**Docker**

Introduction

**What is Docker?**

* **According to Docker –** “Docker allows you to package your application with all of its dependencies into a standardized unit for software development”.
* Docker is a new open source container technology that automates the deployment of application inside isolated Linux container including required runtime dependencies of the application.
* Simply put, Docker packs your application with required dependencies but into a container and allows for rapid deployment.

**Why should I use Docker?**

* Docker containers are the hottest technology nowadays.
* Docker reduces build and deploy time.
* Docker allows you to run many containers simultaneously on any host and can run anywhere.
* Docker provides a safe way to run applications securely, isolated in a container, packaged with all its dependencies and libraries.
* The dockerized application can run on any virtual platform or bare metal without any modification.

**If you are a developer, the Docker technology will give you:**

* faster delivery of your applications.
* writedockerized app and run anywhere on anything.
* can be used as version control.
* You can use a repository to share your containers with other teammembers.

**If you are a DevOps person, the Docker technology will give you:**

* deploying and scaling more easily.
* make the lifecycle more efficient.
* Lightweight footprint – Docker images are very small.
* Simplified maintenance – Docker reduces effort and risk of problems with application dependencies.

**Docker architecture**

|  |  |  |
| --- | --- | --- |
| container 1 | container 2 | container 3 |
| Application 1 dependencies and libraries | Application 2 dependencies and libraries | Application 3 dependencies and libraries |

|  |
| --- |
| Docker engine |
| Host Operating System |
| Infrastructure – VM, Cloud, Bare metal |

Docker components:

**Docker Daemon -** Runs on a host machine. The user does not directly interact with the daemon.

**Docker Client –** Accepts commands from a user communicates with Docker daemon through CLI or REST API.

**Docker Image -** is a template. The image contains all the data and metadata needed to run the containers that are launched from the image.

**Docker Container –** holds everything that is needed for an application to run. Each container is created from a Docker image. Each container is an isolated and secure application platform.

**Docker Registry –** A hosted collection of tagged images that together create the file system for a container. A registry can be public or private.

**Docker Hub –** Software as a Service (SaaS) platform for sharing and managing Docker images.

**Docker Dockerfile** – A text file with a simple syntax for building images.

**Docker Swarm** - is a clustering and scheduling tool for Docker containers.

**Docker installation**

**Docker Editions**

Docker is now coming into editions:

1. Community Edition (CE)
2. Enterprise Edition (EE)

the docker community package is called **docker-ce** and Enterprise Edition called **docker-ee**. CE version is for the developers and ops community. Each month, Docker releases Edge version with the latest features and quarterly releases CE stable version. Docker CE is available for the following operating systems: Ubuntu, Debian, CentOS, Fedora, Windows 10, MacOS, Microsoft Azure and AWS. Docker EE is designed for enterprise development needs (building, shipping, and running business-critical applications in production environment.

Docker also changed its versioning system. The new version format is: **Year.Month.Version, plus either -ce (for Community Edition) or -ee (for Enterprise Edition).**

**Example:**

root@hostname:~# docker -v

Docker version 17.04.0-ce, build 4845c56

The above example states “Community Edition, released in April 2017”.

**Installation Instructions according to operating system**

**Linux**

The docker engine is built on top of the Linux kernel. That means the docker engine runs directly on any Linux distributions. You have several ways to install the Docker engine on Linux:

1. Installation scripts are provided by the Docker community.
2. A docker package is included in official Linux distributions (the package might not be the latest version of docker).
3. Packages are supplied by docker for any Linux distributions.
4. Binary download from Docker repository.

The first option is to use the script provided at <https://get.docker.com> to install Docker engine. This script will check installation requirements, then installs docker depending on your OS. The installation process will include some extra dependencies required by docker if they are missing.

To use this script, we will use the command:

 # curl -s <https://get.docker.com/> | sudo sh

**Install Docker CE on CentOS 7.X or RHEL 7.X**

Add the official docker repository:

 # yum-config-manager –add-repo https://download.docker.com/linux/centos/docker-ce.repo

**Install Docker CS Engine:**

 # yum install docker-ce

**Configure the Docker daemon to start automatically if it is not started automatically after the installation:**

 # systemctl enable docker.service

# systemctl start docker.service

**Install Docker CE on Ubuntu 16.04 LTS**

Add the GPG key for Docker repository:

 # curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -

Add the official Docker repository:

 # add-apt-repository “deb [arch=amd64] <https://download.docker.com/linux/ubuntu> $(lsb\_release -cs) stable”

Update the apt database and install CE Docker Engine:

 # apt-get update update; apt-get -y install docker-ce

**Configure the Docker daemon to start automatically if it is not started automatically after the installation:**

 # systemctl enable docker.service

# systemctl start docker.service

**MacOS**

MacOS is not Linux. MacOS doesn’t have the required kernel features to run a Docker daemon. We still need to have Linux running on our machine as a VM.

You have two options to install Docker on MacOS:

1. **Docker for Mac:** Docker offers a new package called “Docker for MAC” that’s required for Mac OS X El Capitan 10.11 or newer. It uses the open source project called xhyve. Docker for MAC creates a Linux virtual machine on MacOS, that is managed by the Docker application. Mac’s network interface is shared with the VM, making it easier to mount Mac file systems inside containers. These features allow Docker to operate as a native Mac application.
2. **Docker Toolbox:** Docker Toolbox installs additional features like a Docker machine and VirtualBox hypervisor. In Docker Toolbox, the Docker daemon runs inside a boot2docker VirtualBox VM controlled by the Docker machine.

Docker for Mac is the newest offering for the Mac operating system. If you have an earlier Mac that doesn’t meet the Docker for Mac requirements you should use the Docker Toolbox solution.

