker images from one **image base**. Each time you change the initial state of an image and save the existing state, you create a new template with an additional layer on top of it.

Docker images can, therefore, consist of a **series of layers**, each differing but also originating from the previous one. Image layers represent read-only files to which a container layer is added once you use it to start up a virtual environment.

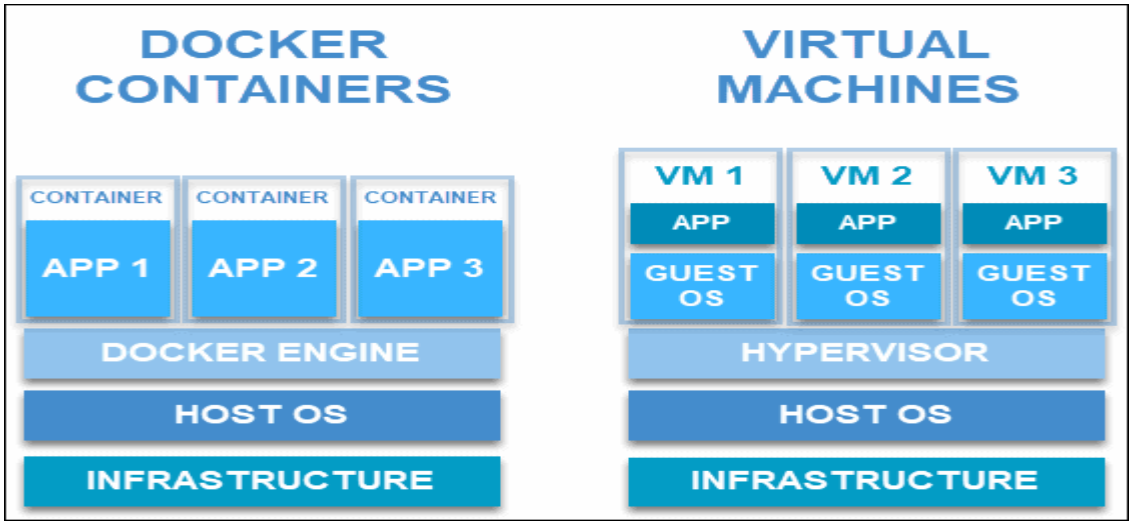
What is a Docker Container?

A **Docker container** is a virtualized run-time environment where users can isolate applications from the underlying system. These containers are compact, portable units in which you can start up an application quickly and easily.

A valuable feature is the **standardization** of the computing environment running inside the container. Not only does it ensure your application is working in identical circumstances, but it also simplifies sharing with other teammates.

As containers are autonomous, they provide strong isolation, ensuring they do not interrupt other running containers, as well as the server that supports them. Docker claims that these units “provide the strongest isolation capabilities in the industry”. Therefore, you won’t have to worry about keeping your machine **secure** while developing an application.

Unlike virtual machines (VMs) where virtualization happens at the hardware level, containers virtualize at the app layer. They can utilize one machine, share its kernel, and virtualize the operating system to run isolated processes. This makes containers extremely **lightweight**, allowing you to retain valuable resources.



Note: If you want to learn more about the difference between virtual machines and containers, how they work, and how to decide which one is best for you, refer to our article Containers vs Virtual Machines (VMs): What’s the Difference?

Docker Images vs Containers

When discussing the difference between images and containers, it isn't fair to contrast them as opposing entities. Both elements are closely **related and are part of a system** defined by the Docker platform.

If you have read the previous two sections that define docker images and docker containers, you may already have some understanding as to how the two establish a relationship.

Images can exist without containers, whereas a container needs to run an image to exist. Therefore, containers are dependent on images and use them to construct a run-time environment and run an application.

The two concepts exist as essential components (or rather phases) in the process of running a Docker container. Having a running container is the final "phase" of that process, indicating it is dependent on previous steps and components. That is why docker images essentially govern and shape containers.

From Dockerfile to Image to Container

It all starts with a script of instructions that define how to build a specific Docker image. This script is called a Dockerfile. The file automatically executes the outlined commands and creates a **Docker image**.

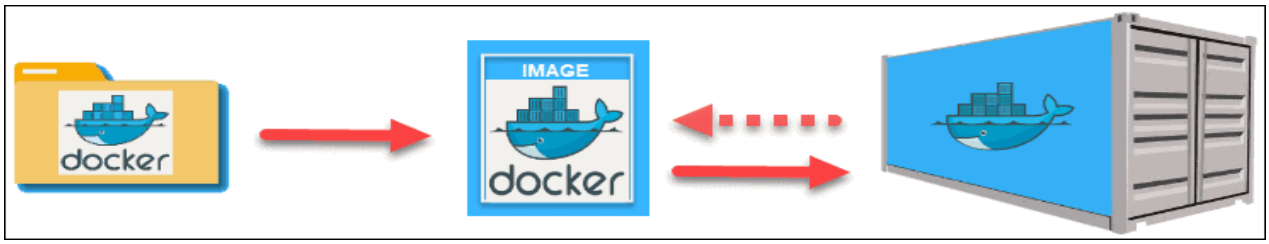
The command for creating an image from a Dockerfile is **docker build**.

The image is then used as a template (or base), which a developer can copy and use it to run an application. The application needs an isolated environment in which to run – a **container**.

This environment is not just a virtual "space". It entirely relies on the image that created it. The source code, files, dependencies, and binary libraries, which are all found in the Docker image, are the ones that make up a container.

To create a container layer from an image, use the command **docker create**.

Finally, after you have launched a container from an existing image, you start its service and run the application.



Creating a Docker Image from a Container.

If you make changes to the initial image and you want to preserve it for future work, you can save the modified image by taking a screenshot of the current state of the container. By doing so, you attach a container layer on top of the image, ultimately building a new immutable image. As a result, you end up with two Docker images derived from the same filesystem.

Conclusion

Once you understand the process of creating a container, you will easily recognize how different images and containers are. After reading this article, you should now have a good understanding of what a Docker image is, what a container is, and how they are connected.

Now that you know the difference between images and containers, check out our article on Docker Copy vs Add command to learn how to copy files in Docker the proper way.

What is container orchestration?

Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.

Why do we need container orchestration?

Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort. Particularly when paired with microservices—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.

This can introduce significant complexity if managed manually. Container orchestration is what makes that operational complexity manageable for development and operations—or DevOps—because it provides a declarative way of automating much of the work.

This makes it a good fit for DevOps teams and culture, which typically strive to operate with much greater speed and agility than traditional software teams.

