# Glossary: Container Basics

| Term | Definition |
| --- | --- |
| **Agile** | is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer issues. |
| **Client-server architecture** | is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. |
| **A container** | powered by the containerization engine, is a standard unit of software that encapsulates the application code, runtime, system tools, system libraries, and settings necessary for programmers to efficiently build, ship and run applications. |
| **Container Registry** | Used for the storage and distribution of named container images. While many features can be built on top of a registry, its most basic functions are to store images and retrieve them. |
| **CI/CD pipelines** | A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation. |
| **Cloud native** | A cloud-native application is a program that is designed for a cloud computing architecture. These applications are run and hosted in the cloud and are designed to capitalize on the |

| | |
|---|---|
| | inherent characteristics of a cloud computing software delivery model. |
| **Daemon-less** | A container runtime that does not run any specific program (daemon) to create objects, such as images, containers, networks, and volumes. |
| **DevOps** | is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. |
| **Docker** | An open container platform for developing, shipping and running applications in containers. |
| **A Dockerfile** | is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile. |
| **Docker client** | 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 Command Line Interface (CLI)** | The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon. |
| **Docker daemon (dockerd)** | creates and manages Docker objects, such as images, containers, networks, and volumes. |
| **Docker Hub** | is the world's easiest way to create, manage, and deliver your team's container applications. |
| **Docker localhost** | Docker provides a host network which lets containers share your host’s networking stack. This approach means that a localhost in a |

| | |
|---|---|
| | container resolves to the physical host, instead of the container itself. |
| **Docker remote host** | A remote Docker host is a machine, inside or outside our local network which is running a Docker Engine and has ports exposed for querying the Engine API. |
| **Docker networks** | help isolate container communications. |
| **Docker plugins** | such as a storage plugin, provides the ability to connect external storage platforms. |
| **Docker storage** | uses volumes and bind mounts to persist data even after a running container is stopped. |
| **LXC** | LinuX Containers is a OS-level virtualization technology that allows creation and running of multiple isolated Linux virtual environments (VE) on a single control host. |
| **IBM Cloud Container Registry** | stores and distributes container images in a fully managed private registry. |
| **Image** | An immutable file that contains the source code, libraries, and dependencies that are necessary for an application to run. Images are templates or blueprints for a container. |
| **Immutability** | Images are read-only; if you change an image, you create a new image. |
| **Microservices** | are a cloud-native architectural approach in which a single application contains many loosely coupled and independently deployable smaller components or services. |
| **Namespace** | A Linux namespace is a Linux kernel feature that isolates and virtualizes system resources. Processes which are restricted to a namespace can only interact with resources or processes that are part of the same namespace. Namespaces are an important part of Docker's isolation model. Namespaces exist for each type of resource, including networking, storage, processes, hostname control and others. |

| | |
|---|---|
| **Operating System Virtualization** | OS-level virtualization is an operating system paradigm in which the kernel allows the existence of multiple isolated user space instances, called containers, zones, virtual private servers, partitions, virtual environments, virtual kernels, or jails. |
| **Private Registry** | Restricts access to images so that only authorized users can view and use them. |
| **REST API** | A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. |
| **Registry** | is a hosted service containing repositories of images which responds to the Registry API. |
| **Repository** | is a set of Docker images. A repository can be shared by pushing it to a registry server. The different images in the repository can be labelled using tags. |
| **Server Virtualization** | Server virtualization is the process of dividing a physical server into multiple unique and isolated virtual servers by means of a software application. Each virtual server can run its own operating systems independently. |
| **Serverless** | is a cloud-native development model that allows developers to build and run applications without having to manage servers. |
| **Tag** | A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other. |