**Windows**

**Install Docker CE on Windows 10**

**Like MacOS, Windows is not Linux. Windows 10 doesn’t have the required kernel features to run a Docker daemon.**

Docker provides Docker for Windows package. Docker Community Edition for Windows currently only ships on Windows 10 Pro, Enterprise, and Education (1511 November update, Build 10586 or later) editions that support Hyper-V. Docker for Windows is a Windows application, including a native user interface and auto-update capability.

Download the installer from <https://download.docker.com/win/stable/InstallDocker.msi> and install it. When the installation finishes, Docker starts automatically.

Open Command Prompt or PowerShell and run Docker commands like: **docker ps**, or **docker info**.

**Install Docker EE for Windows Server 2016**

Docker Enterprise Edition (EE) is already built-in to Windows Server setups. The kernel containerization features are available in all versions of Windows Server 2016. These features allow you to quickly start a new and fully isolated container. This means that you can run Docker containers natively on Windows server.

Open a PowerShell command prompt, and type the following commands:

 PS> Install-Module -Name DockerMsftProvider -Force

PS> Install-Package -Name docker -ProviderName DockerMsftProvider -Force

PS> Restart-Computer -Force

Run hello-world container to check the installation status:

 PS C:Userspcuser> docker run hello-world

**Containers and Images**

**What is a Docker Image?**

An image is a template that can be turned into a container.

1. A docker image is comprised of a series of layers. Each layer is a set of filesystem changes and has a unique ID upon its creation.
2. The image contains all the data and metadata needed to run the containers that are launched from the image.
3. The docker image is immutable. Read-only template.

|  |
| --- |
| **Docker Image** |
| Layer N |
| Layer 1 |
| Layer 0 (ROOT FS) |

|  |
| --- |
| BOOT FS |

**Docker Images - common CLI commands**

* **docker images –** Shows all images
* **docker build**  - Builds an image from a Dockerfile
* **docker commit** – Creates a new image from a container’s changes, pausing it temporarily if it is running.
* **docker rmi** – Removes one or more images
* **docker history -** Shows the history of an image
* **docker tag –** Tags an image into a repository
* **docker import –** Imports the contents from a tarball to create a filesystem image
* **docker export –** Exports a container’s filesystem as a tar archive
* **docker search –** Search the Docker Hub for images

**What is a Docker container?**

1. Each container is created from a docker image.
2. The major difference between a container and image is the top writable layer.
3. When a container is started, a writable layer is added to the top.
4. You can commit the changes made to a container and make a new image out of it.
5. You may have many running containers of the same image.
6. Each container is isolated and secure application environment.

**R/W Container**

|  |
| --- |
| **Docker Container** |
| READ/WRITE LAYER N |
| READ/WRITE LAYER 2 |
| READ/WRITE LAYER 1 |

**R/O Image**

|  |
| --- |
| **Docker Image** |
| Layer N |
| Layer 1 |
| Layer 0 (ROOT FS) |

|  |
| --- |
| BOOT FS |

**Docker Container – common CLI commands**

* **docker create –** Creates a new container, but does not start it.
* **docker run –** creates and starts a container in one operation.
* **docker rm –** Removes one or more containers.
* **docker start –** Starts one or more stopped containers.
* **docker stop –** Stops a running container.
* **docker restart –** Restarts a container.
* **docker kill –** Kills a running container.
* **docker attach –** connects to a running container.
* **docker exec –** Runs a command in a running container.
* **docker ps –** Shows all running containers.
* **docker logs –** Fetches the logs of a container.
* **docker inspect –** Returns low-level information on a container or image.
* **docker port -** lists port mappings or a specific mapping for the container.
* **docker top –** displays the running processes of a container.
* **docker stats –** Displays a live stream of a containers resource usage statistics.
* **docker diff –** Inspects changes on a container’s filesystem.

**Example:**

In this example, we will run a Drupal container. Drupal is an open source content management platform.

 # docker run -dit –name drupal\_container -p 8080:80

drupal

The options **-dit** tell Docker to make the container run as daemon in interactive mode and provide a tty.

The option **–name** assign a name to the container.

The option **-p** publishes a container’s port to the host.

Using docker’s **ps** command we see that the container is running:

 # docker ps

Then, you can access it via http://localhost:8080 or http://host-ip:8080 in a browser.

**Docker Hub and Registry**

**What is Docker Hub?**

Docker Hub is a SaaS platform provided by Docker that makes public and private Docker images available to Docker users. The Docker Hub URL is: <https://hub.docker.com/>

**Docker Hub features:**

1. central repository for public/private images
2. automatically creates new images on changes in source code repository
3. creates work groups to manage access to image repositories
4. lifecycle workflow automation

**Example:**

Search a MySQL image from public repository (Docker Hub).

 # docker search mysql

Pull the image from Docker Hub:

# docker pull mysql

**Docker Private Registry**

A Docker Private Registry is a storage system used to store the private images on a local machine. The registry server is a container provided by docker. Our recommendation is to use registry version 2. This version is a major advance in security, reliability, and efficiency over registry version 1.

**The advantages of a private Docker registry:**

1. Securely deployed in your local environment.
2. Reduce latency for pushing and pulling images, leading to faster deployments.

Example: how to start local registry with specific data store location:

 # docker run -d -p 5000:5000 –restart=always –name registry -v /data/var/lib/registry registry:2

By default, the registry data is persisted as a docker volume on the container file system. If you want to store your volume location were a specific place, use **-v** (volume) parameter.

**Docker pull/push flow**

1. The command # docker run -it ubuntu:16.04 is issued in the CLI of the Docker client
2. Next, an API call is sent from the Docker client to the Docker daemon
3. The Docker daemon searches Docker Hub for Ubuntu images, provided the image does not exist on host
4. The Docker daemon downloads Docker image from Docker Hub