What are the benefits of container orchestration?

Container orchestration is key to working with containers, and it allows organizations to unlock their full benefits. It also offers its own benefits for a containerized environment, including:

- **Simplified operations:** This is the most important benefit of container orchestration and the main reason for its adoption. Containers introduce a large amount of complexity that can quickly get out of control without container orchestration to manage it.
- **Resilience:** Container orchestration tools can automatically restart or scale a container or cluster, boosting resilience.
- **Added security:** Container orchestration's automated approach helps keep containerized applications secure by reducing or eliminating the chance of human error.

What are containers and their benefits?

Containers are a method of building, packaging and deploying software. They are similar to but not the same thing as virtual machines (VMs). One of the primary differences is that containers are isolated or abstracted away from the underlying operating system and infrastructure that they run on. In the simplest terms, a container includes both an application's code and everything that code needs to run properly.

Because of this, containers offer many benefits, including:

- **Portability:** One of the biggest benefits of containers is that they're built to run in any environment. This makes containerized workloads easier to move between different cloud platforms, for example, without having to rewrite large amounts of code to ensure it will execute properly, regardless of the underlying operating system or other factors. This also boosts developer productivity, since they can write code in a consistent manner without worrying about its execution when deployed to different environments—from a local machine to an on-premises server to a public cloud.
- **Application development:** Containers can speed up application development and deployments, including changes or updates over time. This is particularly true with containerized microservices. This is an approach to software architecture that entails breaking up a larger solution into smaller parts. Those discrete components (or microservices) can then be deployed, updated or retired independently, without having to update and redeploy the entire application.
- **Resource utilization and optimization:** Containers are lightweight and ephemeral, so they consume fewer resources. You can run many containers on a single machine, for example.

What is Kubernetes container orchestration?

Kubernetes is a popular open source platform for container orchestration. It enables developers to easily build containerized applications and services, as well as scale, schedule and monitor those containers. While there are other options for container orchestration, such as Apache Mesos or Docker Swarm, Kubernetes has become the industry standard. Kubernetes provides extensive container capabilities, a dynamic contributor community, the growth of cloud-native application development and the widespread availability of commercial and hosted Kubernetes tools. Kubernetes is also highly extensible and portable, meaning it can run in a wide range of environments and be used in conjunction with other technologies, such as service meshes.

In addition to enabling the automation fundamental to container orchestration, Kubernetes is considered highly declarative. This means that developers and administrators use it to essentially describe how they want a system to behave, and then Kubernetes executes that desired state in dynamic fashion.

What is multi-cloud container orchestration?

In the most basic sense, the term "multi-cloud" refers to an IT strategy of using two or more cloud services from two or more providers. In the context of containers and orchestration, multi-cloud usually means the use of two or more cloud infrastructure

platforms, including public and private clouds, for running applications. Multi-cloud container orchestration, then, refers to the use of an orchestration tool to operate containers across multi-cloud infrastructure environments—instead of running containers in a single cloud environment.

Software teams pursue multi-cloud strategies for different reasons, but the benefits can include infrastructure cost optimization, flexibility and portability (including reducing vendor lock-in), and scalability (such as dynamically scaling out a cloud from an on-premises environment when necessary.) Multi-cloud environments and containers go hand-in-hand because of the latter's portable, "run anywhere" nature.

Container orchestration versus Docker

Docker is a specific platform for building containers, including the Docker Engine container runtime, whereas container orchestration is a broader term referring to automation of any container's lifecycle. Docker also includes Docker Swarm, which is the platform's own container orchestration tool that can automatically start Docker containers.

VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, mode switching is completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

1. **Hardware Support for Virtualization**

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Example 3.4 discusses Intel's hardware support approach.

At the time of this writing, many hardware virtualization products were available. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host-based virtualization. Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts. Actually, Xen modifies Linux as the lowest and most privileged layer, or a hypervisor.

One or more guest OS can run on top of the hypervisor. KVM (Kernel-based Virtual Machine) is a Linux kernel virtualization infrastructure. KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively. The VirtIO framework includes a paravirtual Ethernet card, a disk I/O controller, a balloon device for adjusting guest memory usage, and a VGA graphics interface using VMware drivers.

Example 3.4 Hardware Support for Virtualization in the Intel x86 Processor

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance. Figure 3.10 provides an overview of Intel's full virtualization techniques. For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all sensitive instructions in the VMM automatically. For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance. For I/O virtualization, Intel implements VT-d and VT-c to support this.

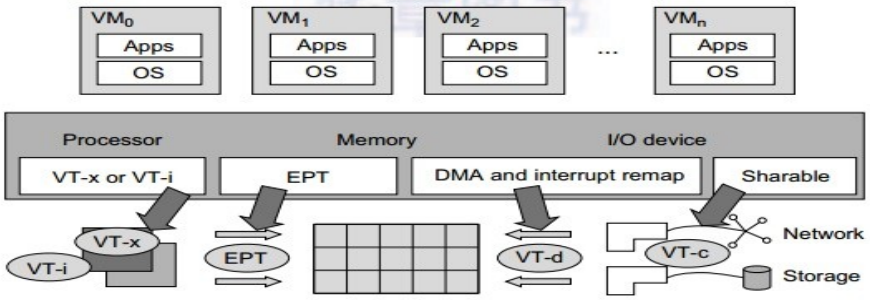


FIGURE 3.10
Intel hardware support for virtualization of processor, memory, and I/O devices.

2. **CPU Virtualization**

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: privileged instructions, control-sensitive instructions, and behavior-sensitive instructions. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior-sensitive instructions of a VM are exe-cuted, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions. When these instruc-tions execute in virtualization, they cannot be trapped in the VMM.

On a native UNIX-like system, a system call triggers the 80h interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call. On a para-virtualization system such as Xen, a system call in the guest OS first triggers the 80h interrupt nor-mally. Almost at the same time, the 82h interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel. Certainly, the guest OS kernel may also invoke the hypercall while it's running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

2.1 Hardware-Assisted CPU Virtualization

This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

Example 3.5 Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtuali-zation of x86 processors is detailed in the

following sections. Intel's VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the

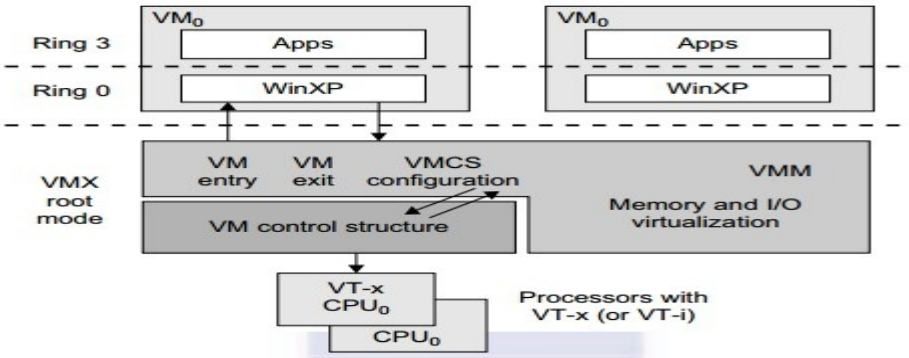


FIGURE 3.11
Intel hardware-assisted CPU virtualization.

CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

3. **Memory Virtualization**

Virtual memory virtualization is similar to the virtual memory support provided by modern operat-ing systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.

That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.

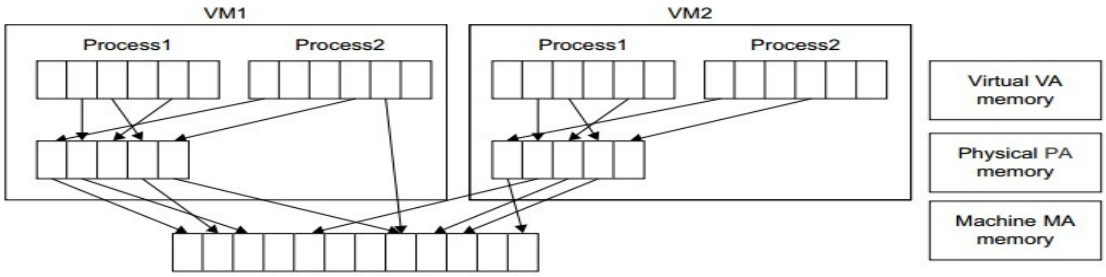


FIGURE 3.12
Two-level memory mapping procedure.

Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded. Consequently, the perfor-mance overhead and cost of memory will be very high.

VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

Example 3.6 Extended Page Table by Intel for Memory Virtualization

Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.

When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4

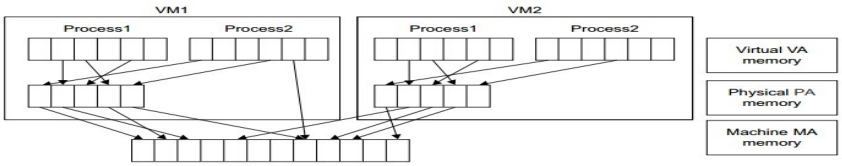


FIGURE 3.12
Two-level memory mapping procedure.

page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page table, as described earlier. To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. There-fore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this short-coming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

4. I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. At the time of this writing, there are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O. Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices.

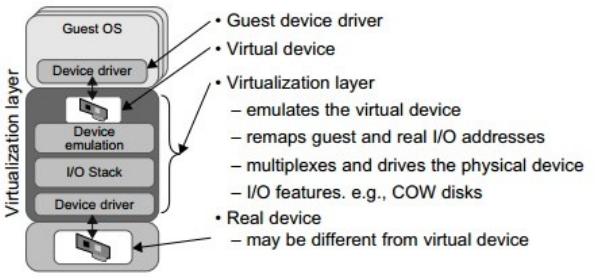


FIGURE 3.14

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices. The full device emulation approach is shown in Figure 3.14.

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates [10,15]. The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend dri-ver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices. For example, when a physical device is reclaimed (required by workload migration) for later reassign-ment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system. Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical. Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or “virtualization-aware” guest OSes.

Another way to help I/O virtualization is via self-virtualized I/O (SV-IO) [47]. The key idea of SV-IO is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-IO defines one virtual interface (VIF) for every kind of virtua-lized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two mes-sage queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

Example 3.7 VMware Workstation for I/O Virtualization

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. Figure 3.15 shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.

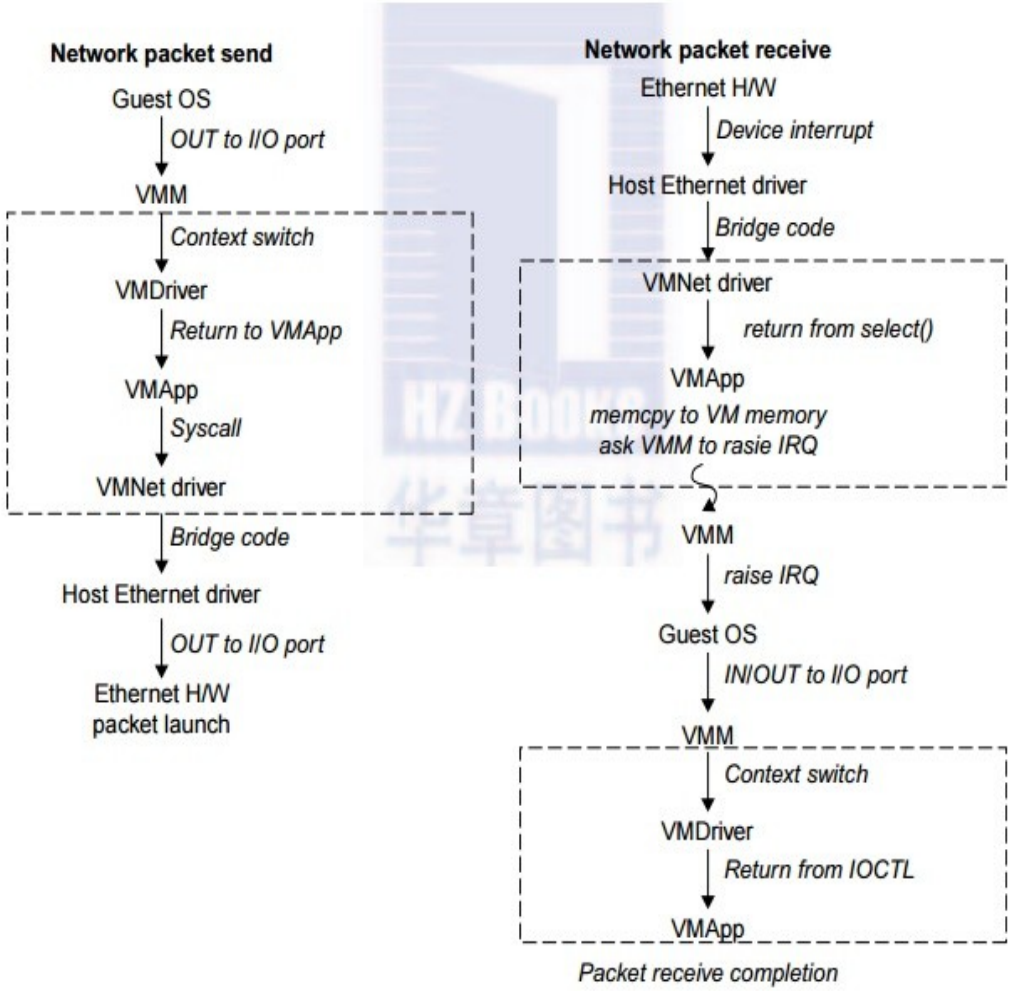


FIGURE 3.15

Functional blocks involved in sending and receiving network packets.

