**Containers vs. virtualization**

* Containers are an abstraction that packages application code and dependencies together.
* Instances of the container can then be created, started, stopped, moved, or deleted using the Docker API or command-line interface (CLI). Containers can be connected to one or more networks, be attached to storage, or create new images based on their current states.
* Containers differ from virtual machines, which use a software abstraction layer on top of computer hardware, allowing the hardware to be shared more efficiently in multiple instances that will run individual applications
* Docker containers require fewer physical hardware resources than virtual machines, and they also offer faster startup times and lower overhead. This makes Docker ideal for high-velocity environments, where rapid software development cycles and scalability are crucial.

Basic components of Docker

The basic components of Docker include:

* [**Docker images:**](https://docs.docker.com/guides/docker-concepts/the-basics/what-is-an-image/)Docker images are the blueprints for your containers. They are read-only templates that contain the instructions for creating a Docker container. You can think of a container image as a snapshot of a specific state of your application.
* [**Containers:**](https://docs.docker.com/guides/docker-concepts/the-basics/what-is-a-container/)Containers are the instances of Docker images. They are lightweight and portable, encapsulating your application along with its dependencies. Containers can be created, started, stopped, moved, and deleted using simple Docker commands.
* [**Dockerfiles:**](https://docs.docker.com/reference/dockerfile/)A Dockerfile is a text document containing a series of instructions on how to build a Docker image. It includes commands for specifying the base image, copying files, installing dependencies, and setting up the environment.
* [**Docker Engine**](https://www.docker.com/blog/blog-how-to-check-docker-version/)**:**Docker Engine is the core component of Docker. It’s a client-server application that includes a server with a long-running daemon process, APIs for interacting with the daemon, and a CLI client.
* [**Docker Desktop**](https://www.docker.com/products/docker-desktop/)**:** Docker Desktop is a commercial product sold and supported by Docker, Inc. It includes the Docker Engine and other open source components, proprietary components, and features like an intuitive GUI, synchronized file shares, access to cloud resources, debugging features, native host integration, governance, security features, and administrative settings management.
* [**Docker Hub:**](https://docs.docker.com/docker-hub/)Docker Hub is a public registry where you can store and share Docker images. It serves as a central place to find official Docker images and user-contributed images. You can also use Docker Hub to automate your workflows by connecting it to your CI/CD pipelines.

**Basic Docker commands**

Docker commands are simple and intuitive. For example:

* **docker run**: Runs a Docker container from a specified image. For example, docker run hello-world will run a container from the “hello-world” image.
* **docker build**: Builds an image from a Dockerfile. For example, docker build -t my-app . will build an image named “my-app” from the Dockerfile in the current directory.
* **docker pull**: Pulls an image from Docker Hub. For example, docker pull nginx will download the latest NGINX image from Docker Hub.
* **docker ps**: Lists all running containers. For example, docker ps -a will list all containers, including stopped ones.
* **docker stop**: Stops a running Docker container. For example, docker stop <container\_id> will stop the container with the specified ID.
* **docker rm**: Removes a stopped container. For example, docker rm <container\_id> will remove the container with the specified ID.