**Question: What does it mean to create a Docker image and why do we use Docker images?**

A [Docker](https://www.techtarget.com/searchitoperations/definition/Docker) image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker [container](https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization), like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.

Docker images have multiple layers, each one originates from the previous layer but is different from it. The layers [speed up Docker builds](https://www.techtarget.com/searchitoperations/tip/Optimize-Docker-images-for-improved-efficiency-and-security) while increasing reusability and decreasing disk use. Image layers are also read-only files. Once a container is created, a writable layer is added on top of the unchangeable images, allowing a user to make changes.

A Docker image has everything needed to run a containerized application, including code, [config files](https://www.techtarget.com/searchitoperations/definition/configuration-file), environment variables, libraries and runtimes. When the image is deployed to a Docker environment, it can be executed as a Docker container. The [docker run command](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Docker-run-vs-docker-compose-Whats-the-difference) creates a container from a specific image.

Docker images are a reusable asset -- deployable on any host. Developers can take the static image layers from one project and use them in another. This saves the user time, because they do not have to recreate an image from scratch.

**Question: Please explain what is the difference from a Container vs a Virtual Machine?**

|  |  |  |
| --- | --- | --- |
| **Characteristic** | **Containers** | **Virtual Machines** |
| Instance security | An individual container is isolated from other containers and the host. However, container boundary security could be compromised if best practices are not followed. Some suggested practices include   * use of verified container base images to prevent malicious code within the container * disable ‘privileged flag’ on containers to prevent unrestricted resource access * restrict communication protocols and processes allowed in and out of the container preventing a container from being hijacked | VM provides strong isolation from other VMs and the host. This allows different apps to be hosted by VMs on the same server or cluster.  As the OS image is part of the VM the associated security protocols can be implemented on the VM. In addition, the OS resources and management tools are available to the app. This makes VMs more secure as compared to containers.  VM platform security capabilities have been leveraged to isolate each container in a lightweight VM, for example, Microsoft VM’s Hyper-V isolation mode. |
| Portability and Deployment | * Containers do not have an OS image and use the host OS, making them inherently lighter and faster to deploy. This is possible through the Bin/Lib (binaries and libraries) in a container image that have instructions/code to support communication with/through the orchestrator. The orchestrator runs on the host OS, thus facilitating communication to/from the container and external applications. This makes containers to easily port across OS * Updating OS bin/lib files in a container is easily done by updating the container’s image file using ‘dockerfile’ and redeploying the container. This allows for quick fixes to containers when host OS is updated * An individual container is deployed through the CLI interface, while the orchestrator can be used to deploy multiple containers | * VMs run the OS in them, making them heavier and needing more time to deploy in comparison to a container. A hypervisor communicates with VM and the host OS. The hypervisor runs on the host OS. The VM advantage being that the most suited OS for an application is encapsulated in it * Updating a VM to ensure that new OS updates are installed, would mean upgrading or at times creating a new VM. This is more effort and time when many VMs are in production * An individual VM is deployed through the CLI interface, while a virtualization management application like VMware’s vSphere can be used to deploy multiple VMs |
| Persistent storage | Use local storage for a single node, or SMB shares for storage shared by multiple nodes or servers. | Use a virtual hard disk (VHD) for local storage for a single VM, or an SMB file share for storage shared by multiple servers. |

**Question: What are 5 examples of container orchestration tools (please list tools)?**

**The top orchestration tools for containers are**

* **Kubernetes**

[Kubernetes](https://kubernetes.io/) is an open-source platform that was designed and developed by Google. Currently, Kubernetes is maintained by [Cloud Native Computing Foundation](https://www.cncf.io/). It is the leading container orchestration system. This is because container solutions widely adopted as it is user-friendly. Kubernetes automates deployment, scaling, and management of container applications.

* **Docker Swarm**

[Docker Swarm](https://docs.docker.com/engine/swarm/key-concepts/) is a container orchestration tool for Docker applications. It gives you the most straightforward route for orchestrating a Docker hosts cluster. Docker is simple and does not need more software. It, thus, suits small-scale enterprises handling container technology.

* **Apache Mesos**

[Mesos](http://mesos.apache.org/documentation/latest/) is an open-source tool developed by [University of California at Berkeley](https://en.wikipedia.org/wiki/Apache_Mesos). It is a cluster management tool that can effectively perform container orchestration.

Apache Mesos handle resource sharing and allocation across different containers. It enables resource allocation using kernel features such as [CGroups in Linux](https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html) and [Zones in Solaris](https://en.wikipedia.org/wiki/Solaris_Containers).

* **Amazon Elastic Container Service (ECS)**

[Amazon ECS](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_instances.html) is an orchestration tool that runs container clusters on [Amazon EC2 instances](https://aws.amazon.com/ec2/instance-types/). ECS powers services such as Amazon Sage Maker, Amazon Batch, and Amazon recommendation engine.

Amazon ECS provides a reliable, scalable, and secure orchestration platform. Therefore Amazon ECS can be considered appropriate for critical and sensitive applications.

* **Azure Kubernetes Service (AKS)**

[AKS](https://azure.microsoft.com/en-us/services/kubernetes-service/) is an open-source container orchestration tool that is a managed Kubernetes service hosted on the Azure cloud platform. AKS offers a serverless continuous integration (CI) and continuous deployment (CD) occurrence. Also, AKS offers scalability, end-to-end deployment, and availability.

Azure Kubernetes offer services that help DevOps in software development, such as planning, administration practices, and development practices. All these services are hosted on Microsoft Azure, and DevOps use them to benefit their requirements.

**Question: How does a Docker image differ from a Docker container?**

In OOP language, if an image is a class, then a container is an instance of a class is a runtime object.

It all starts with a script of instructions that define how to build a specific Docker image. This script is called a [Dockerfile](https://phoenixnap.com/kb/create-docker-images-with-dockerfile" \t "_blank). 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.