The virtual NIC models an AMD Lance Am79C970A controller. The device driver for a Lance controller in the guest OS initiates packet transmissions by reading and writing a sequence of virtual I/O ports; each read or write switches back to the VMApp to emulate the Lance port accesses. When the last OUT instruc-tion of the sequence is encountered, the Lance emulator calls a normal write() to the VMNet driver. The VMNet driver then passes the packet onto the network via a host NIC and then the VMApp

switches back to the VMM. The switch raises a virtual interrupt to notify the guest device driver that the packet was sent. Packet receives occur in reverse.

5. Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors [33,66].

5.1 Physical versus Virtual Processor Cores

Wells, et al. [74] proposed a multicore virtualization method to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

5.2 Virtual Hierarchy

The emerging many-core chip multiprocessors (CMPs) provides a new computing landscape. Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. This idea was originally suggested by Marty and Hill [39]. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

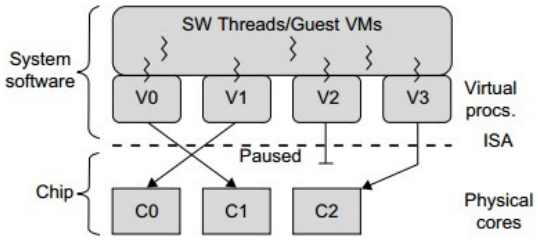


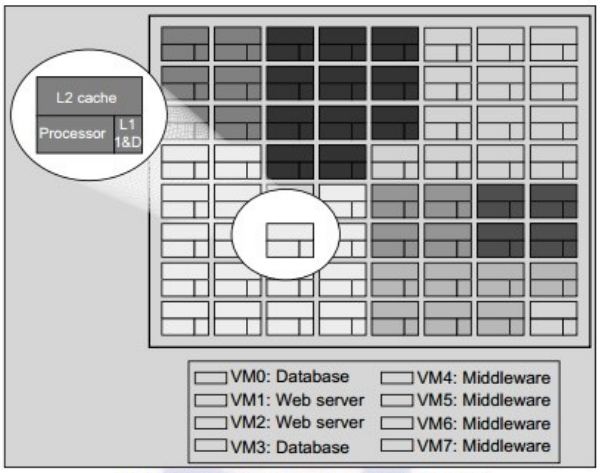
FIGURE 3.16

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.

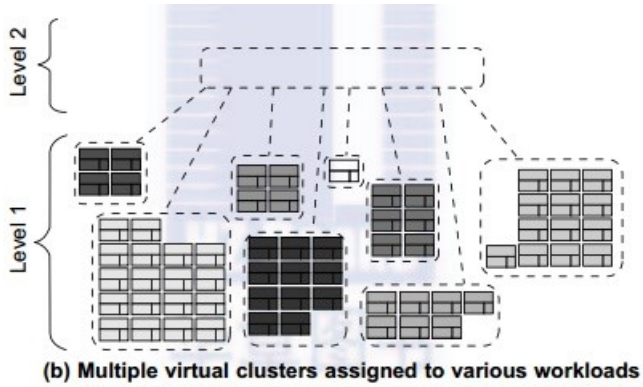
Today's many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads [39]. The hierarchy's first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also pro-vide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

The idea is illustrated in Figure 3.17(a). Space sharing is applied to assign three workloads to three clusters of virtual cores: namely VM0 and VM3 for database workload, VM1 and VM2 for web server workload, and VM4–VM7 for middleware workload. The basic assumption is that each workload runs in its own VM. However, space sharing applies equally within a single operating system. Statically distributing the directory among tiles can do much better, provided operating sys-tems or hypervisors carefully map virtual pages to physical frames. Marty and Hill suggested a two-level virtual coherence and caching hierarchy that harmonizes with the assignment of tiles to the virtual clusters of VMs.

Figure 3.17(b) illustrates a logical view of such a virtual cluster hierarchy in two levels. Each VM operates in a isolated fashion at the first level. This will minimize both miss access time and performance interference with other workloads or VMs. Moreover, the shared resources of cache capacity, inter-connect links, and miss handling are mostly isolated between VMs. The second level maintains a globally shared memory. This facilitates dynamically repartitioning resources without costly cache flushes. Furthermore, maintaining globally shared memory minimizes changes to existing system software and allows virtualization features such as content-based page sharing. A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation. Figure 3.17 shows a case study focused on consolidated server workloads in a tiled architecture. This many-core mapping scheme can also optimize for space-shared multiprogrammed workloads in a single-OS environment.



(a) Mapping of VMs into adjacent cores



(b) Multiple virtual clusters assigned to various workloads

FIGURE 3.17

CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads.

Amazon S3 Features

Amazon S3 has various features you can use to organize and manage your data in ways that support specific use cases, enable cost efficiencies, enforce security, and meet compliance requirements. Data is stored as objects within resources called “buckets”, and a

single object can be up to 5 terabytes in size. S3 features include capabilities to append metadata tags to objects, move and store data across the S3 Storage Classes, configure and enforce data access controls, secure data against unauthorized users, run big data analytics, monitor data at the object and bucket levels, and view storage usage and activity trends across your organization. Objects can be accessed through S3 Access Points or directly through the bucket hostname.

Storage management and monitoring

Amazon S3's flat, non-hierarchical structure and various management features are helping customers of all sizes and industries organize their data in ways that are valuable to their businesses and teams. All objects are stored in S3 buckets and can be organized with shared names called prefixes. You can also append up to 10 key-value pairs called **S3 object tags** to each object, which can be created, updated, and deleted throughout an object's lifecycle. To keep track of objects and their respective tags, buckets, and prefixes, you can use an **S3 Inventory** report that lists your stored objects within an S3 bucket or with a specific prefix, and their respective metadata and encryption status. S3 Inventory can be configured to generate reports on a daily or a weekly basis.

