* **What is a docker container?**

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on

[Docker Engine](https://www.docker.com/products/container-runtime). Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Docker containers that run on Docker Engine:

* **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
* **Lightweight:** Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
* **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry
* **What is the benefit of docker container?**

**The rapid growth of Docker containers is being fueled by the many benefits that it provides. If you have applications that run on VMs or bare metal servers today, you should consider containerizing them to take advantage of the benefits that come from Docker containers. These benefits can be seen across your organization, from developers and operations, to Quality Assurance (QA). The primary benefits of Docker are speed, consistency, density, and portability.**

**Speed**

**Because of their lightweight and modular nature, containers can enable rapid iteration of your applications. Development speed is improved by the ability to deconstruct applications into smaller units. This reduces shared resources between application components, leading to fewer compatibility issues between required libraries or packages. Operational speed is improved, because code built in a container on a developer’s local machine can be easily moved to a test server by simply moving the container. The container startup time primarily depends on the size of the container image, cache, and the time to pull the image and start the container on host. To improve the container startup time, you must keep the size of image as small as possible, using techniques like multi-stage builds and local cache when applicable. For more information, see** [**Best practices for writing Dockerfiles**](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)**.**

**Consistency**

**The consistency and fidelity of a modular development environment provide predictable results when moving code between development, test, and production systems. By verifying that the container encapsulates exact versions of necessary libraries and packages, it is possible to minimize the risk of bugs due to slightly different dependency revisions. This concept easily lends itself to a *disposable system* approach, in which patching individual containers is less preferable than building new containers in parallel, testing, and replacing the earlier. This practice helps avoid *drift* of packages across a fleet of containers, versions of your application, or development, test, and production environments; the result is more consistent, predictable, and stable applications.**

**Density and resource efficiency**

**Containers facilitate enhanced resource efficiency by allowing multiple containers to run on a single system. Resource efficiency is a natural result of the isolation and allocation techniques that containers use. Containers can easily be restricted to a certain number of CPUs and allocated specific amounts of memory. By understanding what resource a container needs and what resource is available to your VM or underlying host server, it’s possible to maximize the containers running on a single host, resulting in higher density, increased efficiency of compute resources, and less wastage on excess capacity. Amazon ECS achieves this through placement strategies. The *binpack* placement strategy tries to optimize placement of containers to be as cost-efficient as possible. Containers in Amazon ECS are part of Amazon ECS tasks placed on compute instances to leave the least amount of unused CPU or memory. This in turn minimizes the number of computed instances in use, resulting in better resource efficiency. The placement strategies can be supported by placement constraints, which lets you place tasks by constraints like the instance type or the availability zone. This further enables you to efficiently utilize resources by verifying that your tasks are running on instance types suitable for your workload, by logically separating your tasks using task groups.**

**Amazon EKS uses the native Kubernetes scheduling and placement strategy, which tries to place pods on nodes to match the requirements of your workloads across nodes and not to place pods on nodes where there aren’t sufficient resources.**

**Kubernetes allows you to limit the resources like CPU and memory to Kubernetes namespaces, pods, or containers. For more information, see** [**Scheduling**](https://docs.aws.amazon.com/whitepapers/latest/docker-on-aws/scheduling.html)**.**

**Portability**

**The flexibility of Docker containers is based on their portability, ease of deployment, and smaller size compared to virtual machines. Like Git, Docker provides a simple mechanism for developers to download and install Docker containers and their subsequent applications using the command docker pull. Because Docker provides a standard interface, it makes containers easy to deploy wherever you like, providing portability among different versions of Linux, your laptop, or the cloud. The images Docker builds are compliant with OCI (Open Container Initiative), which was created to support fully interoperable container standards. Docker can build images by reading the instructions from a *Dockerfile*, which is a text-based manifest. You can run the same Docker container on any supported version of Linux if you have the Docker stack installed on the host. Additionally, Docker supports Windows containers which can run on supported Windows versions. Containers also provide flexibility by making a microservices architecture possible. In contrast to common infrastructure models in which a virtual machine runs multiple services, packaging services inside their own container on top of a host OS allows a service to be moved between hosts, isolated from failure of other adjacent services, and protected from errant patches or software upgrades on the host system. Because Docker provides clean, reproducible, and modular environments, it streamlines both code deployment and infrastructure management. Docker offers numerous benefits for a variety of use cases, whether in development, testing, deployment, or production.**

* **Write a installation step of docker**

* **Step 1: Downloading Docker. ...**
* **Step 2: Configuration. ...**
* **Step 3: Running the instalation. ...**
* **Step 4: Restart. ...**
* **Step 5: License agreement. ...**
* **Step 6: WSL 2 installation. ...**
* **Step 7 — Starting Docker Desktop. ...**
* **Step 8— Testing Docker.**