Storage management

With S3 bucket names, prefixes, object tags, and S3 Inventory, you have a range of ways to categorize and report on your data, and subsequently can configure other S3 features to take action. Whether you store thousands of objects or a billion, **S3 Batch Operations** makes it simple to manage your data in Amazon S3 at any scale. With S3 Batch Operations, you can copy objects between buckets, replace object tag sets, modify access controls, and restore archived objects from S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes, with a single S3 API request or a few clicks in the S3 console. You can also use S3 Batch Operations to run AWS Lambda functions across your objects to execute custom business logic, such as processing data or transcoding image files. To get started, specify a list of target objects by using an S3 Inventory report or by providing a custom list, and then select the desired operation from a pre-populated menu. When an S3 Batch Operation request is done, you'll receive a notification and a completion report of all changes made. Learn more about S3 Batch Operations by watching the video tutorials.

Amazon S3 also supports features that help maintain data version control, prevent accidental deletions, and replicate data to the same or different AWS Region. With **S3 Versioning**, you can easily preserve, retrieve, and restore every version of an object stored in Amazon S3, which allows you to recover from unintended user actions and application failures. To prevent accidental deletions, enable **Multi-Factor Authentication (MFA) Delete** on an S3 bucket. If you try to delete an object stored in an MFA Delete-enabled bucket, it will require two forms of authentication: your AWS account credentials and the concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device, like a hardware key fob or a Universal 2nd Factor (U2F) security key.

With **S3 Replication**, you can replicate objects (and their respective metadata and object tags) to one or more destination buckets into the same or different AWS Regions for reduced latency, compliance, security, disaster recovery, and other use cases. You can configure S3 Cross-Region Replication (CRR) to replicate objects from a source S3 bucket to one or more destination buckets in different AWS Regions. S3 Same-Region Replication (SRR) replicates objects between buckets in the same AWS Region. While live replication like CRR and SRR automatically replicates newly uploaded objects as they are written to your bucket, S3 Batch Replication allows you to replicate existing objects. You can use S3 Batch Replication to backfill a newly created bucket with existing objects, retry objects that were previously unable to replicate, migrate data across accounts, or add new buckets to your data lake. Amazon S3 Replication Time Control (S3 RTC) helps you meet compliance requirements for data replication by providing an SLA and visibility into replication times.

Amazon S3 Multi-Region Access Points accelerate performance by up to 60% when accessing data sets that are replicated across multiple AWS Regions. Based on AWS Global Accelerator, S3 Multi-Region Access Points consider factors like network congestion and the location of the requesting application to dynamically route your requests over the AWS network to the lowest latency copy of your data. S3 Multi-Region Access Points provide a single global endpoint that you can use to access a replicated data set, spanning multiple buckets in S3. This allows you to build multi-region applications with the same simple architecture that you would use in a single region, and then to run those applications anywhere in the world.

You can also enforce write-once-read-many (WORM) policies with **S3 Object Lock**. This S3 management feature blocks object version deletion during a customer-defined retention period so that you can enforce retention policies as an added layer of data protection or to meet compliance obligations. You can migrate workloads from existing WORM systems into Amazon S3, and configure S3 Object Lock at the object- and bucket-levels to prevent object version deletions prior to a pre-defined Retain Until Date or Legal Hold Date. Objects with S3 Object Lock retain WORM protection, even if they are moved to different storage classes with an S3 Lifecycle policy. To track what objects have S3 Object Lock, you can refer to an S3 Inventory report that includes the WORM status of objects. S3 Object Lock can be configured in one of two modes. When deployed in Governance mode, AWS accounts with specific IAM permissions are able to remove S3 Object Lock from objects. If you require stronger immutability in order to comply with regulations, you can use Compliance Mode. In Compliance Mode, the protection cannot be removed by any user, including the root account.

Storage monitoring

In addition to these management capabilities, use Amazon S3 features and other AWS services to monitor and control your S3 resources. Apply tags to S3 buckets to allocate costs across multiple business dimensions (such as cost centers, application names, or owners), then use **AWS Cost Allocation Reports** to view the usage and costs aggregated by the bucket tags. You can also use **Amazon CloudWatch** to track the operational health of your AWS resources and configure billing alerts for estimated charges that reach a user-defined threshold. Use **AWS CloudTrail** to track and report on bucket- and object-level activities, and configure **S3 Event Notifications** to trigger workflows and alerts or invoke AWS Lambda when a specific change is made to your S3 resources. S3 Event Notifications automatically transcodes media files as they're uploaded to S3, processes data files as they become available, and synchronizes objects with other data stores. Additionally, you can verify integrity of data transferred to and from Amazon S3, and can access the checksum information at any time using the GetObjectAttributes S3 API or an S3 Inventory report. You can choose from four supported checksum algorithms (SHA-1, SHA-256, CRC32, or CRC32C) for data integrity checking on your upload and download requests depending on your application needs.

Learn more about S3 storage management and monitoring »

Storage analytics and insights

S3 Storage Lens

S3 Storage Lens delivers organization-wide visibility into object storage usage, activity trends, and makes actionable recommendations to improve cost-efficiency and apply data protection best practices. S3 Storage Lens is the first cloud storage analytics solution to provide a single view of object storage usage and activity across hundreds, or even thousands, of accounts in an organization, with drill-downs to generate insights at the account, bucket, or even prefix level. Drawing from more than 14 years of experience helping customers optimize their storage, S3 Storage Lens analyzes organization-wide metrics to deliver contextual recommendations to find ways to reduce storage costs and apply best practices on data protection. To learn more, visit the storage analytics and insights page.

S3 Storage Class Analysis

Amazon S3 Storage Class Analysis analyzes storage access patterns to help you decide when to transition the right data to the right storage class. This Amazon S3 feature observes data access patterns to help you determine when to transition less frequently accessed storage to a lower-cost storage class. You can use the results to help improve your S3 Lifecycle policies. You can configure storage class analysis to analyze all the objects in a bucket. Or, you can configure filters to group objects together for analysis by common prefix, by object tags, or by both prefix and tags. To learn more, visit the storage analytics and insights page.

Storage classes

With Amazon S3, you can store data across a range of different S3 storage classes purpose-built for specific use cases and access patterns: **S3 Intelligent-Tiering, S3 Standard**, **S3 Standard-Infrequent Access (S3 Standard-IA)**, **S3 One Zone-Infrequent Access (S3 One Zone-IA)**, **S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval**, **S3 Glacier Deep Archive, and S3 Outposts**.

Every S3 storage class supports a specific data access level at corresponding costs or geographic location.

For data with changing, unknown, or unpredictable access patterns, such as data lakes, analytics, or new applications, use **S3 Intelligent-Tiering**, which automatically optimizes your storage costs. S3 Intelligent-Tiering automatically moves your data between three low latency access tiers optimized for frequent, infrequent, and rare access. When subsets of objects become archived over time, you can activate the archive access tier designed for asynchronous access.

For more predictable access patterns, you can store mission-critical production data in S3 Standard for frequent access, save costs by storing infrequently accessed data in S3 Standard-IA or S3 One Zone-IA, and archive data at the lowest costs in the archival storage classes — S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive. You can use **S3 Storage Class Analysis** to monitor access patterns across objects to discover data that should be moved to lower-cost storage classes. Then you can use this information to configure an **S3 Lifecycle** policy that makes the data transfer. S3 Lifecycle policies can also be used to expire objects at the end of their lifecycles.

If you have data residency requirements that can't be met by an existing AWS Region, you can use the S3 Outposts storage class to store your S3 data on premises using **S3 on Outposts**.

Learn more by visiting S3 Storage Classes, S3 Storage Class Analysis, and S3 Lifecycle management »

Access management and security

Access management

To protect your data in Amazon S3, by default, users only have access to the S3 resources they create. You can grant access to other users by using one or a combination of the following access management features: **AWS Identity and Access Management** (IAM) to create users and manage their respective access; **Access Control Lists** (ACLs) to make individual objects accessible to authorized users; **bucket policies** to configure permissions for all objects within a single S3 bucket; **S3 Access Points** to simplify managing data access to shared data sets by creating access points with names and permissions specific to each application or sets of applications; and **Query String Authentication** to grant time-limited access to others with temporary URLs. Amazon S3 also supports **Audit Logs** that list the requests made against your S3 resources for complete visibility into who is accessing what data.

Security

Amazon S3 offers flexible security features to block unauthorized users from accessing your data. Use VPC endpoints to connect to S3 resources from your **Amazon Virtual Private Cloud (Amazon VPC)** and from on-premises. Amazon S3 supports both server-side encryption (with three key management options) and client-side encryption for data uploads. Use **S3 Inventory** to check the encryption status of your S3 objects (see storage management for more information on S3 Inventory).

S3 Block Public Access is a set of security controls that ensures S3 buckets and objects do not have public access. With a few clicks in the Amazon S3 Management Console, you can apply the S3 Block Public Access settings to all buckets within your AWS account or to specific S3 buckets. Once the settings are applied to an AWS account, any existing or new buckets and objects associated with that account inherit the settings that prevent public access. S3 Block Public Access settings override other S3 access permissions, making it easy for the account administrator to enforce a “no public access” policy regardless of how an object is added, how a bucket is created, or if there are existing access permissions. S3 Block Public Access controls are auditable, provide a further layer of control, and use AWS Trusted Advisor bucket permission checks, AWS CloudTrail logs, and Amazon CloudWatch alarms. You should enable Block Public Access for all accounts and buckets that you do not want publicly accessible.

S3 Object Ownership is a feature that disables Access Control Lists (ACLs), changing ownership for all objects to the bucket owner and simplifying access management for data stored in S3. When you configure the S3 Object Ownership *Bucket owner enforced* setting, ACLs will no longer affect permissions for your bucket and the objects in it. All access control will be defined using resource-based policies, user policies, or some combination of these.

Using S3 Access Points that are restricted to a Virtual Private Cloud (VPC), you can easily firewall your S3 data within your private network. Additionally, you can use AWS Service Control Policies to require that any new S3 Access Point in your organization is restricted to VPC-only access.

IAM Access Analyzer for S3 is a feature that helps you simplify permissions management as you set, verify, and refine policies for your S3 buckets and access points. Access Analyzer for S3 monitors your existing bucket access policies to verify that they provide only the required access to your S3 resources. Access Analyzer for S3 evaluates your bucket access policies so that you can swiftly remediate any buckets with access that isn't required. When reviewing results that show potentially shared access to a bucket, you can Block Public Access to the bucket with a single click in the S3 console. For auditing purposes, you can download Access Analyzer for S3 findings as a CSV report. Additionally, the S3 console reports security warnings, errors, and suggestions from IAM Access Analyzer as you author your S3 policies. The console automatically runs more than 100 policy checks to validate your policies. These checks save you time, guide you to resolve errors, and help you apply security best practices.

IAM makes it easier for you to analyze access and reduce permissions to achieve least privilege by providing the timestamp when a user or role last used S3 and the associated actions. Use this “last accessed” information to analyze S3 access, identify unused permissions, and remove them confidently. To learn more see Refining Permissions Using Last Accessed Data.

You can use **Amazon Macie** to discover and protect sensitive data stored in Amazon S3. Macie automatically gathers a complete S3 inventory and continually evaluates every bucket to alert on any publicly accessible buckets, unencrypted buckets, or buckets shared or replicated with AWS accounts outside of your organization. Then, Macie applies machine learning and pattern matching techniques to the buckets you select to identify and alert you to sensitive data, such as personally identifiable information (PII). As security findings are generated, they are pushed out to the Amazon CloudWatch Events, making it easy to integrate with existing workflow systems and to trigger automated remediation with services like AWS Step Functions to take action like closing a public bucket or adding resource tags.

AWS PrivateLink for S3 provides private connectivity between Amazon S3 and on-premises. You can provision interface VPC endpoints for S3 in your VPC to connect your on-premises applications directly with S3 over AWS Direct Connect or AWS VPN. Requests to interface VPC endpoints for S3 are automatically routed to S3 over the Amazon network. You can set security groups and configure VPC endpoint policies for your interface VPC endpoints for additional access controls.

Learn more by visiting S3 access management and security and protecting data in Amazon S3 »

Data processing

S3 Object Lambda

With S3 Object Lambda you can add your own code to S3 GET requests to modify and process data as it is returned to an application. For the first time, you can use custom code to modify the data returned by standard S3 GET requests to filter rows, dynamically resize images, redact confidential data, and much more. Powered by AWS Lambda functions, your code runs on infrastructure that is fully managed by AWS, eliminating the need to create and store derivative copies of your data or to run expensive proxies, all with no changes required to applications.

S3 Object Lambda uses AWS Lambda functions to automatically process the output of a standard S3 GET request. AWS Lambda is a serverless compute service that runs customer-defined code without requiring management of underlying compute resources. With just a few clicks in the AWS Management Console, you can configure a Lambda function and attach it to a S3 Object Lambda Access Point. From that point forward, S3 will automatically call your Lambda function to process any data retrieved through the S3 Object Lambda Access Point, returning a transformed result back to the application. You can author and execute your own custom Lambda functions, tailoring S3 Object Lambda's data transformation to your specific use case.

Learn more by visiting the S3 Object Lambda feature page.

Query in place

Amazon S3 has a built-in feature and complementary services that query data without needing to copy and load it into a separate analytics platform or data warehouse. This means you can run big data analytics directly on your data stored in Amazon S3. **S3 Select** is an S3 feature designed to increase query performance by up to 400%, and reduce querying costs as much as 80%. It works by retrieving a subset of an object's data (using simple SQL expressions) instead of the entire object, which can be up to 5 terabytes in size.

Amazon S3 is also compatible with AWS analytics services Amazon Athena and Amazon Redshift Spectrum. **Amazon Athena** queries your data in Amazon S3 without needing to extract and load it into a separate service or platform. It uses standard SQL expressions to analyze your data, delivers results within seconds, and is commonly used for ad hoc data discovery. **Amazon Redshift Spectrum** also runs SQL queries directly against data at rest in Amazon S3, and is more appropriate for complex queries and large data sets (up to exabytes). Because Amazon Athena and Amazon Redshift share a common data catalog and data formats, you can use them both against the same data sets in Amazon S3.

Learn more by visiting building big data storage solutions and S3 Select »

Data transfer

AWS provides a portfolio of data transfer services to provide the right solution for any data migration project. The level of connectivity is a major factor in data migration, and AWS has offerings that can address your hybrid cloud storage, online data transfer, and offline data transfer needs.

Hybrid cloud storage: **AWS Storage Gateway** is a hybrid cloud storage service that lets you seamlessly connect and extend your on-premises applications to AWS Storage. Customers use Storage Gateway to seamlessly replace tape libraries with cloud storage, provide cloud storage-backed file shares, or create a low-latency cache to access data in AWS for on-premises applications.

Online data transfer: **AWS DataSync** makes it easy and efficient to transfer hundreds of terabytes and millions of files into Amazon S3, up to 10x faster than open-source tools. DataSync automatically handles or eliminates many manual tasks, including scripting copy jobs, scheduling and monitoring transfers, validating data, and optimizing network utilization. Additionally, you can use AWS DataSync to copy objects between a bucket on S3 on Outposts and a bucket stored in an AWS Region. The **AWS Transfer Family** provides fully managed, simple, and seamless file transfer to Amazon S3 using SFTP, FTPS, and FTP. **Amazon S3 Transfer Acceleration** enables fast transfers of files over long distances between your client and your Amazon S3 bucket.

Offline data transfer: The **AWS Snow Family** is purpose-built for use in edge locations where network capacity is constrained or nonexistent and provides storage and computing capabilities in harsh environments. The **AWS Snowball** service uses ruggedized, portable storage and edge computing devices for data collection, processing, and migration. Customers can ship the physical Snowball device for offline data migration to AWS. **AWS Snowmobile** is an exabyte-scale data transfer service used to move massive volumes of data to the cloud, including video libraries, image repositories, or even a complete data center migration.

Customers can also work with third-party providers from the **AWS Partner Network (APN)** to deploy hybrid storage architectures, integrate Amazon S3 into existing applications and workflows, and transfer data to and from the AWS Cloud.

Learn more by visiting AWS cloud data migration services » , AWS Storage Gateway » , AWS DataSync » , AWS Transfer Family » , Amazon S3 Transfer Acceleration » , AWS Snow Family »

Performance

Amazon S3 provides industry leading performance for cloud object storage. Amazon S3 supports parallel requests, which means you can scale your S3 performance by the factor of your compute cluster, without making any customizations to your application. Performance scales per prefix, so you can use as many prefixes as you need in parallel to achieve the required throughput. There are no limits to the number of prefixes. Amazon S3 performance supports at least 3,500 requests per second to add data and 5,500 requests per second to retrieve data. Each S3 prefix can support these request rates, making it simple to increase performance significantly.

To achieve this S3 request rate performance you do not need to randomize object prefixes to achieve faster performance. That means you can use logical or sequential naming patterns in S3 object naming without any performance implications. Refer to the Performance Guidelines for Amazon S3 and Performance Design Patterns for Amazon S3 for the most current information about performance optimization for Amazon S3.

Consistency

Amazon S3 delivers strong read-after-write consistency automatically for all applications, without changes to performance or availability, without sacrificing regional isolation for applications, and at no additional cost. With strong consistency, S3 simplifies the migration of on-premises analytics workloads by removing the need to make changes to applications, and reduces costs by removing the need for extra infrastructure to provide strong consistency.

Any request for S3 storage is strongly consistent. After a successful write of a new object or an overwrite of an existing object, any subsequent read request immediately receives the latest version of the object. S3 also provides strong consistency for list operations, so after a write, you can immediately perform a listing of the objects in a bucket with any changes reflected.

Amazon S3 security and access management

To protect your data in Amazon S3, by default, users only have access to the S3 resources they create. You can grant access to other users by using one or a combination of the following access management features: **AWS Identity and Access Management (IAM)** to create users and manage their respective access; **Access Control Lists (ACLs)** to make individual objects accessible to authorized users; **bucket policies** to configure permissions for all objects within a single S3 bucket; and **Query String Authentication** to grant time-limited access to others with temporary URLs. Amazon S3 also supports **Audit Logs** that list the requests made against your S3 resources for complete visibility into who is accessing what data.

Block Public Access

With a few clicks in the S3 management console, you can apply S3 Block Public Access to every bucket in your account—both existing and any new buckets created in the future—and make sure that there is no public access to any object. S3 Block Public Access settings override S3 permissions that allow public access, making it easy for the account administrator to set up a centralized control to prevent variation in security configuration regardless of how an object is added or a bucket is created.

Object Lock

Amazon S3 Object Lock blocks object version deletion during a customer-defined retention period so that you can enforce retention policies as an added layer of data protection or for regulatory compliance. You can migrate workloads from existing write-once-read-many (WORM) systems into Amazon S3, and configure S3 Object Lock at the object- and bucket-levels to prevent object version deletions prior to pre-defined Retain Until Dates or Legal Hold Dates.

Object Ownership

Amazon S3 Object Ownership disables Access Control Lists (ACLs), changing ownership for all objects to the bucket owner and simplifying access management for data stored in S3. When you configure the S3 Object Ownership *Bucket owner enforced* setting, ACLs will no longer affect permissions for your bucket and the objects in it. All access control will be defined using resource-based policies, user policies, or some combination of these. For more information, see Controlling Object Ownership.

Identity and Access Management

By default, all Amazon S3 resources—buckets, objects, and related subresources—are private: only the resource owner, an AWS account that created it, can access the resource. Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. You may choose to use resource-based policies, user policies, or some combination of these to manage permissions to your Amazon S3 resources. By default, an S3 object is owned by the account that created the object, including when this account is different than the bucket owner. You can use S3 Object Ownership to disable Access Control Lists and change this behavior. If you do, each object in a bucket is owned by the bucket owner. For more information, see Identity and access management in Amazon S3.

Amazon Macie

Discover and protect sensitive data at scale in Amazon S3 with Amazon Macie. Macie automatically provides you with a full inventory of your S3 buckets by scanning buckets to identify and categorize the data. You receive actionable security findings enumerating any data that fits these sensitive data types, including personal identifiable information (PII) (e.g. customer names and credit cards numbers), and categories defined by privacy regulations, such as GDPR and HIPAA. Macie also automatically and continually evaluates bucket-level preventative controls for any buckets that are unencrypted, publicly accessible, or shared with accounts outside of your organization, allowing you to quickly address unintended settings on buckets.

Encryption

Amazon S3 supports both server-side encryption (with three key management options: SSE-KMS, SSE-C, SSE-S3) and client-side encryption for data uploads. Amazon S3 offers flexible security features to block unauthorized users from accessing your data. Use VPC endpoints to connect to S3 resources from your **Amazon Virtual Private Cloud (Amazon VPC)**. Use **S3 Inventory** to check the encryption status of your S3 objects (see storage management for more information on S3 Inventory).

AWS Trusted Advisor

Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to help close security gaps.

Trusted Advisor has the following Amazon S3-related checks: logging configuration of Amazon S3 buckets, security checks for Amazon S3 buckets that have open access permissions, and fault tolerance checks for Amazon S3 buckets that don't have versioning enabled, or have versioning suspended.

AWS PrivateLink for S3

Access Amazon S3 directly as a private endpoint within your secure, virtual network with AWS PrivateLink for S3. Simplify your network architecture by connecting to S3 from on-premises or in the cloud using private IP addresses from your Virtual Private Cloud (VPC). You no longer need to use public IPs, configure firewall rules, or configure an internet gateway to access S3 from on-premises.

Verify data integrity

Choose from four supported checksum algorithms (SHA-1, SHA-256, CRC32, or CRC32C) to check data integrity on your upload and download requests. Automatically calculate and verify checksums as you store or retrieve data from Amazon S3, and access the checksum information at any time using the GetObjectAttributes S3 API or an S3 Inventory report.

Elastic IP addresses

PDFRSS

An *Elastic IP address* is a static IPv4 address designed for dynamic cloud computing. An Elastic IP address is allocated to your AWS account, and is yours until you release it. By using an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account. Alternatively, you can specify the Elastic IP address in a DNS

record for your domain, so that your domain points to your instance. For more information, see the documentation for your domain registrar, or Set up dynamic DNS on Your Amazon Linux instance.

An Elastic IP address is a public IPv4 address, which is reachable from the internet. If your instance does not have a public IPv4 address, you can associate an Elastic IP address with your instance to enable communication with the internet. For example, this allows you to connect to your instance from your local computer.

Contents

Elastic IP address pricing

To ensure efficient use of Elastic IP addresses, we impose a small hourly charge if an Elastic IP address is not associated with a running instance, or if it is associated with a stopped instance or an unattached network interface. While your instance is running, you are not charged for one Elastic IP address associated with the instance, but you are charged for any additional Elastic IP addresses associated with the instance.

For more information, see Elastic IP Addresses on the Amazon EC2 Pricing, On-Demand Pricing page

.

Elastic IP address basics

The following are the basic characteristics of an Elastic IP address:

- An Elastic IP address is static; it does not change over time.
- An Elastic IP address is for use in a specific Region only, and cannot be moved to a different Region.
- An Elastic IP address comes from Amazon's pool of IPv4 addresses, or from a custom IPv4 address pool that you have brought to your AWS account.
- To use an Elastic IP address, you first allocate one to your account, and then associate it with your instance or a network interface.
- When you associate an Elastic IP address with an instance, it is also associated with the instance's primary network interface. When you associate an Elastic IP address with a network interface that is attached to an instance, it is also associated with the instance.
- When you associate an Elastic IP address with an instance or its primary network interface, the instance's public IPv4 address (if it had one) is released back into Amazon's pool of public IPv4 addresses. You cannot reuse a public IPv4 address, and you cannot convert a public IPv4 address to an Elastic IP address. For more information, see Public IPv4 addresses.
- You can disassociate an Elastic IP address from a resource, and then associate it with a different resource. To avoid unexpected behavior, ensure that all active connections to the resource named in the existing association are closed before you make the change. After you have associated your Elastic IP address to a different resource, you can reopen your connections to the newly associated resource.
- A disassociated Elastic IP address remains allocated to your account until you explicitly release it. We impose a small hourly charge for Elastic IP addresses that are not associated with a running instance.
- When you associate an Elastic IP address with an instance that previously had a public IPv4 address, the public DNS host name of the instance changes to match the Elastic IP address.
- We resolve a public DNS host name to the public IPv4 address or the Elastic IP address of the instance outside the network of the instance, and to the private IPv4 address of the instance from within the network of the instance.
- When you allocate an Elastic IP address from an IP address pool that you have brought to your AWS account, it does not count toward your Elastic IP address limits. For more information, see Elastic IP address limit.
- When you allocate the Elastic IP addresses, you can associate the Elastic IP addresses with a network border group. This is the location from which we advertise the CIDR block. Setting the network border group limits the CIDR block to this

group. If you do not specify the network border group, we set the border group containing all of the Availability Zones in the Region (for example, us-west-2).

- An Elastic IP address is for use in a specific network border group only.

Work with Elastic IP addresses

The following sections describe how you can work with Elastic IP addresses.

Tasks

- Allocate an Elastic IP address
- Describe your Elastic IP addresses
- Tag an Elastic IP address
- Associate an Elastic IP address with an instance or network interface
- Disassociate an Elastic IP address
- Release an Elastic IP address
- Recover an Elastic IP address